Zurich University
of Applied Sciences

School of
Engineering

InES Institute of
Embedded Systems

Security by Design for IoT Devices

Mario Noseda, Lea Zimmerli, Andreas Rüst

Unprotected IoT devices are an easy target for cyber-attacks. This white paper shows the application of a systematic development process to identify threats, derive security requirements and implement effective protection measures. The example of a simple WiFi-based sensor illustrates the design process and adequate protection measures. The latest generation of Secure Microcontrollers featuring Trusted Execution Environments (TEE) as well as Secure Elements both provide options to store key material securely and perform cryptographic operations in an energy-efficient way. The interaction of these hardware components together with dedicated firmware and a Public Key Infrastructure (PKI) enables a low-power sensor to connect securely to the cloud.

Connecting a Sensor to the «Cloud»

We use a simple example to show what threats can look like and how we apply a systematic approach to protect our systems adequately. Fig. 1 depicts our setup with our embedded IoT device on the left side. It consists of a microcontroller featuring a temperature and humidity sensor connected via I2C. The microcontroller publishes the readings periodically via a WiFi module using the unsecured MQTT protocol to an MQTT broker on the Internet. The broker, in turn, forwards the values to an IoT service platform to store the gathered measurements in a database and make them available to users via HTTP visualizations. All in all, a straightforward setup, which we can assemble effortlessly using off the shelf development boards.

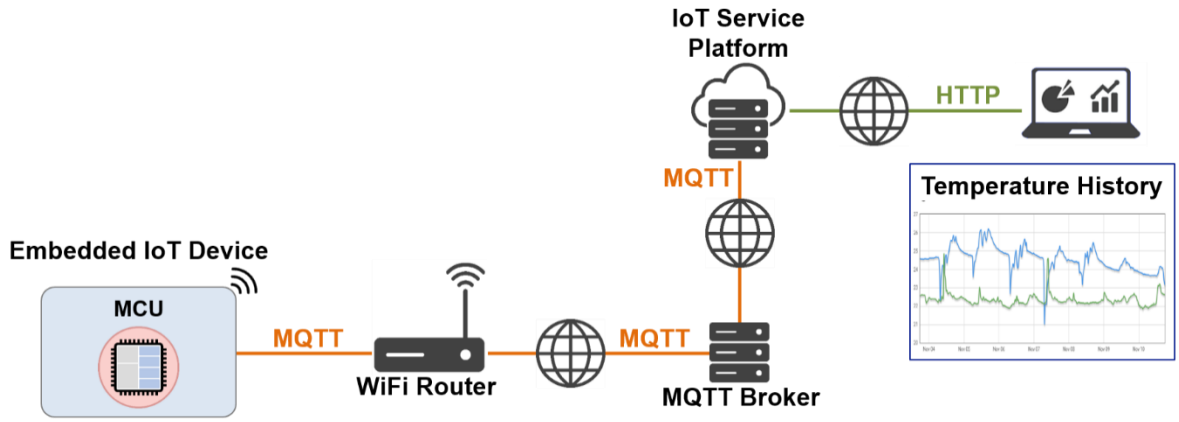

Fig. 1: Unsecured IoT device connecting to the cloud

Unprotected IoT Applications Are an Easy Target

What does this system look like from a security perspective?

Unfortunately, such a system exhibits a large attack surface. For example, an attacker can execute a man-in-the-middle[1] attack even with low-end hardware (e.g. a Raspberry Pi). Fig. 2 illustrates such a scenario where an attacker, using suitable software, gains access to a weak WiFi password (containing insufficient entropy), intercepts the sender's MQTT packet and manipulates its content. During our test runs, it took around 40 minutes for a Raspberry Pi to guess a WiFi password made up of two lowercase English words (with an estimated

[1] Man-in-the-Middle: https://www.enisa.europa.eu/topics/csirts-in-europe/glossary/man-in-the-middle [Accessed 24.06.2020]

entropy of 56 bits[2]). As a result of the attack, a forged temperature value is displayed on the IoT service platform as the attacker modified the values in the MQTT packet.

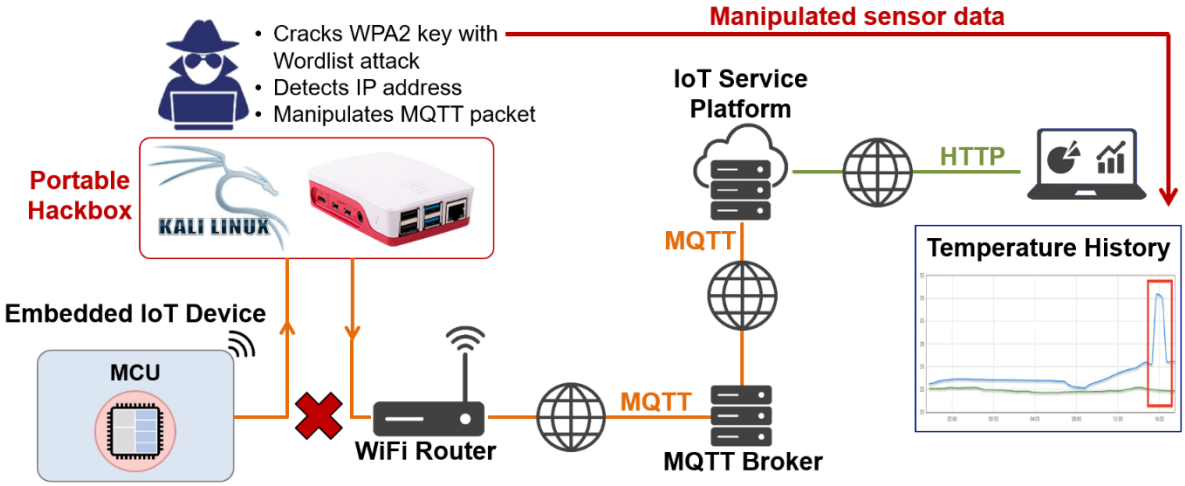

Fig. 2: Man-in-the-Middle attack

Security by Design – a Systematic Process

As a system developer, how can I systematically identify and evaluate such threats and protect my system with a reasonable effort?

For this purpose, we apply STRIDE[3], a well-proven threat modelling process from the IT world. We have supplemented it to meet the specific needs of embedded systems. Fig. 3 illustrates this process. The dataflow diagram, with its simple symbolic notation, helps to understand the data exchanges within the system.

Furthermore, so-called trust boundaries (red dotted lines) identify where information flows between units of different trust levels. Using the well-known STRIDE method from Microsoft, we perform a structured risk analysis on these trust boundaries. Based on the importance of certain services, which is defined by the business objectives of the application, possible threats and their impact on the system are analyzed and evaluated. The results are detailed security requirements for our system.

[2] Calculating Password Entropy: https://www.pleacher.com/mp/mlessons/algebra/entropy.html [Accessed 27.08.2020] E=log2(R^L), (E: Entropy, R: Pool of characters -> lowercase letters -> 26, L: Length of password -> 12)
[3] The STRIDE Threat Model: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20) [Accessed 24.06.2020]

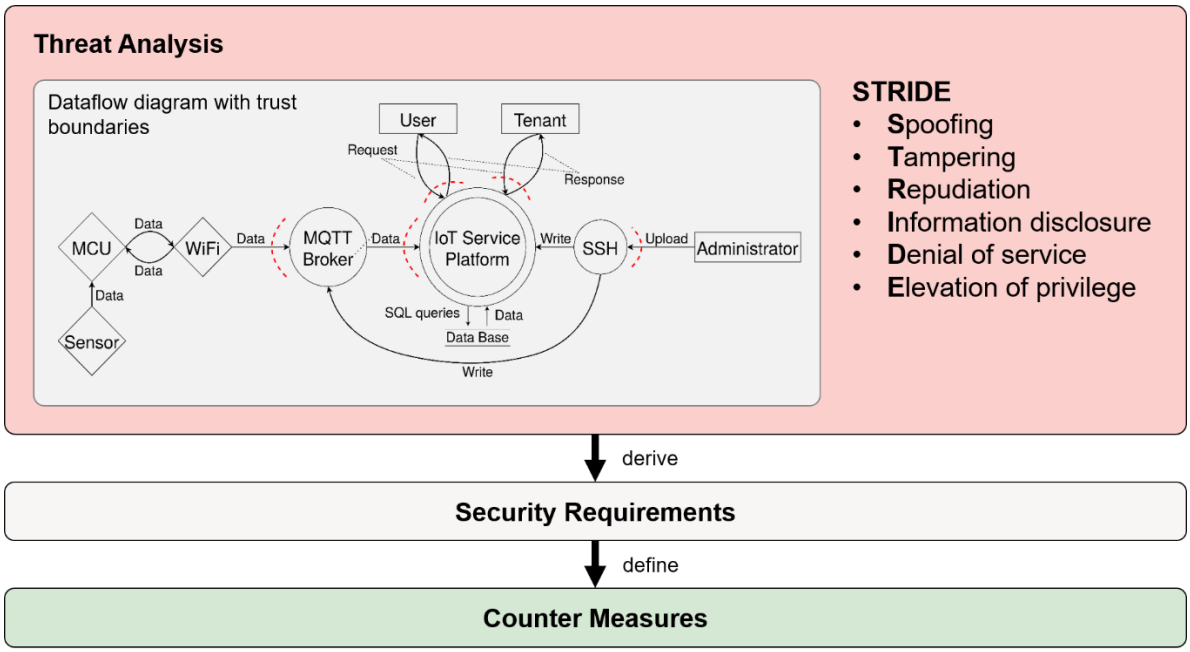

Fig. 3: Defining security requirements using STRIDE

As a next step and based on the defined security requirements, we need to define and implement effective countermeasures to protect our system against attacks. Such implementations are a particular challenge on embedded systems with their constrained resources.

Implement Countermeasures on Your Embedded System

Which resources are available?

Mainly, there are secure elements on the one hand and secure microcontrollers (SMCUs) on the other. Secure elements typically connect to a conventional microcontroller via I2C. Fig. 4 illustrates the internal components of a generic secure element which consists mainly of secure storage and energy-efficient hardware acceleration. We at the Institute of Embedded Systems have hands-on experience with such chips from different manufacturers and have developed an evaluation board to connect and evaluate different secure elements directly on existing microcontroller boards with little effort. Thereby, execution times and energy consumption can be measured directly in the target application.

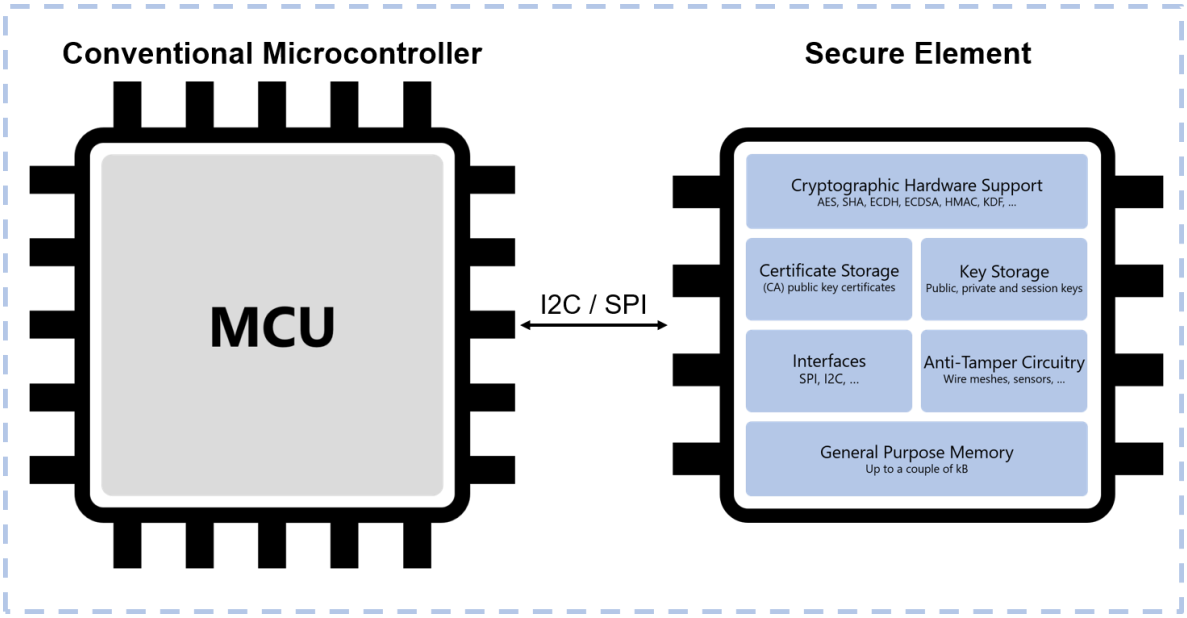

Fig. 4: Secure Element – physically isolate cryptographically sensitive material

Fig. 5 displays the second possibility, which is a new generation of microcontrollers. Based on ARM TrustZone[4] and associated open source firmware, they allow a trusted execution environment (TEE) to be built directly on the microcontroller used for the application. This approach effectively separates the memory, and therefore the executed code, into secure and non-secure partitions while monitoring the exchange between these partitions.

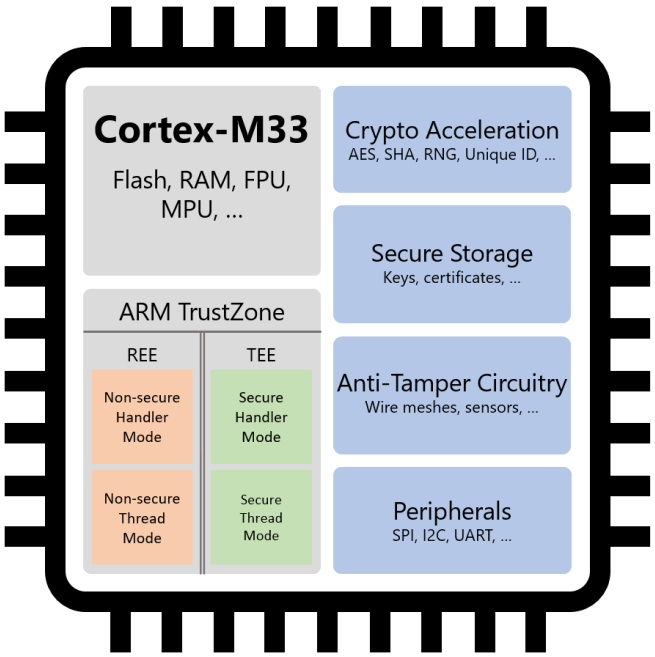

Fig. 5: Secure microcontroller (SMCU)

We can use Secure Elements as well as SMCUs to implement encryption, authentication, and integrity checks securely. In our example, we employ TLS 1.2 (Transport Layer Security) to provide a secure communication channel between MQTT client and server. Thus, the IoT device and the broker both encrypt their messages to prevent eavesdropping by an attacker. Furthermore, checksums/hash functions in TLS ensure the integrity of the transmitted data.

[4] ARM TrustZone: https://developer.arm.com/ip-products/security-ip/trustzone [Accessed 24.06.2020]

Finally, the communication partners mutually authenticate each other through X.509 certificates issued by a trusted authority. All these three components are part of TLS. The trailing 'S' in MQTTS expresses the use of the TLS. As a result, Fig. 6 depicts our protected example system using MQTTS.

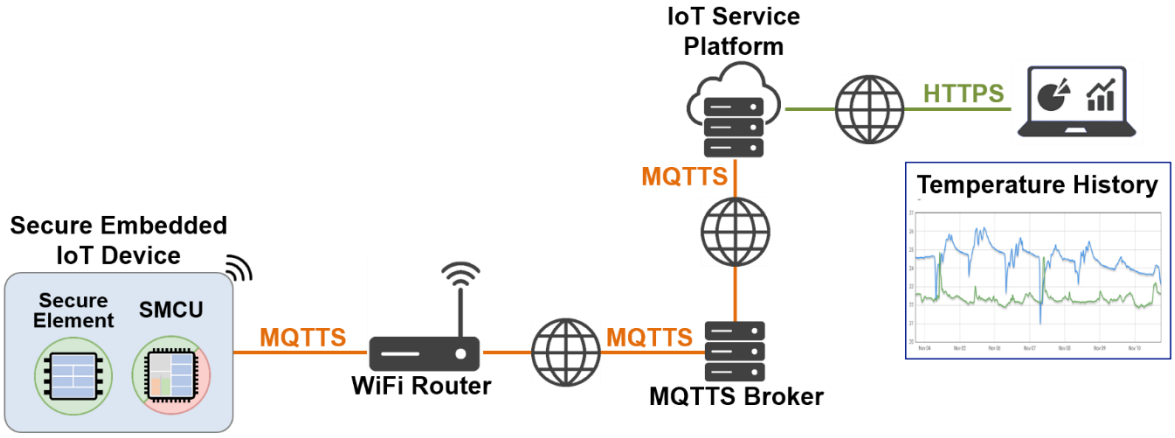

Fig. 6: Secured IoT device connecting to the cloud

Deployment of IoT devices often takes place at readily accessible locations. This factor makes it even more critical to protect keys from physical access. An IoT device not only needs to keep its private keys secret but also needs to prevent the modification of public keys that it received from other entities like the MQTTS broker. Moreover, the constrained resources on the IoT device require specialized hardware support to provide the cryptographic functions in a low-power way. Finally, secure boot sequences assure that the processor only executes authorized code, preventing firmware manipulations. Both options, secure elements, as well as SMCUs, have their specific strengths to tackle these challenges. However, in many cases, the combination of these two components yields an even higher level of security. Such a combination allows running trusted firmware components like the TrustedFirmware-M[5] (TF-M) on the SMCU and benefit from physically isolated and certified storage and resource-efficient processing on the secure element.

Fig. 7 illustrates the IoT devices built for the presented application. For the unsecured device on the left, we have used an NXP LPC55S69 as a conventional MCU by not making use of any Security Extensions. We connected a sensor module to measure the temperature and humidity and a WiFi module to add wireless capability to the device. In contrast, the secured IoT device on the right uses the NXP LPC55S69 as an SMCU by utilizing the integrated TrustZone-M and ARMv8-M Security Extensions in the Cortex-M33 core. Additionally, we have supplemented this with the secure element SE050 from NXP, which is responsible for generating and storing key pairs, as well as the creation and verification of signatures. This chip is located on our custom Secure Element Evaluation Board.

[5] TrustedFirmware-M: https://developer.arm.com/tools-and-software/open-source-software/firmware/trusted-firmware/trusted-firmware-m [Accessed 10.08.2020]

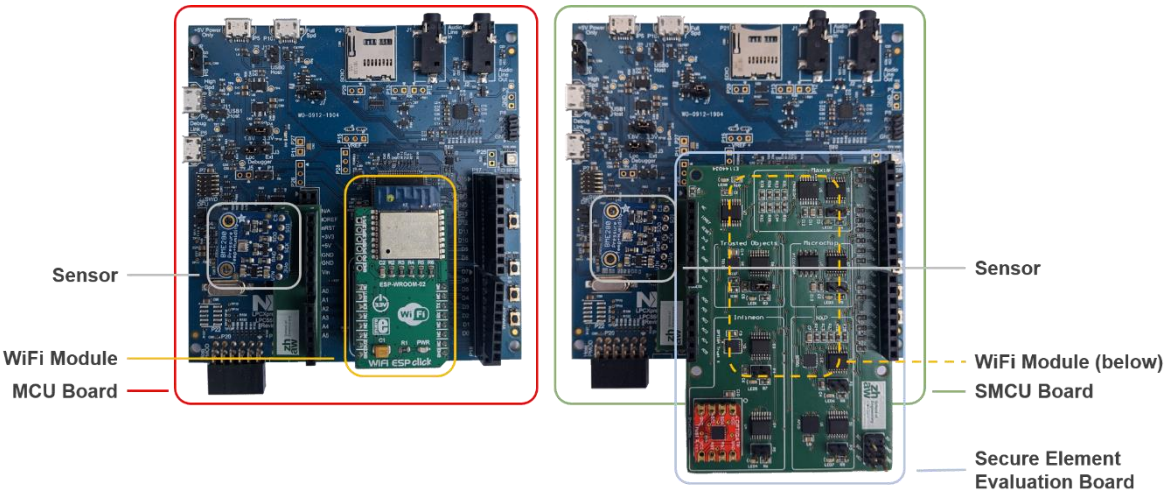

Fig. 7: Unsecured (left) and Secured IoT Device (right)

Embedded IoT Devices in PKI Environments

What infrastructure do we require to associate an IoT device or an MQTTS broker with their respective keys?

The presented system relies on public-key cryptography and digital signatures. For this, a trusted entity needs to issue digital certificates that guarantee the validity and authenticity of the included public key. The setup, technology and procedures to manage such digital certificates are called a public-key infrastructure[6] (PKI).

Fig. 8 illustrates the process for the IoT device and the MQTTS broker to obtain and employ their certificates. Using these certificates, they can mutually authenticate each other. The Certificate Authority (CA) acts as a trusted entity. The process starts with the MQTTS broker issuing a 'certificate signing request' (CSR) to the CA. Primarily, the CSR contains the public key of the MQTTS broker and proof of identity (step 1). On acceptance of the request, the CA provides a digitally signed certificate ensuring the validity and authenticity of the MQTTS brokers public key (step 2). In addition, the CA supplies a certificate that allows everyone to verify the CA signature on the issued certificate. Likewise, the IoT device issues a CSR to the CA (step 3) and receives its certificate (step 4). As a result, the IoT device and the MQTTS broker both hold a CA-signed certificate to prove their identity.

Furthermore, they both possess the CA certificate to verify signatures by the CA. Finally, in steps 5 and 6, the IoT device and the MQTTS broker use their newly received certificates to identify themselves towards the other. I.e. using the CA certificate, they can mutually authenticate each other.

The provisioning process, as indicated in Fig. 8, represents a particular challenge. Notably, system designers have to decide whether steps 3 and 4 take place before or after the deployment of the IoT device. Nevertheless, in both cases, there is usually a need to pre-provision device credentials in a protected environment. Specific services are available for pre-provisioning secure elements.

[6] Public Key Infrastructure: https://www.enisa.europa.eu/topics/csirts-in-europe/glossary/public-key-infrastructure-pki [Accessed 24.06.2020]

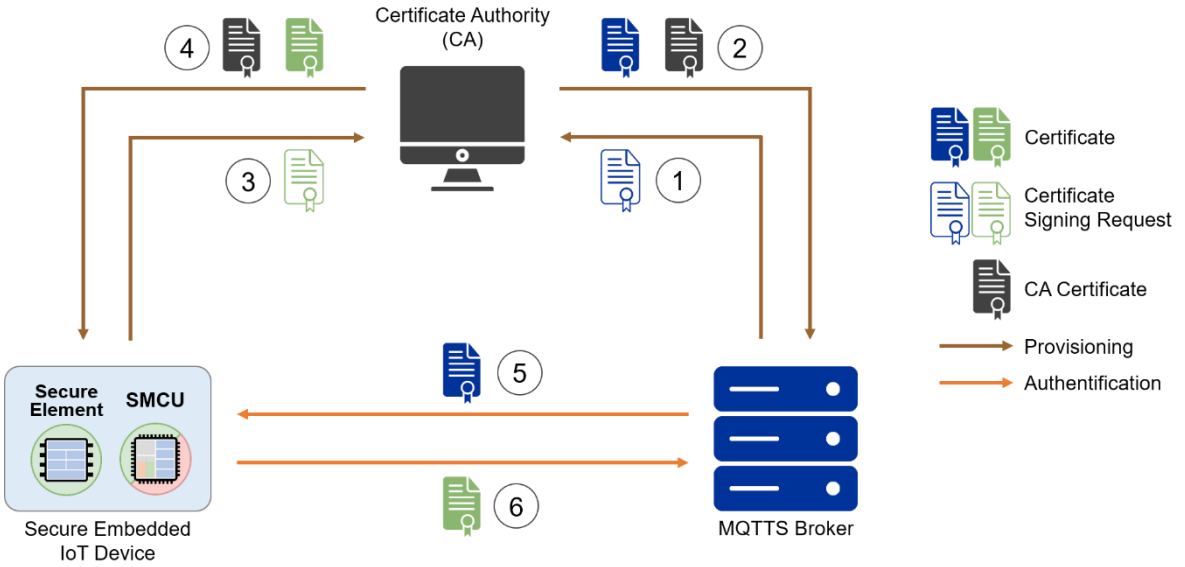

Fig. 8: Obtaining and using certificates in a PKI setup for IoT

It is essential to realize that in the presented example, hardware and dedicated firmware components work together with a Public Key Infrastructure (PKI) to connect a low-power sensor securely to a cloud. We have achieved this by first analyzing the threats. Based on these and the business goals, we have then derived security requirements, which in turn, we used to define and implement adequate countermeasures. A typical product design would additionally include design certification, including the attestation of the followed process steps and their results as well as penetration tests.

Conclusions

If you do not protect your IoT devices, your system is an easy target for attacks. A Raspberry Pi may be enough to perform an attack on an unprotected system. In this white paper, we have outlined a systematic process to identify threats, derive requirements and implement countermeasures for an everyday use case. Furthermore, we have described available hardware, firmware and software components as well as organizational issues (like PKI) to implement effective countermeasures. Moreover, the ZHAW Institute of Embedded Systems has the expertise and prototyping infrastructure available to implement embedded security. Please contact us to apply the presented process and techniques to your application.

Zurich University
of Applied Sciences

School of Engineering

The **Institute of Embedded Systems (InES)** is a leading research centre in the field of embedded systems with more than 50 employees as well as a state-of-the-art development and measurement infrastructure. Our services include:

- Design of system concepts and feasibility studies
- Consulting on the choice of technology
- Rapid prototyping and proof-of-concept
- Selection of radio components
- Realization of exhibition demonstrators
- Design of printed circuit boards (PCB)
- Writing microcontroller firmware: radio, protocols, sensors, actuators
- Interfacing to gateways and service platforms (cloud)
- Verification: Long-term tests and stress testing
- Embedded security: Thread analysis and protection measures

Mario Noseda, Lea Zimmerli, Andreas Rüst
Technikumstrasse 9
CH-8400 Winterthur

mario.noseda@zhaw.ch
andreas.ruest@zhaw.ch
lea.zimmerli@zhaw.ch

www.zhaw.ch/ines