# Security Improvements by Separating the Cryptographic Protocol from the Network Stack onto a Multi-MCU Architecture

Tobias Frauenschläger(✉), Sebastian Renner, and Jürgen Mottok

Laboratory for Safe and Secure Systems (LaS$^3$), Technical University of Applied Sciences Regensburg, 93053 Regensburg, Germany
{tobias.frauenschlaeger,sebastian1.renner, juergen.mottok}@oth-regensburg.de

**Abstract.** The number of IoT devices in SCADA and ICS systems is rising quickly, especially in the domain of critical infrastructures. But these kinds of systems are performing mission critical tasks like controlling devices in industrial facilities or substations in the smart grid. Therefore, they are subject to a lot of regulatory standards. Yet, to provide remote access over the internet, special architectures are developed to integrate a network interface into these devices without inferring with the actual functionality. However, these architectures either lack security measures against cyber-attacks or do not offer the necessary performance for time-critical communication interfaces. To solve that, an architecture consisting of three units is introduced in this paper to provide a network interface with extensive security measures and a high performance. The main feature is the isolation of the cryptographic functionality onto an additional MCU. After proposing the basic concept, the paper presents many implementation details. Based on the current state of implementation, a concept validation of the realized architecture is described.

**Keywords:** Cyber-security · Functional safety · Network security · Industrial Internet of Things · Industrial Control System · Supervisory Control and Data Acquisition System · Multi microcontroller setup · Dos prevention · Critical infrastructures

## 1 Introduction

With the tremendous growth of the Internet-of-Things (IoT), nowadays nearly everything is connected to the internet, which greatly improves the functionality of many different device categories and even enables new use-cases. By now, this trend reached the industrial sector and critical infrastructures in the form of Industrial IoT (IIoT). Supervisory Control and Data Acquisition Systems (SCADA), e.g. the power grid or water supply, or Industrial Control Systems (ICS) like production facilities are connected to the internet to provide an interface to an external instance. This enables new possibilities regarding supervision, maintenance, control and automation.

Most of these systems feature a single point-to-point connection between an end-device in the field and the control unit of the operator. This end-device can be a single PLC (Programmable Logic Controller) or a gateway concentrating local data traffic. Due to the importance of error-free functionality of these systems, intense safety measures are applied to all components.

Unlike functional safety, cyber-security has rather been neglected in the past. For a long time, hardly any security measures were applied to these networks providing a huge attack surface for an adversary to cause serious damage. To prevent a scenario like that, new standards were issued prescribing minimal requirements for cyber-security measures. In order to comply with these new standards, a device has to meet new additional requirements. This turns out to be a non-trivial task, as extensive measures are necessary. Therefore, a comprehensive solution must be developed.

In the research project *Energy Safe and Secure System Module* (ES$^3$M), such a solution is developed at the moment [10]. Currently focused on the power grid, a module consisting of four Microcontroller Units (MCUs) is developed to secure the communication between a substation and the controlling station of an energy provider. However, the created system architecture can easily be ported to any other SCADA, ICS or automotive system. The key characteristic of this architecture is the separation of the cryptographic functionality from the network communication onto two independent MCUs. This separation with its characteristics and implementation details will be further presented in this paper.

## 1.1 Contribution

Building upon existing work, the paper contributes the following points to the topic of secure communication architectures.

- Higher security confidence: Complete isolation of the cryptographically sensitive data from the network communication onto two separate MCUs
- Small size: The reduction of complexity and code size results in more testable and maintainable software for each MCU
- Transparent functionality: No influence on the actual task of the system
- Efficiency: Performance guarantees are given

## 1.2 Structure

The paper is structured as follows. In Sect. 2, the background and the context of the paper is presented. Based on this, Sect. 3 evaluates related work. Section 4 describes the basic concept of the architecture, while in Sect. 5 the concrete implementation is introduced. As the implementation isn't completely finished at the time of writing, Sect. 6 only presents basic performance characteristics of the system and mainly depicts the concepts we plan for a comprehensive validation of the architecture in the future. Section 7 concludes the paper with an outlook to future work.

## 2 Background

### 2.1 Regulatory Context

Within the current research context, in [2] the regulatory security measures are outlined that must be applied to communication interfaces inside the power grid. In this standard, the usage of the *Transport Layer Security* (TLS) protocol is prescribed for all TCP/IP based connections. This results in the application of both symmetric and asymmetric cryptography as well as X.509 certificates for securing the communication channel. Because most of the application specific protocols running on top of the communication interface assume a persistent connection, the maximum TLS session time is set to 24 h in the standard. This is a trade-off between the lifetime of the secure channel and the time in between new connection setups.

Next to the security related prescriptions, the field of application within critical infrastructures or industrial facilities results in extensive functional safety requirements. In [1], the definition of so called *Safety Integrity Levels* (SIL) can be found. Based on this classification, specific measures can be derived that must be implemented by a device. In the given context, many of the devices in question can be classified to be SIL3, which implies a device availability of $\geq$99.99999% and an error-rate of $\leq 10^{-7}$. To reach such numbers, both a periodic self-test of each MCU and additional monitoring by an independent instance is necessary.

### 2.2 Attack Vectors

Based on analyses of cyber-attacks on SCADA systems [5] and on IoT smart-world critical infrastructures [6], two different attack vectors can be identified. Firstly, due to a lack of proper security measures regarding confidentiality, integrity and authenticity, the communication can easily be eavesdropped or even modified by an adversary. On the one hand, this can reveal sensitive data, but on the other hand, the attacker can also harm both communication parties in various ways. Through modifications of the data traffic, the operation of a device or the whole system can be manipulated in a malicious way, so the actual functionality is not executed correctly. This could stop the system or even cause serious physical damage. Also, modified data may result in wrong status information about the system leading to incorrect operation or maintenance steps. The second attack vector is a *Denial-of-Service* (DoS) attack. In this scenario, the communication interface is flooded with data, so proper communication is not possible anymore. This, again, may cause the system to fail in its actual task and prevent surveillance or control functionality.

The usage of TLS in the communication channel will prevent all possible attacks of the first attack vector. Because a DoS attack is very hard to prevent, it must be ensured that such attacks will not interfere with the actual task, causing malfunction in the device functionality. Also, the network interface of the device must be fully operational as soon as the DoS attack is over. In order to assure that, the internal functionality and also the functional safety measures must be well prepared for this kind of situation.

## 3 Related Work

To overcome the possible attacks while conforming to the regulatory context described in Sect. 2, an extensive security solution is necessary. Niedermaier et al. proposed a Dual-MCU architecture that secures a device in an ICS system from a DoS attack [8]. Instead of putting both the control and the network functionality onto a single MCU, the features are split onto two MCUs. One handles all the network communication, in the following referred to as *NW-MCU*, while the second one performs the actual control job relevant to the overall system, further named *IO-MCU*. The communication between the two is done over a SPI (Serial Peripheral Interface) connection in a timely deterministic fashion. In the case of a DoS attack, all additional processing is done on the NW-MCU without influencing the IO-MCU. The result is an unaltered behavior regarding the control functionality during and also after a DoS attack. An additional benefit of this split architecture is the reduction in complexity and code size on each MCU. This simplifies software testing, reduces bugs and enables an easier certification.

This proposed architecture is a proper security solution for the second attack vector, but it lacks cryptographic measures against eavesdropping or manipulating the data traffic described in the first attack scenario. Therefore, adequate measures must be integrated, namely in the form of inserting TLS into the protocol stack. This could be done directly on either the NW-MCU or the IO-MCU, keeping the proposed architecture as is. However, this would on the one hand lead to cryptographically sensitive data being stored in memory that is directly accessible over the network interface. Due to bugs and vulnerabilities in the software in use, this sensitive data, e.g. certificates or private keys, can be obtained by an adversary. In high-class processors, this problem is normally addressed by hardware additions called *Trust Zones* [7] that isolate memory regions from unauthorized processes. But within this research context, only simple MCUs in the form of System-on-Chip modules are used that do not provide such functionality. On the other hand, adding TLS to the IO-MCU would increase the workload and the complexity of its software, resulting in the need for a more powerful MCU. However, this should be avoided, as it would create other challenges related to the regulatory context and certification efforts.

A Possible Solution for that are *Secure Elements*. In 2016, Pascal Urien presented so called security modules based on secure elements that include complete TLS/DTLS protocol functionality for the application in IoT devices [11]. These modules are low power and low priced external chips with their own CPU and, most importantly, tamper-proof memory. The communication between the module and a main MCU is done using the ISO 7816 communication interface [3]. To provide TLS functionality to the application, a software bridge runs on the main MCU. It receives the cipher text from the network stack and sends it to the secure element for processing. After decryption, the plain text is sent back to the MCU, where the software bridge forwards the data to the actual application software. A transmission of plain text over the secure channel works accordingly in the opposite direction. This enables a secure communication using TLS without storing cryptographically sensitive data on the main MCU.

The combination of both approaches, namely the addition of a NW-MCU and a secure element implementing TLS, could address all in Sect. 2.2 described attacks. However, this solution would still have problems. Firstly, when connecting the secure element to the NW-MCU, the decrypted plain text sent back from the secure element is stored on the NW-MCU until it is forwarded to the IO-MCU. Thus, it would still be possible for an adversary to read or modify the plain text due to vulnerabilities in the software. Connecting the secure element to the IO-MCU would prevent that issue, but in this case the workload and complexity of the IO-MCU would again be increased, as it would have to communicate with two parties at the same time. Secondly, the ISO 7816 based communication between a MCU and the secure element and the CPU inside the secure element itself are both very slow, causing a large delay of up to several hundred milliseconds in the processing of the data [11]. In case of high traffic, this may quickly become a bottleneck, no matter to which MCU the secure element would be connected.

Based on this work, in the next section we introduce an extended architecture to resolve the issues in current designs. This architecture introduces an additional MCU to further isolate the TLS functionality from the network stack without increasing the load of the MCU running the actual application.

## 4 Basic Concept

Building on the introduced architecture from Sect. 3, another split of functionality is performed. Providing a clear and consistent naming scheme, the name *NW-MCU* is kept for the existing MCU handling all the network related functionality. The MCU running the actual application, named IO-MCU in [8], is now called *APP-MCU*, as the application is not limited to I/O control in the context of this paper. In addition to these two MCUs, a new MCU is added implementing the TLS functionality, named *Crypto-MCU*. It is inserted in between the NW-MCU and the APP-MCU, keeping the functionality of both unchanged. The resulting architecture is shown in Fig. 1.

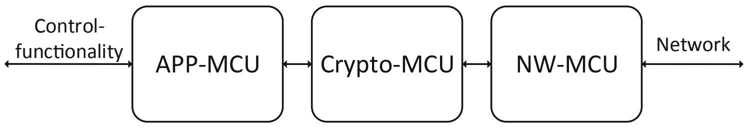

**Fig. 1.** New architecture with the additional Crypto-MCU

As can be seen in Fig. 1, the NW-MCU still handles all network related functionality including the protection against DoS attacks. The raw TCP payload received over the network interface is forwarded to the Crypto-MCU without any processing. This payload contains the TLS records, which are then processed on the Crypto-MCU. Thereafter, the decrypted plain text is sent to the APP-MCU, which is finally using and interpreting it. Data sent from the APP-MCU to the network is processed in the opposite direction through all three MCUs. The communication between the MCUs is based on SPI with additional hardware flow control for timely determinism and improved robustness.

With this architecture, the cryptographically sensitive data, like keys, certificates and the decrypted plain text, are completely isolated from the network interface and therefore not accessible from the outside. Even if an adversary gains access to the NW-MCU due to a software vulnerability, he cannot obtain or even modify the sensitive data because of the physical separation onto two different MCUs.

To further increase the security of the architecture, two additional components are added. On the one hand, a dedicated *Random Number Generator* (RNG) is placed on the printed circuit board (PCB), generating high entropy random numbers. The selected device is certified in the strongest class PTG.3 [9], which is suitable for any cryptographic application. With it, proper ephemeral keys can be generated. On the other hand, a secure element, as already mentioned in Sect. 3, is added to the system, connected to the Crypto-MCU. However, it is not used to implement the complete TLS functionality, but merely for authentication during the TLS handshake. Certified to Common Criteria EAL 5+ [4], it provides a tamper-proof storage for certificates and private keys, and even features an on-device key generation, resulting in the private keys never leaving the secure element. This ensures maximum security. All in all, the proposed architecture builds an extensive security solution that can protect a device, meaning the APP-MCU in this context, from both attack vectors described in Sect. 2.

## 5 Implementation

To prove the security improvements of our proposed architecture, we created a prototype containing all of the described components. The details of specific implementations are presented in this section.

### 5.1 Hardware Setup

The created prototype with all the described components can be seen in Fig. 2. It shows three boards, each containing one MCU. The green board in the middle is a custom PCB containing the Crypto-MCU, the secure element and the dedicated RNG. The boards on the left and the right side are off-the-shelf development boards from STMicroeletronics[1], representing the APP-MCU and the NW-MCU. For easy development, all three MCUs are of the same type in this setup. The two development boards contain the MCU STM32H743ZIT[2] that is

[1] https://www.st.com/en/evaluation-tools/nucleo-h743zi.html.
[2] https://www.st.com/en/microcontrollers-microprocessors/stm32h743-753.html.

based on an ARM Cortex-M7 core with a 480 MHz clock frequency. The Crypto-MCU is of the type STM32H753ZIT, which offers the same features as noted above, except for additional hardware accelerators for the Advanced Encryption Standard (AES) algorithm. The MCU type has been chosen due to the high performance while still being a System-on-Chip design, the huge amount of communication interfaces for potential future evaluations and the extensive options for hardware-based network packet filtering.

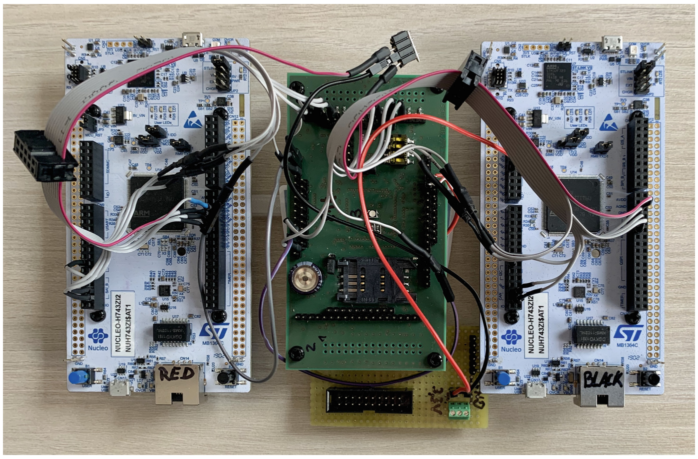

**Fig. 2.** Current hardware setup with the separation onto three controllers

## 5.2 Communication Between the MCUs

Before we dive into the specific software details of each MCU, the communication interface between the MCUs is presented. As already mentioned, the communication is based on SPI. However, not the default master-slave topology is used, but a more flexible multi-master system is deployed. This way, a communication with equally distributed access rights is possible, enabling both MCUs to initiate a data transmission whenever they want to. To achieve this, both participants can act as either master or slave depending on the transmission direction. This is configured in software using a flow control based on additional I/O lines. By sharing the SPI lines between the two participants, only half-duplex transmission is possible. For the current prototype, we use this interface for both communications between the three MCUs. But in case of another, maybe simpler APP-MCU, the interface between it and the Crypto-MCU can be changed to a different connection type, e.g. standard SPI or UART (Universal Asynchronous Receiver Transmitter).

For the message transmission over this interface, a proprietary protocol consisting of a *Header* and optional *Payload* has been defined. The header contains the type of the message, the length of the optionally following payload and a

CRC (Cyclic Redundancy Check) field, each occupying 2 bytes. Currently, there are five different message types defined, further described in Table 1.

**Table 1.** Message Types and their Meaning

| **Message type** | **Meaning** |
|---|---|
| Connection_Start | Command to start a new network connection. This can either mean actively connecting to a server or listening for incoming connections |
| Connection_Established | Notification that a new connection has been established |
| Connection_Stop | Command to stop the current network activity. This can either mean to close an active connection or to stop listening for incoming connections |
| Connection_Closed | Notification that all network activities are stopped |
| Payload | Transmission of network payload |

The first four message types are used for controlling and synchronizing the state machines on the different MCUs. Messages with the 'Payload' type are then used to actually exchange payload data between the MCUs. Based on these messages, the cooperation of the MCUs with their distinct functionality is managed.

### 5.3 Software of the Crypto-MCU

To reduce the amount of additional work for the critical APP-MCU, the Crypto-MCU is considered to be the master of the system related to network functionality. This means that it controls the NW-MCU with its functionality, while simultaneously exchanging the plain text network data with the APP-MCU. In order to provide a clear and scalable architecture, the software is written in the C++ programming language (Version 2014). This enables bundling functionality inside classes with a properly abstracted interface. This way, a loose coupling of the different software components is possible. Additionally, the *FreeRTOS*[3] kernel (Version 10.3.1) is integrated to provide a runtime environment. This real-time operating system is well-suited for MCUs and built with an emphasis on reliability and ease of use. The main functionality of the Crypto-MCU is modeled in three functional units called *PayloadProcessor*, *PayloadTransceiver* and *InterControllerConnection*, each represented by a single class. The PayloadTransceiver and InterControllerConnection classes are used for the payload exchange between the Crypto-MCU and the NW-MCU as well as the APP-MCU. This leads to a double instantiation of both classes. The PayloadProcessor class isolates the actual TLS functionality from the remaining code. The overall structure is shown in Fig. 3, with each unit and other implementation related details explained in more detail in the following sections.

[3] https://www.freertos.org/.

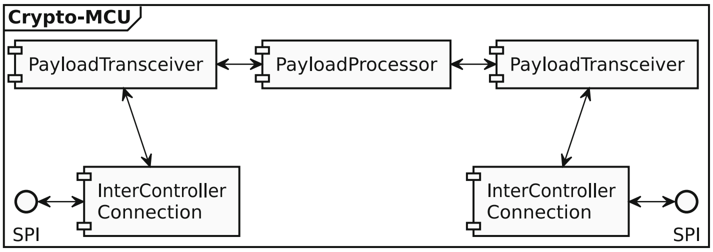

**Fig. 3.** Structure of the functional units inside the Crypto-MCU software

**PayloadProcessor.** This class forms the core of the Crypto-MCU software. Here, all the TLS related functionality is isolated from the rest of the software. The program execution is based on an *Event-driven architecture*. At startup, a task is created to handle all incoming events. These events are created in the PayloadTransceiver objects and are sent to the PayloadProcessor over an asynchronous event-queue. There are two categories of events: state-change events and payload-processing events. A state-change event either contains the command to start a new or stop a currently active network connection, or indicates the establishment or termination of a connection. A payload-processing event either means encryption or decryption of actual payload with subsequent forwarding of the processed data.

For the TLS capabilities, the open-source *mbedTLS*[4] library is used. It offers a simple API and is widely used in the embedded community. The code is slightly modified in some places to enable the usage of the RNG, the secure element and the AES hardware accelerators of the MCU. Due to the isolation of TLS and therefore all cryptographically sensitive data into a single task with a defined communication interface using the event-queue, the sensitive data can easily be protected with a *Memory Protection Unit* (MPU). This, in combination with the additional usage of the secure element for storing private keys and certificates, greatly improves the security of the whole system.

**PayloadTransceiver.** In this class, the state of a single external MCU is managed. Therefore, this class is instantiated twice, both for the NW-MCU and the APP-MCU (see Fig. 3). Internally, this class works in a very similar way as the PayloadProcessor class. It also features an Event-driven architecture with an event-queue that stores events for sequential processing. In this case, there are, again, two categories of events: Either there is an external message available from the other MCU or a message from the PayloadProcessor has been received. These messages can either contain payload to forward to the other MCU or are used to change the state of the network connection. In case of an external message, the header is parsed and proper events for the PayloadProcessor are created and added to its event-queue.

[4] https://tls.mbed.org/.

The fact that there are three event-queues in total on the Crypto-MCU may seem overly complicated at first, but this architecture results in many advantages. The most important one is the independence of each processing unit. This results in improved timely behavior compared to an otherwise single bigger event-driven system that provides the same functionality, because each task can process the events at its own pace without slowing down the others. Furthermore, the CPU load is reduced by a heavy usage of *Direct Memory Access* controllers (DMAs) for the communication interfaces and the hardware accelerators. The free CPU resources can then be used for processing the remaining event-queues. Another positive aspect of the different event-queues is the possibility to overcome temporary bottlenecks in the processing pipeline, for example caused by a faster reception of incoming data from the NW-MCU compared to the actual decryption, due to the storing capacity of the queues. Lastly, this separation simplifies the usage of a MPU to further secure the decrypted payload from unauthorized access.

**InterControllerConnection.** The last of the three classes handles the actual communication with the other MCU, as described in Sect. 5.2. This way, the physical communication interface is independent from the logic implemented in the PayloadTransceiver class. As shown in Fig. 3, there are two objects of this class, each one connected to one MCU via SPI and to one of the PayloadTransceiver objects. Furthermore, this abstraction enables a simple replacement of the communication interface, which can benefit future developments. Internally, the transmission and the reception of messages is split. The reception is handled in a distinct task, while the transmission is done from within the PayloadTransceiver task in a blocking manner. The synchronization between the two is done using a mutex.

All in all, the modularity of the Crypto-MCU software with the three event-queues enables responsive and efficient data processing in both directions. Moreover, by splitting the functionality, additional security measures in the form of a MPU can be applied. Finally, the use of FreeRTOS allows scalability for future software additions.

### 5.4 Software of the NW-MCU

Following the concepts of the Crypto-MCU software, the NW-MCU software also features an event-driven architecture. To ease the development efforts and to minimize the written code, as much code as possible is shared between the MCUs. The result of this effort is the structure shown in Fig. 4.

Compared to the software structure of the Crypto-MCU, there are only a few differences observable in Fig. 4. Mainly, the PayloadProcessor object is gone. As there is no TLS functionality needed on this controller, we do not need an object of this class. Additionally, there is only one PayloadTransceiver object, because we only have to handle one state machine on this controller. The last difference is the replacement of one InterControllerConnection object with an object of the

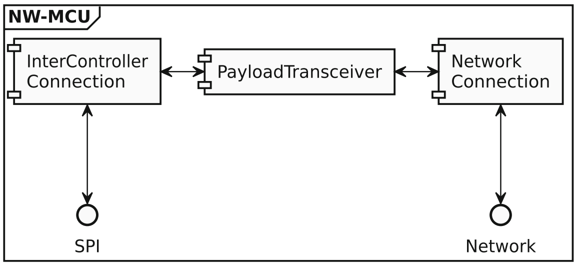

**Fig. 4.** Structure of the functional units inside the NW-MCU software

class *NetworkConnection*. With the presence of an event-queue on the NW-MCU, this software has the same advantages as described for the Crypto-MCU software. Also, the complete fundamental software framework including the FreeRTOS kernel is shared between the MCUs.

The NetworkConnection class mimics the interface of the InterController-Connection class in order to work with the existing PayloadTransceiver object. However, the implementation is very different. Inside this class, the actual network connection is handled, using the *Lightweight IP*[5] stack (LWIP). This open-source library provides a complete TCP/IP network stack with support for many additional features.

Based on the already described functionality of the data processing and the different messages that are exchanged between the MCUs, a state machine has been created and implemented on the NW-MCU. It is shown in Fig. 5.

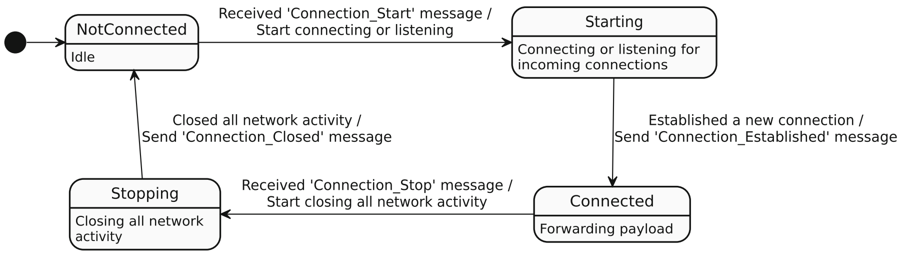
NotConnected
Idle
Received 'Connection_Start' message / Start connecting or listening
Starting
Connecting or listening for incoming connections
Established a new connection / Send 'Connection_Established' message
Connected
Forwarding payload
Received 'Connection_Stop' message / Start closing all network activity
Stopping
Closing all network activity
Closed all network activity / Send 'Connection_Closed' message

**Fig. 5.** State machine implemented in the NW-MCU software

As you can see in Fig. 5, there are four different states defined: *NotConnected*, *Connected*, *Starting* and *Stopping*. The first two states are the persistent ones, in which an active connection is established or not. In the *NotConnected* state, the NW-MCU is idle. This is also the default state after system startup. In the *Connected* state, the NW-MCU actively forwards payload both from the Crypto-MCU to the network endpoint and vice versa. The latter two states are more

[5] https://savannah.nongnu.org/projects/lwip/.

of a temporary kind. The *Starting* state indicates that the NW-MCU is trying to establish a new connection. Depending on the configuration, this can either mean that it actively tries to connect to a remote host or that it is acting as a host listening for an incoming connection on a given port. The *Stopping* state is the counterpiece to this, meaning that currently all network activity is being terminated. This again can imply closing an active connection to a host or to stop listening for an incoming connection. In the current setup, the NW-MCU is able to handle only a single connection at a time. However, this limitation can easily be removed in future developments.

The state transitions are also shown in Fig. 5. There are two types of transitions: Commands from the Crypto-MCU and events from the network stack. The two message types *Connection_Start* and *Connection_Stop*, already shown in Table 1, trigger transitions to the *Starting* and *Stopping* states respectively. As soon as the network stack indicates a successfully established connection or that all network activity is terminated, the state changes to *Connected* or *NotConnected*. In either case, a message of the type *Connection_Established* or *Connection_Closed* is sent to the Crypto-MCU announcing the state transition (see Table 1). Not shown in Fig. 5 are the state transitions caused by errors. If such a situation is encountered, either the *Stopping* or the *NotConnected* state is entered, depending on the current state and the actual error.

With the presented state machine and the code shared with the Crypto-MCU, a flexible, responsive and robust software handling the network connection is created. In cooperation with the Crpyto-MCU, both attack vectors described in Sect. 2.2 are addressed.

### 5.5 Software of the APP-MCU

The last MCU in the proposed architecture is the APP-MCU. It runs the actual application, to which a secure network interface, implemented by the Crypto-MCU and the NW-MCU, is provided. With the presented architecture, no restriction is given related to the application running on the APP-MCU. It can be anything from a real-time I/O control to a more complex gateway device. Independent from the main functionality, the software of the APP-MCU has to run the already known functional units consisting of a slightly modified *PayloadTransceiver* object and an *InterControllerCommunication* object. This is necessary for the APP-MCU to communicate with the Crypto-MCU. The modified PayloadTransceiver provides an interface for the actual application to send and receive data over the secured network connection.

Within the current research project, the application running on the prototype's APP-MCU is not the endpoint of the network data, but rather acts as a network gateway forwarding the payload to another network host. This way, the prototype represents a gateway device that provides a secured network channel using TLS. For the software of the APP-MCU, this means that the structure is almost identical to the one of the NW-MCU aside from an inverted network behavior. This enables sharing most of the code between the NW-MCU and the APP-MCU.

## 6 Concept for Validation

The current prototype, with the APP-MCU mirroring the functionality of the NW-MCU creating a network security gateway, provides a solid setup for validation of the proposed security architecture. At the time of writing, the implementation described in Sect. 5 is a work in progress. The software for each of the three MCUs is in an working state, but not all features are completely done or well optimized yet. Therefore, comprehensive and sound validation results cannot be created at the moment. However, some basic performance measurements are presented to prove the viability of our architecture.

- Currenty, the TLS handshake, including the secure element for authentication, takes around 1.2 s to complete. However, this process is not yet fully optimized.
- The delay caused by the processing chain of the NW-MCU and the Crypto-MCU is around 5 ms for network payload to finally reach the APP-MCU.
- DoS attacks are completely handled by the NW-MCU and the Crypto-MCU without affecting the actual functionality.

Based on these first promising results, the concept for the comprehensive validation of the system is already defined. Using the gateway functionality created within the current research project, the following tests, with additional comparison to other in this paper presented architectures, are planned for a future work.

- Measurement of the processing delay under different network traffic loads
- Behavior during and after different DoS attacks related to TCP and TLS
- Penetration tests regarding security aspects
- Tests related to functional safety and reliability of the system

## 7 Conclusion and Outlook

In this paper, a Multi-MCU security architecture has been presented. In addition to an APP-MCU running the actual application, two MCUs are added to provide security functionality in the form of TLS (Crypto-MCU) and a DoS protected network interface (NW-MCU). We showed that related work already has partial solutions against the identified attack vectors on SCADA and ICS systems relevant for this paper. However, all presented solution either lack proper security measures, do not provide the performance necessary in some of these critical systems or imply the need of a more powerful APP-MCU. Following this, we propose a new architecture featuring two additional MCUs for providing a secure network interface. One of them takes care of all network related functionality, while the second one is solely handling the security functionality. This physically isolates all cryptographically sensitive data from a remote access, highly increasing the security while providing protection against DoS attacks.

In conclusion, the current state of the prototype seems promising. The basic functionality is working as described in this paper, with no problems resulting from the use of a Multi-MCU architecture. In a future work, the architecture will be further verified against the dependability objectives like functional safety and IT-security. Also the performance characteristics will be analyzed.

## References

1. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. Technical report International Electrotechnical Commission, April 2010
2. IEC 62351–3: Power systems management and associated information exchange - data and communications security; Part 3: Communication network and system security - Profiles including TCP/IP. Technical report International Electrotechnical Commission, October 2014
3. International Organization for Standardization: Identification cards - Integrated circuit cards - Part 3: Cards with contacts - Electrical interface and transmission protocols. Standard ISO/IEC, 7816–3 (2006)
4. International Organization for Standardization: Information technology - Security techniques - Evaluation criteria for IT security. Standard ISO/IEC 15408–1/2/3:2009, December 2009
5. Irmak, E., Erkek, I.: An overview of cyber-attack vectors on SCADA systems. In: 2018 6th International Symposium on Digital Forensic and Security, ISDFS, pp. 1–5, March 2018. https://doi.org/10.1109/ISDFS.2018.8355379
6. Liu, X., Qian, C., Hatcher, W.G., Xu, H., Liao, W., Yu, W.: Secure Internet of Things (IoT)-based smart-world critical infrastructures: survey, case study and research opportunities. IEEE Access **7**, 79523–79544 (2019). https://doi.org/10.1109/ACCESS.2019.2920763
7. Mukhtar, M.A., Bhatti, M.K., Gogniat, G.: Architectures for security: a comparative analysis of hardware security features in Intel SGX and ARM TrustZone. In: 2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE), pp. 299–304, March 2019. https://doi.org/10.1109/C-CODE.2019.8680982
8. Niedermaier, M., Merli, D., Sigl, G.: A secure Dual-MCU architecture for robust communication of IIoT devices. In: 2019 8th Mediterranean Conference on Embedded Computing, MECO, pp. 1–5, June 2019. https://doi.org/10.1109/MECO.2019.8760188
9. Schindler, W., Killmann, W.: A proposal for: Functionality classes for random number generators. Bundesamt für Sicherheit in der Informationstechnik, September 2011

10. Frauenschläger, T., Dentgen, M., Mottok, J.: Systemarchitektur eines Sicherheitsmoduls im Energiesektor. In: 2. Symposium Elektronik und Systemintegration: Intelligente Systeme und ihre Komponenten: Forschung und industrielle Anwendung, April 2020. https://www.haw-landshut.de/fileadmin/Hochschule_Landshut_NEU/Ungeschuetzt/ITZ_Cluster_Forschung/ClusterMST/Symposium-ESI/2020/Tagungsbandbeitraege/A1-3_OTH-Regensburg_Frauenschlaeger_ESI_2020.pdf
11. Urien, P.: Innovative TLS/DTLS security modules for iot applications: concepts and experiments. In: Mandler, B., Marquez-Barja, J., Mitre Campista, M.E., Cagáňová, D., Chaouchi, H., Zeadally, S., Badra, M., Giordano, S., Fazio, M., Somov, A., Vieriu, R.-L. (eds.) IoT360 2015. LNICST, vol. 169, pp. 3–15. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47063-4_1