Evaluating the Impact of Integrating a Security Module on the Real-Time Properties of a System [⋆]

Sunil Malipatlolla[1] and Ingo Stierand[2]

[1] OFFIS - Institute for Information Technology
Oldenburg, Germany
sunil.malipatlolla@offis.de
[2] Carl von Ossietzky Universität Oldenburg
Oldenburg, Germany
stierand@informatik.uni-oldenburg.de

Abstract. With a rise in the deployment of electronics in today's systems especially in automobiles, the task of securing them against various attacks has become a major challenge. In particular, the most vulnerable points are: (i) communication paths between the Electronic Control Units (ECUs) and between sensors & actuators and the ECU, (ii) remote software updates from the manufacturer and the in-field system. However, when including additional mechanisms to secure such systems, especially real-time systems, there will be a major impact on the real-time properties and on the overall performance of the system. Therefore, the goal of this work is to deploy a minimal security module in a target real-time system and to analyze its impact on the aforementioned properties of the system, while achieving the goals of secure communication and authentic system update. From this analysis, it has been observed that, with the integration of such a security module into the ECU, the response time of the system is strictly dependent on the utilized communication interface between the ECU processor and the security module. The analysis is performed utilizing the security module operating at different frequencies and communicating over two different interfaces i.e., Low-Pin-Count (LPC) bus and Memory-Mapped I/O (MMIO) method.

Keywords: Security, FPGA, Interfaces, Real-Time Systems

1 Introduction and Related Work

Real-time applications such as railway signaling control and car-to-car communication are becoming increasingly important. However, such systems require high quality of security to assure the confidentiality and integrity of the information during their operation. For example, in a railway signaling control system, the

[⋆] This work was supported by the Federal Ministry for Education and Research (BMBF) under support code 01IS11035M, 'Automotive, Railway and Avionics Multicore Systems (ARAMiS)

control center must be provided with data about position and speed of the approaching train so that a command specifying which track to follow may be sent back. In such a case, it must be assured that the messages exchanged between the two parties are not intercepted and altered by a malicious entity to avoid possible accidents. Additionally, it is mandatory to confirm that the incoming data to the control center is in fact from the approaching train and not from an adversary. Similar requirements are needed in a car-to-car communication system. Thus, there is a need to integrate a security mechanism inside such systems to avoid possible attacks on them. Furthermore, above considered systems are highly safety relevant, thus the real-time properties typically play an important role in such systems. In general, the goal of a real-time system is to satisfy its real-time properties, such as meeting deadlines, in addition to guaranteeing functional correctness. This raises the question, what will be the impact on these properties of such a system when including, for example, security as an additional feature? To understand this, we integrate a minimal security module in the target real-time system and evaluate its impact on the real-time properties as a part of this work.

There exist some work in the literature which addresses the issue of including security mechanisms inside real-time applications. For example, Lin et al. [9] have extended the real-time schedulability algorithm Earliest Deadline First (EDF) with security awareness features to achieve a static schedulability driven security optimization in a real-time system. For this, they extended the EDF algorithm with a group-based security model to optimize the combined security value of selected security services while guaranteeing schedulability of the real-time tasks. In a group-based security model, security services are partitioned into several groups depending on the security type and their individual quality so that a combination of both results in a better quality of security. However, this approach had a major challenge as how to define a quality value for a certain security service and to compute the overhead due to those services. In another work, authors Marko et al. have designed and implemented a vehicular security module, which provides trusted computing [12] like features in a car [13]. This security module protects the in-vehicle ECUs and the communication between them, and is designed for a specific use in e-safety applications such as emergency breaking and emergency call. Further, the authors have given technical details about hardware design and prototypical implementation of the security module in addition to comparing its performance with existing similar security modules in the market. Additionally, the automotive industry consortium, autosar, specified a service, referred to as Crypto Service Manager (CSM), which provides a cryptographic functionality in an automobile, based on a software library or on a hardware module [2]. Though the CSM is a service based on a software library, it may be supported by cryptographic algorithms at the hardware level for securing the applications executing on the application layer. However, to the best of our knowledge, none of the aforementioned approaches addresses the issue of analyzing the impact on the real-time properties of a system when integrating a hardware security module inside it.

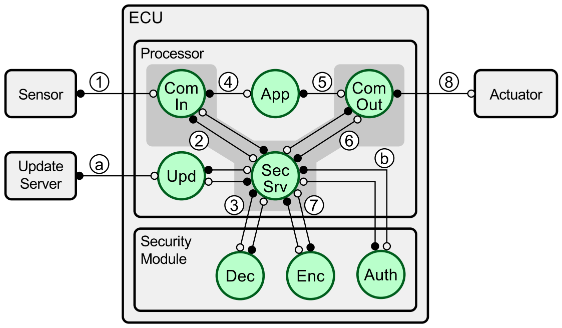

Fig. 1. System Scenario

The rest of the paper is organized as follows. Section 2 gives a detailed description of the system under consideration and its operation with the security module internals and the adversarial model. Section 3 evaluates the system operation with three different test scenarios, and presents the corresponding analysis results. Section 4 concludes the paper and gives some hints on future work.

2 System Specifications

2.1 System Model

The system under consideration is depicted in Figure 1. It comprises of a sensor, an actuator, an electronic control unit (ECU) with a processor & a security module, and an update server. The system realizes a simple real-time control application, where sensor data are processed by the control application in order to operate the plant due to an actuator. The concrete control application is not of interest in the context of this paper. It might represent the engine control of a car, or a driver assistant system such as an automatic breaking system (ABS).

The scenario depicted in Figure 1 consists of the following flow: `Sensor` periodically delivers data from the plant over the bus (1), which is in an encrypted form to avoid its interception and cloning by an attacker. The data is received by the input communication task `ComIn`, which is part of the operating system (OS). Each time the input communication task receives a packet, it calls the security service (`SecSrv`), which is also part of the OS, for decryption of the packet (2). The security service provides the hardware abstraction for security operations, and schedules service calls. The decryption call from the communication task is forwarded to the security module (3), which processes the packet data. The cryptographic operations of the security module modeled by `Dec`, `Enc`, and `Auth` are realized as hardware blocks. The decrypted data is sent back to the security service, which is in turn returned to the `ComIn`. Now the data is ready for transmission to the application (4), which is modeled by a single task `App`. The application task is activated by the incoming packet, and processes the sensor data. The controller implementation of the task calculates respective actuator data and sends it to the communication task ComOut (5) for transmission to the Actuator. However, before sending the data to the Actuator, the communication task again calls the security service (6), which in turn accesses the security module for data encryption (7). After Enc has encrypted the data, it is sent back to the communication task ComOut via SecSrv, which delivers the packet to the Actuator (8). It is required that the control application finishes the described flow within a single control period, i.e., before the next sensor data arrives.

Additionally, the system implements a function for software updates. To update the system with new software, the UpdateServer sends the data to the Upd task (a) via a communication medium (e.g., over Internet) to the outside. This received data must be authenticity verified and decrypted before loading it into the system. For this, the Upd forwards the data to the SecSrv, which utilizes the Auth block of the security module (b). Only after a successful authentication, the data is decrypted and loaded into the system else it is rejected.

The security module, integrated into the ECU, is a hardware module comprising of cryptographic hardware blocks for performing operations such as encryption, decryption, and authenticity verification. Though these operations are denoted as tasks in the system view, they are implemented as hardware blocks. Further, a controller (a state machine), a memory block, and an I/O interface are included inside the security module (not depicted in Figure 1). Whereas the controller executes the commands for the aforementioned cryptographic operations, the memory block acts as a data buffer. The commands arrive as requests form the SecSrv on the processor, and the responses from the security module are sent back. In essence, the SecSrv acts as a software abstraction of the hardware security module, for providing the required cryptographic operations to the executing tasks on the processor.

The security module is equipped with particular support for update functionality i.e., authenticity verification. In normal operation, the data is temporarily stored in the memory block of the security module to compare the attached Hash-based Message Authentication Code (HMAC) value by the update server with the computed HMAC value in the security module before decrypting and loading it. This kind of operation, where all update data is stored in memory of the security module, and authentication being applied at once on the data, is however not appropriate in the context of the considered real-time application. This is because, while the security module is performing authentication, other operations such as decryption of incoming sensor data or encryption of outgoing actuator data are blocked. Given this, large update data can block the device for a long time span, resulting in a violation of allowed delay by the control algorithm.

To avoid such a situation in the considered scenario, for authenticity verification, the HMAC is calculated in two steps. In the first step, a checksum of the update data is calculated using a public hash algorithm such as Secure Hash Algorithm (SHA-1). In the second step, the HMAC is computed on this checksum. The Upd task thus calculates and sends only the checksum of the data instead of whole data itself to the security module via SecSrv. Since the update process is not time critical, Upd task is executed with low priority, preventing any undesired interference with the real-time application. Therefore, only the interference for authentication of the single checksum has to be considered. However, since the update data being encrypted, Upd task needs to access the security module for its decryption. To this end, the data is split into packets and decrypted piece-wise. The impact of these operations has to be considered in the real-time analysis.

Another possibility to verify the authenticity of the update data would be to compute the HMAC iteratively within the security module. For this, the update data from the Upd task is sent to the security module in a block-by-block basis via the SecSrv. The computed HMAC on the received block is stored inside the memory block of the security module. Before sending the next block, the SecSrv checks for any pending requests for encryption or decryption operation from other high priority tasks. If there exists such a request, it is executed before sending the next block of data for HMAC computation. In order to handle this procedure, the security module should be equipped with an additional hardware block performing the scheduling of cryptographic operations. Further, the communication interface between the SecSrv and the security module has to be modified. Though, this method is currently not supported by our security module it is definitely a desired feature.

The goal of the security module is to provide a secure communication path between the sensor and the actuator and to provide authentic updates of the system. For this, the cryptographic blocks of the security module utilize standardized algorithms for providing the cryptographic operations such as encryption, decryption, hash computation, and HMAC generation & verification. Having said that, all the aforementioned cryptographic operations performed by the security module in the considered system utilize a single block cipher algorithm i.e., Advanced Encryption Standard (AES) as the base [10]. It is a symmetric key algorithm i.e., it utilizes a single secret key for both encryption and decryption operations. In addition to being standardized by National Institute for Standards and Technology (NIST), the AES-based security mechanisms consume very few computational resources, which is essential in resource constrained embedded systems.

2.2 Adversarial Model

To describe all possible attack points in the considered system, an adversarial model is formulated as depicted in Figure 2. The model highlights all the components (with a simplified ECU block) and the corresponding internal and external communication paths (i.e., numbered circles) of the original system (c.f. Figure 1). The adversary considered in the model is an active eavesdropper (c.f. Dolev-Yao Model [5]), i.e., someone who first taps the communication line to obtain messages and then tries everything in order to discover the plain text. In particular he is able to perform different types of attacks such as classical cryptanalysis and implementation attacks, as defined in taxonomy of cryptographic attacks by Popp in [11]. While classical cryptanalysis attacks include cloning by

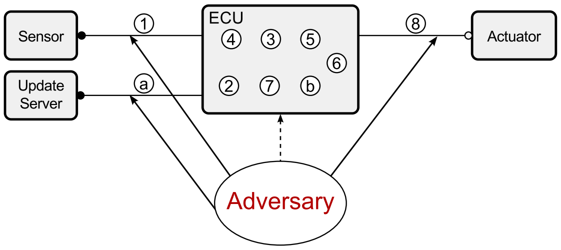

Fig. 2. Adversarial Model

interception, replay, and man-in-the-middle attacks, the implementation attacks include side-channel analysis, reverse engineering, and others.

For our analysis, we assume that the attacker is only able to perform classical cryptanalytic attacks on the external communication links (indicated by thick arrows coming from adversary) i.e., from sensor to ECU, ECU to actuator, and update server to ECU. In specific, under cloning by interception attack, the adversary is capable of reading the packets being sent to the ECU and store them for using during a replay attack. Whereas in a man-in-the-middle attack, the adversary can either pose as an ECU to authenticate himself to the update server or vice versa. In the former case, he would know the content of the update data and in the latter he may update the ECU with a malicious data to destroy the system. However, to protect the systems against the classical cryptanalytic attacks, strong encryption and authentication techniques need to be utilized. With reference to this, the security module in here provides techniques such as confidentiality, integrity, and authenticity which overcome these attacks. We rule out the possibility of attacker being eavesdropping the ECU's internal communication (indicated by a dotted arrow coming from adversary) because such an attack implies that the attacker is having a physical access to the ECU and thus control the running OS and the tasks themselves.

3 System Analysis

To analyze the impact of including the security feature on the real-time properties of the system, we consider three different test cases as detailed in the sequel. A brief description about the system set-up and the utilized tools is given before delving into the obtained results with the test cases.

The control application is executed with a frequency of 10 kHz, i.e., the `Sensor` sends each 100 μs a data packet to the ECU. The update service is modeled as a sporadic application with typically very large time spans between individual invocations. All tasks of the processor are scheduled by a fixed priority scheduling scheme with preemption, where lower priority tasks can be interrupted by higher priority tasks. Furthermore, all tasks belonging to the OS (depicted by a dark gray shaded area of Figure 1) get higher priority than the application tasks. The priorities in descending order are `ComIn`, `ComOut`, `SecSrv`,

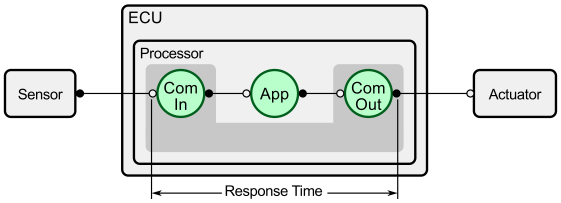

Fig. 3. Scenario 1 w/o security feature

`App`, and `Upd`. The operations of the security module are not scheduled, and the module can be considered as a shared resource. The `SecSrv` task processes incoming security operation requests for the security module in first-in-first-out (FIFO) order.

For all test cases, the processor of the ECU is a 50MHz processor (20 ns cycle time) that is equipped with internal memory for storing data and code. Internal memory is accessed by reading and writing 16 Bit words within a single processor cycle. Communication between the ECU, the sensor, and the actuator is realized by a controller area network (CAN) bus. For simplicity, we assume that all data are transferred between the processor and the CAN bus interface via I/O registers of 16 bit width, and with a delay of four processor cycles. Communication over CAN is restricted to 64 Bit user data, and we assume that this is also the size of packets transmitted between the ECU and the sensor/actuator. In order to transmit 128 Bit data as required by the operation of the security module, each transmission consists of two packets. Receiving and transmitting data thus requires 16 Byte data transfer between the CAN bus controller and the processor, summing up to 64 processor cycles (1.28 μs). Storing the packet into the OS internal memory costs additional 320 ns. Bus latencies are not further specified in our setting, as we concentrate on the timing of the ECU application.

The utilized AES algorithm inside the security module operates on 128 Bit blocks of input data at a time. Thus, all the blocks (`enc`, `dec`, and `Auth`) of the security module, operate on same data size because they utilize the same algorithm. The security module is implemented as a proof-of-concept on a Xilinx Virtex-5 Field Programmable Gate Array (FPGA) [7] platform. The individual cryptographic blocks of the security module are simulated and synthesized utilizing the device specific tools. Utilizing an operating frequency of 358 MHz for the FPGA, the execution time for each of encryption, decryption, and authentication operations is determined (by simulation) to be 46 ns for a 128 Bit block of input data. The timing parameters for other operating frequencies of the security module are obtained by simple scaling. The utilized FPGA device supports storage in the form of block RAM with 36 kb size, which is large enough to be used as memory block of the security module.

We apply timing analysis in order to find the worst-case end-to-end response time of the control application, starting from the reception of sensor data up to the sending of actuator data (shown in Figure 3). Various static scheduling

	Scenario 1	Scenario 2	Scenario 3		
		50MHz	50MHz	50MHz	358MHz
Task		LPC	LPC	MMIO	MMIO
App	50.0 μs	50.0 μs	50.0 μs	50.0 μs	50.0 μs
ComIn	1.6 μs	1.6 μs	1.6 μs	1.6 μs	1.6 μs
ComOut	1.6 μs	1.6 μs	1.6 μs	1.6 μs	1.6 μs
SecSrv	—	80 ns	80 ns	80 ns	80 ns
Comm. CPU/SM	—	1.46 μs	1.46 μs	400 ns	200 ns
Dec	—	358 ns	358 ns	358 ns	46 ns
Enc	—	358 ns	358 ns	358 ns	46 ns
Auth	—	—	358 ns	358 ns	46 ns
Response Time	53.2 μs	60.1 μs	62.0 μs	56.7 μs	54.96 μs

Table 1. Analysis Results

analysis tools are available for this task (e.g. [6, 1]). The system however is sufficiently small for a more precise analysis based on real-time model-checking [4]. To this end, the system is translated into a Uppaal model [3]. The worst-case response time is obtained by a binary search on the value range of respective model variable.

3.1 Target system without any security features

In the first scenario, which is shown in Figure 3, the communication tasks send the data directly to the control application and to the communication bus without encryption or decryption. The resulting end-to-end response time is shown in column "Scenario 1" of Table 1. As expected, the analysis shows that the execution times are simply summed up for the involved tasks since no further interferences occur in this simple setting. Indeed the situation would be different when multiple application tasks are executed on the same ECU, which would cause additional interferences.

3.2 Target system with secure communication feature

In the second scenario, only the secure communication feature between the ECU and the sensor and the actuator is enabled. The update service however is switched off. This implies that the security module has to perform only encryption and decryption but no authentication. For this scenario, we assume that the security module communicates with the processor via a Low-Pin-Count (LPC) bus [8]. LPC is a 4 Bit wide serial bus defined with a clock rate of 33 MHz. According to the specification [8], the transfer of 128 Bit data plus 16 Bit command requires about 1.46 μs, when the bus operates with typical timing parameters. Each invocation of the `SecSrv` involves a transfer of the data to or from the security module, plus the execution time of the task of 80 ns for internal copying operations. The security module operating at a clock rate of 50 MHz results in an individual execution time of 358 ns for encryption and decryption (from simulation results). With this set-up, the timing analysis shows (column "Scenario 2" of Table 1), that enabling only the secure communication feature results in a significant raise in the response time of the system i.e., about 13% more than in the previous scenario.

3.3 Target system with secure communication and secure update features

For the third scenario, both secure communication and authentic update features are enabled. This scenario has been analyzed with three different sets of timing parameters.

The first setting assumes the same parameters as for Scenario 2. Hence the security module operates at a clock rate of 50 MHz, and communicates with the processor via LPC. The results show a further raise in the end-to-end response time of the application because the update service might call the authentication service, which incurs an additional execution time to the pending encryption and decryption operations of the security module. Thus, it can be seen that the end-to-end response time is around 16% higher than in the first scenario.

For the second setting, we assume that the security module communicates with the processor via a Memory-Mapped I/O (MMIO) interface. Memory transfers are assumed to operate with 16 Bit words, and a delay of four processor cycles, resulting in transfer times of 80 ns. A transfer between the processor and the security module now sums up to 400 ns.

The final setting works with a very fast security module, and memory transfers only having two cycles delay. The security module operates at 358 MHz, which results in all cryptographic operations with 46 ns of execution time.

Surprisingly, the end-to-end response time in the second and the final setting has reduced and is only around 6.5% and 3.5% respectively. This implies that the type of communication interface between the processor and the security module has a significant impact on the resulting overall response time of the system.

In all settings, the software update function is assumed to perform the operations as discussed in Section 2. After calculating the checksum of the update data, task `Upd` sends an authentication request to `SecSrv`. When the authentication is successful (which is always true in the considered scenario), the task successively sends decryption requests for each packet of the update data, while waiting for the reply before sending a new request. The results shown in the table represent the worst-case behavior obtained with various values of the execution time (between 10 ns and 1 μs) needed by `Upd` between successive decryption requests. The impact of the operation of `Upd` remains rather small, which can be explained by the fact that the task is executed with low priority. However, the selection of the execution times was not exhaustive, and thus do not guarantee absence of race conditions. To enforce limited impact of the update function, the `SecSrv` should be modified by running with priority inheritance, where requests are executed with the same priority as the calling task. A more comprehensive analysis of this issue is subject of future work.

4 Conclusion

In this work, a real-time system integrated with a security module is analyzed to determine the impact of the latter on the worst-case response time of the system. For this, different communication interfaces such as LPC bus and MMIO method are utilized between the security module and the control unit processor of the system, for performing cryptographic operations such as encryption, decryption, and authentication. It is observed that the worst-case response time of the system is high for a slower interface (i.e., LPC) and decreases drastically for a faster interface (i.e., MMIO). Thus, when including a security mechanism in real-time systems it is necessary to consider about the type of communication interface being utilized. Though, the target system in here has a single ECU, a sensor, and an actuator, the typical systems have multiple such components which need a further investigation.

References

1. Anssi, S., Albers, K., Dörfel, M., Gérard, S.: chronVAL/chronSIM: A Tool Suite for Timing Verification of Automotive Applications. In: Proc. Embedded Real-Time Software and Systems (ERTS) (2012)
2. Autosar Organization: Specification of Crypto Service Manager. http://www.autosar.org/download/R4.0/AUTOSAR_SWS_CryptoServiceManager.pdf (2011)
3. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal 2004-11-17. Tech. rep., Aalborg University, Denmark (Nov 2004)
4. Dierks, H., Metzner, A., Stierand, I.: Efficient Model-Checking for Real-Time Task Networks. In: International Conference on Embedded Software and Systems (ICESS) (2009)
5. Dolev, D., Yao, A.C.: On the security of public key protocols. Tech. rep., Stanford University, Stanford, CA, USA (1981)
6. Hamann, A., Jersak, M., Richter, K., Ernst, R.: A framework for modular analysis and exploration of heterogeneous embedded systems. Real-Time Systems 33(1-3), 101–137 (jul 2006)
7. Inc., X.: Xilinx, http://www.xilinx.com/support/documentation/virtex-5.htm
8. Intel: Low Pin Count (LPC) Interface Specification. Intel Corp. (August 2002)
9. Lin, M., Xu, L., Yang, L., Qin, X., Zheng, N., Wu, Z., Qiu, M.: Static security optimization for real-time systems. IEEE Transactions on Industrial Informatics 5(1), 22 –37 (Feb 2009)
10. National Institute of Standards and Technology (NIST): Advanced Encryption Standard (AES) (2001)
11. Popp, T.: An Introduction to Implementation Attacks and Countermeasures. In: Proceedings of IEEE/ACM International Conference on Formal Methods and Models for Co-Design (MEMOCODE'09). pp. 108 –115 (Jul 2009)
12. Trusted Computing Group, Inc.: Trusted Platform Module (TPM) specifications. http://www.trustedcomputinggroup.org/resources/tpm_main_specification (2010)
13. Wolf, M., Gendrullis, T.: Design, implementation, and evaluation of a vehicular hardware security module. In: Kim, H. (ed.) Information Security and Cryptology - ICISC 2011, Lecture Notes in Computer Science, vol. 7259, pp. 302–318. Springer Berlin Heidelberg (2012)