Modeling and Generation of Secure Component Communications in AUTOSAR*

Cinzia Bernardeschi
Department of Information Engineering
University of Pisa, Italy
cinzia.bernardeschi@ing.unipi.it

Marco Di Natale
TeCIP Institute
Scuola Superiore Sant'Anna, Pisa, Italy
marco.dinatale@sssup.it

Gianluca Dini
Department of Information Engineering
University of Pisa, Italy
gianluca.dini@ing.unipi.it

Dario Varano
Department of Information Engineering
University of Pisa, Italy
dario.varano@for.unipi.it

ABSTRACT

The AUTOSAR standard acknowledges the need for improved security in automotive communications by providing a set of standard modules for encryption and authentication, to ensure confidentiality and integrity. However, these modules are not currently matched by corresponding models for security at the application level, and their use is somewhat in violation of the established AUTOSAR methodology that relies on code generation from high level specifications for all the communications and scheduling features. In this paper we present modeling extensions and code generation features, developed in the context of the EU project Safure, that aim at bridging this gap.

CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Redundancy;* •**Networks** → Network reliability;

Keywords

AUTOSAR; security;

1. INTRODUCTION

Modern motor vehicles contain an increasing number of Electronic Control Unit (ECU), executing vehicle functions (including automated driving and active safety). ECUs are interconnected by wired networks such as the Controller Area Network (CAN) or Ethernet, and wireless connectivity is increasingly used for additional flexibility and bandwidth for features like keyless entry, diagnostic, and entertainment. This increased connectivity leads to an increasing number of potential cyber security threats. Stephen Checkoway et al. demonstrate that remote exploitation is feasible via a broad range of attack vectors including CD players and Bluetooth [9]. Thus security in automotive is becoming increasingly important and should be taken into account from the early stages of software development. In this work, we focus on the AUTOSAR standard, which is the standard for the modeling and development of SW components in the automotive industry.

AUTOSAR defines a modeling language for automotive application components and a set of standard services integrated in a common architecture framework. As part of recent extensions and developments, AUTOSAR now offers a set of security-related services, including the Crypto Service Manager (CSM) [5], which provides a set of security functionalities (such as hash function, encryption, MAC computation) that can be used by software components, and the Secure On-board Communication (SecOC) component for the authentication of messages exchanged over networks.

Currently, the steps that the developers have to follow in order to specify a communication making use of the AUTOSAR security mechanisms are complex and error prone. AUTOSAR does not provide any means to specify high level security requirements at the level of the application component models, but rather requires the application developers to directly use the standard services of the CSM. This is in contrast with the AUTOSAR approach for scheduling and communication. Application components are not allowed to use the basic software services for communication over network buses or the operating system services. Rather, high level specifications are provided and, based on them, code is automatically generated by tools to define the threads and invoke the appropriate network communication services. In this work, we propose a similar approach to handle the needs of confidentiality and integrity, allowing the developers to specify a security level (confidentiality and/or integrity) for the communication of the application components (in the AUTOSAR model). In addition, we developed a prototype

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

SAC 2017,April 03-07, 2017, Marrakech, Morocco

http://dx.doi.org/10.1145/3019612.3019682

model and code generation tool, which automatically synthesizes the right services to use to achieve the security level specified by the developers. This work was conducted as part of the European project SAFURE [3] (Safety And Security By Design For Interconnected Mixed-Critical Cyber-Physical Systems).

1.1 Background

Security is increasingly considered in the development of automotive electronics systems, both by enforcing software programming standards that prevent software defects that may enable cyber-attacks, as well as by implementing security mechanisms for secure communication.

Several publications demonstrate the possibility of attacks on messages and nodes in embedded automotive networks [9][12]. Countermeasures for secure communications on the CAN bus have been proposed including [13]. Factors like Required Resources and Required Know-How have been considered in the SAHARA (Security-Aware Hazard Analysis and Risk Assessment) method for defining threats criticality [15].

With respect to modeling, a UML-based approach for security and engineering processes is MDS [7]: a modular methodology for combining languages for modeling system designs with languages for modeling security. It uses transformations on security-enhanced models in order to generate implementations. SecureUML [14] and umlsec [11] are based on this approach. SecureUML can be used to model systems with role-based access control policies. Saadatmand et al. have proposed and discussed benefits of extending MARTE (an extension of the UML modeling language for real-time and embedded systems design) with UML stereotypes to specify confidentiality properties of message communication [16]. Later they have proposed concepts and mechanisms that allow to model confidentiality and authentication requirements at a higher abstraction level and automatically derive the corresponding security implementations [17]. In the SeSaMo project, Gashi et al. explore how security and safety in embedded systems are affected by redundancy and diversity [10].

In a preliminary version of this work [8] we presented the main ideas behind our AUTOSAR modeling approach. Here, we introduce a methodology for the automatic generation of secure components and Run-Time Enviroment (RTE) code.

1.2 The AUTOSAR Modeling and Development Process

AUTOSAR defines a three-layered architecture consisting of: *Application layer*, *Runtime Environment (RTE) layer*, and *Basic Software (BSW) layer* [1]. The application layer contains the Software Components (SWCs) developed for the automotive system functions by suppliers. The RTE layer is a middleware layer, automatically generated by tools and providing an actual implementation of the communication services requested by the software components, the definition of the threads and their scheduling. Finally, the BSW provides basic platform services (including drivers, the communication stack and the operating system), and basic

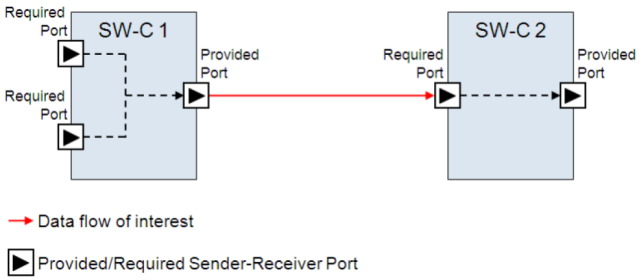

Figure 1: Example of AUTOSAR port interaction.

software modules to software components.

The AUTOSAR application model consists of a set of application software components that communicate using ports that express client-server relationships (in this case the port is typed by an operation interface) or send-receive data interactions, where the port is typed by a data interface consisting of a set of typed data items. An example of an AUTOSAR communication is shown in Figure 1, in which two components SW-C1 and SW-C2 exchange data over a pair of send-receive port (indicating a data item that is provided on one side, required on the other). The internal behavior of AUTOSAR components consists of runnables or functional units, represented by a function entry point. Each runnable indicates the port is uses. Internally, the runnable code accesses the ports through a set of standard API for port communication and port service request (denoted as RTE services). A set of events triggering the execution of runnables completes the set of the main modeling entities.

The coder that provides the internal representation of the runnable behavior has the responsibility of using the standard API for communication over the ports. This API is independent from the component placement and translated by the RTE code into the actual communication mechanism. An example of RTE function is the API for writing data over a send-receive port:

```
Std_ReturnType  Rte_Write_<port>_<comp>(..,data,..)
```

In general, an AUTOSAR software component cannot directly access BSW modules [1]; software components communicate between each other and with BSWs only through the RTE. The RTE is an abstraction layer that is generated by tools and provides for communication and scheduling. The RTE provides the actual implementation of the RTE port communication primitives (using network services or direct local communication primitives) and has the responsibility of organizing the execution of runnables in threads. In the architecture framework, the RTE code uses the services of the Basic Software (BSW), that is, the set of the drivers and the operating system. This model allows software components to be developed independently of the underlying hardware, which means that they are transferable and reusable.

2. SECURITY IN AUTOSAR

The AUTOSAR standard provides a number of mechanisms and modules that can be used by the software developers to build safe and secure software.

- Secure On-board Communication (SecOC) [6]: the purpose of the SecOC module is to allow the transmission of secured data between two or more peers over a network;
- Crypto Abstraction Library (CAL) [4]: the CAL provides other BSW modules and application software components with cryptographic functionalities. As a library, the CAL is not related to a special layer of the AUTOSAR Layered Software Architecture.
- CSM: the CSM is an AUTOSAR service which provides cryptographic functionalities to other software modules, based on a software library or based on a hardware module (HSM).

The AUTOSAR CSM and CAL specifications define the same cryptographic functionalities, including hash calculation; the generation and verification of message authentication codes; random number generation; encryption and decryption using symmetrical and asymmetrical algorithms; the generation and verification of digital signature and key management operations.

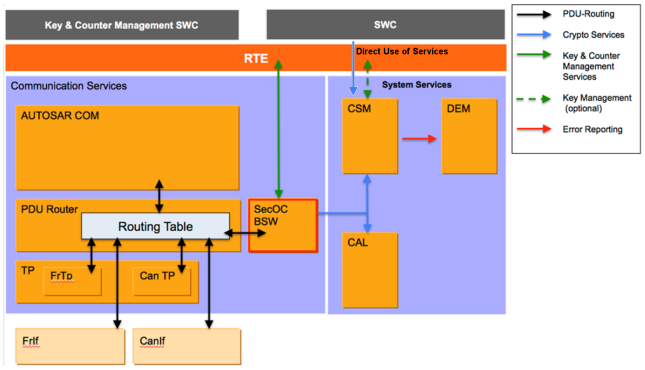

Figure 2: AUTOSAR standard modules for security.

The Crypto Service Manager [5] is an AUTOSAR service, part of the AUTOSAR service layer, as shown in Figure 2. The CSM provides cryptographic services to SecOC for securing/unsecuring data over the network. The CSM offers also a standardized interface to higher software layers to access cryptographic functionalities. The CSM service can be configured to provide the services that are strictly needed and the selection of synchronous or asynchronous processing. The CSM also controls the concurrent, multiple and synchronous/asynchronous access of one or multiple clients to one or more services (i.e. it performs buffering, queuing, arbitration, multiplexing).

A sample list of CSM interfaces is: CsmMacGenerate, CsmMacVerify, CsmSymEncrypt, CsmSymDecrypt, CsmSymBlockEncrypt and CsmSymBlockDecrypt.

Service name	Csm_MacGenerateStart	
Syntax	`Std_ReturnType Csm_MacGenerateStart( Csm_ConfigType cfgId, const Csm_SymKeyType *keyPtr)`	
Parameters (in)	cfgId	Identifier of the CSM module configuration
	keyPtr	Pointer to the key structure, used by the MAC generation algorithm
Return values	Std_ReturnType	E_OK: request successful E_NOT_OK: request failed CSM_E_BUSY: request failed, service is still busy
Description	The interface is used to initialize the MAC generate service of the CSM module.	

Figure 3: Example of the CSM function for the initialization of the Mac Generation service.

The interfaces specified above are implemented using the streaming approach of start, update and finish functions. For example the CsmMacGenerate interface is: Csm_MacGenerateStart(), Csm_MacGenerateUpdate(), Csm_MacGenerateFinish().

Figure 3 shows the detailed description of the CSM service that is supplied for the generation of message authentical codes (MACs).

The services offered by the CSM can be used locally only. If remote access is needed, it is up to the provider to specify, implement and provide some proxy for access to CSM. If the CSM is used remotely (via a proxy), it must be taken into account that this raises security implications: any communication between ECUs is done via unprotected communication buses (e.g. CAN). Unencrypted data, not yet signed data, would be transmitted and might become stolen or manipulated.

CSM services use cryptographic algorithms that are implemented using a software library or cryptographic hardware modules - both are out of scope and not specified by AUTOSAR. Note that there is no user management in place, which prevents unauthorized access to any of the CSM services. This means, that if any access protection is needed, it must be implemented by the application; access protection is not target of the CSM.

3. AUTOSAR EXTENSIONS FOR MODELING SECURE COMMUNICATIONS

The AUTOSAR security model defines the mechanisms that should be implemented as part of the BSW layers and mostly disregards the application level, that is, how the designer of an application with security concerns should specify that its communications need to be suitably protected. These requirements should be applicable to the elements of the model, that is to runnables and any port interaction, of sender/receiver or client server type, as well as on events.

In this section we summarize the main modeling security concepts defined in the SAFURE project [3] and realized as stereotypes extending the AUTOSAR implementation provided by the IBM Rhapsody tool. The security metamodel defined in Safure builds upon the results of the EVITA project [2] and, in particular, its metamodels. The EVITA

project defines a number of models, many of which refer to basic concepts such as trust and attacks, but essentially focuses on intra-platform security issues whereas, in contrast, SAFURE focuses on the security of in-vehicle communication and its relationship with performance and safety. The main concepts of the SAFURE security modeling at the functional level are shown in Figure 4. The project also defines concepts for the in-vehicle secure communication model that are not relevant for this work.

SAFURE defines two main concepts at the functional level: the trust level of a functional element and the security requirements of communications between elements. A functional element, either a component or a runnable (an executable function), 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 its functionality. A trust specification consists of: i) a *trust specification identifier* (trustSpecID), which identifies the specification, and ii) 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, 1 being the highest trust level and 5 the basic one. The notion of trustLevel recurs in other projects. For example, it is similar to the attack potential defined in EVITA as a measure of the effort required to create and carry out an attack.

In the SAFURE metamodel, security requirements can be specified on communications among elements at the functional level. The security requirement can apply to an entire data interface (all the elements in it), to a single data item within an interface, to all the communications outgoing from a component, or to an interaction between a sender and a receiver runnable (possibly further qualified by the communication interface or data item). A security requirement defines a required amount of trust in terms of a system-specific criterion and a minimum level of an associated quality metric that is necessary to meet one or more trust policies. SAFURE defines a secure communication requirement by four attributes: i) a *security requirement identifier*, ii) a *security level*, iii) a *confidentiality level indicator* (defined as the required key length), and, iv) an *integrity level indicator* (defined as the required tag length).

We defined a Rhapsody implementation of AUTOSAR extensions for the elements represented in Figure 4. AUTOSAR is a closed standard and cannot be extended as such. However, a prototype implementation of a possible extension is possible when using the IBM Rhapsody tool. Rhapsody provides AUTOSAR modeling as a profile extension of its UML metamodel. If additional UML stereotypes are defined on top of the base entities that are used to derive the AUTOSAR base entities, then they they will also be applicable to the AUTOSAR concepts, extending them. Hence, we provided a prototype AUTOSAR extension implementaion as a profile in Rhapsody. The definition of stereotypes is the only feature that possibly separates Rhapsody from other tools. The processing of the AUTOSAR model, however, is absolutely general, since it makes use of the standard ARXML model output format. The ARXML model output contains full information on all the model features (including the use of service calls and their parameters).

The Rhapsody implementation first requires the definition of stereotypes for trust specifications and security requirements.

The security requirements that are added to the communication are realized by stereotyping a constraint. According to the abstract metamodel of Figure 4, we defined a stereotype SecurityRequirement, with the properties of the metamodel of Figure 4. Similarly, a stereotype TrustSpecification is defined (details on the stereotypes definitions are omitted for space reasons). Constraints that are stereotyped as SecurityRequirement can be added in the Rhapsody AUTOSAR model to the communication elements (such as a pair of sender and receiver runnables) using dependencies. In the case of a pair of sender and receiver runnables, the (single) sender needs to be identified by stereotyping the dependency with the sender runnable.

Of course constraints can also be added to components, interface definitions and ports, realizing the desired flexibility as in the abstract metamodel. A sample model showing the applicabiity of the proposed extensions is represented in Figure 5. It consists of two components with two runnables accessing as reader and writer a pair of sender and receiver ports. The ports are typed by a data interface with a data item. In our example, a TrustSpecification is applied to the component SWC1 and the runnable run21 (on the left of Figure 5). In Figure 5, SecureCommunication_if, SecureCommunication_de, SecureCommunication_swc and SecureCommunication_re represent the security requirement applied to an entire data interface, to a single data item within an interface, to all the communications outgoing from a component, and to an interaction between a sender and a receiver runnable, respectively.

4. CSM LIBRARY IMPLEMENTATION

To support our experimental work and test the execution of the generated code, we implemented a CSM library compliant with the AUTOSAR standard. The library has been implemented in C-language and relies on OpenSSL. The services implemented are the symmetrical encryption/decryption and the MAC generation/verification, which allow us to fulfil the confidentiality and integrity security level, respectively. The CSM supports the processing of a single instance of each service at a time. Each service makes use of a configuration structure, where all the information regarding the current request are stored. When a new instance of a service is created, a configuration structure is allocated. New service requests cannot be served until the previous instance is completed and the structure deallocated. The CSM is based on the streaming approach with start, update and finish functions. Although it is possible to configure synchronous or asynchronous job processing, in this work only synchronous services are implemented, and the interface functions immediately compute the result. An example of implementation of a CSM function is shown in Figure 6.

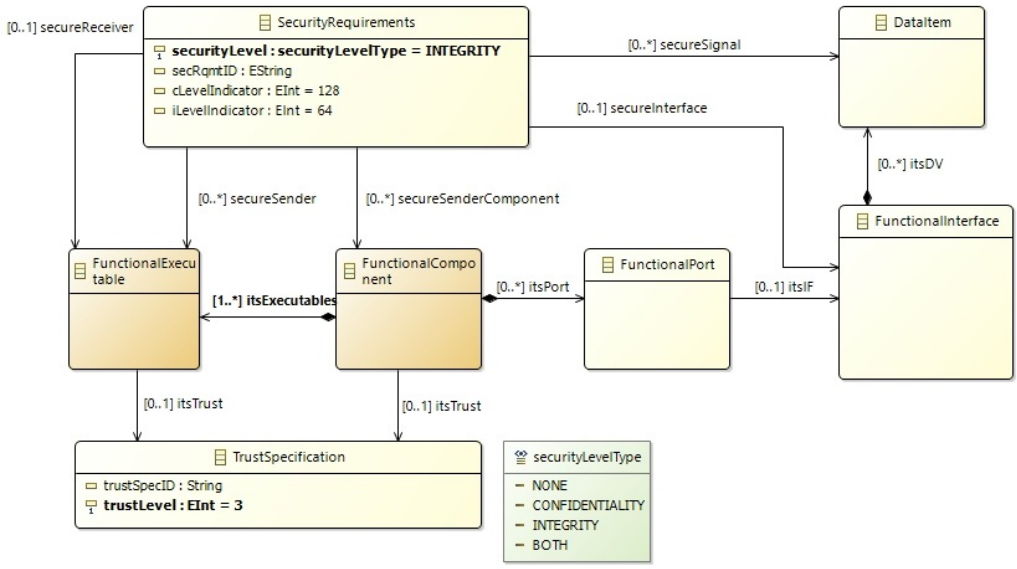

Figure 4: The Safure metamodel of extensions for the definition of communication security.

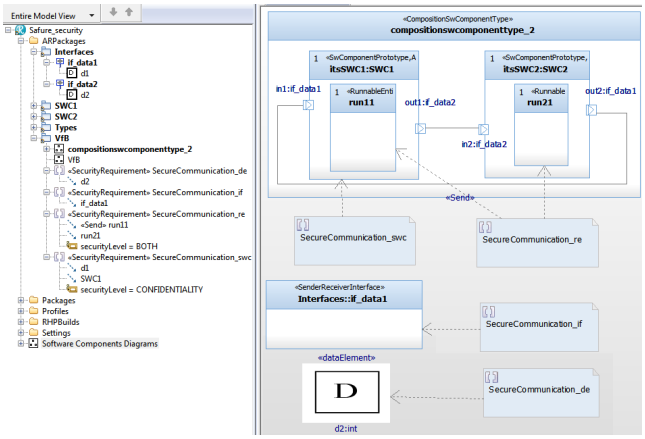

Figure 5: Example of use of the Safure extensions.

5. AUTOMATIC GENERATION OF COMPONENTS FOR SECURITY

The definition of an AUTOSAR model with security extensions applied to communication links or to the ports involved in the communication results in the production (by model export) of a model description file in the standard ARXML (XML) format.

In our prototype implementation, developers can insert the security specification by using an AUTOSAR-compliant tool like Rhapsody, and then export the system as ARXML; or insert the security tag directly in the ARXML file.

In the ARXML, the security levels are expressed as a tag in the form of a pair [name; value] within the description field of the two ports involved in the communication and they can assume the following values:

- `SecurityNeeds=INTEG`, which stands for "integrity": in a communication between two entities (A and B), if A sends a message to B, B is able to verify that the received message was not altered by an external entity;
- `SecurityNeeds=CONF`, which stands for "confidentiality": in a communication between two entities (A and B), a third (non-authorized) entity (C), is not able to understands the content of the message exchanged between A and B;
- `SecurityNeeds=BOTH`, which means that both (integrity and confidentiality) are required.

Our tool consists of a Python script that parses the ARXML file and automatically generates the required elements based on the specified security level. The security requirements are fulfilled by using the services provided by the CSM. Client ports and interfaces to the CSM services are generated automatically. The script allows the developers to choose if these elements are added directly within the component, which requires a specific security level, or if they are to be added within a new purposely generated component, which acts as a proxy or filter. For this purpose, we define an additional tag in the description field of the component:

- `NewComponent=TRUE`: for each secure component the script adds a new *filter* component, which transparently encrypts and decrypts sent and received data. This is useful when dealing with legacy components that cannot be modified.
- `NewComponent=FALSE`: (default value) the required security elements are added directly within the component which requires security services.

We show an example of automatic generation applied to a sample model of an active safety system, as outlined in Figure 7.

```
Std_ReturnType
   Csm_SymEncryptStart (Csm_ConfigIdType cfgId,  const  Csm_SymKeyType *keyPtr,
                        const uint8* InitVectorPtr,  uint32 InitVectorLength) {
        // Check parameters;
        // Initialize configuration structure;
        // call the underlying cryptographic primitive
        return_value = Cry_SymEncryptStart((const void *)csm_encrypt->PrimitiveConfigPtr,
                                           keyPtr, InitVectorPtr, InitVectorLength);
        return return_value;
}
```

Figure 6: A sample function of CSM library

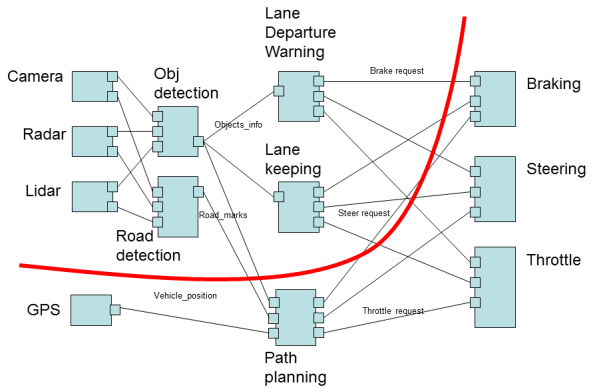

Figure 7: Braking system model overview.

The system consists of a portion of an active safety subsystem, in which multiple sensor components are feeding information on the environment around the car to subsystems dedicated to the detection of objects and the road profile. Following the sample model of two active safety functions (Lane departure warning and Lane keeping) and a Path planning component, a subset of the typical actuation systems of a car is represented (braking, steering, and throttle).

Confidentiality and integrity requirements are added to the communications between the GPS and the PathPlanning components, and between the latter and all the actuators, by the means of security tags. Other communications are only characterized by integrity requirements.

After the system is modelled using Rhapsody, as shown in Figure 8, the ARXML file exported by Rhapsody, is processed by our script, which generates a new ARXML file.

Consider the subsystem composed by GPS, PathPlanning and actuators (components below the red line in Figure 7).

If NewComponent=TRUE, the resulting model is shown in Figure 9 where filters are shown in grey.

To outline the filter automatic generation process, consider the PathPlanning component and the corresponding PathPlanningFilter (in Figure 9).

To ensure security for the communication through the receiver port pathRec3 the filter adds a Receiver port pathRec3_r and a Sender port pathRec3_s. The filter receives secured data from pathRec3_r, decrypts and forwards it to PathPlanning through the sender port pathRec3_s. Similarly, for port pathSend1 the PathPlanning filter adds a receiver port pathSend1_r and a sender port pathSend1_s. The filter receives unsecured data from PathPlanning through the port pathSend1_r, encrypts, and forwards it to (the filter of) Braking component, through port pathSend1_s. The filter has client ports to call the CSM functions for integrity (MAC) and confidentiality (symmetric encryption), shown on the right hand side of the filter component.

If NewComponent=FALSE, the resulting model is shown in Figure 10. Client ports are added as shown in the figure. The component uses these ports to call cryptographic functionalities through the CSM interfaces and implement the security level.

6. AUTOMATIC GENERATION OF SECURE RTE CODE

Consider the sender-receiver transmission of data elements over the GPS - PathPlanning communication in the sample model.

Depending on the allocation of the components on the system processor, specified in the AUTOSAR model allocation, the implementation of the RTE communication API can be of different types.

In the case of intra-ECU communication, the data trasmission is performed by a write on a global variable (shared memory), no encryption is required (we assume the ECU is trusted), and no security elements are added by our script. For inter-ECU communications, the data transmission occurs over a communication bus (in our example, a CAN bus, but this does not affect the generated code), and security statements are added by the script in the prototype implementation of the RTE write.

To show an example of a code generated for the automatic call of the encryption functions, we operate on a configuration in which the GPS and PathPlanning subsystems reside on different ECUs. The example only details the code generation features that are added to satisfy the confidentiality requirements (encryption). The integrity requirements are satisfied by the internal implementation of the AUTOSAR COM function, by invoking the services of the SecOc. The code generated for the runnable operation that writes into the gpsPort in case of NewComponent=FALSE is:

```
Std_ReturnType
```

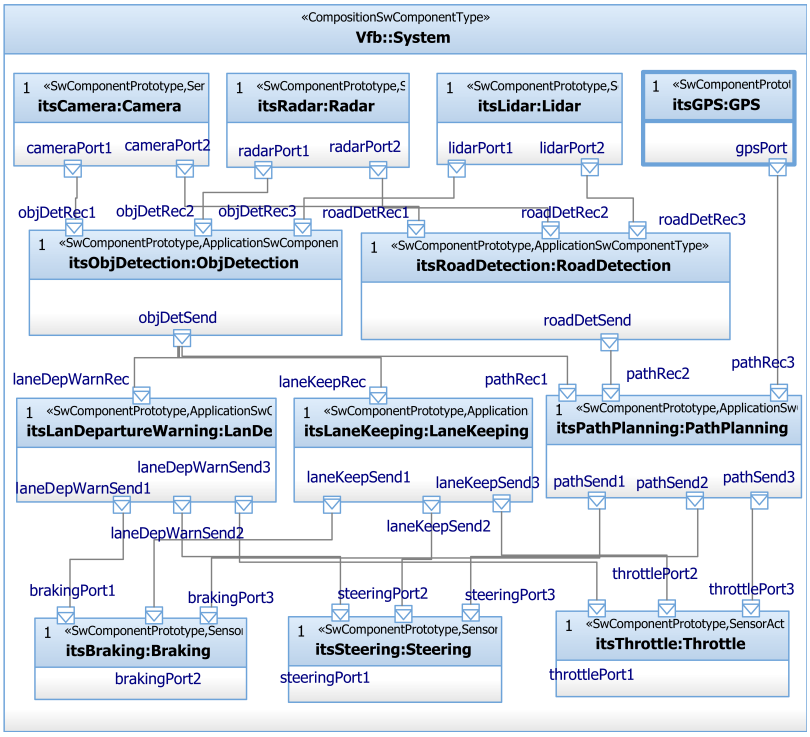

Figure 8: Braking system modeled on IBM Rational Rhapsody.

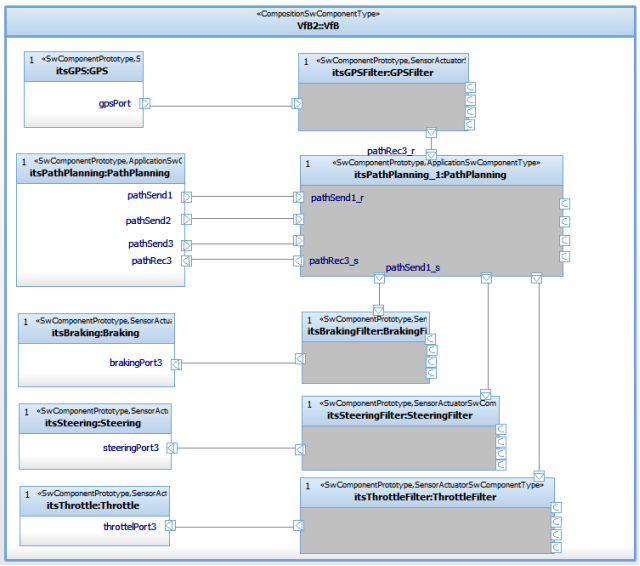

Figure 9: Braking system after security tags processing (NewComponent=TRUE).

```
  Rte_Write_gpsPort_GPS(uint32 datum) {
...
Csm_SymEncryptStart(cfgId, ..., sizeof(uint32));
Csm_SymEncryptUpdate(cfgId, datum_buf, ... );
Csm_SymEncryptFinish(cfgId, ...);
res = Com_SendSignal(sigID, datum_buf);
 ...                          // AUTOSAR COM
```

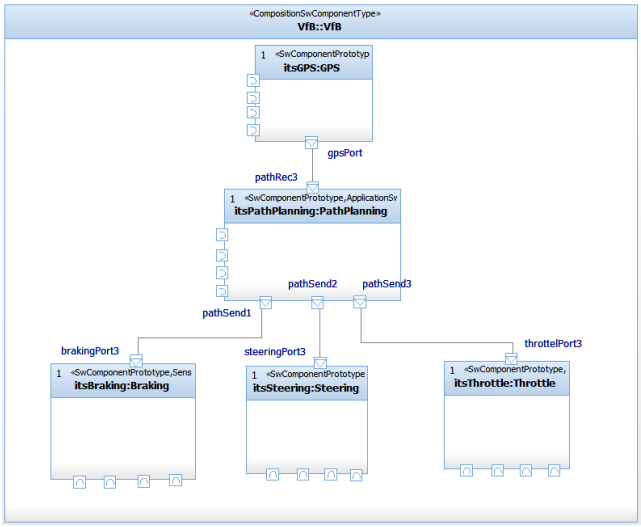

Figure 10: Braking system after security tags processing (NewComponent=FALSE).

```
return res;
}
```

The code implementation of the RTE function performs, transparent to the user, the encoding of the data value and then its transmission using the higher level communication functions prescribed by AUTOSAR (in its COM layer).

In the case the user specifies NewComponent= TRUE a filter

component is automatically added. In this case the code generated for the RTE Write function performs a simple copy into the buffer variable that is shared with the filter component

```
Std_ReturnType
  Rte_Write_gpsPort_GPS(uint32 datum) {
...
Rte_RxBuf = datum;
Rte_Write_Port0_GPSFilter(Rte_RxBuf);
...
return RTE_E_OK;
}
```

The encryption code in this case is added to the communication outgoing from the filter, as shown below for the function `Rte_Write_Port0_GPSFilter(...)`:

```
Std_ReturnType
   Rte_Write_Port0_GPSFilter(uint32 datum)  { ...
Csm_SymEncryptStart(cfgId, ..., sizeof(uint32));
Csm_SymEncryptUpdate(cfgId, datum_buf, ... );
Csm_SymEncryptFinish(cfgId, ...);
res = Com_SendSignal(sigID, datum_buf);
 ...                          // AUTOSAR COM
return res;
}
```

This approach enhances the security of AUTOSAR design because it is in the direction of security by design. On a security annotated AUTOSAR model, security formal properties can be proved and security components can be automatically generated so alleviating the developer's task and reducing software vulnerabilities.

7. CONCLUSIONS

In this paper we present a methodology for the specification and the automatic generation of security features for the communication between AUTOSAR components. The framework consists of modeling extensions that allow AUTOSAR designers to add a security specification to the model of the communication among components and a code generation tool. The modeling extensions have been implemented (in prototype form) in Rhapsody, and the code generation tool takes as input the model export and generates filter components that perform the required encryption or an implementation of the communication API that encrypts the data automatically, without manual coding by the user.

8. ACKNOWLEDGMENTS

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.

9. REFERENCES

[1] AUTOSAR, (http://www.autosar.org/).

[2] EVITA - E-safety vehicle intrusion protected applications, Seventh Research Framework Programme of the European Community, Project reference: 224275. http://evita-project.org/.

[3] SAFURE - Safety And Security By Design For Interconnected Mixed-Critical Cyber-Physical Systems, horizon 2020, project reference: 644080. https://safure.eu//.

[4] AUTOSAR. *AUTOSAR Specification of Crypto Abstraction Library: AUTOSAR Release 4.2.2.*

[5] AUTOSAR. *AUTOSAR Specification of Crypto Service Manager: AUTOSAR Release 4.2.2.*

[6] AUTOSAR. *AUTOSAR Specification of Module Secure Onboard Communication: AUTOSAR Release 4.2.2.*

[7] D. Basin, J. Doser, and T. 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.

[8] C. Bernardeschi, G. Del Vigna, M. Di Natale, G. Dini, and D. Varano. Using autosar high-level specifications for the synthesis of security components in automotive systems. In *Intl. Work. on Modelling and Simulation for Autonomous Systems*, pages 101–117. Springer, 2016.

[9] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

[10] I. Gashi, A. Povyakalo, L. Strigini, M. Matschnig, T. Hinterstoisser, and B. Fischer. Diversity for safety and security in embedded systems. In *Proceedings of the IEEE Intl. Conf. on Dependable Systems and Networks*, pages 1–2, 2014.

[11] J. Jürjens. UMLsec: Extending UML for secure systems development. In *UML 2002—The Unified Modeling Language*, pages 412–425. Springer, 2002.

[12] K. Koscher, Czeskis, et al. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462. IEEE, 2010.

[13] C.-W. Lin and A. Sangiovanni-Vincentelli. Cyber-security for the Controller Area Network (CAN) communication protocol. In *2012 International Conference on Cyber Security*, pages 1–7. IEEE, 2012.

[14] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML 2002–The Unified Modeling Language 2002*, pages 426–441. Springer, 2002.

[15] G. Macher, M. Stolz, E. Armengaud, and C. Kreiner. Filling the gap between automotive systems, safety, and software engineering. *e & i Elektrotechnik und Informationstechnik*, 132(3):142–148, 2015.

[16] M. Saadatmand, A. Cicchetti, and M. 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] M. Saadatmand and T. Leveque. Modeling security aspects in distributed real-time component-based embedded systems. In *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*, pages 437–444. IEEE, 2012.