D2.1. Requirements Specification for the Use Cases

This document serves as deliverable **D2.1. Requirements Specification for the Use Cases** in the CORNET-funded **Trusted IoT project**.

This deliverable contains the technical, functional and security related requirements for each use case.

USE CASE 1: Industry 4.0 (BTU)

Decomposition of the Setup

In case of BTU Cottbus – Senftenberg (BTU), the use-case would revolve around the Industry 4.0 and use of embedded systems (e.g. overlay CGRA architectures on FPGAs involving Machine Learning) for certain applications under the umbrella of Industry 4.0. It may include applications such as product automation or machine-learning driven predictive maintenance (PdM) of equipment using a variety of decentralized sensors (such as smart sensor nodes).

Industry 4.0

Industry 4.0 which is mostly referred to the 4th industrial revolution, is a concept of an industrial environment which incorporates characteristics like interconnectivity of sensors, machines and industrial components, big data and information, as well as the partial or complete autonomy of cyber-physical systems [1].

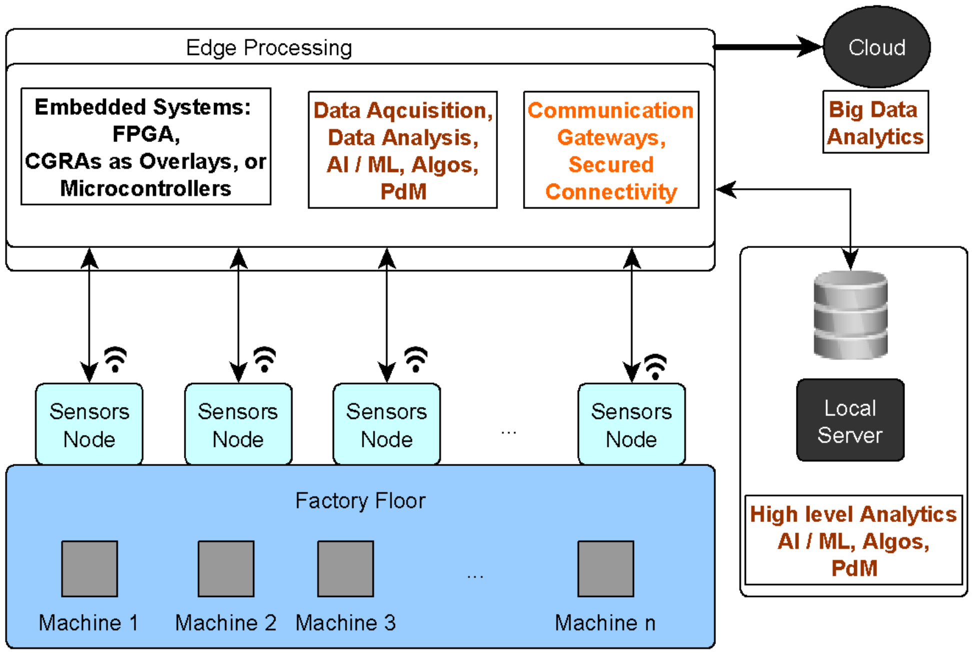

Edge Processing
Cloud
Embedded Systems: FPGA, CGRAs as Overlays, or Microcontrollers
Data Aqcuisition, Data Analysis, AI / ML, Algos, PdM
Communication Gateways, Secured Connectivity
Big Data Analytics
Sensors Node
Sensors Node
Sensors Node
...
Sensors Node
Local Server
Factory Floor
Machine 1
Machine 2
Machine 3
...
Machine n
High level Analytics AI / ML, Algos, PdM

Fig. 1: A typical Industry 4.0 environment

An application scenario in this environment e.g. can be predictive maintenance of components such as BLDC motors used in CNC or milling machines. Such a system can be monitored using different decentralized smart sensors such as accelerometer, vibration sensor, microphones etc. [2]. In such a setup, deep learning based analysis and edge processing can be used to perform these tasks. Edge processing would involve embedded systems including FPGAs, CGRAs (as devices or overlays on FPGAs) or Microcontrollers. A typical Industry 4.0 environment for this case would look like as depicted in Figure 1.

Coarse Grained Reconfigurable Architecture (CGRAs) and Field Programmable Gate Arrays (FPGAs)

Coarse Grained Reconfigurable Architecture or CGRAs as the name suggests are constructed with a more coarsely spread mesh of functional units for computations in contrast to FPGAs. In a CGRA, many functional units (FUs) or processing elements (PEs) are interwined with each other by a mesh style network. It may consist of registers as immediate memories attached to each or spread throughout the CGRA to hold data and can be accessed by a group of FUs depending on the architecture. Shown in Figure 2 is an example of a typical CGRA. In our case, we would consider an overlay architecture of CGRA that can be mapped onto a commercial FPGA [3] and can be used for specific applications such as running machine learning algorithms or deep neural networks e.g. for PdM.

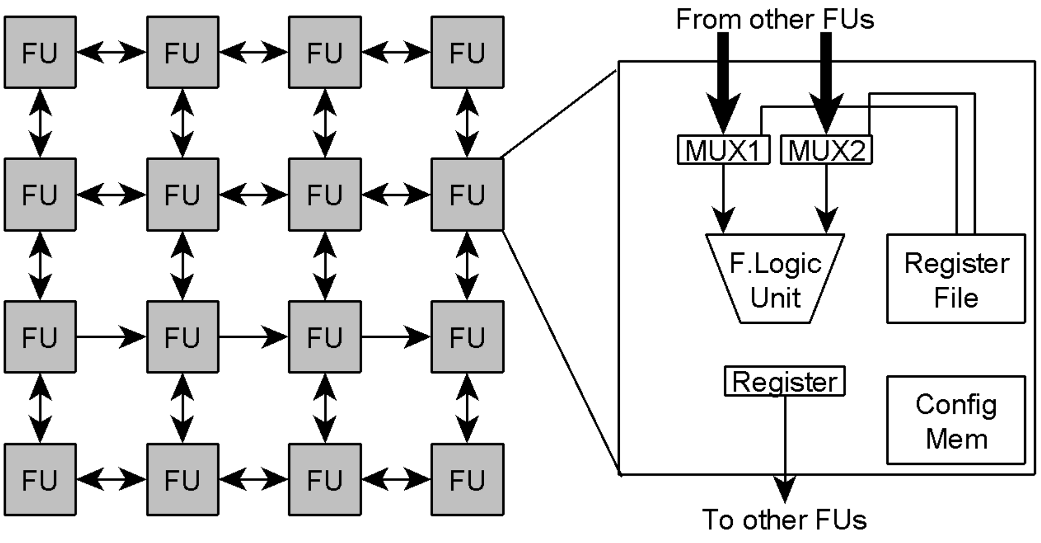


Fig. 2: Example of a typical CGRA

Scope and Analysis of Threats

For the scope of threats, we limit our vulnerability to hardware level threats or low level application threats, thus, excluding any application-level vulnerabilities or security threats. In addition to that, network layers are not covered or considered for vulnerability knowledge building. Attackers or Adversaries have a security level of 3 i.e. possessing an average level of resources but highly skilled.

In our case, following scope of threats but not limited to, can be taken into consideration:

1. In the use-case of Industry 4.0, leakage or manipulation of sensitive sensor data and information in an industrial environment such as for predictive maintenance:
    a. Manipulation or attack of/on smart sensors hardware with malicious intent:
        i. Users Risk: Users acting as an adversary who have access to the sensors can interact and manipulate the hardware
        ii. Physical mal-interference in the communication interfaces resulting in corrupt data transfer
        iii. Eavesdropping: Low cost hardware on sensors without encryption capabilities hence making data and information transfer unsafe
        iv. Smart sensor devices having the capability of easy-pairing can be looped out of the system by an attacker (Easy device access)

v. Adversaries with a higher attacking capability can hack into smart sensors and transmit dummy/malicious commands or transmit garbage data

2. Cyber-attacks on reconfigurable devices through manipulation of hardware bit-streams [4] [5]:
    a. Insertion of low-level hardware malwares such as energy draining Ring-Oscillators
    b. Unencrypted bitstreams can be tweaked to insert malwares which can steal important information stored in low-level components of reconfigurable devices.

To further extend the threat model we consider the well-known STRIDE methodology but applying to non-software type threats with the following description:

a. Spoofing (S)
b. Tampering (T)
c. Repudiation
d. Information Disclosure (I)
e. Denial of Service (DoS)
f. Elevation of Priviledge (EoP)

Threat	**Type**	**Description**	**Possible counter-responses**
1	S, TEoP	An user acting as an adversary spoofs a sensor component or node	Certain kinds of manipulation that change the properties or settings of a component should be dealt with dual authorization
2	S, T	Smart sensor data is altered, or produces garbage by either way of Spoofing or Tampering	Ensure that data is encrypted, Ensure that data can only be transferred to the embedded devices unless there is a proper handshake between the sensors and device receiving the data
3	S	PdM and Analytics sabotage due to sensor sending old data	Important and sensitive data communication should be guaranteed by time stamping and subsequent attestation
4	R, D	Sensors deny sending or receiving data, either by Repudiation or DoS due to smart sensor hardware manipulation	Firmware should be protected from alterations, only authorized updates should be allowed and it should be encrypted
5	T, EoP	Edge device processing is sabotaged or does not perform as intended based on the correct data from sensors	Scanning and checking of firmware before programming will ensure such attacks are not successful

6	I	A reads sensitive data from the sensors	Data should be encrypted
7	D	Data received from smart sensors for the purpose of PdM is overflowed or cannot be transferred due to continuous congestion	Firmware on the smart sensors should be protected from alterations, only authorized updates should be allowed and it should be encrypted. Moreover any interference should be mitigated between sensors and receiving devices
8	D	Communication gateways between sensors and embedded devices and subsequently local server or cloud are congested	Authentication should be provided and only specific authorized users should be granted access to render communication
9	EoP, I	‘A’ gets rights to read or wire tap bitstreams	Encryption would help in such attacks
10	EoP, I, T	‘A’ party gets rights to read/write bitstreams to the FPGA device	Only authorized users should be able to program critical devices, user authentication is needed
11	EoP, T	‘A’ gets to write malicious/spyware bitstreams or hardware file to the device	Scanning bitstreams before programming, through specific methodologies to prevent unintended bitstreams to be written to the device [4], [5]
12	EoP, T	‘A’ inserts malicious hardware in the firmware of the device resulting in physical consequences such as overheating or high energy usage	Malware scan or attestation before programming the device should be mitigate such attacks

.
Note: Particulars presented in all the above mentioned and aforementioned sections and sub-sections are subject to evolve and to be tweaked for further improvements, for knowledge building/understanding of the topics and for better approach to analyze threats more effectively, hence better research possibilities.

References

[1] N. Gronau, M. Grum, and B. Bender, “Determining the optimal level of autonomy in cyber-physical production systems,” *IEEE Int. Conf. Ind. Informatics*, vol. 0, pp. 1293–1299, 2016, doi: 10.1109/INDIN.2016.7819367.
[2] P. Suawa, T. Meisel, M. Jongmanns, M. Huebner, and M. Reichenbach,

"Modeling and Fault Detection of Brushless Direct Current Motor by Deep Learning Sensor Data Fusion," *Sensors*, vol. 22, no. 9, pp. 1–17, 2022, doi: 10.3390/s22093516.
[3] F. Fricke, A. Werner, K. Shahin, F. Werner, and M. Hubner, "Automatic tool-flow for mapping applications to an application-specific CGRA architecture," *Proc. - 2019 IEEE 33rd Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2019*, pp. 147–154, 2019, doi: 10.1109/IPDPSW.2019.00033.
[4] J. Rettkowski, S. Mahmood, A. Shallufa, M. Hubner, and D. Gohringer, "Inspection of partial bitstreams for FPGAs using artificial neural networks," *Proc. - 2019 IEEE 33rd Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2019*, pp. 83–86, 2019, doi: 10.1109/IPDPSW.2019.00023.
[5] S. Mahmood, J. Rettkowski, A. Shallufa, M. Hubner, and D. Gohringer, "IP core identification in FPGA configuration files using machine learning techniques," *IEEE Int. Conf. Consum. Electron. - Berlin, ICCE-Berlin*, vol. 2019-Septe, pp. 103–108, 2019, doi: 10.1109/ICCE-Berlin47944.2019.8966236.

USE CASE 2: Mobile Robots (TUD)

Decomposition of the setup

Mobile robots can move autonomously without assistance from external human operators. This autonomy is possible with the help of a perception system that helps it. It also needs a cognition unit or a control system to coordinate all the robot subsystems. These algorithms for perception and control are very sophisticated and require high processing power. Since part of the robots, autonomy is also power-wise, and these systems are usually battery-powered, they cannot be equipped with high-performance processors. On the other hand, FPGA (Field Programmable Gate Array) devices contribute to speeding up all processes and algorithms on the perception and control pipeline.

In this context, the targeted use case for TU Dresden will be using low-powered FPGAs on mobile robots.

The main components of a mobile robot can be split between hardware and software:

Hardware

- **Sensors:** Typically small, low-cost, and power-efficient. They provide input data for autonomous navigation. Integrated laser scanners such as Light Detection and Ranging (LiDAR), 3D cameras, accelerometers, gyroscopes, and wheel encoders are examples of them. These technologies have become popular due to their dynamic use. However, their integration can become problematic and a threat point for cyberattacks. Centralization of this data is significant to mitigate error access points.
- **Robot locomotion mechanism:** Essential for stability and ability. It is also a limiting factor according to the application.

- **Batteries:** For increased autonomy a good battery management might help. However, another big energy drain of a mobile robot, besides the actuators, is the processing device.
- **Manipulating equipment:** They can be robotic manipulators and bring the possibility to perform services for material handling operations.
- **Processing devices:** This component maintains the mobile robot's ability to navigate and operate in dynamic environments and make real-time decisions. With the introduction of power--efficient AI processors, such as ultra-low-power FPGAs, real-time decision-making for autonomous mobile robots became possible. Enabling calculations of complex decisions allows new ways of dynamic routing and scheduling, navigating and classifying, and reacting to obstacles appropriately.

Software

- **Simultaneous localization and mapping (SLAM):** This supportive technology for real-time navigation encompasses the two activities of creating detailed area maps of the environment and calculating the position of an AMR on a map[1]. For high accuracy and reliability, SLAM can be supported indoors by real-time location systems using ultra-wideband technology and outdoors by global positioning systems using network satellites placed in orbit.
- **Motion planning:** Considering the robot's size and dynamics, it calculates a feasible, collision-free path from point A to point B. SLAM complements this, and the decisions that must be made about the guide path, routing, and obstacle avoidance should all be taken by the mobile robot itself. This is because relying on wireless networks to offload this computation is only sometimes available or allows for real-time calculation. Additionally, it also adds an extra layer of risk for cyberattacks.
- **Artificial intelligence:** AI techniques can support mobile robots in navigation and any service the application might need. AI techniques such as vision systems and machine learning (ML) enable identifying and classifying obstacles.
- **Middleware:** The mobile robot requires a middleware for interfacing between all the components, sensors, and software applications. In other words, it needs a software component that different applications use to communicate. It should provide functionality to connect applications intelligently and efficiently so that it's faster to innovate.

Threat analysis overview

In this section, first, the scope, assets, and entry points of access are detailed. Then, the considered potential threats of the system are described.

Scope

The security scope here is limited to robot middleware and medium-level software vulnerabilities. It excludes threats covering application software or its implementation, nor threats on the network layer are considered. In particular, three types of attackers are considered:

- Human attackers interact physically with the robot (Robot User),
- Another robot or system is capable of physical interaction with the robot. (Third-Party Robotic System), and
- A human teleoperation the robot or sending commands to it through a client application (e.g., smartphone app) (Teleoperator / Remote User)

Assets

These are categorized as Privacy, Integrity, and Availability. The Privacy category determines that the robot's private data should not be accessible by attackers. In other words, sensor data, as well as persistent robot data (logs, software, etc.), must not be accessible by unauthorized actors.

The mobile robot integrity determines that attacks should not modify the device's behavior. The mobile robot system must not harm its users, environment, or itself. Similarly, unauthorized actors cannot control robot actuators or disrupt their tasks.

Finally, the robot's availability determines that it should continue to operate even under attack. This means that the embedded computing resources should be protected. Starving a robot of its computing resources can prevent it from functioning correctly. The robot should answer commands in a reasonable time.

Entry Points

These are the system attack surface area:

- **Robot Components Communication Channels:** The mobile robot is in constant communication with users or other robots. This communication layer is generally composed of multiple devices talking over a shared bus. This bus may be accessible over the robot link.
- **Remote Application Interface:** For user-grade devices, remote applications (cloud, smartphone application, etc.) can be used to read robot data or send robot commands.
- **Sensors:** Sensors are capturing data which usually end up being injected into the robot middleware communication channels.
- **Embedded Computer Physical Access:** External (HDMI, USB...) and internal (PCI Express, SATA...) ports.

Threat model

SImilar to previous sections, the STRIDE methodology is used for the threat model. For each threat a type is given: Spoofing (S), Tampering (T), Repudiation (R), Information Disclosure (I), Denial of Service (DoS), and Elevation of Privilege (EoP).

Table #: List of the assumed threats for ultra-low-power FPGA mobile robot

Threat	**Type**	**Description**	**Response**

1	S, T, I, EoP	An attacker spoofs a component identity.	Components should authenticate themselves, not be attributed similar identifiers, and they should be chosen carefully.
2	T	An attacker intercepts and alters a message.	Messages should be signed and/or encrypted.
3	T	An attacker writes to a communication channel without authorization.	Components should only communicate on encrypted channels. Sensitive inter-process communication should be done through shared memory whenever possible.
4	I	An attacker listens to a communication channel without authorization.	
5	DoS	An attacker prevents a communication channel from being usable.	Sensitive inter-process communication should be done through shared memory whenever possible. Robot behaviors should be tolerant of a loss of communication.
6	T	A node accidentally writes incorrect data to a communication channel.	Components should always validate received messages. Invalid message events should be logged and users should be notified.
7	S	An attacker spoofs a robot sensor (by e.g. replacing the sensor itself or manipulating the bus).	Sensors should embed an identifier to detect hardware tampering. Components should try to explicitly refer to which sensor ID they expect data from. Sensor data should be signed and ideally encrypted over the wire.
8	S	An attacker spoofs a robot actuator.	Actuators should embed an identifier. Commands should be signed (ideally encrypted) to prevent manipulation.
9	T	An attacker modifies the command sent to the robot actuators. (intercept & retransmit)	
10	R	An attacker intercepts the robot actuators command (can	

		recompute localization).	

References

[1] R. Bloss, “Simultaneous sensing of location and mapping for autonomous robots,” Sensor Review, vol. 28, no. 2. Emerald, pp. 102–107, Mar. 28, 2008. doi: 10.1108/02602280810856651.

USE CASE 3: Drones 4.4 (KU Leuven)

Decomposition of the setup

The targeted use-case for KU Leuven will be the use of FPGAs on drones. On a drone there are multiple components that work together to achieve the intended goal: flying a drone.

To improve security, the use case will try to implement these different components on a single FPGA. The main motivator for this is that it reduces the number of individual components, and therefore the number of attack vectors.

The different components on a drone are typically made by different companies. Suggesting that they all implement their solutions on the same job is not a light request. To help companies to trust each-other this use case will be built on open-source, widely used hardware.

RISC-V

RISC-V is an open Instruction Set Architecture (ISA) [3]. This can be best seen as a **model** or an **idea** on how a computer should run software. An ISA defines, among other:

- list of instructions
- anatomy of instructions
- data types
- registers

It is worth pointing out that RISC-V is an open specification and is not an implementation. In this project we are using an **open source** RISC-V **implementation**.
Several people or groups of people have taken the RISC-V specification and made an implementation.

Every RISC-V implementation must have the base integer instruction set. A choice that is open to the implementation is the width of the processor (e.g. 32-bit, 64-bit or 128-bit) and the number of registers in the register file (e.g. 32 or 16). These choices are reflected in the naming of an implementation. For example:

- **RV32I** is a RISC-V implementation which has the base integer instruction set and the register file has 32 registers with a 32-bit width
- **RV32E** is a RISC-V implementation which has the base integer instruction set and the register file has **16** registers with a 32-bit width
- **RV64I** is a RISC-V implementation which has the base integer instruction set and the register file has 32 registers with a **64-bit** width

Next to the base integer instruction set, there are multiple extensions. These provide additional instructions that serve a certain category of operations. For example:

- the M-extension adds instructions for integer multiplication and division
- the B-extension adds instructions for bit manipulation
- the D-extension adds instructions for double-precision floating point operations.

As with the base-implementations, the naming of an implementation typically gives a summary on which extensions are added. For example, an RV32IMAC implementation is:

- a 32-bit implementation,
- with 32 registers
- supporting:
    - the base integer instructions,
    - integer multiplication and division,
    - atomic instructions,
    - and, has compressed instructions.

For a complete and more in-depth description of the instruction set, we refer to the ISA specification [1].

Open-source implementations

A number of open-source implementations of RISC-V are available. A short list is given on Wikipedia: https://en.wikipedia.org/wiki/RISC-V#Open_source. A more extensive list can be found here: https://github.com/riscvarchive/riscv-cores-list.

Model of the proposed solution

The proposed solution will replace all the processors and non-sensor modules, on a single node with a single FPGA chip. This is visualized in Fig. 3.

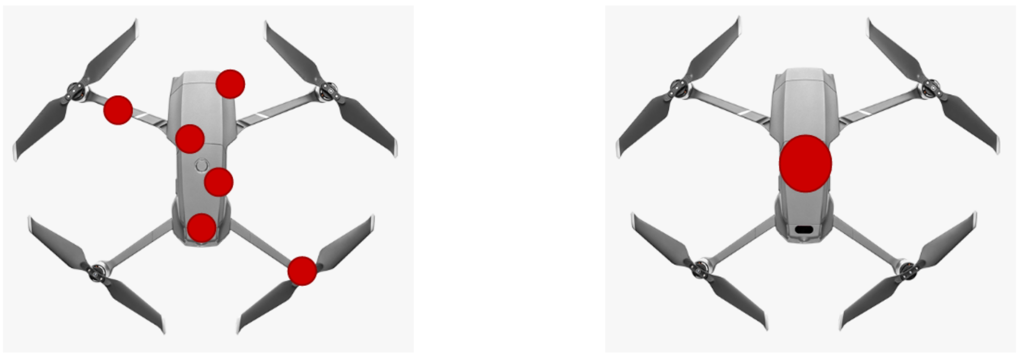


Fig. 3: Centralisation of multiple processors into a single FPGA chip

An abundance of components are already available as commercial off-the-shelf (COTS) products. To allow compatibility with existing products, a RISC-V implementation is chosen that, to certain extent, provides all the functionalities that are required. As different components have different requirements, three different implementations are used. This will illustrate that not all RISC-V implementations have to be identical. Additionally, different sizes/features of implementations can live alongside each other.

Choices of RISC-V implementations

A number of open source implementations are available. Depending on various parameters a choice can be made. For the Trusted IoT project, three different implementations are considered.

Ibex

The CPU core is heavily parameterisable and well suited for embedded control applications. Ibex is being extensively verified and has seen multiple tape-outs. Ibex supports the Integer (I) or Embedded (E), Integer Multiplication and Division (M), Compressed (C), and B (Bit Manipulation) extensions.

source: https://github.com/lowRISC/ibex
intended use: MCU / Flight computer
license: Apache License Version 2.0
HDL: SystemVerilog

NEORV32

The NEORV32 Processor is a **customizable microcontroller-like system on chip (SoC)** built around the NEORV32 RISC-V CPU and written in **platform-independent VHDL**. The processor is intended as auxiliary controller in larger SoC designs or as *ready-to-go* stand-alone custom microcontroller that even fits into a Lattice iCE40 UltraPlus low-power & low-density FPGA. The project is intended to work *out of the box* and targets FPGA / RISC-V beginners as well as advanced users.

source: https://github.com/stnolting/neorv32
intended use: Comm

license: 3-clause BSD
HDL: Verilog

PicoRV32

PicoRV32 is a CPU core that implements the RISC-V RV32IMC Instruction Set. It can be configured as RV32E, RV32I, RV32IC, RV32IM, or RV32IMC core, and optionally contains a built-in interrupt controller.

source: https://github.com/YosysHQ/picorv32
intended use: ESC
license: ISC
HDL: VHDL

With these 3 implementation, a number of varieties in available choices are covered:

- different HDL languages
- different sizes of implementations
- different licenses

This will enforce that the proposed solution is not depending on a single chosen parameter or combination of parameters.

On the licenses of open source implementations

The following is in no way complete nor can it be the basis for any legal argumentation.

The three RISC-V implementations that are chosen above each use a different license. Here is a short summary on these licenses with links to the full license texts. Table 1, below, attempts to illustrate the overlaps and differences between the three.
The info below is taken from the website: https://tldrlegal.com/

ISC license

A short, permissive software license. Basically, you can do whatever you want as long as you include the original copyright.

This license is functionally identical to the MIT/Expat and the Simplified BSD license, discarding some language that was made unnecessary by the Berne convention.

More info:

- https://tldrlegal.com/license/-isc-license#fulltext
- https://joinup.ec.europa.eu/license/isc-license

3-clause BSD license (new BSD license)

The BSD 3-clause license allows you almost unlimited freedom with the software so long as you include the BSD copyright and license notice in it (found in Fulltext).

More info:
- https://tldrlegal.com/license/bsd-3-clause-license-(revised)#fulltext
- https://joinup.ec.europa.eu/license/bsd-3-clause-new-or-revised-license

Apache License, Version 2.0

You can do what you like with the software, as long as you include the required notices. This permissive license contains a patent license from the contributors of the code.

More info:
- https://tldrlegal.com/license/apache-license-2.0-(apache-2.0)#fulltext
- https://intellectual-property-helpdesk.ec.europa.eu/news-events/news/how-apache-20-2020-02-13_en

		ISC	***BSD 3-clause***	***Apache License 2.0***
CANNOT	Hold Liable	x	x	x
	Use Trademark		x	x
MUST	Include copyright	x	x	x
	Include license	x	x	x
	State changes			x
	Include notice			x
CAN	Commercial use	x	x	x
	Modify	x	x	x
	Distribute	x	x	x
	Place warranty		x	x
	Private Use			x
	Use patent claims			x
	Sublicense			x

Table 1: High-level comparison of the relevant licenses on the chosen implementations

Adversary Model & Threat analysis

In this section, first, the assumptions of the analysis are detailed. Then, the potential threats of the system and a solution strategy will be described.

The scope of security here is limited to hardware and low-level software vulnerabilities. It excludes threats covering application software or its implementation, nor threats on the network layer are considered. Moreover, supply chain attacks are deemed to be outside the scope of this report. In particular, there are three types of attackers that are taken into account: attackers with physical access, attackers with user access, and attackers who already have access to the system.

A. **Assumptions.**
Table 2 lists the assumptions that were identified for this threat model study. We outline the assumptions regarding the threat analysis as follows.

Assumption	***Description***
A1	We do not consider stealthy non-intrusive physical attackers, such as side channel attacks
A2	We do not consider threats covering application software or its implementation, nor threats on the network layer
A3	We do not consider supply chain attacks

Table 2: List of assumption, made in the threat model

B. Threat model
A widely used threat modelling technique is STRIDE - a methodology created by Microsoft in 1999.The STRIDE methodology is a software-based approach that starts from the system model. For each communication link or entity involved, the threat model considers a number of threats. For each threat a type is given: Spoofing (S), Tampering (T), Repudiation (R), Information Disclosure (I), Denial of Service (DoS), and Elevation of Privilege (EoP).

One can then choose the most relevant risks from the complete list of threats. Table 3 provides a summary of these threats.

Threat	**Type**	**Description**
1	S	MCU receives old data from the Sensors (replay attack)
2	S	MCU receives old data from M (ESC) (replay attack)

3	S	Sensors receive old data from the MCU (replay attack)
4	S	M (ESC) receives old data from the MCU (replay attack)
5	T	MCU (non-firmware) instructions modified
6	T	Sensors data modified
7	T	Control (RX/TX) data is modified
8	T	MCU does not perform based on received data from Sensors and ESCs (firmware)
9	T	Attack on communication module
10	R	MCU denied sending data
11	R	Sensors denied sending data
12	R	ESCs denied sending data
13	I	A reads data from the ESCs
14	I	A reads Sensor data
15	I	A read MCU instructions (non-firmware)
16	I	A read MCU instructions (firmware)
17	D	Performance issues; MCU cannot handle the amount of data received from sensors/EScs
18	D	SoC congestion: ESCs cannot handle send/receive data to/from MCU
19	D	Network congestion: communication core cannot manage incoming/outgoing data to the controller
20	E	A gets read/write access to MCU
21	E	A gets read/write access to ESCs
22	E	A gets read/write access to sensors

Table 3: List of the assumed threats, together with their STRIDE-type

Threat 1 – MCU receives old data from the Sensors (replay attack)
Response: Mitigate. Sensitive data communicated to the MCU should be authenticated and freshness will guarantee the prevention of replay attack.

Threat 2 – MCU receives old data from M (ESC) (replay attack)
Response: Mitigate. Sensitive data transferred to the MCU should be verified (authentication/data encryption), and freshness will ensure that a replay attack is prevented.

Threat 3 – Sensors receive old data from the MCU (replay attack)
Response: Mitigate. Sensitive data communicated to the sensors should be authenticated and freshness will guarantee the prevention of replay attack.

Threat 4 – M (ESC) receives old data from the MCU (replay attack)
Response: Mitigate. Sensitive data communicated to the ESCs should be authenticated and freshness will guarantee the prevention of replay attack.

Threat 5 – MCU (non-firmware) instructions modified
Response: Mitigate. Data encryption will prevent unauthorized entities to modify non-firmware data in the MCU. Access authorization will allow only legitimate entities to access these data.

Threat 6 – Sensors data modified
Response: Mitigate. Sensitive sensor data should be encrypted.

Threat 7 – Control (RX/TX) data is modified
Response: Mitigate. Sensitive control (RX/TX) data should be encrypted and protected. Attestation will also aid in verifying the authenticity of the data.

Threat 8 – MCU does not perform based on received data from Sensors and ESCs (firmware)
Response: Mitigate. Attestation of MCU through eHSM will guarantee the health and detection of any potential adversarial presence.

Threat 9 – Attack on communication module
Response: Mitigate. Performing attestation will identify the presence of adversary in a communication module.

Threat 10 – MCU denied sending data
Response: Mitigate through authenticity. Sensitive data communicated from the MCU should be authenticated to verify the data origination authenticity.

Threat 11 – Sensors denied sending data
Response: Mitigate. Sensitive data communicated from the sensors should be authenticated based on sensor’s ID.

Threat 12 – ESCs denied sending data
Response: Mitigate. Sensitive data communicated from the ESCs should be authenticated and verified based on ESC-ID.

Threat 13 – A reads data from the ESCs
Response: Mitigate. Sensitive ESC data should be protected through encryption.

Threat 14 – A reads Sensor data
Response: Mitigate. Sensitive sensor data should be encrypted.

Threat 15 – A read/write MCU instructions (non-firmware)
Response: Mitigate. Encryption of sensitive data will safeguard against unauthorized access.

Threat 16 – A read/write MCU instructions (firmware)
Response: Mitigate. Unauthorized firmware changes should be detected during the attestation process. Additionally, user authentication will prevent unauthorized access to firmware.

Threat 17 – Performance issues; MCU cannot handle the amount of data received from sensors/ESCs
Response: Mitigate. Data origin authentication through verification and attestation will identify malicious sensors/ESCs and prevent Denial of Service attack on MCU.

Threat 18 – SOC congestion: ESCs cannot handle send/receive data to/from MCU
Response: Mitigate. Identify compromised ESCs through attestation and isolate it from communicating to the MCU.

Threat 19 – Network congestion: communication core cannot manage incoming/outgoing data to the controller
Response: Mitigate. Authentication will prevent unauthorized parties to communicate and flood communication channel with data packets. eHSM module will also aid in this process by performing attestation of the entities and detecting adversarial presence.

Threat 20 – A gets read/write access to MCU
Response: Mitigate. User authentication and verification should be established while giving read/write access to the sensitive data. Only operator(s) should be allowed to perform modification during the offline phase.

Threat 21 – A gets read/write access to ESCs
Response: Mitigate. User authentication and verification should be established while giving read/write access to the sensitive data. Only MCU/Operator should be allowed to perform any modification.

Threat 22 – A gets read/write access to sensors
Response: Mitigate. Only MCU/Operators should be allowed to perform any modification to the sensors. Attestation will aid in detecting adversarial presence in sensors.

eHSM

Finally the embedded Hardware Security Module (eHSM) will contain all the required co-processors to achieve attestation and secure communication. This module will be customly developed.

The intended platform for use case 4 is an FPGA. There are different ways to configure an FPGA, but the most frequently occurring technique is through SRAM. With this type of FPGA, the FPGA can be seen as two parallel planes. One layer consists of the logical building blocks, like: Look-Up Tables, Memories, and DSP components. The routing that allows interconnection between these building blocks is also in this reconfigurable fabric.

To give this reconfigurable fabric its design, these primitives need to be configured. This configuration resides in the second plane: the configuration memory. As mentioned above, this memory is implemented on the used FPGA-type as SRAM. This is functionally represented in Fig. 3.

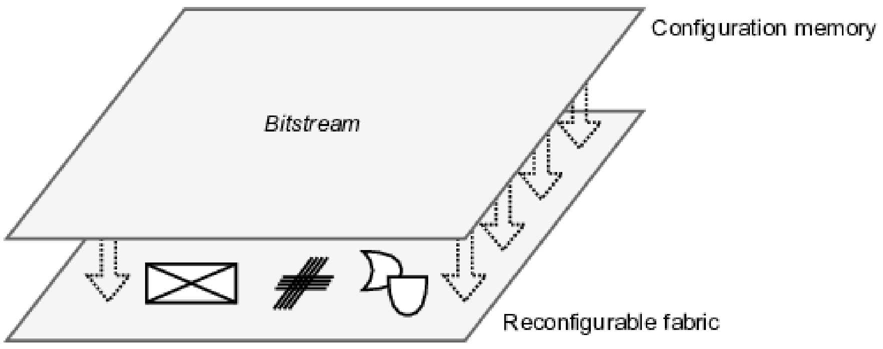


Fig. 3: Functional representation of the configuration memory and reconfigurable fabric in an SRAM-based FPGA

Each RISC-V implementation will be implemented in a certain region. These regions might have some communication, but are ideally completely separated. This separation is how the setting is, prior to this project.

To secure the communication and to be able to do attestation, an isolated implementation is added. This contains the eHSM.

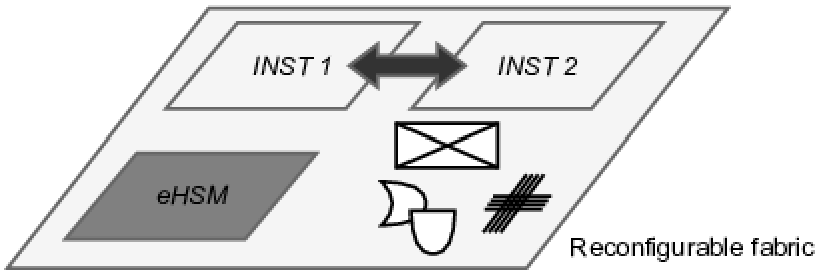


Fig. 4: Architectural representation of the envisioned design, with the eHSM

Bibliography

[1] ISA Specification, Volume 1, Unprivileged Spec v. 20191213, https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf
[2] PX4 - Open source Autopilot for Drones https://px4.io/
[3] https://riscv.org/

USE CASE 4: Environmental Monitoring (VUB)

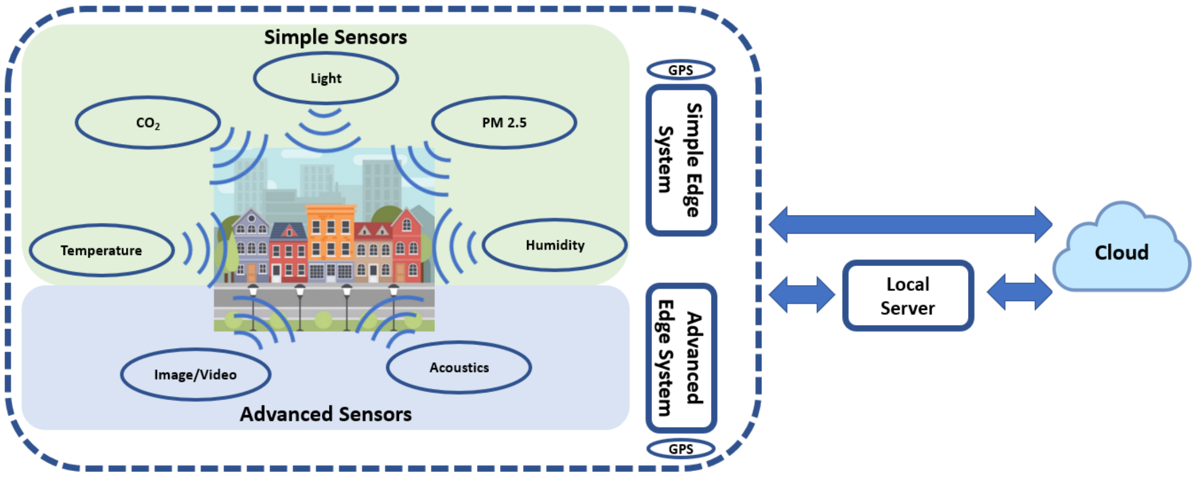


Fig. 5: Environmental monitoring system considered in the VUB use case.

Decomposition of the setup

The targeted use case for VUB will be environmental monitoring (Fig 5.) with different levels of complexity. On the one hand, a simple monitoring system will comprise single sensors to measure environmental parameters such as pollution, temperature, humidity, ... demanding a low-end computational device to process and transmit the collected data. On the other hand, a more advanced monitoring system composed of arrays of sensors (e.g., microphones) and/or complex sensors (e.g., cameras) will present higher system capabilities and different security demands.

Threat analysis

Both types of systems acquire sensor data and have some form of local processing capabilities, local storage, a wireless communication channel, and a GPS. For the scope of threats, we limit our threat analysis to hardware or low-level application threads. The following points highlight the possible security threats the two types of systems might face.

Simple monitoring system:

- In case the device is physically accessed by an undesired user, the acquired sensor data can be read out. In the worst scenario that same user could manipulate the data resulting in false and/or incorrect data.
  Counter-response: data encryption makes data unreadable by third parties. Any modifications to the data voids data integrity.
- The device itself can be moved to an undesired location.
  Counter-response: The GPS location help mitigate the risks of collecting data at unexpected locations.
- Security measures need to be taken in order to detect a device that is not configured/programmed in an appropriate way.
  Counter-response: Firmware/configuration integrity should be detected during firmware attestation.
- The wireless communication channel allows third parties to read out the transmitted data, spoofs data or jam the transmitted data.
  Counter-response: the communication between the edge device and the remote data collection point will be encrypted. Jammed transmitted data will be stored locally until successful transmission.
- Over-the-air updates can be compromised with spoofed firmware/configurations.
  Counter-response: Appropriate authentication and encryption mechanisms should be implemented to circumvent this thread.

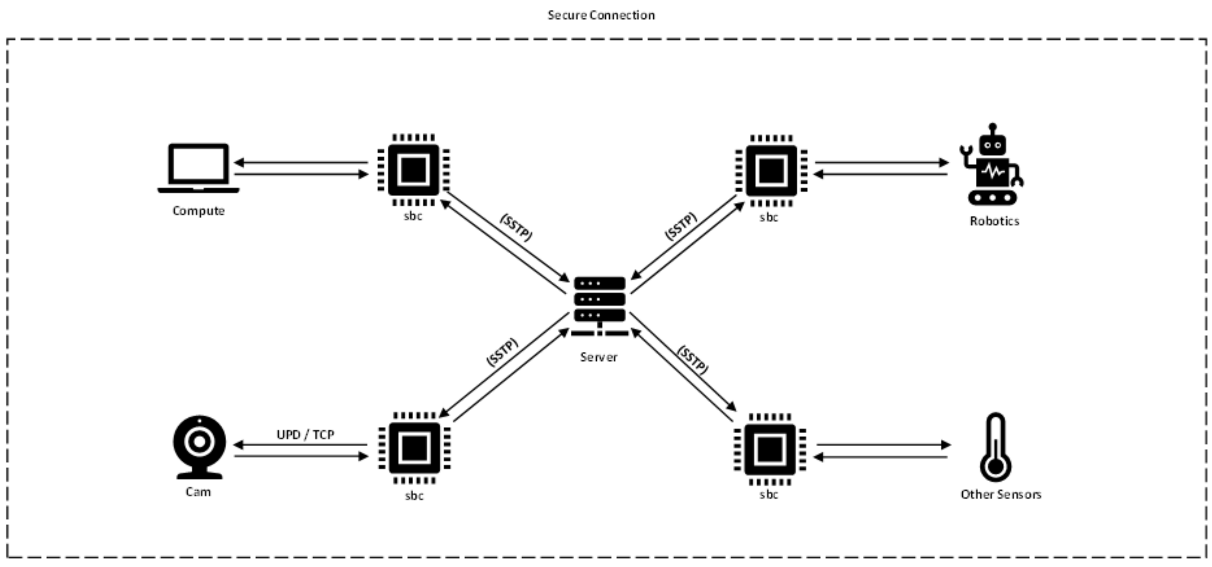
Advanced monitoring system:

- Comprises the same points as listed for the Simple monitoring system.
- Unauthorized access by a third party.
  Counter-response: all communication will be authenticated and encrypted
- Read-out and/or manipulation of the audio and/or video streams by an unauthorized third party. This might lead to the leakage of sensitive recordings.
  Counter-response: All sensitive data must be encrypted locally before it is transmitted. Once transmitted, the data must be securely erased from the local storage.

USE CASE 5: Cooperative robotics (GFaI)

Decomposition of the setup

The targeted use case for GFaI will be to develop a system, which will inspect components manufactured in the industry. The system is illustrated in the following graphic.



The data stream is tunneled via SSTP (Secure Socket Tunneling Protocol) for a secure connection. The server is the centre of the system and the PC can be used to monitor the current status of the system. The Internet of Things (IoT) should be dynamically extensible, so the devices are designed as plug and play. The communication between each device and the server is hosted by one single board computer (sbc) respectively. The goal is to communicate with all devices independently.

Threat analysis

- Identifiable Weak Points in the IoT System are:
  - Devices – Devices themselves can easily be a vulnerability. If not secured the devices, the physical interfaces and firmware can be tampered
  - Communication Channels – The type of communication protocol used within the system can result in a security hole which makes the overall system vulnerable to specific attacks resulting in an overall weaker system

- Countermeasures to protect the IoT System from possible Threats:
  - VPN Tunnel: Building a VPN Tunnel that can ensure an encrypted communication between the devices within the IoT System.

Requirement specifications

Based on the end goal being an IoT environment consisting of actuators, sensors and single-board computers (SBC). The need for a secure communication protocol is paramount. The below listed points summarize the possible protocols which can be used:

- Constrained Application Protocol (CoAP)
  - can be secured through the use of DTLS or TLS
  - operates similar to HTTP
  - runs **over UDP and not TCP**
  - CoAP devices should typically support RSA and AES or ECC and AES
- MQ Telemetry Transport for Sensor Nodes (MQTT-SN)
  - focused on constrained devices (used as sensors/actuators)
  - many-to-many communication protocol (M2M)
- Secure Socket Tunnelling Protocol (SSTP)
  - form of VPN tunnel
  - Point-to-Point Protocol (PPP) for data transfer
  - secured through key negotiation, encryption and traffic integrity checks
- 6LoWPAN
  - Standard IP networking used on low-power mesh and sensor networks

In regard to our user case we chose the SSTP as the primary communication protocol.