SPARTA

D6.3

First Release of Demonstration

Project number	830892
Project acronym	SPARTA
Project title	Strategic programs for advanced research and technology in Europe
Start date of the project	1st February, 2019
Duration	36 months
Programme	H2020-SU-ICT-2018-2020

Deliverable type	Demonstrator
Deliverable reference number	SU-ICT-03-830892 / D6.3 / V1.0
Work package contributing to the deliverable	WP6
Due date	January 2021 – M24
Actual submission date	2nd February, 2021

Responsible organisation	CINI
Editor	Gabriele Costa
Dissemination level	PU
Revision	V1.0

Abstract	This document describes the first release of the WP6: High Assurance Intelligent Infrastructure Toolkit (HAII-T) demonstration.
Keywords	Intelligent infrastructure, secure orchestration, security-by-design

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 830892

Editor
Gabriele Costa (CINI)

Contributors (ordered according to beneficiary numbers)
Branka Stojanović, Katharina Hofer-Schmitz (JR)
Manon Knockaert, Jean-Marc Van Gyseghem (UNamur)
Lukas Malina, Petr Dzurenda (BUT)
Tewodros Beyene (FTS)
Marius Momeu (TUM)
Raimundas Matulevičius, Mari Seeba, Jake Tom (UTartu)
Joaquin Garcia-Alfaro, Jean-Max Dutertre, Jean-Luc Danger, Aimilia Tasidou, Maryline Laurent (IMT)
Emmanuel Baccelli, Michel Hurfin, Ludovic Mé (Inria)
Gianluca Roascio, Paolo Prinetto, Gabriele Costa, Alessandro Armando (CINI)
Giorgio Bernardinetti, Francesco Mancini (CNIT)
Nerijus Morkevičius (KTU)
Uwe Roth, Qiang Tang (LIST)
Marek Pawlicki (ITTI)

Reviewers (ordered according to beneficiary numbers)
Artsiom Yautsiukhin (CNR)
Martin Zadnik (CESNET)

Executive Summary

This document presents the first release of the demonstration of the HAII-T Program. The document starts by introducing a use case based on a real intelligent infrastructure. The use case was designed to embed all the technologies and aspects that are relevant for the activities of the Program. After a general description of the use case and its features we provide an architectural overview of its network. Then, for each of the demonstrated technologies, we focus on a specific application scenario, i.e., on specific subsystems that have a role in the demonstration. Each application scenario chapter provides the details of the corresponding demonstration, staged inside the use case. Finally, each application scenario description includes an evaluation section that provides a critical review of the demonstration.

Table of Content

List of Figures

List of Tables

Chapter 1 Introduction

Modern II are extremely complex and interconnected systems that promise to improve many aspects of our society. For instance, II can integrate many aspects of our society so to provide citizens with faster and better services. An average II can include many technologies, e.g., Cloud applications and IoT devices, and environments, e.g., industry 4.0 production plants and smart buildings. Although II bring concrete benefits to all of these contexts, they also carry security concerns.

In the context of SPARTA, the HAII-T Program focuses on the technologies that are central to any II, e.g., IoT, Fog and Cloud computing, and their security. In particular, HAII-T proposes a security-by-design methodology for ensuring that security requirements are evaluated at each stage of the lifecycle of II. The goal is to obtain overall security guarantees in line with the state-of-the-art security mechanisms which, often, are difficult to be applied to technologies such as the IoT. Our main objective is to provide practical tools implementing such a methodology.

In this document, we describe the year 2 demonstration of the HAII-T Program. All the demonstrations are staged in a single, common use case that we introduce in Chapter 2.1. The network infrastructure for the use case is then given in Chapter 2.2. The use case provides a replica of a real intelligent infrastructure that we extended with technologies relevant to our demonstrations.

Each demonstration is presented through an application scenario. Starting from Chapter 3, each chapter details an application scenario. Every application scenario describes specific security and technological aspects of the use case and presents a methodology to deal with them. Eventually, application scenarios provide an evaluation section for discussing the current level of maturity of the proposed methodology as well as an open issue and future developments.

Chapter 2 Use case

2.1 Overview

The use case is based on a smart building scenario. The scenario is inspired by the Zero-Emission Building (ZEB), which is hosted inside the Genova University Campus, located in Savona (Italy). Besides standard facilities, e.g., study rooms and canteen, the university campus includes research infrastructures and facilities. Among them, there is a smart grid (called Savona Polygeneration Micro-grid – SPM) consisting of several nodes that generate electricity for supplying the internal demand. Figure 2.1 shows a sky view of the campus and, highlighted in red, ZEB location. A closer view of ZEB appears in Figure 2.2.

Figure 2.1: The Savona campus and the zero-emission building.

ZEB is a smart building built with innovative technologies and materials in order to rationalize its energy consumption. It contains several laboratories as well as a gym. Sensors and actuators have been deployed to monitor the environment (e.g., room temperature) and reconfigure it (e.g., by opening a window).

A parking lot for e-vehicles is located in front of ZEB. The parking lot is provided with a Charging Unit (CU) for two vehicles. The CU has two working modalities, i.e., grid-to-vehicle (G2V) and vehicle-to-grid (V2G). G2V is the standard modality of e-vehicles during the recharging process. Instead, V2G allows the SPM to retrieve energy stored in vehicle batteries, e.g., to respond to a demand peak. The CU and the parking lot appear in Figure 2.3.

The use case presented in this deliverable revolves around a fictional, intelligent infrastructure for the

Figure 2.2**:** The zero-emission building (front view).

Figure 2.3**:** The parking lot for e-vehicles.

smart building scenario described above.

2.2 Network infrastructure

In this section, we describe the network infrastructure of the use case.

Figure 2.4 shows the reference network scheme of the use case. Briefly, the smart building hosts a three-segment network. The three segments are *dmz*, *intranet* and *iot*. Servers hosting public services, i.e., those that are accessible from outside the network perimeter, are connected to *dmz*. Other servers and hosts are instead connected to *intranet*. Finally, *iot* is used for connecting various field devices, e.g., sensors and actuators. The smart building network resides behind a router firewall that delimits the perimeter with the external network, i.e., the public Internet.

Networks and nodes are labeled with their *symbolic* names, e.g., ns and www, and IP addresses. Node names are managed by the DNS service running on node ns. Network address space represent the interval of IP addresses that can be assigned to connected devices. For brevity, statically assigned IPs are only represented by the last address segments. For instance, the IP address of node www is 198.51.100.3. If a node has no address label, its IP address is dynamically assigned. Finally, when relevant, we label connections with a specification of the used channel, e.g., IEEE 802.15.4.

More details about the components, subsystems and applications hosted by the smart building network are give in the following chapters.

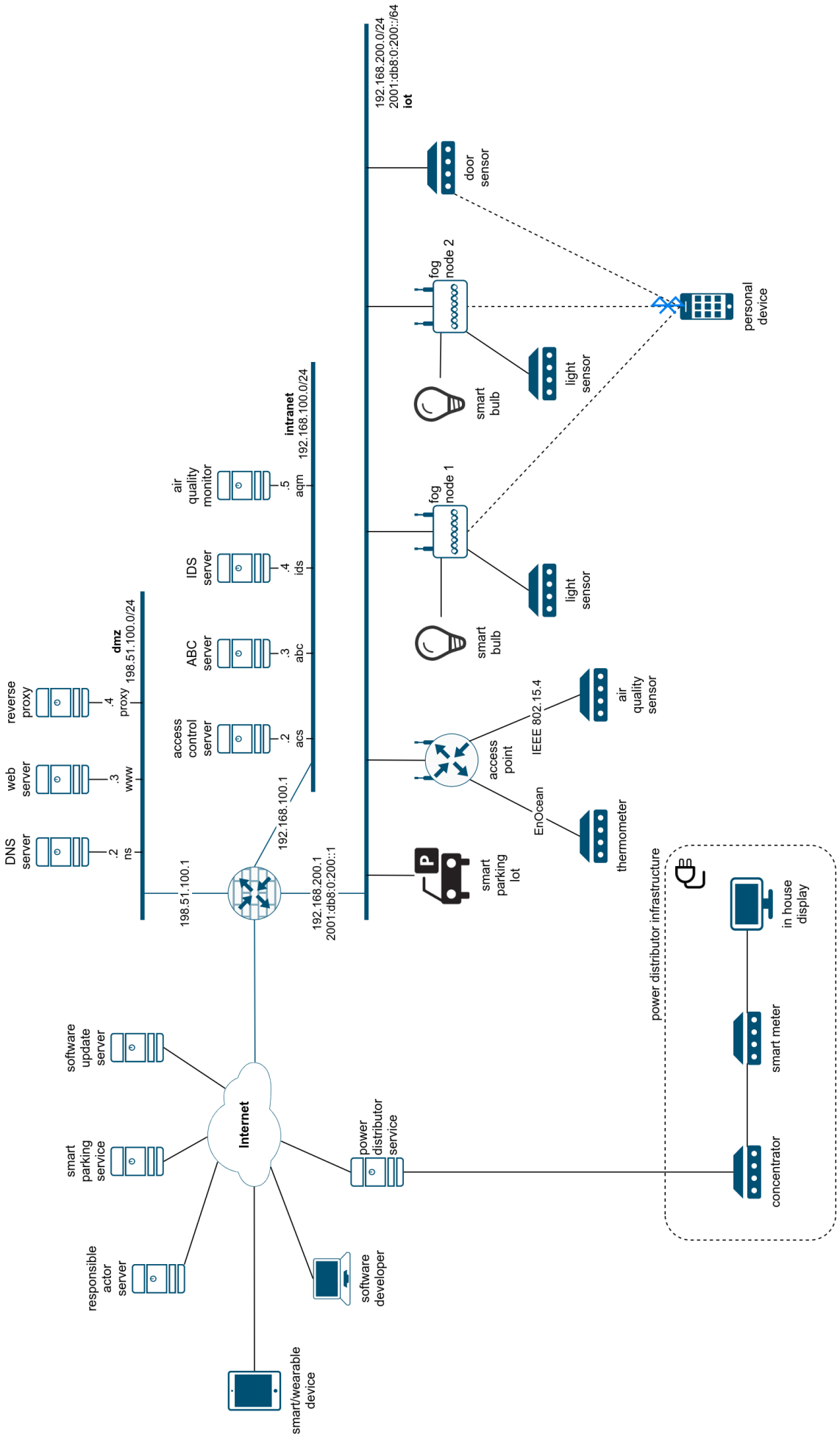

Figure 2.4: Network infrastructure overview.

Chapter 3 Orchestration framework

In this section we describe the design and implementation of the network infrastructure described in Section 2.2. The design and implementation processes are central for the orchestration framework. Through the incremental implementation of the use case, this section demonstrates the main features of the framework.

3.1 Implementation and usage

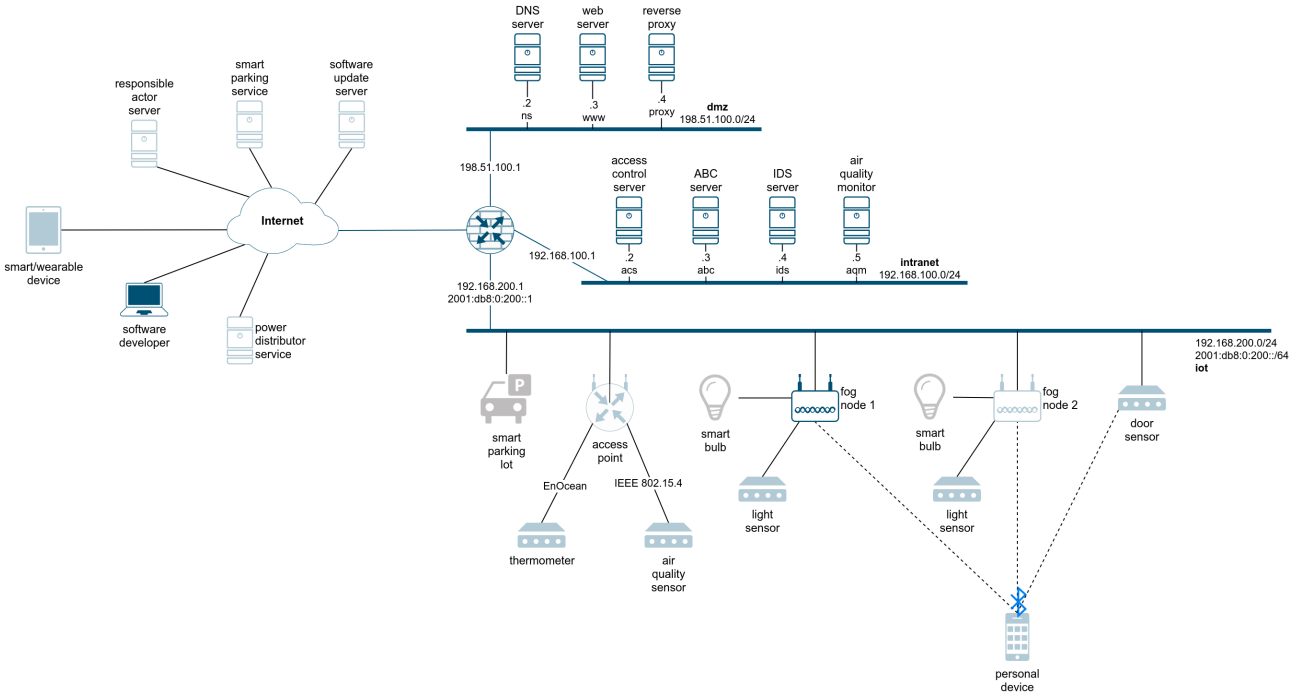

Figure 3.1: Portion of the scenario currently implemented.

The current version does not entirely implement the use case infrastructure. Nevertheless, it includes several components and it covers aspects of interest for the application scenarios (see following chapters). Figure 3.1 shows the currently implemented portion of the scenario. The future evolution of the demonstrator will populate the current implementation with the missing components.

Figure 3.2 highlights the implementation details of the network infrastructure. Below we discuss its core aspects.

- Public network simulation. Several devices connect with the scenario infrastructure by means of the public Internet (see Figure 2.4). Some of them are actual, remote entities (e.g., web servers), while others must be deployed with the scenario. To support this hybrid structure, we rely on a *simulated internet*. The simulated internet is implemented by means of a router (rt-simint) which connects three networks, i.e., *ext*, *simint* and *outside*. Briefly, ext is connected with a virtual interface of the host platform. By means of a virtual bridge, the host interface provides direct connectivity with the Internet.
- VPN access. To make the infrastructure extensible, we provide direct access to each network by means of a virtual private network (VPN) server. VPN server is connected to ext. In this way, VPN can be accessed from both software running on the host platform and even from the Internet. The server accepts connections on different ports. Each port is uniquely mapped into one of the network in the infrastructure. In Figure 3.2, port numbers are used to label (in red) the connection with the corresponding network. For instance, establishing a session with 172.16.255.10:8886 will connect the VPN client to the intranet.

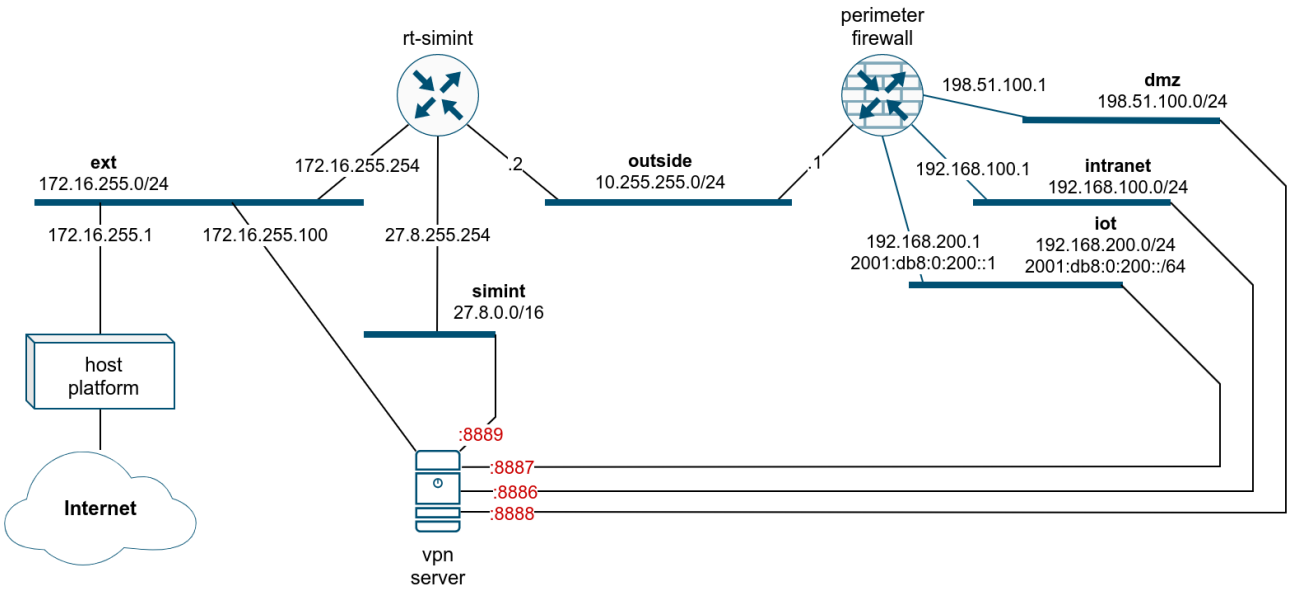

Figure 3.2: Scenario implementation overview.

3.2 TOSCA model

We used TOSCA to design the use case infrastructure. The reasons behind this choice are highlighted in [19]. Below, we present the TOSCA specification used to describe part of the use case infrastructure. Most of the elements implement native TOSCA v1.3 types [29], e.g., networks. The only new type we introduced extends the native Compute node for allowing docker containers integration.

```
node_types:
  docker.nodes.Compute:
    derived_from: tosca.nodes.Compute
    properties:
      ...
```

Briefly, a Tosca specification consists of a list of nodes, each of them having the following structure.

```
name:
  type: ...
  properties:
    ...
  requirements:
    ...
  capabilities:
    ...
```

The meaning is that each Tosca node, uniquely identified by a certain name, must instantiate a certain predefined type. Then, the node definition is populated by assigning values to its type properties, requirements and capabilities. Intuitively, properties include data characterizing the node, e.g., the IP address of a certain network port, requirements include relationships that the node *must* establish, e.g., a network port only exists in relationship to a certain device, and capabilities include relationship that the node *can* establish, e.g., a network port can be connected to a certain network. For more details about the full Tosca syntax we refer the reader to [29].

Networks

Network specification follows the diagram of Figure 3.2. In particular, we have the following network specifications.

```
dmz-net:
  type: tosca.nodes.network.Network
  properties:
    cidr: 198.51.100.0/24
    gateway_ip: 198.51.100.1

intra-net:
  type: tosca.nodes.network.Network
  properties:
    cidr: 192.168.100.0/24
    gateway_ip: 192.168.100.1

iot-net:
  type: tosca.nodes.network.Network
  properties:
    cidr: 192.168.200.0/24
    gateway_ip: 192.168.200.1

iot-net6:
  type: tosca.nodes.network.Network
  properties:
    ip_version: 6
    cidr: 2001:db8:0:200::/64
    gateway_ip: 2001:db8:0:200::1

outside-net:
  type: tosca.nodes.network.Network
  properties:
    cidr: 10.255.255.0/24
    gateway_ip: 10.255.255.1

simint-net:
  type: tosca.nodes.network.Network
  properties:
    cidr: 27.8.0.0/16
    gateway_ip: 27.8.0.1

ext-net:
  type: tosca.nodes.network.Network
  properties:
    cidr: 172.16.255.0/24
    gateway_ip: 172.16.255.1
```

Basically, networks are defined through their address space `cidr` and default gateway `gateway_ip`. The only exception is for IoT network. Since IoT devices use both IPv4 and IPv6, we need to specify two distinct networks. In particular, `iot-net6` models this behavior. To implement the network backbone, we also introduce the following nodes representing the routers of Figure 3.2 and their ports establishing the connection with the corresponding network.

```
rt-simint:
```

```
  type: docker.nodes.Compute
  properties:
    build: ./ubuntu-cracker
    image: spartawp6/ubuntu
    hostname: simint.sparta.ii
    cap_add:
      - NET_ADMIN
      - NET_RAW
    environment:
      - MASQUERADE=172.16.255.0
      - ROUTES=198.51.100.0/24 via 10.255.255.1;blackhole 192.168.0.0/16
      - ROOTPW=test123

rt-simint-port-outside:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 10.255.255.2
  requirements:
    - link: outside-net
    - binding: rt-simint

rt-simint-port-simint:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 27.8.0.254
  requirements:
    - link: simint-net
    - binding: rt-simint

rt-simint-port-ext:
  type: tosca.nodes.network.Port
  requirements:
    - link: ext-net
    - binding: rt-simint

extclient:
  type: docker.nodes.Compute
  properties:
    build: ./ubuntu-cracker
    image: spartawp6/ubuntu
    hostname: extclient
    environment:
      - DNS=198.51.100.2
      - DEFGW=27.8.255.254
      - ROOTPW=test123
    cap_add:
      - NET_ADMIN
      - NET_RAW

extclient-port-simint:
  type: tosca.nodes.network.Port
  requirements:
    - link: simint-net
    - binding: rt-simint
```

```
fw:
  type: docker.nodes.Compute
  properties:
    build: ./firewall-cracker
    image: spartawp6/firewall
    hostname: fw.sparta.ii
    cap_add:
      - NET_ADMIN
      - NET_RAW
    security_opt:
      - seccomp:unconfined
    tmpfs:
      - /tmp
      - /run
      - /run/lock
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:ro
    environment:
      - ROLES=10.255.255.1,red 192.168.200.1, green
              192.168.100.1,green 198.51.100.1,orange
      - DEFGW=10.255.255.2
      - ROOTPW=test123

fw-port-intranet:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 192.168.100.1
  requirements:
    - link: intra-net
    - binding: fw

fw-port-dmz:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 198.51.100.1
  requirements:
    - link: dmz-net
    - binding: fw

fw-port-iot:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 192.168.200.1
  requirements:
    - link: dmz-net
    - binding: fw

fw-port-iot6:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 2001:db8:0:200::1
  requirements:
    - link: dmz-net
```

```
    - binding: fw

fw-port-outside:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 10.255.255.1
  requirements:
    - link: outside-net
    - binding: fw
```

Hosts

The current TOSCA specification includes all the nodes connected to dmz and intranet. Moreover, we have nodes for the software developer node and one of the two fog nodes (see Figure 2.4). To establish connections between nodes and networks, port nodes are also introduced. The specification of nodes and ports follows.

```
acs:
  type: docker.nodes.Compute
  properties:
    build: ./ubuntu-cracker
    image: spartawp6/ubuntu
    hostname: acs.sparta.ii
    cap_add:
      - NET_ADMIN
      - NET_RAW
    environment:
      - DNS=198.51.100.2
      - DEFGW=192.168.100.1
      - ROOTPW=test123

acs-port-intranet:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 192.168.100.2
  requirements:
    - link: intra-net
    - binding: acs

abc:
  type: docker.nodes.Compute
  properties:Secur
    build: ./ubuntu-cracker
    image: spartawp6/ubuntu
    hostname: abc.sparta.ii
    cap_add:
      - NET_ADMIN
      - NET_RAW
    environment:
      - DNS=198.51.100.2
      - DEFGW=192.168.100.1
      - ROOTPW=test123

abc-port-intranet:
```

```
    type: tosca.nodes.network.Port
    properties:
      ip_address: 192.168.100.3
    requirements:
      - link: intra-net
      - binding: abc

ids:
    type: docker.nodes.Compute
    properties:
      build: ./ubuntu-cracker
      image: spartawp6/ubuntu
      hostname: ids.sparta.ii
      cap_add:
        - NET_ADMIN
        - NET_RAW
      environment:
        - DNS=198.51.100.2
        - DEFGW=192.168.100.1
        - ROOTPW=test123

ids-port-intranet:
    type: tosca.nodes.network.Port
    properties:
      ip_address: 192.168.100.4
    requirements:
      - link: intra-net
      - binding: ids

aqm:
    type: docker.nodes.Compute
    properties:
      build: ./ubuntu-cracker
      image: spartawp6/ubuntu
      hostname: aqm.sparta.ii
      cap_add:
        - NET_ADMIN
        - NET_RAW
      environment:
        - DNS=198.51.100.2
        - DEFGW=192.168.100.1
        - ROOTPW=test123

aqm-port-intranet:
    type: tosca.nodes.network.Port
    properties:
      ip_address: 192.168.100.5
    requirements:
      - link: intra-net
      - binding: aqm

fog:
    type: docker.nodes.Compute
    properties:
```

```
    build: ./ubuntu-cracker
    image: spartawp6/ubuntu
    hostname: fog.sparta.ii
    cap_add:
      - NET_ADMIN
      - NET_RAW
    environment:
      - DNS=198.51.100.2
      - DEFGW=192.168.200.1
      - ROOTPW=test123

fog-port-iot:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 192.168.200.2
  requirements:
    - link: iot-net
    - binding: fog

dns:
  type: docker.nodes.Compute
  properties:
    build: ./dns-cracker
    image: spartawp6/ubuntudns
    hostname: dns.sparta.ii
    cap_add:
      - NET_ADMIN
      - NET_RAW
    environment:
      - DNS=127.0.0.1
      - DEFGW=198.51.100.1
      - ROOTPW=test123

dns-port-dmz:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 198.51.100.2
  requirements:
    - link: dmz-net
    - binding: dns

www:
  type: docker.nodes.Compute
  properties:
    build: ./www-cracker
    image: spartawp6/ubuntuwww
    hostname: www.sparta.ii
    cap_add:
      - NET_ADMIN
      - NET_RAW
    environment:
      - DNS=198.51.100.2
      - DEFGW=198.51.100.1
      - ROOTPW=test123
```

```
www-port-dmz:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 198.51.100.3
  requirements:
    - link: dmz-net
    - binding: www

proxy:
  type: docker.nodes.Compute
  properties:
    build: ./ubuntu-cracker
    image: spartawp6/ubuntu
    hostname: proxy.sparta.ii
    cap_add:
      - NET_ADMIN
      - NET_RAW
    environment:
      - DNS=198.51.100.2
      - DEFGW=198.51.100.1
      - ROOTPW=test123

proxy-port-dmz:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 198.51.100.4
  requirements:
    - link: dmz-net
    - binding: proxy
```

VPN management

As explained above, we use a VPN server to provide direct connectivity with each network. To do this, we introduce the following specification.

```
vpn:
  type: docker.nodes.Compute
  properties:
    build: ./vpn-cracker
    image: spartawp6/vpn
    hostname: vpn.sparta.ii
    cap_add:
      - NET_ADMIN
      - NET_RAW
    environment:
      - MASQUERADE=172.16.255.0
      - ROUTES=198.51.100.0/24 via 10.255.255.1;blackhole 192.168.0.0/16
      - ROOTPW=test123

vpn-port-dmz:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 198.51.100.100
```

```
  requirements:
    - link: dmz-net
    - binding: vpn

vpn-port-intranet:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 192.168.100.100
  requirements:
    - link: intra-net
    - binding: vpn

vpn-port-iot:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 192.168.200.100
  requirements:
    - link: iot-net
    - binding: vpn

vpn-port-simint:
  type: tosca.nodes.network.Port
  properties:
    ip_address: 27.8.0.100
  requirements:
    - link: simint-net
    - binding: vpn
```

3.3 Scenario deployment and execution

For scenario deployment we opted for Docker Compose,[1] i.e., the default Docker orchestrator. The choice of Docker is mainly driven by the limited resources requirede by Linux containers w.r.t. other virtualization techniques, e.g., hypervisor-based virtual machines. To support continuous integration, the scenario code is hosted on a private github repository at https://github.com/enricorusso/spartawp6.[2]

Moreover, to favor portability and testing, we released a Linux appliance for executing the use case. The appliance is released as a OVA image for both VirtualBox[3] and VMWare Player.[4] The virtual machine includes three scripts for updating, starting and stopping the infrastructure. Briefly, the update script downloads the last version of the infrastructure from the official repository, the start script invokes Docker Compose to launch all the infrastructure elements and the stop script interrupts them.

When the infrastructure is running, it is possible to extend it via VPN. To do this, it is sufficient to configure a VPN client[5] on a connected host. Below, is a minimal VPN configuration file for connecting a host to the intranet with IP address 192.168.100.200. A general purpose script is available on the VM.

```
# OpenVPN - SPARTA client
```

[1] https://docs.docker.com/compose/
[2] A public repository will be provided for the year three demonstrator.
[3] https://www.virtualbox.org/
[4] https://www.vmware.com/products/workstation-player.html
[5] These instructions have been tested with OpenVPN.

```
remote 192.168.58.139 # server ip address (IP of the host running docker)
port 8886 # intranet

dev tap
proto tcp-client

ifconfig 192.168.100.200 255.255.255.0 # intranet

ifconfig-nowarn

secret key.txt
ping 10
comp-lzo
verb 3

intranet
route 198.51.100.0 255.255.255.0 192.168.100.1
route 10.255.255.0 255.255.255.0 192.168.100.1
route 192.168.100.0 255.255.255.0 192.168.100.1

script-security 2
dhcp-option DNS 198.51.100.2
dhcp-option DOMAIN sparta.ii
```

3.4 Evaluation

The main goal of the orchestration framework is to support the entire life cycle of an intelligent infrastructure. In the previous sections, we highlighted how the use case infrastructure of Chapter 2.1 can be modeled in TOSCA. Then, we converted the TOSCA specification into directives for deploying and configuring a virtual replica. Both the specification and the virtual implementation are crucial assets for the demonstrations of the application scenarios.

We could model a significant portion of the target infrastructure in TOSCA. The entire modeling only required few hundreds of lines of code. In terms of performances, we can execute the existing infrastructure inside a relatively small virtual machine, only requiring 2 GB memory and 20 GB disk to run.

Clearly these results are only preliminary. As a matter of fact, several details are still to be integrated. In particular, actual implementations of services are still missing. Also, part of the IoT infrastructure must be modeled and implemented. Finally, physical devices and existing services must be integrated via VPN. All in all, the current results are promising, but further investigation and experiments are needed.

Chapter 4 Door sensor

Placed within the IoT network, the door sensor has the task of monitoring the access of individuals inside the building. To do this, it pairs in Bluetooth with the mobile device of the person accessing, and requests the transmission of a unique identifier. Subsequently, upon request, it is able to transmit this information via Wi-Fi for central analysis. Figure 4.1 shows the position of the device in the overall scenario.

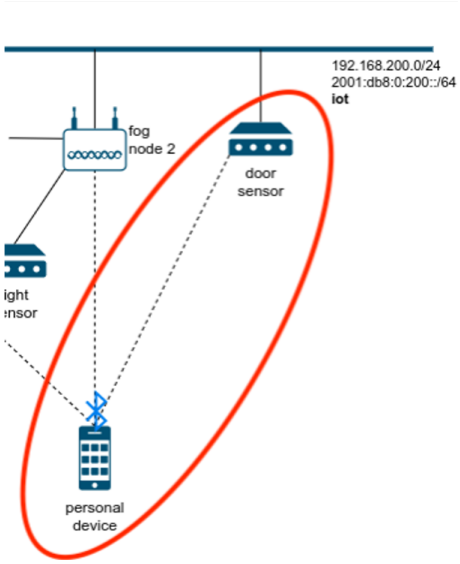

Figure 4.1: Position of the door sensor inside the demonstrator, with reference to Figure 2.4.

An app is installed on the personal device, capable of interacting with the door sensor to pass it the person ID. The firmware on the sensor mistakenly considers the mobile app as a trusted actor. Therefore, knowing the length of the information in terms of bytes, it does not check the size of the input, and truncates it at the first string termination byte. This is a very serious vulnerability, because it allows an attacker to corrupt the memory of the IoT device and initiate an attack to steal data, enter the network, change the credentials for the device setup, and so on. In particular, if the attacker somehow gets the firmware binary running on the device (e.g., by discovering that a commercial known binary runs on it), he would be able to bypass even a possible *non-executable-stack* defense, and to mount a *Return-Oriented Programming (ROP)* attack [37][45][46].

Here comes the defense. The door sensor is a particular IoT device, because its main chip on board is a hybrid MCU-FPGA system. Therefore, to remedy possible vulnerabilities in the exchange of information and in coding (often entrusted to third parties), the FPGA inside the main chip of the door sensor is instructed to act as a *Control-Flow Integrity (CFI) Monitor* [6]. Once received by the third-party, the firmware to be flashed on the device is instrumented by a proprietary, certified and trusted tool, which traces the *Control-Flow Graph (CFG)* and inserts, for each program flow transfer considered as dangerous, single communication instructions to the FPGA regarding the position inside the code. Such information is also encoded in a particular format to be inserted in the architecture implemented within the FPGA. The programming of the door sensor is therefore split in two: (i) loading an instrumented binary into the Flash memory of the MCU and (ii) loading a specific bitstream into the Flash-based FPGA cells. During the execution, the FPGA waits for the CPU to transmit control flow information, and without stopping the CPU, and in a very efficient way from the computational point of view, it is able to detect flow anomalies taken by the program, and to stop it in few clock cycles, with the possibility of even reporting the accident to the network administrator.

More details on this FPGA-based solutions used to secure the door sensor can be found in [31].

4.1 Advantages of the FPGA-based defense against binary attacks

The door sensor is implemented through a hybrid MCU-FPGA SoC called **SE***cube*™ [1]. Hybrid SoCs are no longer that rare in the IoT world, given the ever increasing need to distribute the computational load more towards the source (edge computing). It is expected that, over the next few years, hardware vendors will focus on delivering computing hardware to execute complex, compute-intensive functions at the edge. In this context, hybrid chips based on CPU and FPGA components, will be the easiest and most power/cost-effective way to meet the new edge computing hardware requirements. Although there are still a few examples on the market, mostly provided by FPGA vendors who embed ARM or NIOS cores in their devices, hybrid CPU+FPGA chips are expected to become increasingly popular in the next years.

The use of an FPGA that lives on the same chip as the processor to defend against software exploits is a solution that meets both the needs of reconfigurability and upgradeability typical of purely software solutions, and those of greater efficiency and isolation typical of purely hardware solutions. The defense does not degrade system performance, as (i) the CPU only has to take the time of a single **OUT** instruction to the FPGA interface to transfer a constant data, already inserted in the binary, and (ii) the integrity check is done in parallel without stopping the CPU activity, within a dedicated architecture that takes a very short time to complete its taks.

The static analysis tool and the infrastructure for programming the door sensor are of course exclusively owned by the network administrator, who physically proceeds to secure it via offline programming (i.e., by using physical interfaces such as JTAG or SPI with authentication). The door sensor cannot be attacked from a physical point of view, as the **SE***cube*™ packaging is resistant to common side-channel or fault injection attacks [21][9], as well as desoldering and other more invasive attacks immediately determine its unusability. Therefore, it is obvious that disconnecting the FPGA from inside the chip is also impossible.

Furthermore, the firmware running on the door sensor is not even aware of the presence of the FPGA on board, so in the binary there are no information-exchange instructions, which are only added by the instrumentation prior to the physical programming. In any case, the IP core of the CFI monitor on the FPGA only supports data input and never data output: at the time of programming, the pins are forced to allow transmission in one direction only, so there is no the possibility of any sensitive information leakage.

4.2 Evaluation

The proposed solution certainly has a notable theoretical advantage, for all the reasons discussed in the section above. The experiments conducted so far have seen the physical implementation of the monitor on the **SE***cube*™ FPGA, the instrumentation of some benchmark applications already known in the literature, and the execution of these in the protected environment. An occupation of less than 3% was found in relation to the pure logic cells of the FPGA, while most of the resources employed concentrate on the memories, necessary to store the information for applying the protection. These data are comforting, considering that the reconfigurable hardware is very modest in size (just 7K LUTs and 30KB of available memory). The impact of the instrumentation on the additional code memory occupied and on the execution times was found to be very small up to a few percentage points more, except in specific cases. All experimental data can be found in [31].

The binary instrumentation needed to support the FPGA defense has so far been carried out manually or in a very poorly automated way. We are already working on a complete automatic software for the recognition of vulnerable points and their instrumentation, which is increasingly refined and less impactful also thanks to a semi-formal approach adopted.

To keep pace with today's hardware development trends, the portability of this solution within a RISC-V-based architecture[2] is under evaluation. Security-responsible hardware modules would no longer

[1] www.secube.eu
[2] www.riscv.org

be implemented in external reconfigurable hardware, but they would be directly placed within the microcontroller architecture to be protected, resulting in a higher level of transparency, reliability and performance.

Chapter 5 Secure low-power OS software updates with RIOT and SUIT

In this scenario, we address the security of software update for low-power IoT devices (e.g. based on small microcontrollers) over the network (e.g. a low-power radio link). We co-author an IETF standard in this field: the SUIT manifest standard [32], which specifies a security architecture, and the necessary metadata and cryptography to secure software updates, applicable on microcontroller-based devices, such as the ones RIOT runs on. RIOT [8] is an open source general-purpose operating system for low-power IoT devices, which we base our work on and develop in the context of this project.

5.1 Indoor air quality monitor

We have built a prototype of low-cost, low-power indoor air quality monitor, depicted in Fig. 5.2. This prototype bundles a popular COTS system-on-chip (an Arm Cortex-M microcontroller communicating with IEEE 802.15.4 low-power radio) connected via I2C/SPI bus on-board to a variety of sensors: a humidity sensor (the Bosch bme280 sensor), a metal oxide (MOX) gas sensor (the GY-CC811 sensor) to detect a wide range of Volatile Organic Compounds (VOCs), and a dust particle sensor using an optical sensing method (the Telaire sm-pwm-01c sensor) to measure concentration of fine dust particles of diameter over 1 micrometer.

This AirMonitor prototype targets indoors deployments, and integrates in the global HAII-T infrastructure as shown in Fig. 5.2. The AirMonitor is connected to the network via a low-power radio access-point, through which it communicate via IPv6 (6LoWPAN) protocols with a remote software udpate server. The software running on the low-power prototype is based on RIOT Release 2020.10 [36], which we contributed to upstream, to enhance it with security features as described below.

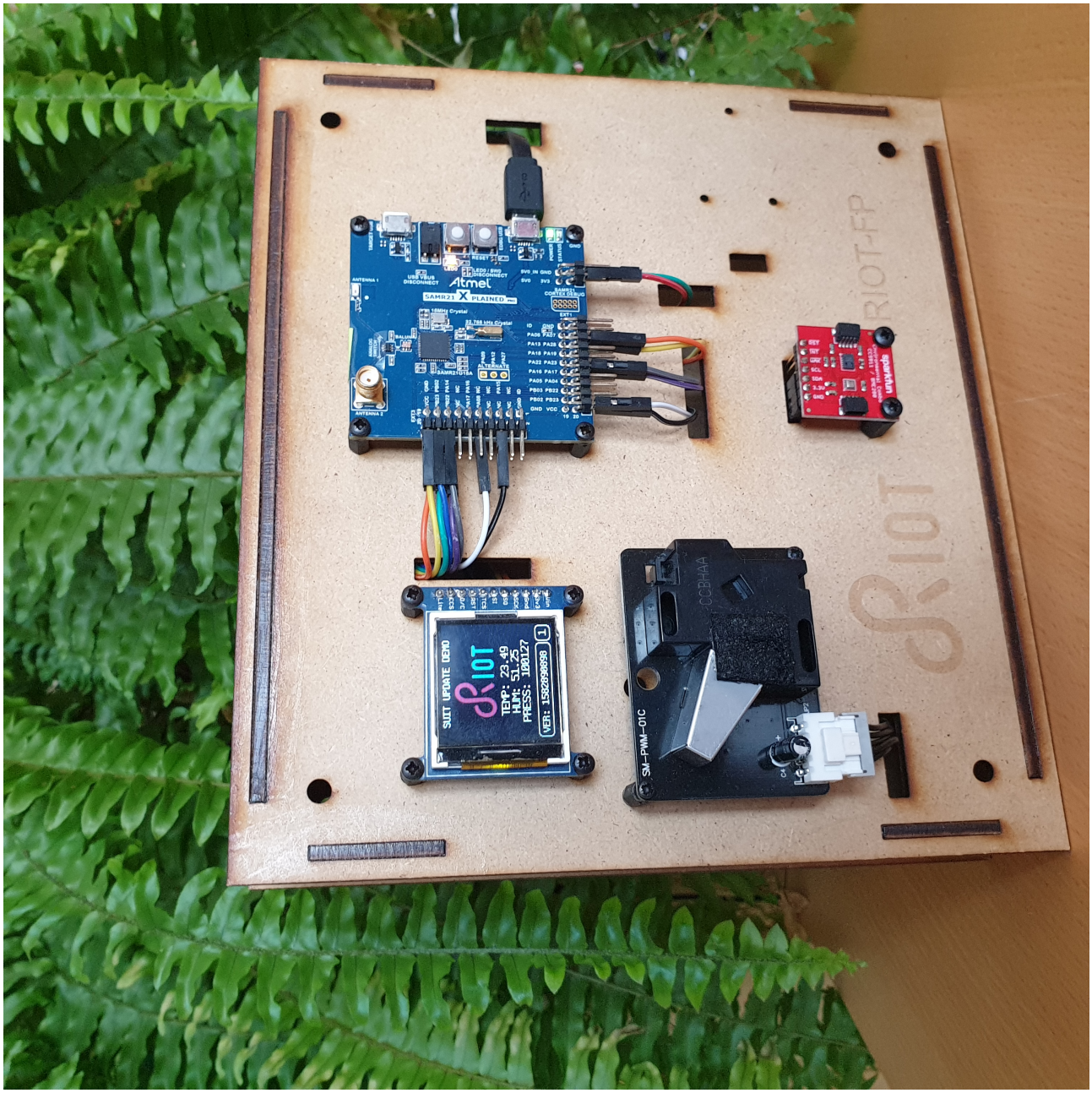

Figure 5.1: Indoor Air Quality Monitor Prototype.

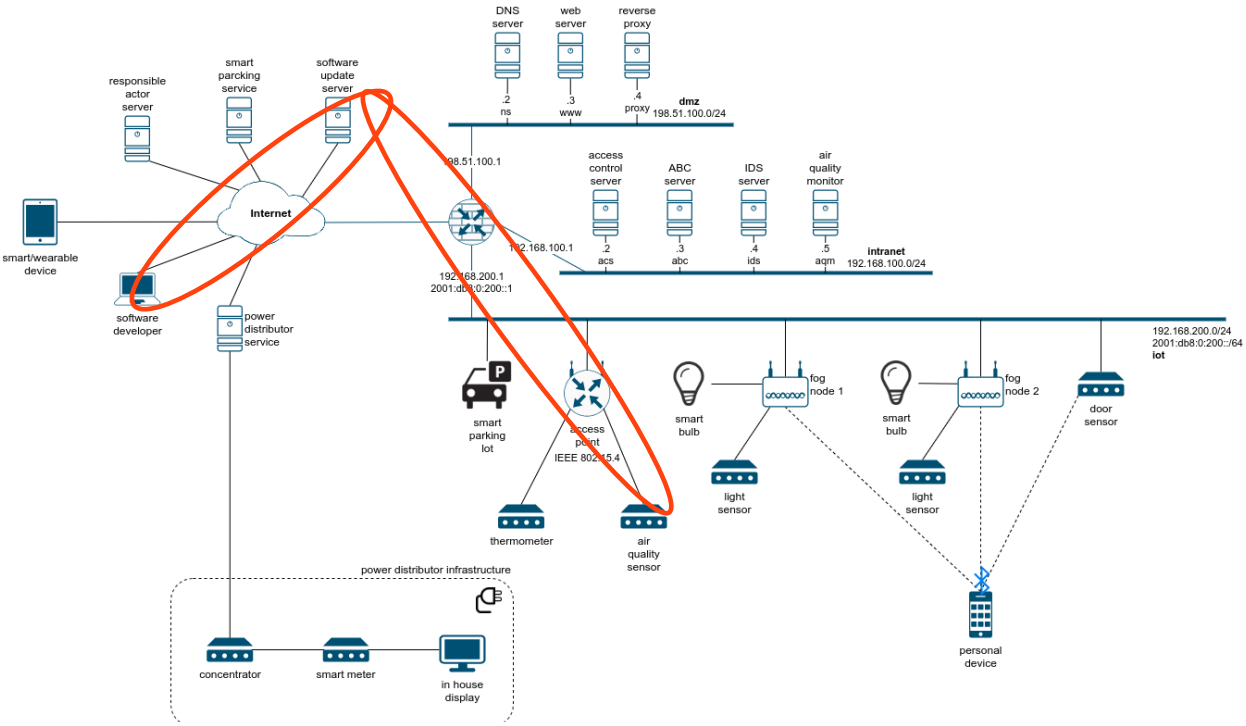

Figure 5.2**:** AirMonitor integration in the global HAII-T infrastructure.

5.2 AirMonitor low-Power software update scenario: deployment & execution

A commissionning phase provisions the AirMonitor with a bootloader, an initial operating system (RIOT) image, and cryptographic material including the public key of the authorized software maintainer.

Next, software udpate and maintenance life-cyles can take place on the Air monitor, over the low-power network, in compliance with the SUIT standard (draft-ietf-suit-manifest-09 [32]). The authorized AirMonitor software maintainer can:

1. build a RIOT software update,
2. produce the corresponding SUIT manifest, i.e. metadata and cryptographic material,
3. upload the SUIT manifest and the RIOT update to the software update server.

In turn, the AirMonitor can

1. poll the software update server for available software updates,
2. download and verify SUIT manifests,
3. download and validate RIOT software updates, reboots if validated (else: sends an notification).

As such, this scenario demonstrates end-to-end security for software updates on low-power IoT devices, providing strong cryptographic guarantees, for example, based on ed25519 digital signatures and SHA256 hashing – the use of alternative cryptographic primitives is also possible. By end-to-end, we here mean from the authorized software maintainer to the AirMonitor.

Using the mechanisms specified by SUIT, the prototype we demo can for example mitigate the below attacks.

5.2.0.0.1 Mitigating Tampered Firmware Update Attacks

An attacker may try to update the IoT device with a modified and intentionally flawed firmware image. To counter this threat, our prototype based on SUIT uses digital signatures on a hash of the image binary and the metadata to ensure integrity of both the firmware and its metadata.

5.2.0.0.2 Mitigating Unauthorized Firwmare Update Attacks

An unauthorized party may attempt to update the IoT device with modified image. Using digital signatures and public key cryptography, our prototype based on SUIT ensure that only the authorized maintainer (holding the authorized private key) will be able to update de device.

5.2.0.0.3 Mitigating Firmware Update Replay Attacks

An attacker may try to replay a valid, but old (known-to-be-flawed) firmware. This threat is mitigated by using a sequence number. Our prototype based on SUIT uses a sequence number, which is increased with every new firmware update.

5.2.0.0.4 Mitigating Firmware Update Mismatch Attacks

An attacker may try replaying a firmware update that is authentic, but for an incompatible device. Our prototype based on SUIT includes device-specific conditions, which can be verified before installing a firmware image, thereby preventing the device from using an incompatible firmware image.

5.3 Evaluation

We published work in [48] [47] [32] [18] [35] describing and evaluating the mechanisms we designed and used in the AirMonitor scenario. This workflow aims to be both general purpose and applicable to secure software maintenance on very low-power IoT devices.

We studied the applicability of this workflow on different types of low-power network technologies (including but not limited to IEEE 802.15.4, BLE...), over various protocols for secure transport [43] including but not limited to the DTLS/TLS variants we study in [35]. In future work, we plan to study our scenario using other radio technologies (e.g. 6TiSCH) and other communication security primitives (e.g. OSCORE/EDHOC).

This workflow is applicable either for full firmware updates as we studied in [48], or for modular software updates. For instance, this workflow can be applied to secure the update of small virtual machines hosted in RIOT as we designed in our work on rBPF [47]. In this later work, we show that using such VMs is promising to enable modular updates, as rBPF requires very small (10%) memory overhead. Our preliminary evaluation of SUIT overhead on various microcontrollers in [48] shows that this approach fits, but relative resource budget varies quite a bit, and needs further experimental evaluation. In future work, we plan to integrate and study additional cryptographic primitives (e.g. a selection of post-quantum NIST competition candidates) in the workflow of our secure low-power IoT software udpate scenario.

Chapter 6 Privacy-enhancing authentication system

This section presents the demonstration of Privacy-Enhancing Authentication System (PEAS) and its integration in chosen intelligent infrastructure environment, i.e. smart building. The authenticated protocol is privacy-enhancing, since it does not disclose whole user identity to a verifier. Only necessary pieces of the user identity (e.g. age, gender, membership etc.) are provided during the verification phase. Furthermore, the authentication sessions are mutually unlinkable. Therefore, the protocol protects user identity, and avoid profiling and trucking users.

PEAS is based on an Attribute-based Credentials (ABC) cryptography protocol, namely, Keyed-Verification Anonymous attribute-based Credentials (KVAC), published in [13]. KVAC employs algebraic MAC and Boneh-Boyen signatures. The current version integrates the revocation mechanism for the identification and revocation of misbehaving users. This Revocable Keyed-Verification Anonymous attribute-based Credentials (RKVAC) system is designed in order to offer revocation and be secure and practical also for running on constrained platforms such as smartcards, smartphones, wearables and single-board computers widely used in intelligent infrastructures. PEAS supports a user centric approach, i.e., users are real owners of them personal data and they decide which the information provide to who and when. Hence, PEAS mitigates storing user personal information that are irrelevant for service providers which is in line with the GDPR regulation.

The selected demonstration scenarios of PEAS are:

- Smart building access control – PEAS works as the privacy-preserving access control system for secure and privacy-enhancing access of persons (users) inside smart buildings, to protected areas, rooms, departments, machines.
- Smart parking access control — PEAS works as the privacy-preserving access control system in privacy-enhancing parking services and parking areas used by drivers.

6.1 Description of PEAS

The PEAS topology is depicted in Fig. 6.1 with the following entities:

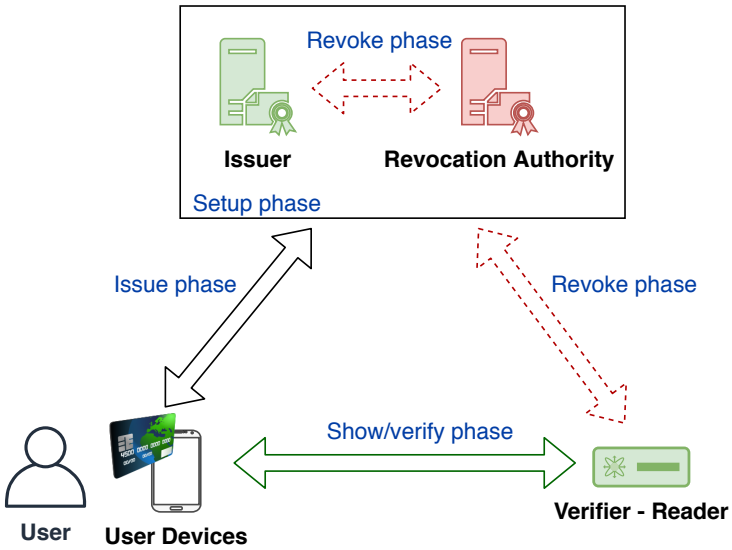

Figure 6.1: Highlevel Topology of PEAS.

- **Issuer** (I): is responsible for issuing attributes (personal attributes) to a user aggregated in a cryptographic credential *cred* using the `Issue` algorithm. The credential is digitally signed by the issuer's secret key. The issuer can be implemented as the ABC server in the network infrastructure depicted in Fig. 2.4.

- **Revocation Authority** (RA): assigns and issues a unique revocation handler m_r to each user using the `Issue` algorithm. The Revocation Authority can revoke users by revoking their revocation handlers. The revocation handler is linked to the user identifier. Thus, if the handler is revoked, the user with all his/her attributes is revoked in the scenario. RA can be deployed as the web server or as an external server in the network infrastructure, Fig. 2.4.
- **User** (U): gets the credential *cred* that includes issued his/her attributes from the Issuer and anonymously proves the possession of attributes to the Verifier using the `Show` algorithm. Further, the user has to compute a one-time per-session pseudonym C which is linked to the credential *cred* via the revocation handler m_r and prove that this one-time pseudonym is not revoked. It is recommended to use secure handheld devices as user items such as smartcards or smartphones with secure elements. The data exchange is mainly via a wireless communication interface, e.g., Bluetooth, NFC, ISO/IEC 14443. The user is represented as the personal device in the network infrastructure, Fig. 2.4.
- **Verifier** (V): verifies the possession of required attributes from the user, and checks the revocation status of the revocation handler using the `Verify` algorithm and the verifier's secret key sk_V and RA's public key pk_{RA}. The Verifier can be represented by a small PC device (embedded PC), i.e., door sensor, or as the access control server in the network infrastructure, Fig. 2.4.

PEAS consists of 4 main phases: Setup, Issue, Show, Revoke. Fig. 6.2 depicts these phases and their interactions with entities. The phases are described in the following text:

- `Setup`: Issuer and Revocation Authority input the security parameter κ and generate the public system parameters and private/public keys. The setup phase runs on I and on RA separately (steps 1. and 2. in Fig. 6.2). The Issuer creates the private key also for the Verifier. The RA generates an empty revocation list RL and an empty list of revocation handlers RH. The RA outputs private and public keys (sk_{RA}, pk_{RA}) and parameters. Public parameters, pk_{RA} and $params_{RA}$, are securely distributed to other entities.
- `Issue`: The phase consists of two sub-algorithms: `IssueRA` and `IssueI`. The `Issue` phase is firstly run at RA (3.a in Fig. 6.2) and after that it is run at the Issuer (3.b in Fig. 6.2). The algorithm inputs the private key of the Revocation Authority sk_{RA}, the issuer's private key sk_I, a list of personal attributes $(m_1, \ldots, m_{n-1})$, user's ID and the list of all revocation handlers of all users RH. The algorithm outputs signature on all user attributes σ (i.e., cryptographic credential), the revocation handler m_r, the (signed) randomizers and updates the list of revocation handlers RH.
- `Show/verify`: The algorithms `Show` and `Verify` are run between the User and the Verifier. On the user's side, the algorithm inputs user's attributes $(m_1, \ldots, m_{n-1}, m_r)$, the signature σ (i.e. cryptographic credential), the set of randomization pairs $\{(e_1, \sigma_{e_1}), \ldots, (e_k, \sigma_{e_k})\}$, the indices of disclosed attributes $m_{z \in D}$ and the identifier of the current time epoch $epoch$. On the verifier's side, the algorithm inputs the verifier's private key sk_V, the revocation lists RL and the epoch identifier. The user outputs the pseudonym C and the cryptographic proof π of the attributes possession. In the **smart building** use case, the attributes can be customized as floor/room/machine access attributes for groups of legitimate users. In the **smart parking** scenario, the user can show attribute proving that is a legitimate user of the parking lot. The Verifier outputs logical value 0/1, i.e., permit or deny access.
- `Revoke`: This phase is typically run between the Verifier and the Revocation Authority. The algorithm inputs the RA's private list of revocation handlers RH, the public revocation list RL, the RA's private key and the communication transcript π, C received from the Verifier. The algorithm outputs updated revocation list RL'. The RA is able to reconstruct all pseudonyms of all users for every epoch. RA is able to identify the owner of the pseudonym (since RA stores the link between the revocation handler m_r and the identity of the user in RH).

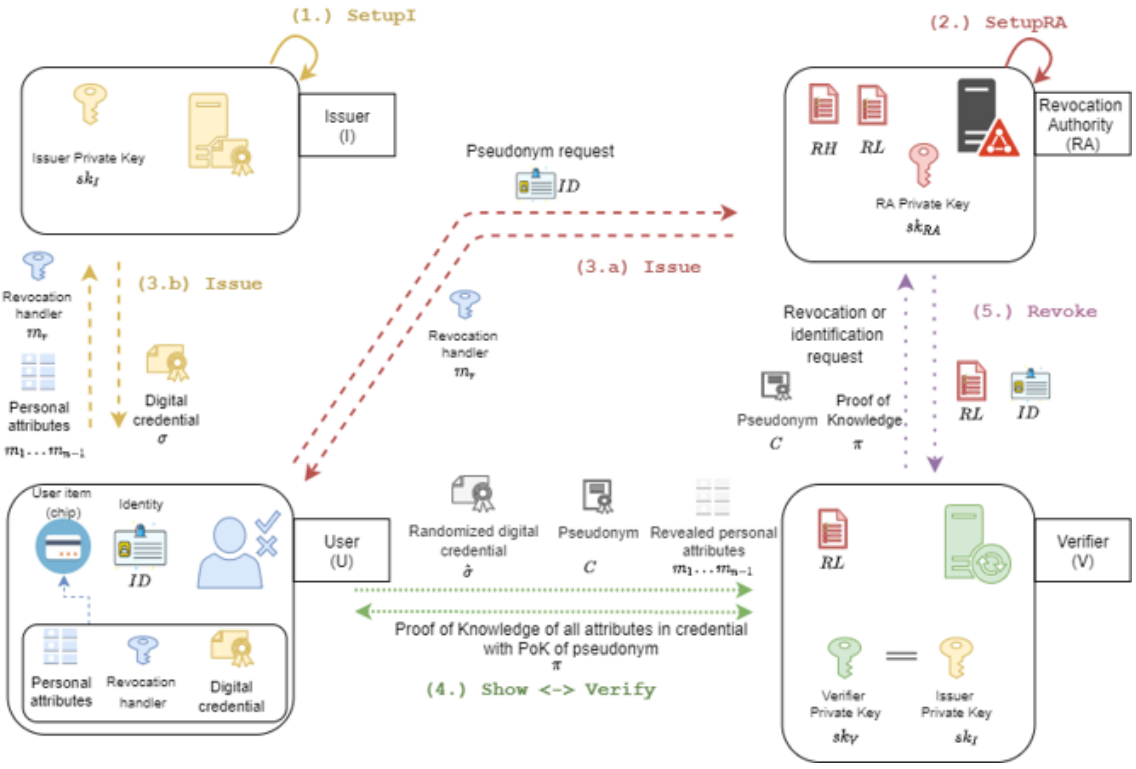

Figure 6.2: Main Phases of Revocable ABC in PEAS.

6.2 Security analysis of PEAS

The PEAS cryptographic core (RKVAC) is composed of two crucial components: the keyed-verification credential scheme [13] and the scalable revocation scheme [12]. More details, including the formal security models and formal security proofs, can be found in our aforementioned papers on these components.

The keyed-verification credential scheme has been proven secure in the random oracle model in [13] under the n-Strong Computational Diffie-Hellman Inversion Problem. This PEAS part follows the security model defined by Chase *et al.* [17] with these security properties:

- *Correctness*: honest users are almost always accepted (there is negligible probability that a honest user is rejected due to a hash collision when the revocation list is compared),
- *Soundness* dishonest users are almost always rejected,
- *Anonymity*: protocol transcripts are indistinguishable from simulations,
- *Key-Parameter Consistency*: there is a unique private key to every public key/parameter.

The revocation scheme has been proven secure in the random oracle model in [12] under the l-Decisional Bilinear Diffie-Hellman Inversion Problem (l-DBDHI) assumption [20]. This component follows the security model defined in [12] with these requirements:

- *Revocation Completeness*: unrevoked users will almost always pass the revocation check.
- *Revocation Soundness*: revoked users will almost always be rejected by the revocation check.
- *Revocation Privacy*: protocols will not release any private data during proving the possession of an unrevoked revocation handler.

To be noted that the RKVAC scheme is symmetric, therefore, RA is able to create a valid proof of a user. Further, all parameters from I and RA should be digitally signed and stored in a security storage/element. In addition, the user should keep the history of epochs and control if the epoch is not reused in order to prevent linkability.

6.3 PEAS implementation

The first release of the PEAS implementation consists mainly of a cryptographic RKVAC library which is written in the C programming language. The library also requires the third-party libraries such as OpenSSL [5], MCL [39] and GMP [4]. The RKVAC library provides main APIs for all phases in PEAS and supports 10 attributes per user. The `Show` and `Verify` (between U and V) algorithm employs a pairing-friendly BN-254 curve generated by the MCL library. The PEAS implementation could be run on each system entity, i.e., I, RA, V and U, as the separated proof-of-concept application with a simple user interface.

In future work, the PEAS implementation will be enhanced by a web-based GUI for V, I and RA, the Android application on the user side and other communication layers connecting enhanced applications of all entities.

6.4 Evaluation of PEAS: experimental results

The overall performance of PEAS can be evaluated mainly by the running time in the Show/Verify phase, i.e., the interaction of a user item with a verifier device. Figure 6.3 depicts the comparison between the RKVAC scheme (red) and the original keyed-verification (KVAC) scheme lacking revocation [13] (blue) for different numbers of attributes stored and disclosed on single-board PC platform (Arduino DUE) with 84MHz ARM Cortex-M3 CPU, 512 kB and 96 kB SRAM. The total code size of the implemented library is approx. 32 kB in this experiment. Running functions require approx. 2.6 kB in RAM. Thus, this implementation of PEAS can be run on various constrained devices with limited RAM and a memory storage on the user side such as embedded devices, smartcards or smart watches. The results are the arithmetic means of 10 measurements in milliseconds.

In this experimental measurement, the `Show` and `Verify` algorithm employs the micro-ecc library with secp160r1,secp192r1, secp224r1, secp256r1,secp256k1 elliptic curves that allow us to get approximated runtimes on the verifier and user side.

Independently of the number of attributes used, the RKVAC scheme requires always only ca. 1.6 s to generate the ownership proof with all but revocation attributes disclosed. Nevertheless, the KVAC scheme (without revocation) requires only ca. 170 ms and is faster than RKVAC. Therefore, KVAC can be considered as a suitable solution in use cases which require fast verification (even on constrained platforms) and no user revocation.

The complexity of both schemes increases with the number of undisclosed attributes, each undisclosed attribute increases the `Show` time by ca. 100 ms. This experimental implementation is limited to 10 attributes per user due its design for memory constrained devices, but the available memory resources of some smart devices such as watches or mobiles promise storing up to tens to hundreds attributes. Furthermore, current smart devices such as watches or mobiles with powerful computational capacities, e.g. CPU 800 - 1000MHz, can provide shorter runtimes of both schemes (KVAC and RKVAC) than the ARM-84-MHz-based CPU device.

Our future plans include also the extended implementations of the user and the verifier for smart devices with the support of various communication interfaces, and the complex integration with all PEAS entites inside the network infrastructure.

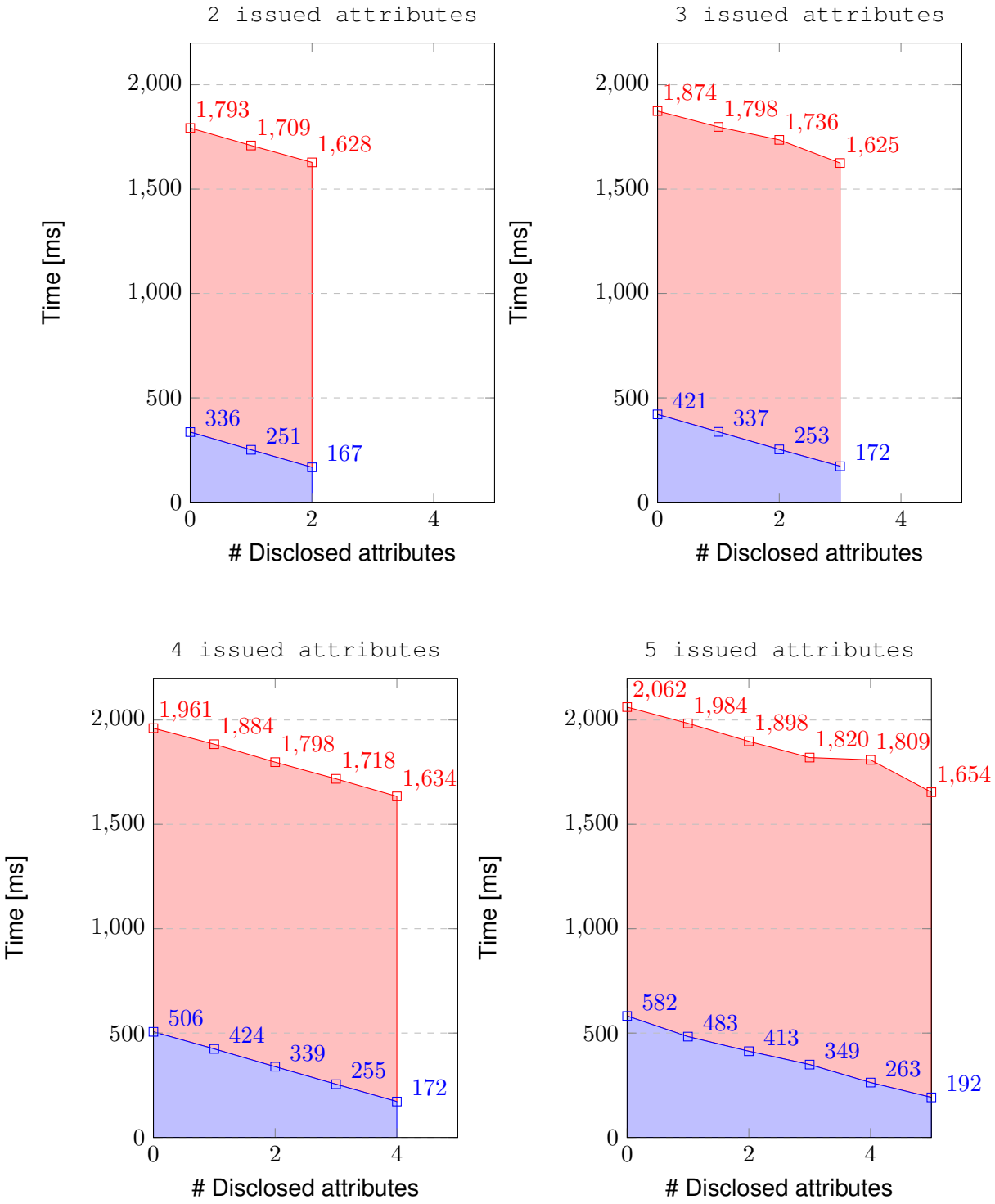

Figure 6.3: The `Show` and `Verify` runtimes of KVAC and RKVAC on Arduino DUE. Blue – KVAC scheme; red – RKVAC scheme.

Chapter 7 Intrusion detection

The intrusion detection tools proposed by Inria and ITTI analyze (in a centralized way) the information collected during an execution to identify if an attack has occurred or not during this execution. In the anomaly detection approaches, this analysis determines whether the observed behavior is normal or deviates from the normal behaviors specified by some previously defined models. The construction of these models is done during a so-called learning phase. During this phase, several executions of the supervised application (or of the supervised system) are taken into account. Thus the use of datasets is crucial when testing, validating and evaluating intrusion detection mechanisms. This input information comes either from parts of the network (captures of network traffic) or from computers (logs and traces describing activities performed by processes). At their heart the intrusion detection tools rely on data to find the indications of attacks. For the first demonstration, the cybersecurity detection tools rely on well-known benchmark datasets, with the inclination to switch to partner-generated data tied into the use-cases as soon as this data is available. Several datasets were identified as being of possible interest during this study. These four potential sources correspond either to application-level execution traces (a fault-tolerant distributed file system called XtreemFS) or to network traffic captures (CICIDS-2017 https://www.unb.ca/cic/datasets/ids-2017.html, IoT-23 https://www.stratosphereips.org/datasets-iot23, and possibly pcap files generated by Sparta partners using tools like tcpdump). As shown in Figure 7.1, a same selected dataset is used both during the learning and detection phases of the same experiment.

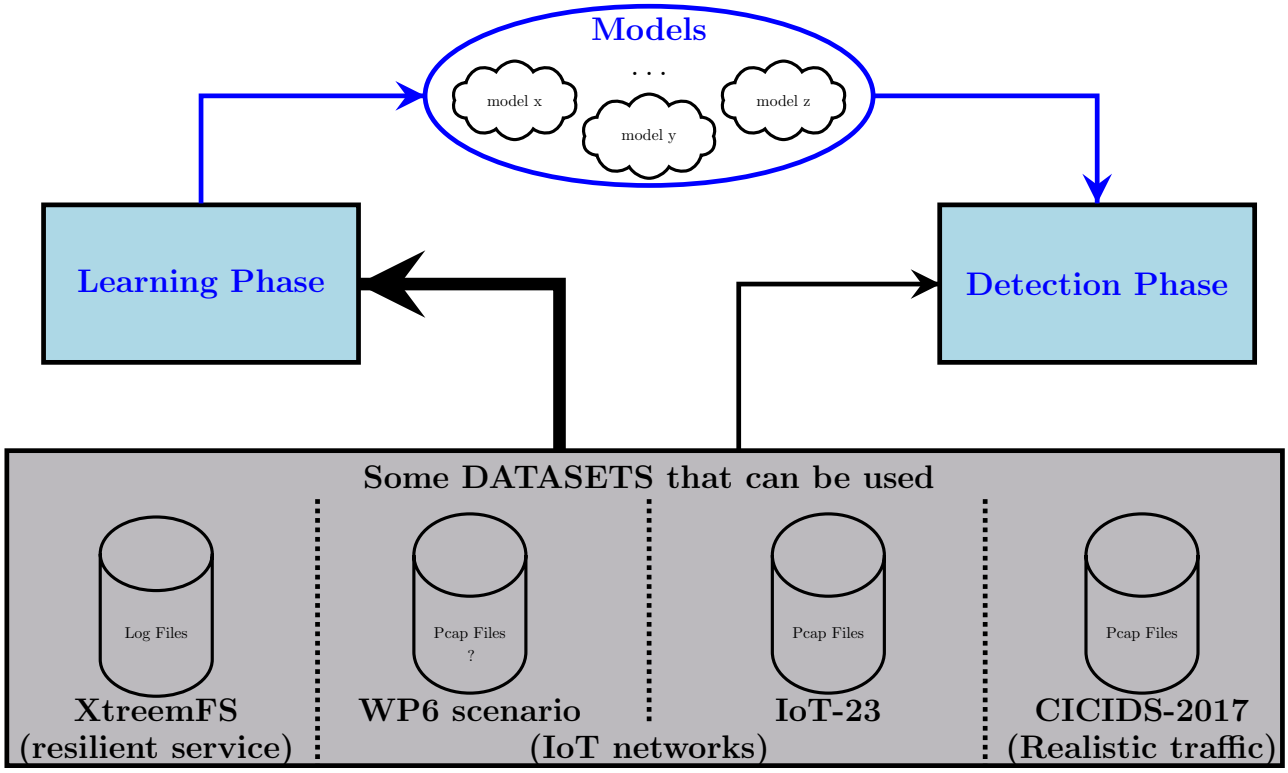

Figure 7.1: Intrusion Detection: Learning Phase and Detection Phase

To evaluate the proposed solutions, our first experiments have considered an open source application whose code can be modified to generate traces (XtreemFS) and a recent and publicly available dataset (CICIDS-2017) that is recognized as a reference when evaluating intrusion detection services. In both cases, the datasets contain information captured during normal executions and executions disrupted by several attacks.

In addition to the evaluation of our proposed solutions using the existing public datasets mentioned previously, a first link is also established with the use-case. Our goal is not to build a complete new dataset (including numerous traces of normal behaviors and real attacks): this is clearly out of the

scope of this study. However we propose to illustrate the use of our solutions with applications developed by other partners. Collecting network traffic (rather than logs of executed events) has the advantage of not requiring instrumentation of the software used. A trace can be collected using for example tcpdump. Among the applications integrated in the use-case, a first project partner provided us a trace corresponding to an execution of his application. More precisely, this application considers SUIT-compliant secure software updates for low-power IoT devices (See chapter 5). The trace (communications via IPv6 6LoWPAN protocols) corresponds to an execution where four update requests are triggered: only two succeed, the other two being rejected because the device fetches SUIT manifest with an invalid signature or because the SUIT manifest is pointing to firmware with lower sequence number. Once it has been broken down into four consecutive parts, this trace allows us to distinguish two types of behavior: two full updates and two failed attempts which can be considered as anomalies. Obviously, the tests carried out considering this very small dataset have just an illustrative character.

7.1 Different models for anomaly detection

The models proposed by Inria rely on the fact that events observed during an execution of a distributed application are partially ordered. Models based on recent advances in the field of machine learning methods for network anomaly detection are considered by ITTI.

7.1.1 Models relying on partially ordered sets of events

The anomaly detection solution investigated by Inria is designed rather to supervise distributed applications. For each process which participates in the distributed computation, the input trace describes the sequence of events which occurred locally on this process. An event corresponds to an occurence of an action: an internal activity like for example a system call, a sending of a message to another process, or the reception of a message. Obtaining such traces may require instrumentation of the application code (for monitoring and logging). Based on these traces, each distributed computation can be observed as a partially ordered set of events and be represented by a lattice of consistent cuts. This intermediate representation of a learned normal behavior is potentially very large in size. It is used to obtain smaller models that characterize the acceptable sequences of events. These models can take the form of an automaton or a list of temporal properties that have to be satisfied (likely invariants). Several types of model are constructed and used in parallel because these different models are often complementary during the detection phase: using several types of model is a key to reduce false negative. To reduce false positive during the detection phase, the training phase must consider multiple distinct correct executions: the resulting models are obtained by combining the intermediate models defined during the learning of each normal execution.

Although most of its functionalities are designed to analyze a partially ordered set of interactions between several components, the mechanism described above can also be used (in a degraded mode) to model and to analyse network traffics captured at a particular point of a network. In that case, the totally ordered set of observations is interpreted as if it was a sequence of internal events of a single process.

7.1.2 Machine learning approach enhanced with data balancer

In general, the step-by-step process of ML-based Intrusion Detection System (IDS) can be succinctly summarised as follows: a batch of annotated data is used to train a classifier. The algorithm 'fits' to the training data, creating a model. This is followed by testing the performance of the acquired model on the testing set - a batch of unforeseen data. In order to alleviate the data balancing problem present in the utilised IDS dataset (i.e., the number of training samples belonging to some classes is larger in contrast to other classes), an additional step is undertaken before the algorithm is trained (as seen in Fig. 7.2).

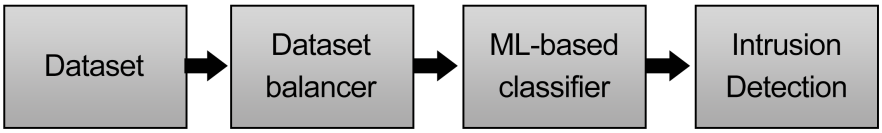

Figure 7.2: IDS training pipeline with dataset balancing

The ML-based classifier block of Fig. 7.2 can be realised by such models as **Artificial Neural Network** [40][41] and **Random Forest** [11].

The ANN in use is set up as follows: two hidden layers of 40 neurons, with the Rectified Linear Unit as activation function, and the ADAM optimizer, batch size of 100 and 35 epochs. The setup emerged experimentally.

7.2 Evaluation

7.2.1 Experiments and first results with event-based models

First experiments have been conducted using the XtreemFS distributed file storage application. A fault-tolerant distributed file system is a key service in a resilient system. The choice of XtreemFS, a general purpose storage system (http://www.xtreemfs.org), is motivated by the fact that this open-source project developed in the context of European projects proposes also an unsecured version of the file system. Thus we have the possibility to instrument the code to gather application level traces and, in addition, it is quite easy to attack this system by exploiting some existing vulnerabilities. Five attacks, which focus on five different commands (namely, NewFile, DeleteFile, OsdChange, Chmod, Chown) have been carried out to put the system in an inconsistent state or to grant additional access rights to an attacker. For each attack, four different contexts are considered (c1, c2, c3, c4) depending on whether the attacker is internal or external to the monitored system and depending on how the steps of the attack and the normal actions interleave.

Attacks	*NewFile*				*DeleteFile*				*OsdChange*				*Chmod*				*Chown*			
Contexts	c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4	c1	c2	c3	c4
`Automaton`	-	-	-	5/5	-	1/5	-	5/5	4/5	4/5	4/5	5/5	-	-	-	5/5	-	-	-	-
`Inv-Order`	5/5	5/5	-	-	5/5	5/5	-	-	5/5	5/5	-	-	5/5	5/5	-	-	-	-	-	-
`Inv-Occur`	-	5/5	5/5	-	-	5/5	5/5	-	-	5/5	5/5	-	-	5/5	5/5	-	-	5/5	5/5	1/5

Figure 7.3: Complementarities between automaton-based and invariant-based models (XtreemFS Dataset)

In Figure 7.3, we describe some results related to the detection of attacks during an execution of XtreemFS. We consider the five attacks mentioned above and for each of them, the attack is conducted during four distinct executions corresponding to the four different contexts. Our main goal is to assess the interest of using several different models simultaneously. Therefore in this experimentation, three different classes of models are considered. One is based on automaton. The two others are based on likely invariants defined for couples of types of events: invariants about their order (Inv-Order) and invariants about their number of occurrences (Inv-Occur). For each class of model, five different models are constructed using different subsets of learned normal behaviors. In each scenario (for a given attack and a given context) and for each class of model, the detection phase is executed five times (with each constructed model). In each cell of Figure 7.3, the notation

$x/5$ indicates that the analysis leads to raise an alert in x out of 5 cases while the symbol $-$ indicates that no model of this class was able to detect the attack in the corresponding context. When models are considered separately, the results show that false negatives occur in more than half of the twenty cases. Yet, specificities and complementarities of the models used during the detection phase appears clearly (see the colored notations $5/5$ that indicate the lack of false negative with a given class of model). This confirms the interests of using several models in parallel: using three models, a single attack (Chown) is almost undetectable in only two contexts (c1 et c4).

In the context of Sparta, we also study how to reduce the time required to compute such models and their size without having a noticeable impact on the false negative rate.

The studied models are suitable for the analysis of a partially ordered set of events. However, they can also be used to analyze a totally ordered set of events. Thus, we use an automaton-based model and an invariant-based model to analyze the network traffic captured by a partner while running a protocol that secures the firmware updates of a Riot device. The built models accept the two behaviors corresponding to successful updates and generate alerts in the case of the two rejected requests. This experiment, while encouraging, is mainly intended to illustrate the approach and cannot be considered as a complementary evaluation.

7.2.2 Experiments and first results with the machine learning approach

CICIDS2017 [38] is an effort to create a dependable and recent cybersec dataset. The Intrusion Detection datasets are notoriously hard to come by, and the ones available display at least one of frustrating concerns, like the lack of traffic diversity, attack variety, insufficient features etc. The authors of CICIDS2017 offer a dataset with realistic benign traffic, created as an interpolation of the behaviour of 25 users using multiple protocols. The dataset is a labelled capture of 5 days of work, with 4 days putting the framework under siege by a plethora of attacks, including malware, DoS attacks, web attacks and others. This work relies on the captures from Tuesday, Wednesday, Thursday and Friday. CICIDS2017 constitutes one of the newest datasets available to researchers, featuring over 80 network flow characteristics. The Imbalance Ratio of the Majority Class to the sum of all the numbers of samples of the rest of the classes was calculated to be 2.902.

CICIDS 2017 dataset consists of 13 classes - 12 attacks and 1 benign class.

There is a wide discrepancy among the classes in terms of the number of instances, especially the benign class as compared to the attack classes.

Random Subsampling, along with a number of other subsampling methods were used to observe the influence dataset balancing has on the performance of the reference ML algorithms - an Artificial Neural Network (ANN) and the RandomForest algorithm.

The tests proved that changing the balance ratio undersampling the majority classes improves the recall of the minority classes, but degrades the precission of the classifier on those classes. This basically means that dataset balancing causes the ML algorithms to misclassify the (previously) majority classes as the instances of the minority classes, thus boosting the false positives.

The results of the best of the tested approaches are to be seen in Table 7.1. In addition to the precision, recal and f1-score metrics the table features the 'support' column, which indicates how many instances of the class in question were found in the testing set.

Table 7.1: CICIDS2017 / Random Subsampling down to 7141 instances per class / RandomForest

	precision	recall	f1-score	support
0	1.00	0.98	0.99	162154
1	0.13	0.99	0.23	196
2	1.00	1.00	1.00	12803
3	0.92	1.00	0.96	1029
4	0.98	1.00	0.99	23012
5	0.85	0.99	0.92	550
6	0.93	0.99	0.96	580
7	0.93	1.00	0.96	794
8	0.17	1.00	0.29	1
9	1.00	1.00	1.00	15880
10	0.73	1.00	0.85	590
11	0.63	0.98	0.77	301
12	0.07	1.00	0.14	4
13	0.32	0.48	0.39	130
accuracy			0.9872	218024
macro avg	0.69	0.96	0.74	218024
weighted avg	0.99	0.99	0.99	218024

Chapter 8 IoT protocols formal modeling

This chapter presents the demonstration of IoT Protocols formal modeling. This demonstrator consists of two demo scenarios:

- Demo 1: Formal verification of IoT protocols and security relevant parts of protocols (Figure 8.1)
- Demo 2: IoT network risk analysis based on probabilistic model checking (Figure 8.2)

The general goal of this research is to review both the methods of formal verification and their practical application within this domain, see also [23, 24].

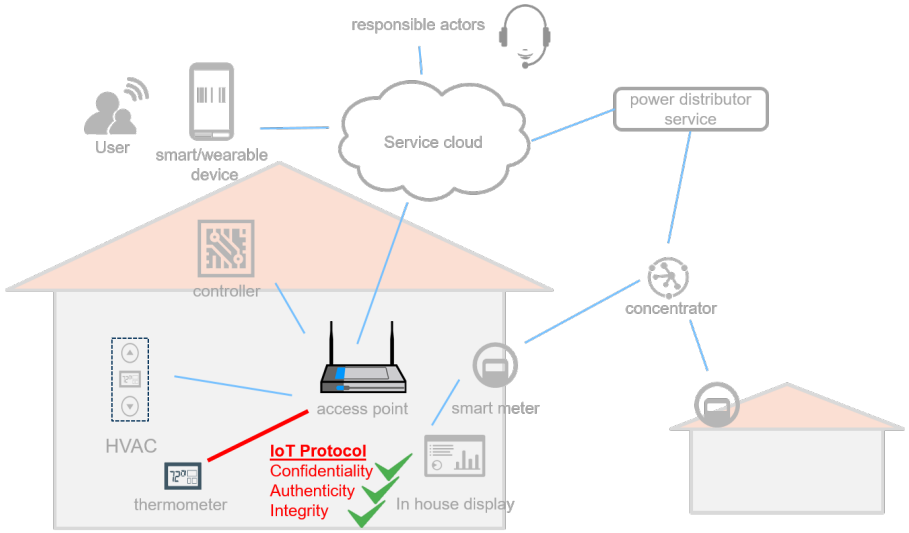

Figure 8.1: Formal verification of IoT protocols

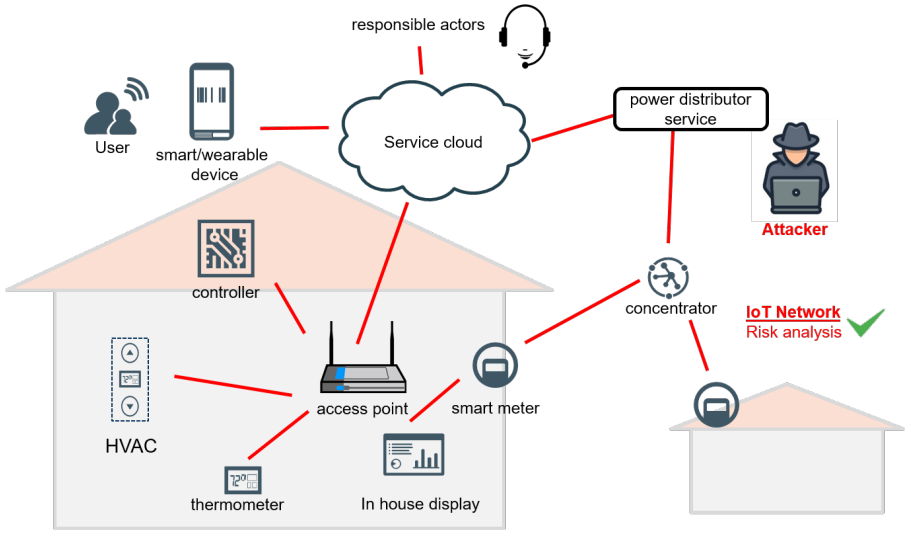

Figure 8.2: Formal verification of IoT protocols

The smart home scenario, as part of WP6 common use case, is selected in order to select the most relevant protocols and to ensure their significance. The scenario describes the network infrastructure of a smart home environment.

The house is equipped with different smart appliances and sensors, including:

- HVAC – smart humidity ventilation air-condition system
- Controller – presenting HEMS – smart home energy management system

- In house display
- Thermometer
- Power distributor service
- Smart/wearable devices
- Access point
- Smart meter

In order to enable a communication between these devices different IoT protocols can be used, including *EnOcean*, *Z-Wave*, *ZigBee*...

Formal verification of protocols is the act of proving or disproving the correctness of intended protocol with respect to a certain formal specification or property, using formal methods.

Possible checks include verification or falsification of security properties, functional correctness, qualitative and quantitative analysis of protocol's specifications or implementations.

Some common model checkers as *AVISPA*, *ProVerif*, *Scyther* and *Tamarin* have their focus on security protocols.

There are also model checkers for statistical and/or probabilistic checking, as *UPPAAL* and *PRISM* that can be used for more general cases of IoT network modeling and risk analysis.

8.1 Demo 1: formal verification of IoT protocols

As initial model for SPARTA, we consider the protocol EnOcean. This protocol[1], mainly used in Smart Home domain, was developed in 2001. EnOcean provides authentication, integrity checking, encryption and replay protection.

Demo 1 presents formal verification of EnOcean protocol, used in smart home environment to connect thermometers. Figure 8.3 presents components of the SPARTA WP6 common use case involved in this demo.

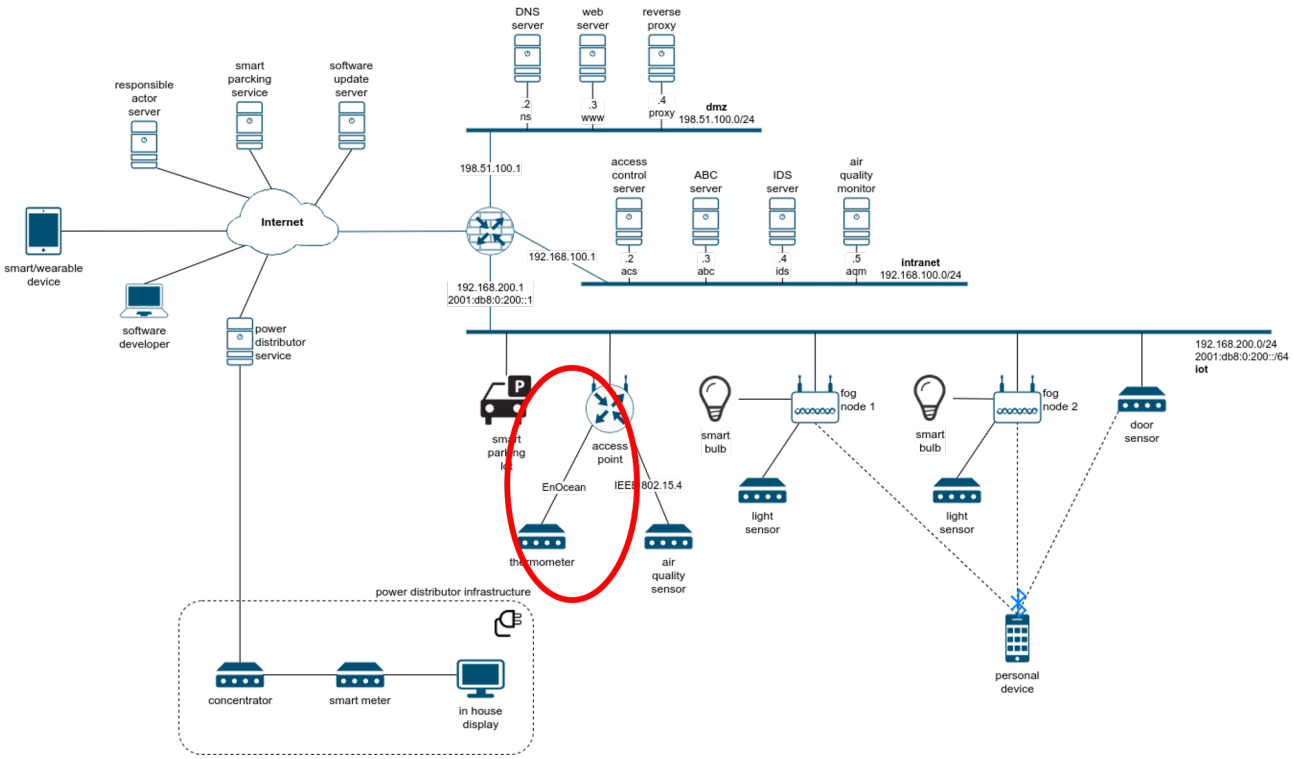

Figure 8.3: Formal verification of IoT protocols demo infrastructure

[1] https://ieeexplore.ieee.org/document/8260940

For this use scenario, the focus is on formal verification of EnOcean protocol using the ProVerif tool, and specifically the unilateral teach-in procedure and the unilateral authentication within protocol specification.

In order to apply a selected model checker on a given protocol, first a model as input has to be created according to the specification. Since it is in general not possible to verify the whole protocol specification, it is important to model the most important parts of it according to the problem statement (e.g. functional check, check of authentication property or check if given attack is possible). Figure 8.4 presents a general protocol verification workflow.

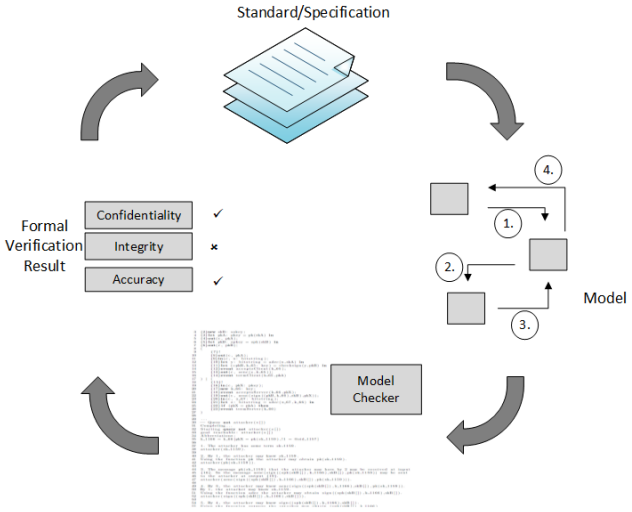

Figure 8.4: Formal verification of IoT protocols workflow

The output of the formal verification process is either usually a proof of correctness or a discovery of potential vulnerabilities. As for all tools which can handle an unbounded number of sessions, the result can also be neither a verification nor a falsification of the statement. The reason is that the verification of protocols for an unbounded number of sessions is undecidable in general. In such cases ProVerif gives an attack derivation, which potentially can give hints how to reconstruct an attack when manually inspecting it.

This demo will be implemented in two phases – *phase 1* that includes EnOcean unidirectional teach-in formal modeling, and *phase 2* that includes unidirectional teach-in and authentication.

Phase 1, as part of SPARTA WP6 first release of demonstration, includes ProVerif formal model of unidirectional teach-in procedure in EnOcean protocols (Figure 8.5). In a first step a suitable model is extracted based on EnOcean's specification, where the focus is the unidirectional teach-in procedure with a preshared key, which is written on the device.

Based on the model in Figure 8.5 the protocol is encoded in ProVerif's input language, which is a variant of the applied pi calculus. The focus on the parameter to be included in the model are encrypted parameters and parameters, which are necessary for the device identification. It has to be noted that several parameters as the SLF (security level format), the cmac size, the rlc (rolling code) window size and the IDs of the devices are sent in plain text and therefore visible to any attacker. Moreover, security properties are formulated. Due to the model relevant security properties in phase 1 are especially the secrecy of Key A and the RLC.

The capabilities of the attacker are modeled under the Dolev-Yao model. That means the attacker has the complete control of the communication channel, is able to manipulate any data, but for reading encrypted messages the attacker needs to be in the possession of the corresponding key. The two formulated security properties within the model – the secrecy of Key A and the RLC (Rolling Code, increase every time a message is sent to the transmitter) – can be proved.

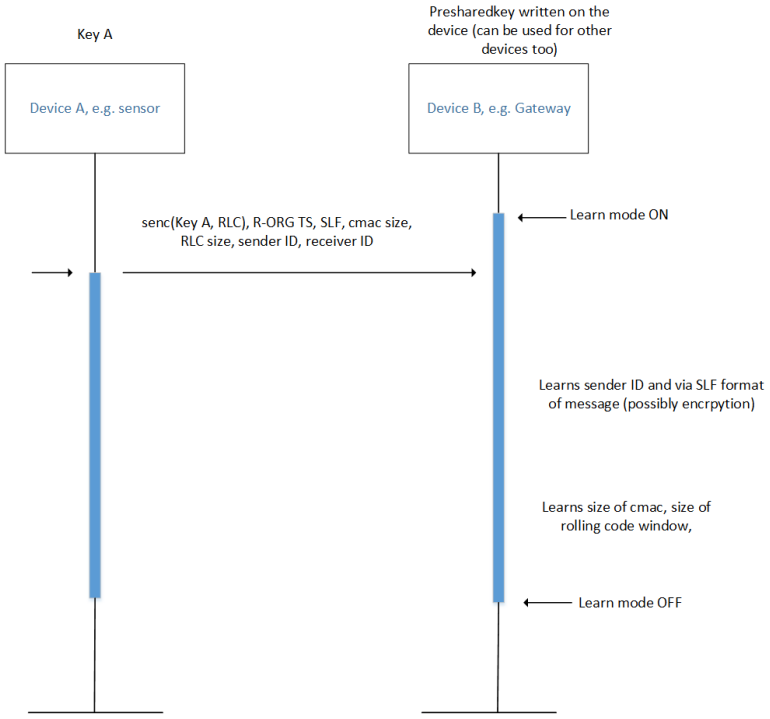

Figure 8.5: Formal model of EnOcean protocol unidirectional teach-in procedure

8.2 Demo 2: IoT network risk analysis based on model checking

Demo 2 presents a smart home IoT network risk analysis based on probabilistic model checking. This use-scenario focuses on a probabilistic risk analysis of two different smart home system configurations through threat modeling and model checking.

The first step is modeling a system in a threat modeling tool, in order to identify potential threats and vulnerabilities. The second step includes the attack scenario definition by exploiting selected vulnerabilities. The last step is probabilistic modeling of attack scenarios within a defined smart home system architecture, and a risk analysis.

This demo will be implemented in two phases – *phase 1* includes modeling of hijacking of smart HVAC system attack within smart home environment, and *phase 2* should include modeling of hijacking of smart meter attack:

- *Hijacking of smart HVAC* system attack is modeled in order to estimate the probability of a successful man-in-the-middle attack, with two different attacker goals – hijacked heating and high power consumption, both as part of ransomware attacks.
- *Hijacking of smart meter* is modeled in order to estimate the probability of a successful man-in-the-middle attack within this scenario, where attackers main goals are to lower his power consumption, and commit fraud.

Phase 1, as part of SPARTA WP6 first release of demonstration, involves components of the SPARTA WP6 common use case involved in this demo presented on Figure 8.6.

Inputs for this model are system components and protocols used for their communication (Figure 8.7).

This architecture is then modeled in a threat modeling tool, in order to detect potential system vulnerabilities and threats. The next step is to select subset of vulnerabilities relevant to the desired attack scenario, from the long list of threats discovered using the threat modeling tool, for example:

- *thermometer–access point link* – Link jammed: Denied Transmission
- *access point* – Tailored Context/Incorrect Actuation (open point for Man-in-the-middle attack)
- *access point* – Tailored Context/Incorrect Reading (open point for Man-in-the-middle attack)

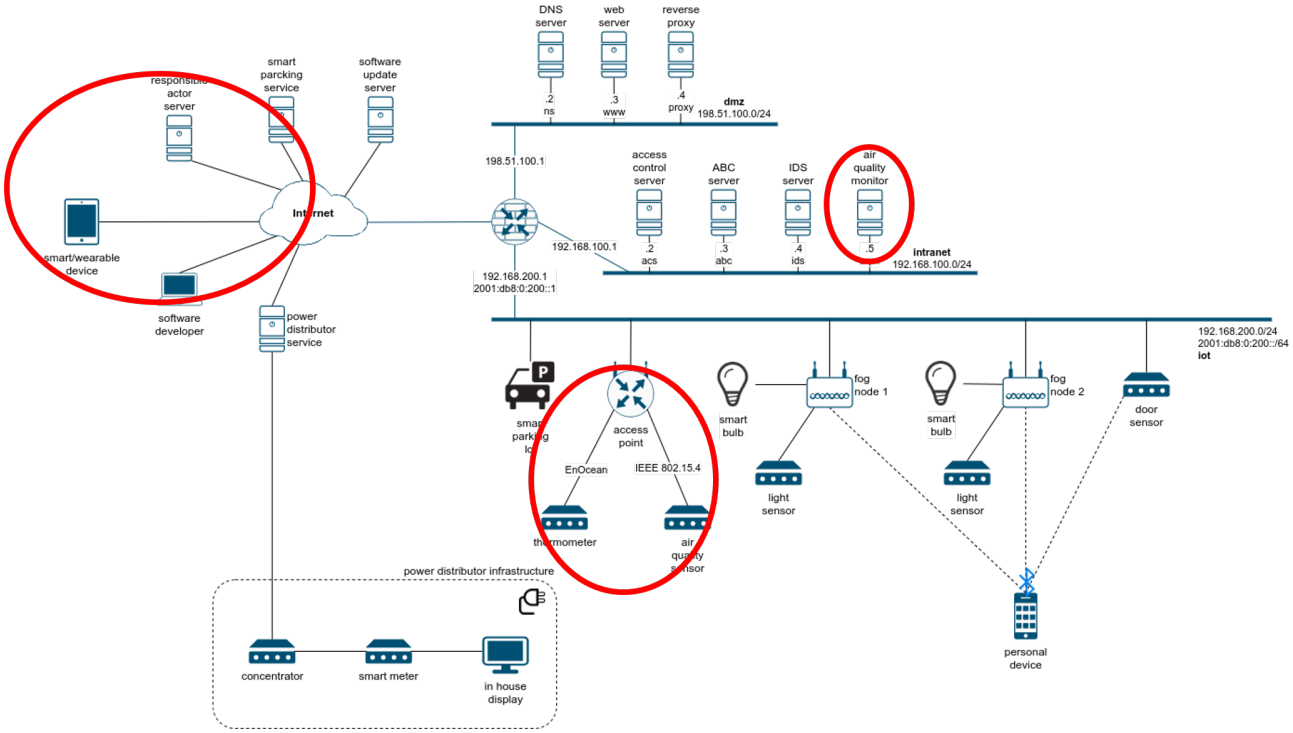

Figure 8.6: IoT network risk analysis based on model checking demo phase 1 infrastructure

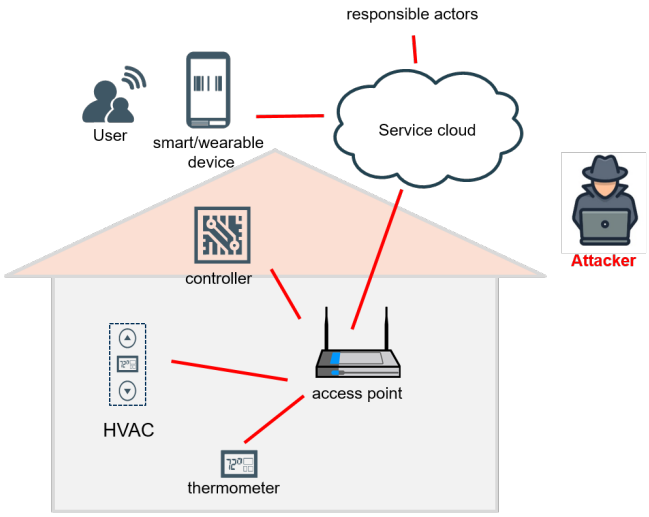

Figure 8.7: Hijacking of smart HVAC risk analysis architecture

- *controller* – Tailored Context/Incorrect Actuation (open point for Ransomware attack)

In addition to vulnerabilities, attack parameters and pre-conditions should be defined, for example:

- *Hijacked heating* – The attacker takes control over heating, high temperature is detected, heating is turned on, resulting in damage; this attack relates to two types of attacks: Ransomware attack and Man-in-the-middle attack;
- *High power consumption* – The attacker takes control over HVAC, both cooling and heating is turned on, resulting in damage and high power consumption; this attacks relates to two types of attacks: Ransomware attack and Man-in-the-middle attack.

All these preconditions and input data are then modeled in PRISM model checker in form of a sequential flow of events. An additional parameter that should be defined is the cost – the maximum number of vulnerabilities that can be exploited during an attack, as a limiting factor for attacker capabilities.

The output of this process is probability of successful attack within given assumptions. All tests are

conducted for different cost values – 1-4. Resulting console output example and risk probabilities are presented on figures below.

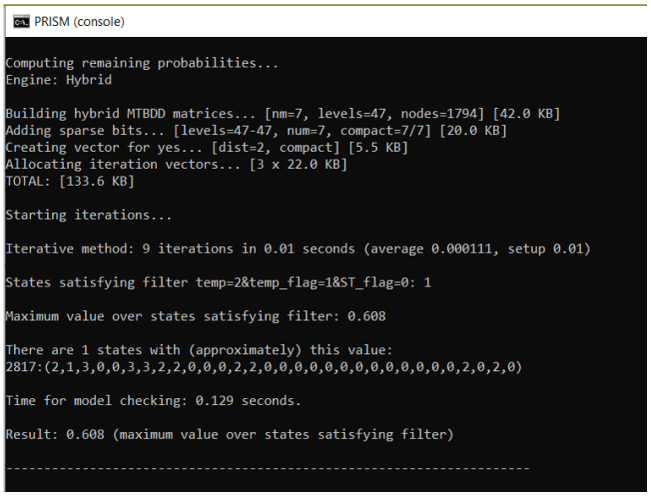

Figure 8.8**:** Hijacking of smart HVAC risk analysis - console output example

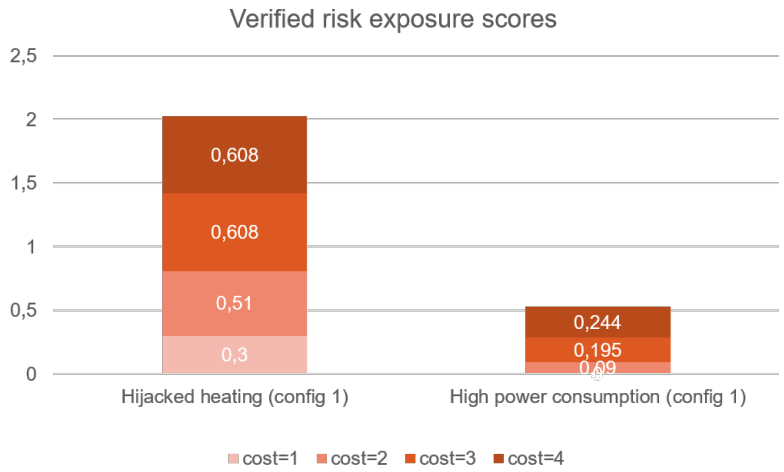

Figure 8.9**:** Hijacking of smart HVAC risk analysis - resulting risk probabilities; cost: maximum number of vulnerabilities that were exploited during an attack.

8.3 Evaluation

The main goal of this research is to review both the methods of formal verification and their practical application within this domain in two specific demo cases described above. Both demos include two phases, where phase 1 relates to implementation and evaluation in D6.3, while phase 2 will be implemented and evaluated in the next year and reported in D6.4. Evaluation of phase 2 also includes publishing of the reported work in relevant journals/conferences.

Demo 1: Formal verification of IoT protocols

Phase 1 of the first demo includes includes ProVerif formal model of the unidirectional teach-in procedure in EnOcean protocol. All relevant parts, including security properties, are formulated and modeled, and it has to be noted that several parameters as the SLF (security level format), the cmac size, the rlc (rolling code) window size and the IDs of the devices are sent in plain text and therefore are visible to any attacker. The result of this modeling shows that the secrecy of Key A and the RLC can be proved. Phase 2 includes formal modeling of the high security authentication process. Preliminary results for phase 2 indicate the existence of an attack trace in EnOcean protocol authentication procedure.

Demo 2: IoT network risk analysis based on model checking

Phase 1 of the second demo includes a risk analysis of hijacking of smart HVAC system attack within smart home environment based on PRISM modeling. Vulnerabilities, attack parameters and pre-conditions are defined and modeled in the PRISM model checker in form of a sequential flow of events. The variable in this process is the cost – the maximum number of vulnerabilities that can be exploited during an attack, as a limiting factor for attacker capabilities, which takes values 1 – 4. Resulting risk probabilities are presented on 8.8. Evaluation of phase 2 will include modeling of additional scenario that includes smart meters in smart home environment.

Chapter 9 Fog security orchestration

One of the goals of SPARTA project was to develop effective orchestrator working in the Fog layer of the Fog computing architecture by providing effective means for solving QoS and security related problems of orchestration of heterogeneous Fog and Edge layer devices and services. The main idea of this demonstrator is to check the placement of Fog and Edge devices and services for possible QoS and security related issues and find the non-optimal distribution of services between Fog nodes.

Fog computing based solutions frequently have insufficient security due to the fact that they use intensive communications with Constrained devices located at the Edge layer. If one of many Edge devices doesn't support communications protocol of the sufficient strength, security of the whole solution may be compromised. Moreover, in some Fog solutions roaming services are supported, when the service follows human, vehicle, etc., and travels from one Fog node to another. In such cases, when the lesser security capable service "arrives" to the secure Fog node, the security of this Fog node may be compromised. Similar problems may also occur with other QoS parameters such as latency, bandwith, range, etc. Service Orchestrators placed in Fog nodes may be used to monitor the whole situation in the Fog node (including communications with neighbouring Fog nodes) and take required measures in cases of potential violations of security and other QoS parameters.

9.1 Fog security orchestration scenario

The selected demonstration scenario is a smart home scenario, Figure 9.1 shows the place of Fog orchestration related infrastructure elements inside the WP6 common use case.

Two Fog nodes are placed in two different locations (e. g. different rooms). These Fog nodes are capable of running services required to communicate with Edge devices connected to the corresponding Fog nodes (Figure 9.2).

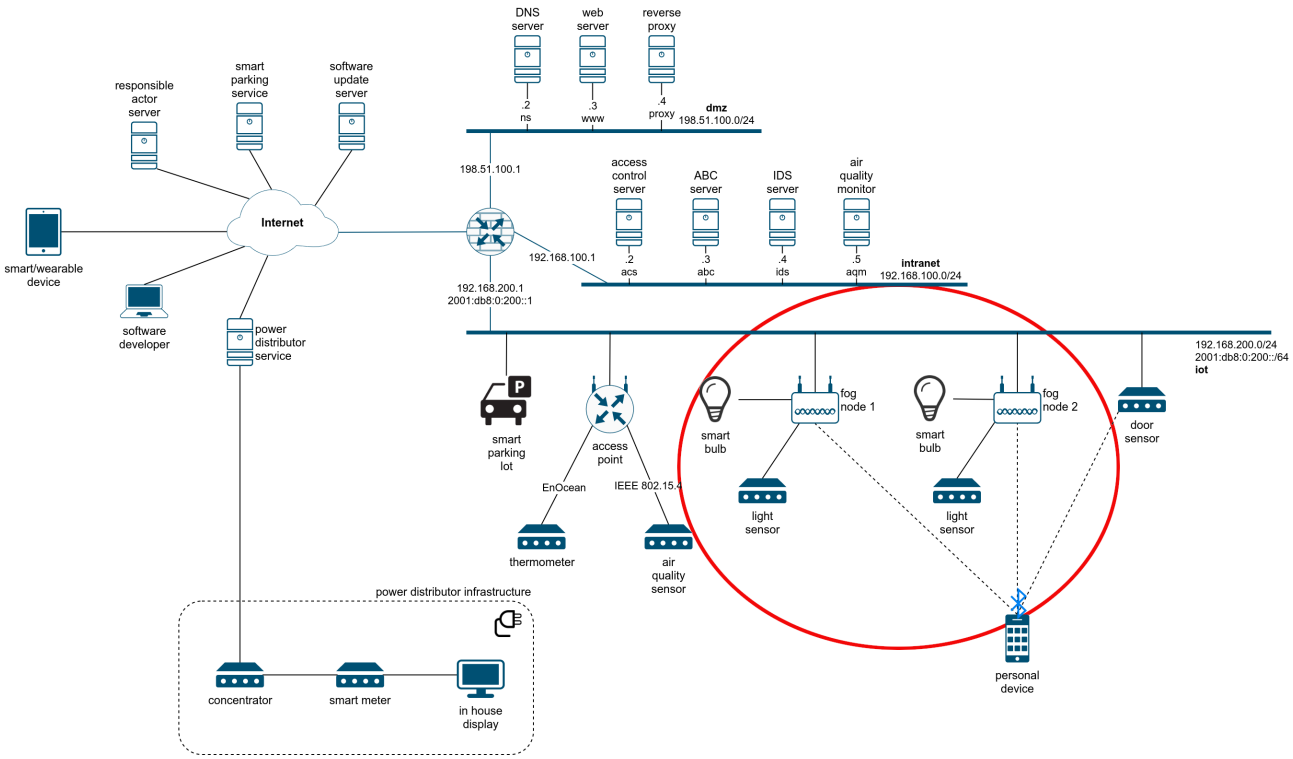

Figure 9.1: Fog orchestration components in the common WP6 use case infrastructure

Fog nodes host services which monitor the lighting characteristics in the rooms using light sensors and are capable to adjust the lighting according to the preferences of the human by activating smart bulbs. Location and Presence services are running on the Fog nodes to detect the presence of

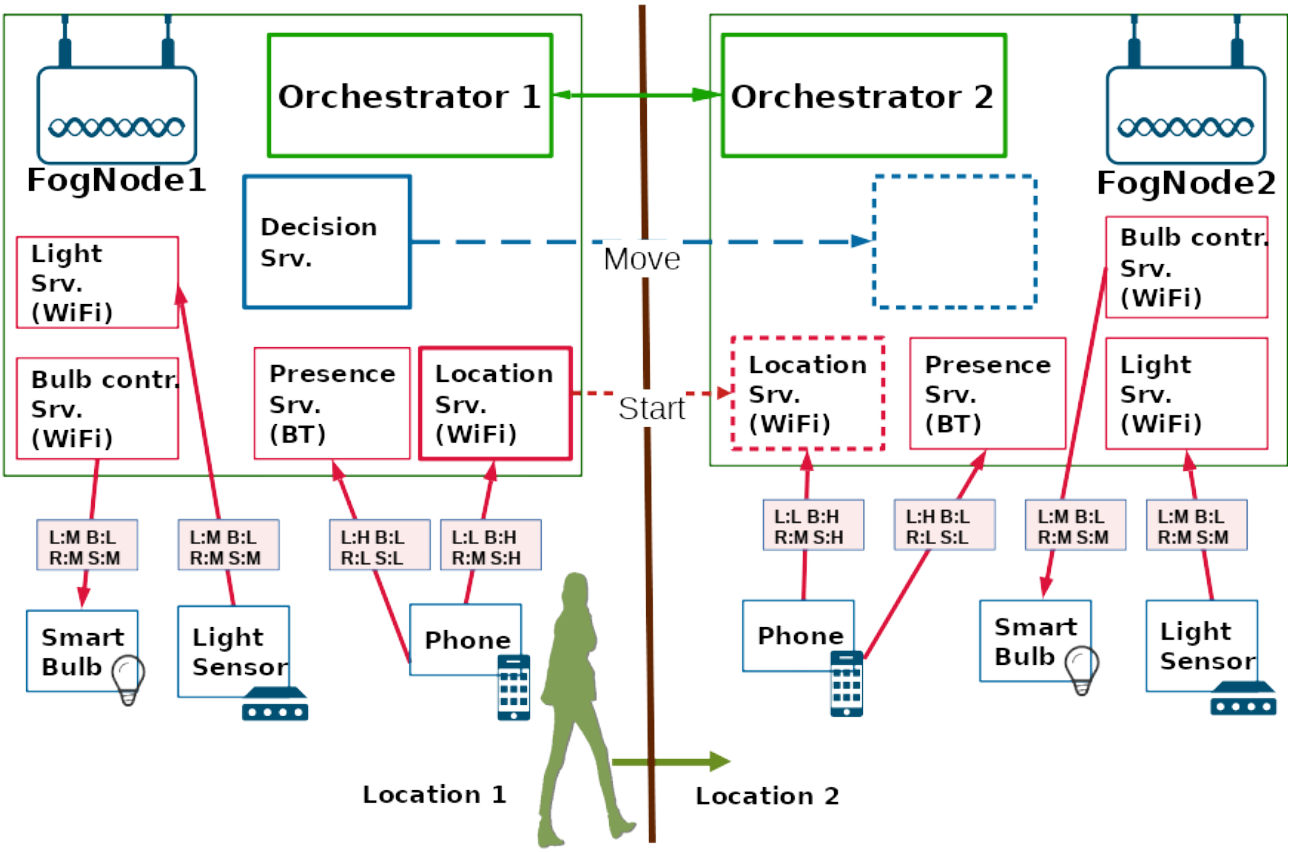

Figure 9.2: Fog orchestration scenario

the human in the room (Presence service) and to monitor the exact position in the room (Location service). Decision service “knows” the lighting preferences of the particular human and is capable to make decisions on how environment actuators (e. g. smart bulbs) should be controlled according to the position of the human inside the room. For example, if the human is siting on the couch near the TV, then the lighting should be dimmed, etc.

To ease the optimization problem in first implementations of Orchestrator we decided to use simplified set of possible QoS and security settings of devices and communication protocols. We will use four categories of parameters (”Latency”, ”Bandwidth”, ”Range” and ”Security”), each with possible values of ”Low”, ”Medium” and ”High”. Each Fog and Edge device used in this scenario will be specified in terms of these parameters. In Figure 9.2 all requirements for communication parameters are indicated by red boxes on links (e. g. ”L:H” means that required latency for this communication is High, etc.)

In this demonstration scenario Location service requires increased amount of resources (CPU time, RAM, energy, etc.) to run, and it is preferable to turn it off when the human is not in the room. The Orchestrators running on both Fog nodes monitor the surrounding services, communicate with each other and are capable of stopping, starting and moving other services from one Fog node to another. Decision service, which is responsible for adjusting the lighting according to the personal preferences of the particular human is a personalized service. This means that if the human leaves the room the service should be suspended, thus freeing some resources on the Fog node, but keeping the collected human related information intact. If the human lately reenters the same room, the corresponding Decision service is resumed and is instantly ready to assist the human with personalized decisions. On the other hand, if the human enters the second room, then the Orchestrators must communicate to each other and make decision to move the corresponding personalized Decision service from the first Fog node to the second Fog node and to resume it there. In the proposed scenario Orchestrators should also turn off the Location service as soon as human leaves the room and should start one as soon as the Presence service detects the presence of the human in the

room.

The goal of this demonstrator is to show how Orchestrators make decisions on controlling the services according to the QoS and security requirements. Each decision of the Orchestrators on starting/stopping/suspending/moving services must be checked for the satisfaction of the minimal requirements imposed by various hardware and software restrictions of the involved physical devices as well as requirements arising from the specifics of the area of application (e. g. health related data should be protected better than environment monitoring data).

The Orchestrators should work in three main steps:

- Each Orchestrator should apply requirements on latency, bandwidth, security, and range imposed by the application area and hardware/software capabilities of each Fog node and decide if it is possible to start all required services without violating these requirements.
- The second step is to find the optimal distribution of the available services between different Fog nodes. This is useful for saving energy and computation resources in cases when some services may be stopped, suspended or moved to another Fog nodes.
- The third step is dynamic service allocation, which happens when the situation changes during runtime, and Orchestrators must change the distribution of the services between available Fog nodes according to the new conditions.

If the infrastructure presented in Figure 9.2 is used, then the demonstrator of the first step should collect all the information of the service and device placement (including coordinates of the devices) in both Fog nodes independently, check the minimal QoS and security requirements and report if the initial service and physical device placement satisfies the requirements. In this case the situation when both Fog nodes are running the Location service is acceptable, because all minimal requirements are satisfied.

The problems in this step may occur if for example Edge node ”Smart Bulb” is implemented using Constrained device which only supports RC4 encryption (which means security level ”Low”), but application area (according to the Figure 9.2) requires ”Medium” security level for communication between Fog node and ”Smart Bulb”. In such case, the Orchestrator should detect the security requirements violation problem.

During the second step, Orchestrators should search for the optimal placement of the services and decide to stop one of the Location services as this variant of service placement saves some energy and computing resources in the first Fog node (assuming the human is in the first room). To demonstrate the third step the Presence services should detect the movement of the human from one room to another, report this situation to the Orchestrators, which on their own turn should reallocate the personalized Decision service from the first Fog node to the second Fog node.

9.2 Evaluation

The main goal of this demonstrator is to evaluate the influence of Orchestrators running in Fog nodes to the overall performance and security of the Fog architecture based Intelligent Infrastructure. Currently the implementation of prototype described in this demonstration scenario is under way. First isolated experiments on evaluating the security and QoS impact of different types of protocols used for communications between Fog node and Edge node devices have been conducted [44]. These results will be useful for performing future experiments on the influence of Orchestrator’s decisions on the various parameters of Inteligent Infrastructure.

For the evaluation we plan to finalize the prototype and assess the performance of all three steps of Orchestrator’s behavior. Further we plan to investigate how Orchestrators’ decisions influence security of the whole system, how the requirements of memory and processing power are affected in Fog nodes and Egde nodes, evaluate changes on the energy requirements, etc. The full implementation of this scenario and evaluation of all parameters will be finished in the next year and reported in D6.4

Chapter 10 Fog hardening

Fog nodes play an important role in the IoT network of the Intelligent Infrstructure scenario (Figure 3.2), as they serve edge computation for end-users in their proximity, sharing the load on the resources provided by cloud servers. However, this network topology optimization exposes the security of the user- and kernel-space software running on fog servers, as they interact with the end-user device directly, which may be malicious. Such an exposure poses security risks which threaten the confidentiality of the data processed by user-space fog applications, the integrity of the kernel-space fog operating system, and, inherently, of the whole fog layer of the ZEB (Figure 2.2).

Attackers leverage memory corruption vulnerabilities to establish primitives for reading from or writing to the address space of a vulnerable application. These primitives form the foundation for code-reuse and data-oriented attacks. For strenghtening the security-by-design goal of the Intelligent Infrastructure, we equip user-space applications and kernel-space software running in the fog layer with mechanisms that mitigate data-oriented attacks. The security enhancement ensures the confidentiality of sensitive data, such as personal user information or user authentication material, that user-space application running on fog nodes process. Moreover, the security extension hardens the underlying operating system kernel against data-oriented attacks, preventing an attacker from taking over the fog node, and, potentially, the whole fog layer.

10.1 Hypervisor-assisted selective memory protection

We arm the operating system running on the fog node with the xMP framework [34]. xMP leverages virtualization extensions offered by the x86 architecture to provide memory isolation primitives which allow us to protect sensitive data in both the user- and kernel-space fog software. In addition, xMP displays a software-based Pointer Authentication Code (PAC) mechanism, similar to the equivalent PAC feature shiped with ARM processors that operates in hardware. xMP's PAC functionality allows us to ensure the integrity of memory pointers, by tagging them with Keyed-Hash Message Authentication Codes (HMACs), immutably bound to the container that stores them, which get authenticated before each memory dereference.

xMP is based on top of the x86 architecture and the Xen hypervisor [3], as it relies on virtualization extensions that are exclusive to Intel chips and are already implemented by Xen. xMP employs Xen's `altp2m` subsystem [28] as a building block for providing disjoint isolation domains which we use to protect sensitive data. Xen's `altp2m` was introduced to add support for the Intel virtualization extension that allows VMs to switch among Extended Page Tables (`EPTs`) [25], Intel's implementation of SLAT tables. Using Xen's `altp2m`, xMP instructs the underlying VMM to isolate and relax permissions to selected memory regions, on-demand.

10.2 xMP design

An xMP domain may exist in one of two states, the permissions of which are configured as desired. In the protected state, the most restrictive permissions are enforced to prevent data leakage or modification. In the relaxed state, the permissions are temporarily loosened to enable legitimate access to the protected data as needed. Figure 10.1 shows the overarching architecture of xMP. To enforce the access restrictions of all xMP domains in each `altp2m` view, xMP propagates the permissions of each domain across all available `altp2m` views. Setting up an xMP domain requires at least two `altp2m` views. The framework dedicates one view, the *restricted view*, to unify the memory access restrictions of all other isolation domains. xMP configures this view as the default on every vCPU, as it collectively enforces the restrictions of all xMP domains. xMP uses the second view to *relax* the restrictions of (i.e., *unprotect*) a given xMP domain and to allow legitimate access to its data. For example, refer to this view as *domain[id]*, with *id* referring to the xMP domain of this view. By entering *domain[id]*, the system switches to the `altp2m` view *id* to bring the xMP domain into its relaxed state—crucially, all other xMP domains remain in their protected state. By switching to the *restricted*

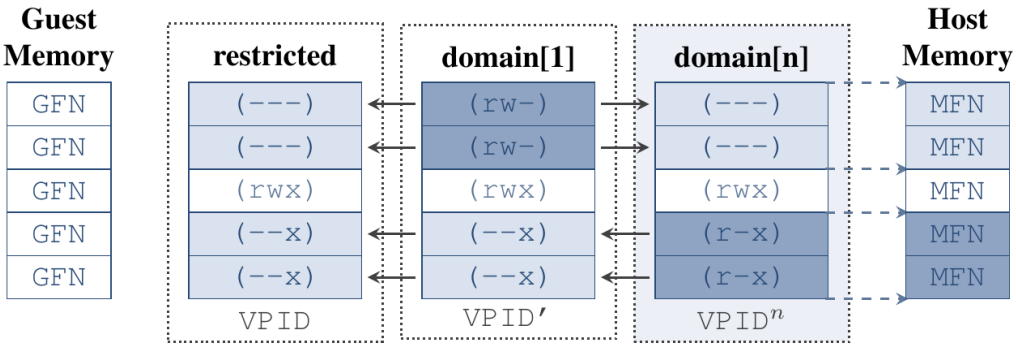

Figure 10.1: The system configures $n+1$ `altp2m` views to create n disjoint xMP domains. Each $\{domain[i] \mid i \in \{1, ..., n\}\}$ relaxes the permissions of a given memory region (dark shade) and restricts access to memory regions belonging to other xMP domains (light shade).

view, the system switches *all* domains to their protected state. The Intel architecture defines the unprivileged `VMFUNC` instruction [25] to enable VMs to switch among 512 `EPTs` without involving the VMM. xMP executes the `VMFUNC` instruction for switching between xMP isolation domains to relax and restrict access permissions to sensitive data.

In addition, xMP provides software support for PAC to ensure the integrity of pointers to sensitive data within isolated domains. With this enhancement, by exploiting memory corruption vulnerabilities, adversaries are not able to redirect pointers to injected, attacker-controlled objects outside the protected domain, or existing, high-privileged objects inside the xMP domain. To facilitate PAC, xMP leverages lightweight HMAC frameworks provided by the underlying OS, such as the SipHash [7] in the Linux kernel. We apply HMACs inserted by xMP in selected pointers to authenticate them. xMP reuses the available sign-extension bits of virtual addresses to store the HMAC truncated to 15 bits. For ensuring that pointers cannot be illegally redirected to existing objects, we bind them to a specific context that is unique and immutable.

10.3 Hardening user-space applications in the fog layer

xMP grants user processes running in user-space the ability to protect selected memory regions by extending the Linux kernel with four new system calls. We use this trait to harden the user-space *Decision* service that the fog orchestrator (Section 9) deploys on fog nodes, which adjusts the light level based on user preferences recieved from their end-device. Specifically, we isolate in a dedicated xMP domain the end-user data that the service processes and stores in its address space. This way, we prevent the user information from being leaked in case an attacker exploits a memory corruption vulnerability that may emerge in the *Decision* service.

We instrument the *Decision* service to invoke the underlying OS for allocating a new xMP isolation domain via a system call capable of setting up `EPT` tables. Transparently, xMP configures the restricted isolation domain as well, and loads its corresponding `EPT` as the default memory view on the executing `CPU`. Next, before the application loads the user's light level preferences, we isolate it in the non-restricted xMP domain. We perform this step by executing the xMP system call responsible with adjusting physical address permissions in `EPT` tables. The same operation is performed by the *Decision* service when it receives user light-level preferences from a service running on a different fog node, as a consequence to the user's transition from one room to another (Figure 9.2). Before the *Decision* service accesses the isolated data, it executes the non-privileged `VMFUNC` instruction to switch to the xMP domain which grants access to it. After it performs the access, the application switches back to the privileged xMP domain, which restricts access to the isolated data. Thus, as the domain that relaxes permissions to user light preferences is only active for the short duration while the *Decision* services accesses them, even in the presence of a memory corruption vulnerability, we ensure that an attacker cannot alter or exfiltrate the sensitive data.

10.4 Evaluation

The flexibility of the xMP framework allows us to extend this functionality and protect other sensitive data that may be processed by other user-space applications deployed by the fog orchestrator. For example, to provide user authentication, services may use shared libraries capable of cryptographic operations, such as OpenSSL [33], which store cryptographic material in their own process memory. In the presence of a memory corruption vulnerability in the crypto library, the cryptographic material may be compromised (e.g. Heartbleed [22]), allowing attackers to bypass authentication. However, the crypto library could be instrumented to isolate the sensitive material in an xMP domain, and allow access to it only in trusted location.

Moreover, xMP support selective memory isolation in the kernel-space. This functionality is available to kernel developers that can configure custom xMP domains dedicated to protect snesitive in-kernel data structures, such as process credentials represented by `struct cred` in the Linux kernel. For future releases of the Intelligent Infrastructure demonstrator, we also consider hardening kernel software on fog nodes and potentially other critical nodes from the networks deployed in the use-case (Figure 3.2).

Chapter 11 Managing personal data in vehicle's recharge process

11.1 Initial scenario

Fig. 11.1 highlights the main components of the vehicle's recharge process. Hence the driver using his personal device initiates the charging process in the smart parking lot. Once the charging process is done, the power distribution (based on the smart meter) sends the charged energy amount to be paid.

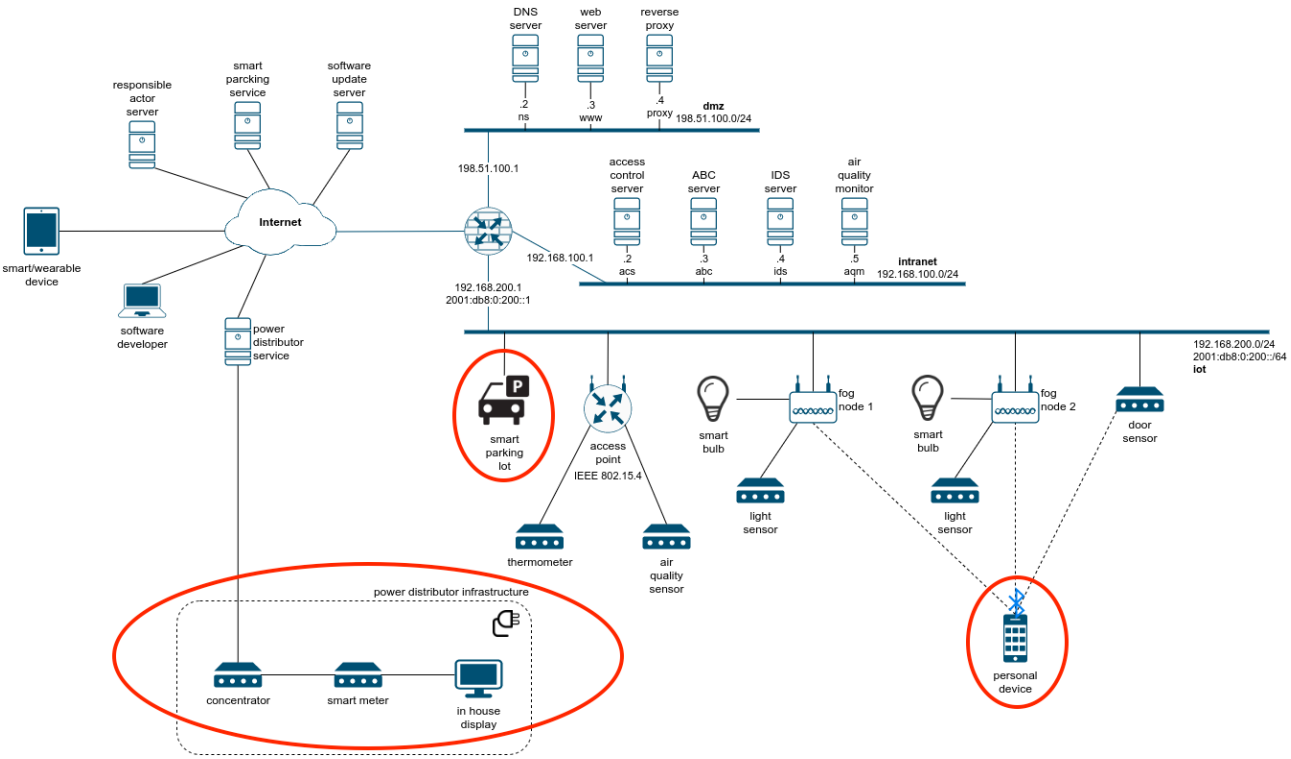

Figure 11.1: Vehicle's Recharge components n in the Global HAII-T infrastructure

Figure 11.2 presents the vehicle recharging payment process. The process includes four actors:

- Driver (i.e. Driver's PersonalDevice) – this actor can be represented by the driver's device (e.g., mobile phone or any other device) used to communicate the payment for the car charge. It presents the point of sales at the recharge point in the parking lot;
- ParkingLot (PL, i.e., smart parking lot) – the lot where the car is parked;
- EnergyServiceProvider (ESP, i.e., power distributor infrastructure) – manages the vehicle recharging infrastructure;
- SmartBuilding (SB) – runs the payment operation.

The scenario focuses on the Payment details, which potentially could include - the driver's name, bank account and/or credit card information, and authorisation to debit/credit the bank account and/or credit card for the service. Let's consider Payment details as the Driver's personal information. The process is as follows: the driver submits her payment details to the PL (see activity 1. Submit payment details), where the charging is happening. Once done, the PL initiates charge (see activity 2. Initiate charge) by sending initiation command to the ESP. Then ESP charges the car (see activity 3. Charge car with energy), sends information about the charged energy amount (see activity 4. Send charged energy amount) to the SB for the payment. Also, the ESP informs PL (see activity 5. Inform about the charge being done) that charge is completed.

After PL sends the Payment details to the SB (see activity 6. send payment details), the SB can to

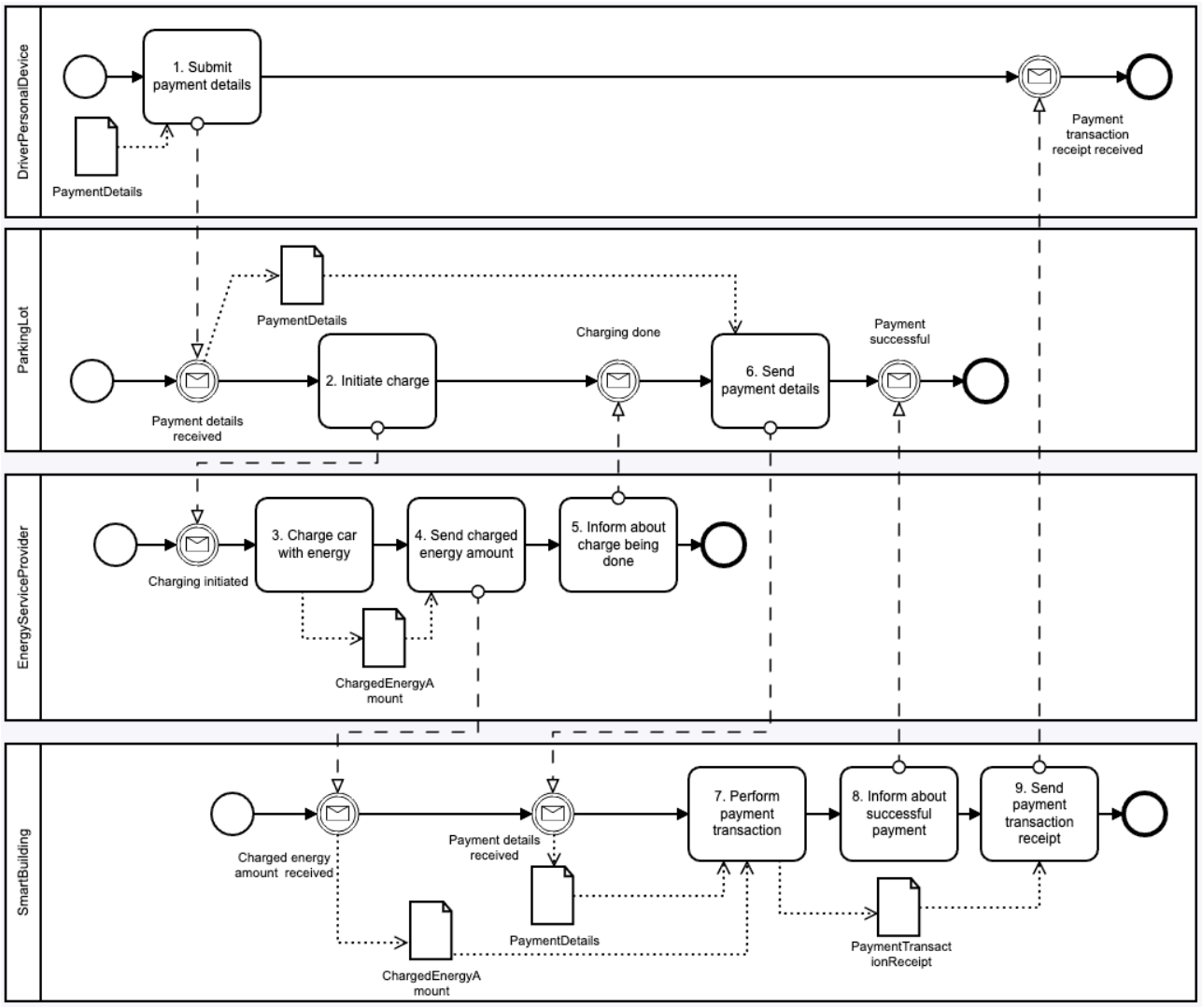

Figure 11.2: Vehicle recharge process

perform the payment transaction for the charged energy amount (see activity 7. Perform payment transaction). Then the SB informs the PL (see activity 8. Inform about successful payment) about the success of the payment (potentially Boolean value) and sends the payment transaction receipt (e.g., including the charged energy amount, the debited/credited amount) to the Driver (see activity 9. Send payment transaction receipt).

The problem in the above scenario is that it is not clear how the Driver's personal data (i.e., Payment details) are handled during the scenario and whether the treatment of the personal data respects the principles of the General Data Protection Regulation. The DPO tool[1] can help to assess how much the described process (see Fig. 11.2) is compliant to the GDPR and recommend means to achieve this compliance [30]. For example, in the scenario above, we can determine that the data owner is Driver whose personal data Payment Details are processed by the controller Smart Building. Analysis of the process using the DPO tool results in the list of non-compliances (see Fig. 11.3), such as:

- Consent is missing (GDPR [1], Art. 7)
- Privacy policy is missing (GDPR [1], Art. 13, 14)
- No security measures are present (GDPR [1], Art. 25)
- Processing task is not being recorded (GDPR [1], Art. 30)
- Record of processing task is missing (GDPR [1], Art. 30)

[1] https://dpotool.cs.ut.ee/

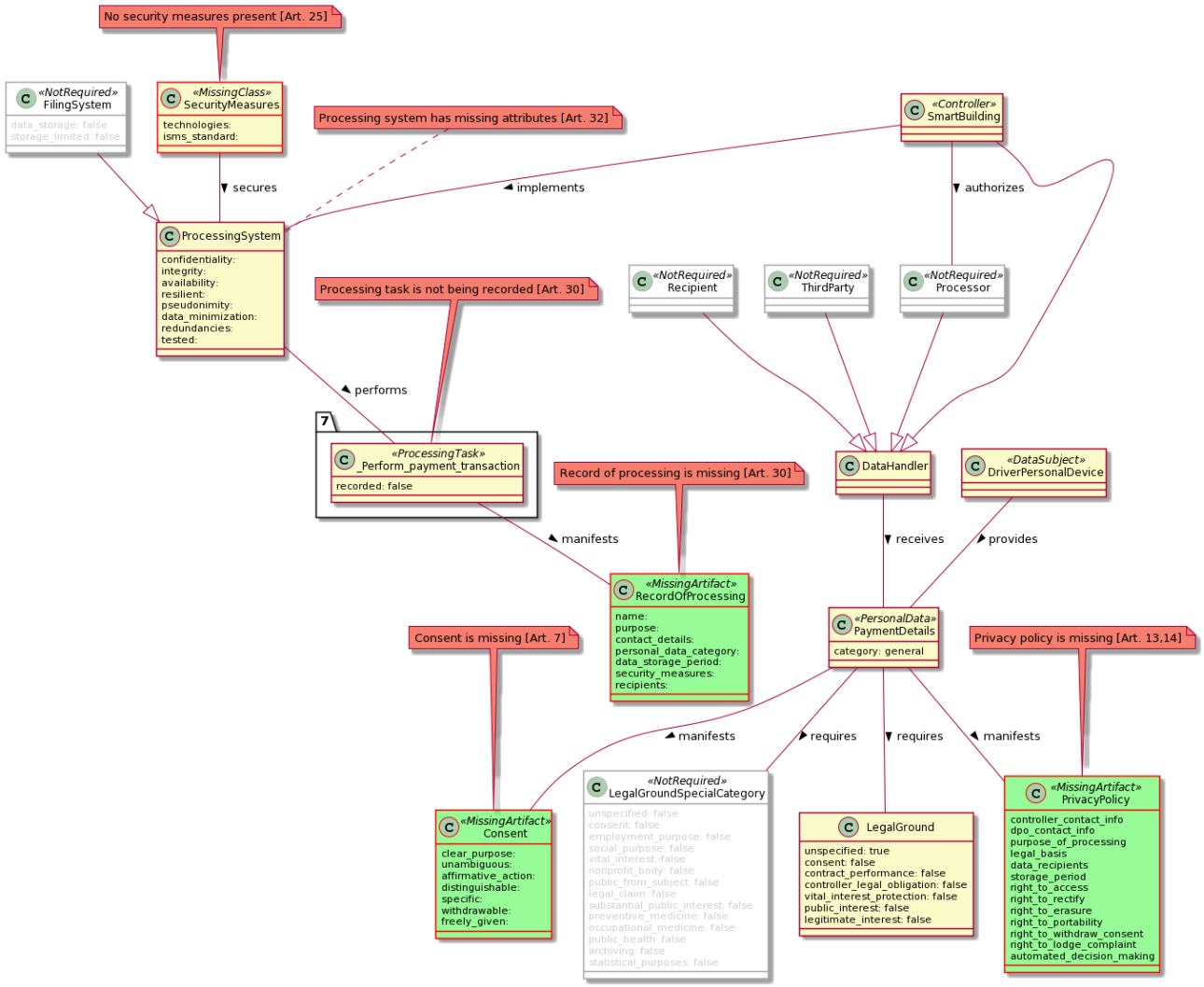

Figure 11.3**:** Compliance check of initial Vehicle charge process

- Processing system has missing security attributes (GDPR [1], Art. 32)

The DPO tool supports the process analysis during which the GDPR compliance could be achieved in the iterative manner, as illustrated in the next section.

11.2 Updated scenario

In this section, we discuss how each identified non-compliance to the regulation is addressed in the business process. The updated Vehicle charge process is presented in Fig. 11.4.

Introduction of the consent. To eliminate non-compliance of *Consent is missing*, one need to introduce process on how the consent should be shared between the personal data subject and controller. In Fig. 11.4 this is done by activities C1.1 where the Driver provides consent for processing payment details to SB and C1.2 where SB keeps consent for processing payment details. When the personal data are received, the SB checks consent for processing the payment details (C1.3). If it is not valid, the SB informs the Driver about the invalid consent (C1.5); otherwise, it proceeds with sending the permission to charge the vehicle (C1.4). In addition, Fig. 11.4 illustrates that the data subject's consent is collected prior processing task performed by the SB (controller).

Introduction of security measures. In Fig. 11.4 two security measures are introduced. Firstly, the sensitive personal information is communicated using the TLS protocol (see 1. Submit payment

details and 2. Send payment details). Secondly, PKencryption schema is applied. Hence, in the DriverPersonalDevice, the payment details are encrypted (C2.1) before sending it to the PL. In the SB, the payment details are decrypted (C2.2) before performing the payment transaction.

Record processing task. In a given example, processing task is 7. Perform payment transaction (see Fig. 11.2). Once the transaction is performed, the logging details (such as Name, Purpose, Contact Details, Personal Data Category, Data Storage Period, Security Measures, Third Countries Transfer, and Recipients) are logged in the next step (see, C3. Log payment transaction details).

Introduce the processing system's security attributes. The processing system must be designed in the way that security/privacy properties (such as Confidentiality, Integrity, Availability, Resilient, Pseudonymity, Data Minimization, Redundancies, and Tested) would be respected. If this is the case, these attributes could potentially be introduced to the SB as illustrated in Fig. 11.4.

Once the non-compliances are addressed in the business process model, on car run again the check of the business process model using the DPO tool [30]. The resulting model is illustrated in Fig. 11.5. It illustrates that non-compliance on *missing privacy policy* is still not addressed in the updated process model. This means that it is not clear whether the required information is provided to the data subject prior to data collection. Potentially this non-compliance should be addressed in the next iteration of the business process model analysis.

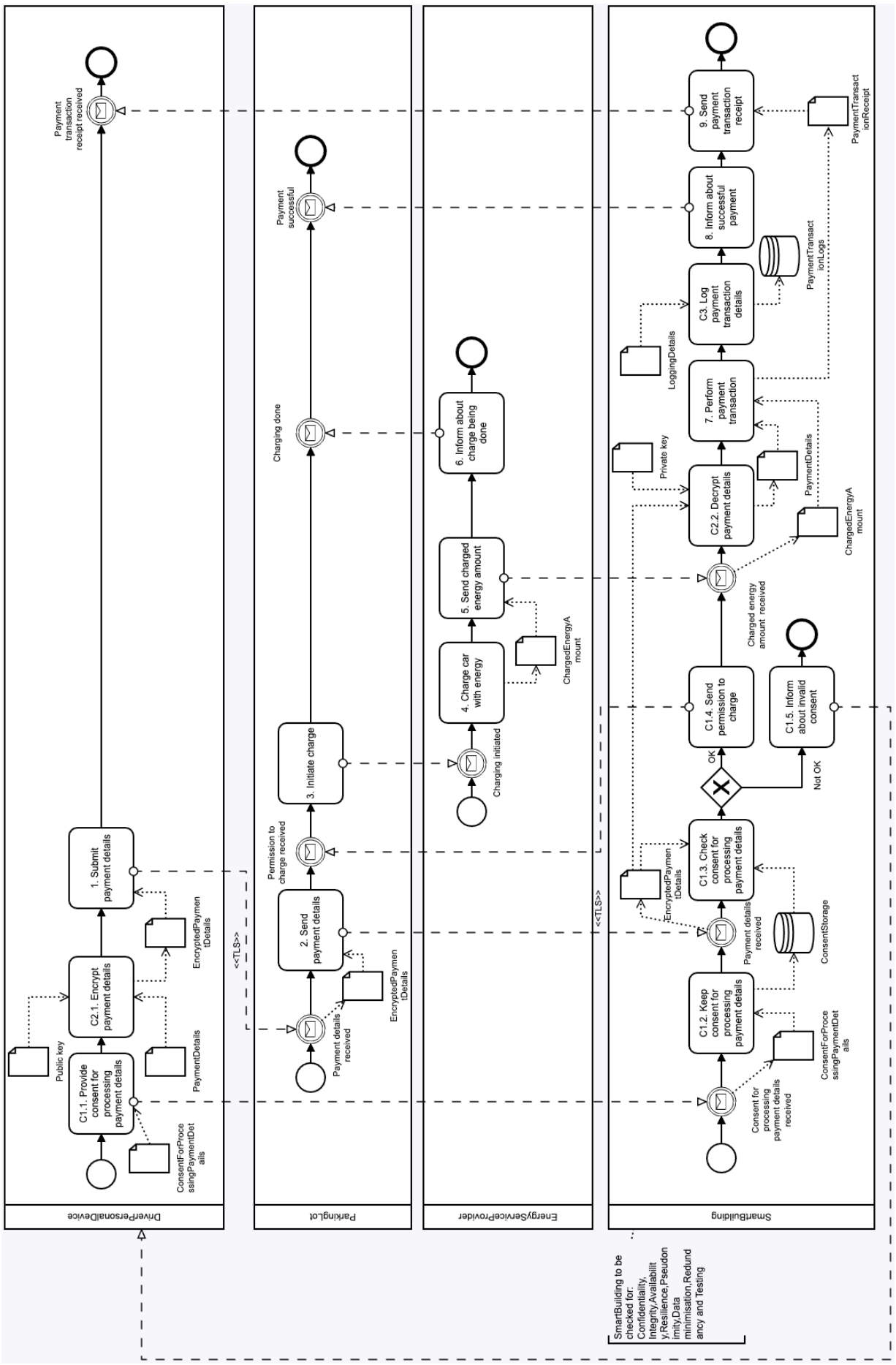

Figure 11.4: Updated Vehicle charge process

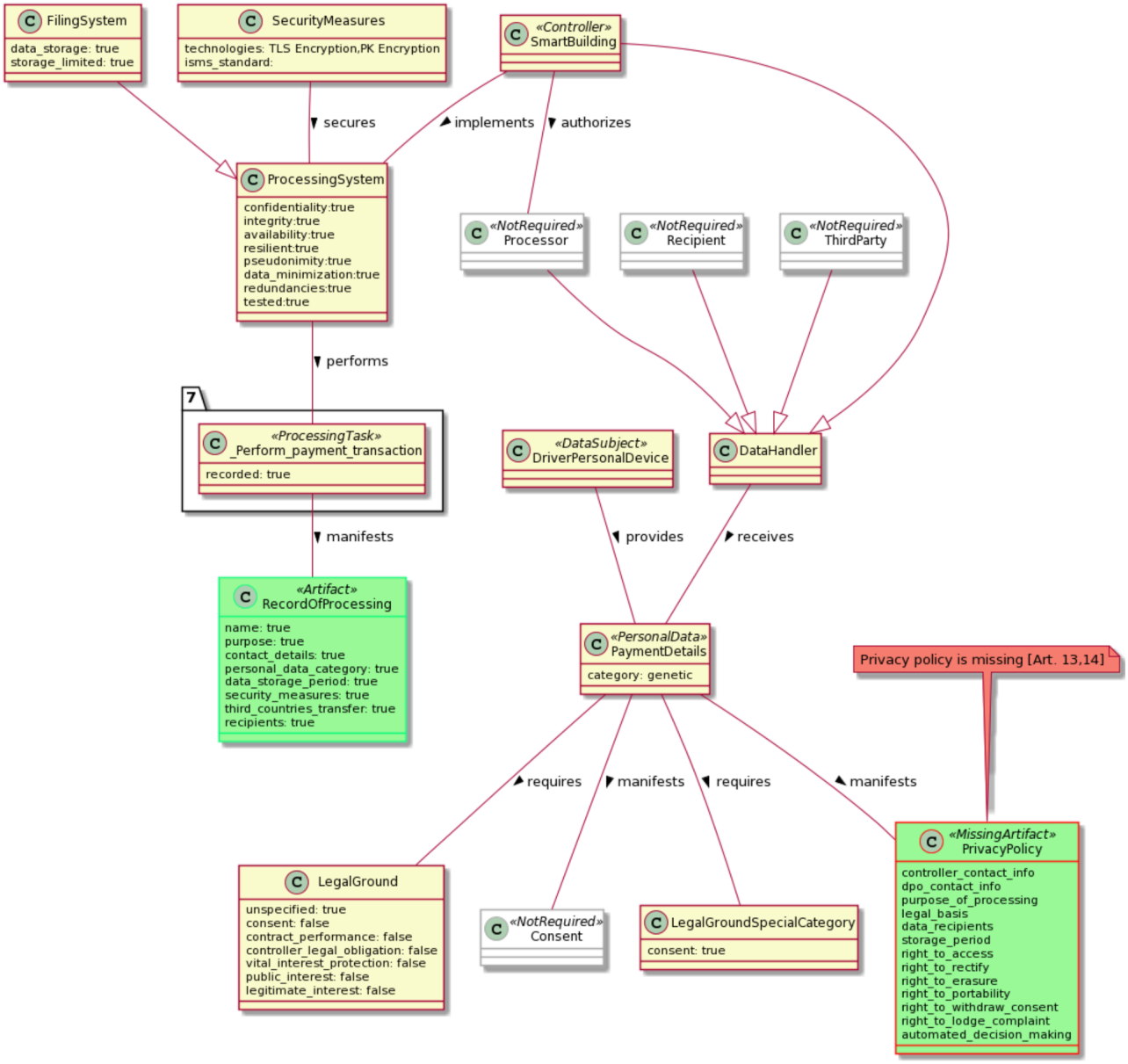

Figure 11.5: Compliance check of updated Vehicle charge process

11.3 Evaluation of the DPO tool

11.3.1 Compliance to ISO/IEC 27701:2019

ISO27701 [ISO27701] extends ISO27001 requirements to take into account privacy issues in compliance with the EU GDPR. Its annex A is composed to be an extension of ISO27001 statement of applicability for controllers of personally identifiable information. The DPO tool [30] evaluation requirements are generally the same to ISO27701 as described in the annex A controls, in addition, the DPO tool adds the visualisation part of business process and some automatisation steps to evaluate the compliance as illustrated above.

Table 11.1: Summary of DPO tool evaluation to ISO/IEC 27701:2019 Annex A controls

Controls included in the DPO tool	Controls partially included in the DPO tool	Controls not included in the DPO tool
• Identify and document purpose • Identify lawful basis • Determine when and how consent is to be obtained • Obtain and record consent • Privacy impact assessment • Records related to processing PII • Determining and fulfilling obligations to PII principals • Determining information for PII principals • Providing information to PII principals • Providing mechanism to modify or withdraw consent • Providing mechanism to object to PII processing • Access, correction and/or erasure • Providing copy of PII processed • Handling requests • Automated decision making • Limit collection • Limit processing • Retention • Disposal	• PII minimization objectives • PII de-identification and deletion at the end of processing • PII transmission controls • Identify basis for PII transfer between jurisdictions • Records of transfer of PII	• Contracts with PII processors • Joint PII controller • PII controllers' obligations to inform third parties • Accuracy and quality • Temporary files • Countries and international organizations to which PII can be transfered • Records of PII disclosure to third parties

Thus the DPO tool compliances were checked against the ISO27701 annex A controls. 31 controls were taken into account and out of these 19 controls were found as included in the DPO tool, 7 controls – not included and 5 were found as partially included in the DPO tool. Summary of the finding is presented in Table 11.1. The table illustrates some shortages of DPO tool models. It is important to add Artefact Contract between Controller and Processor, also improve some attributes related to the third country, add joint Controllers and multiple Processors functionality and refine consent issues related to parental care. Some use cases or processes were missing or could be interpreted as partial compliance. These can also be added to increase the previous work scope, for example, national law extensions, data transfer channels details, version management of privacy policy and record of processing, and some change management issues related to data changes or erasure.

11.3.2 Evaluation by experts

The DPO tool was evaluated by different expert groups with varied expertise and professional views [42]. The activity was carried in several iterations. Before the next iteration, the DPO models were refined based on prior expert group suggestions. The evaluation was conducted along with four parameters - accuracy, completeness, usability and reliability of the models used.

The evaluation was performed using semi-formal interviews with expert group members. During the interview, the DPO tool's DPIA, Data Breach, and Business Process Compliance models were explained, and their performance was presented through the illustrative scenario (similar to Fig. 11.3). The demonstration was done using the DPO tool. The main focus was on syntax and semantics of the models (including both BPMN and UML notation). After each interview, the models were improved and fixed based on the interview and its analysis.

Expert groups were composed of professionals with increasing experience of knowledge and implementation of GDPR. The first expert was a government office lawyer. His area of specialisation was in drafting legislation and contracts. The second expert group representative was a university data protection officer, who had two years of experience with practical use cases. The third expert group included a law firm that specialises in data protection consultation services and data audits to evaluate the compliance to GDPR. The last expert group was composed of Estonian Data Protection Authority (DPA) officials and included representatives from the technical and legal specialisations. All expert groups had a legal background but not any experience with BPMN or UML models.

Accuracy and completeness. The feedback of experts groups related to accuracy and completeness can be divided into categories: issues related with UML models (includes ambiguous class names, missing classes and attributes); minor issues with process improvements (includes missing artefacts and additional tasks); and requirements to improve corner case details (additional use case-based classes and attributes). All suggested changes were validated against the corresponding GDPR article clauses.

Usability and Reliability. At the beginning, the expert groups were intimidated by BPMN and UML notations as they have not used them in their day to day activities. However, after a brief introduction, the models were understood at the level to determine the necessary issues. All expert groups preferred BPMN over UML models as visualisation part of the models/tool due to its intuitive nature. However, they pointed out that the GDPR model in UML (see details in [30]) helps save a significant amount of time in processing the GDPR requirements. The expert groups found that the DPO Tool concept and the models have an excellent potential use for refining and re-evaluating business processes from the legal requirements perspective.

Chapter 12 Privacy-preserving data processing

12.1 User data processing within the demonstration use case

User data processing within the intelligent infrastructure can offer useful insights on the system usage characteristics and suggestions for its improved configuration. Privacy preservation during user data processing is important, both to protect from data breaches but also to prevent unauthorized data use within the system.

The proposed privacy-preserving data processing solution is applicable in the smart parking service in the HAII-T infrastructure, as shown in Fig.12.1. At the moment the system is intended to be applied in the smart parking service, that manages the smart parking lot and the charging units. The reservation and billing transaction data from this service provide compatible data items for the data processing mechanisms. For example, this data can be processed to produce statistics on the parking and charging spots demand, peak hours and availability, to determine appropriate pricing policies for the service.

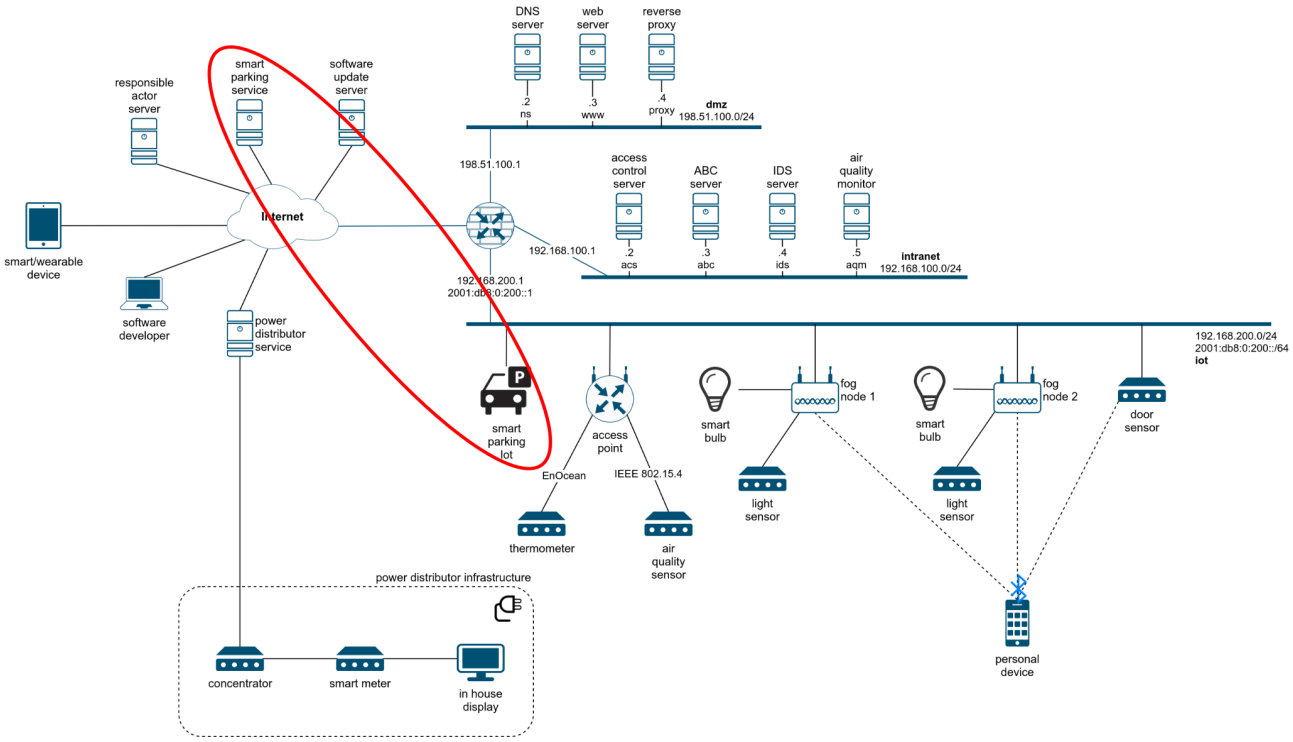

Figure 12.1: Privacy-preserving data processing in the HAII-T infrastructure.

12.2 Using searchable encryption for privacy-preserving data processing

Searchable Encryption (SE) offers the ability to store information in an encrypted form, while allowing search operations to be performed on the encrypted dataset, so that authorised users can locate the information of interest to them without the need to decrypt the dataset.

SE can be used for data retrieval from untrusted cloud servers, without revealing any sensitive information to the service provider or any other entities. It is also a suitable technique for privacy-preserving computations, and counting queries in particular, to determine how many of the data records in the dataset contain a specific keyword, representing an attribute of interest. Although data remain encrypted on the server side, SE records can be decrypted by the recipient of the search results, therefore data minimisation still needs to be applied during data collection for the encrypted dataset, as re-identification could be possible if the decrypted data contain identifiable information.

Searchable encryption has been a very active research topic over the last decade, producing increasingly effective and applicable solutions [10, 16]. By applying technologies of cryptography, data structures, algorithms, information retrieval and databases, state of the art constructs make it possible to achieve different compromises between security, efficiency and query expressiveness.

Searchable Symmetric Encryption (SSE) is a practical method for searchable encryption providing a balance between efficiency, functionality and security. A searchable symmetric encryption scheme consists of an initial database setup algorithm and a search protocol. A database DB is a list of identifier and keyword-set pairs. During setup, a database DB is used as input with a list of document decryption keys, producing a secret key K along with an encrypted database EDB. The search protocol proceeds between a client C and server E, where C takes as input the secret key K and a query (a tuple of keywords and a boolean formula) and E takes as input EDB. At the end of the protocol, C outputs a set of document identifiers. This basic structure is further enhanced to enable multi-client functionality and dynamic record additions to the encrypted dataset [14, 15, 26]. The privacy requirements of a searchable encryption scheme include: index privacy, document privacy, query privacy, search pattern privacy and access pattern privacy.

12.3 Evaluation

The goal of the privacy-preserving data processing system is to allow the utilization and valorization of user data within the intelligent infrastructure, while protecting the data privacy. Although at the moment the system is intended to be applied in the smart parking use case, where the reservation and billing transaction data will provide compatible data items for the data processing mechanisms, we expect this functionality to be applicable to other parts of the infrastructure that produce service usage data, to offer insights on the system usage statistics while preserving data privacy.

At this stage the properties and goals of the system have been identified along with viable tools for its implementation [2, 27]. The system is still in preliminary stage and a more detailed specification is yet to be defined. Implementation and experimental evaluation will be carried out during the next year to examine the practical applicability of the solution. The evaluation results will be reported in D6.4. In total, the privacy-preserving data processing system is expected to enhance the intelligent infrastructure functionality.

Chapter 13 Scenarios in relation to edge-, fog- and cloud-computing

All described scenarios consist of a number of devices and systems that provide services and functionality. With that information given, a focus also has to be taken on the management of privacy and security. Security principles have been discussed in the first deliverable (D6.1-Security-by-Design Framework for the Intelligent Infrastructure) [19] and potential security and privacy-specific related key components have been listed (sub-chapter 4.5). One goal is to map these security principles and components to functional and service components in the scenarios, or to suggest the extension with essential components. Generic properties should be defined that can be added to the TOSCA-based description of the scenarios.

A first step is the mapping of the known scenarios and their components to the technological architecture of IoT frameworks in relation to edge-, fog- and cloud-computing. The view on that has been extended with the user devices like smart phones, smart cards or personal computers, which might access the infrastructure either from the edge side, mainly wireless, or from the cloud side, which also includes wired computers. To visualize this, user devices have been put on both sides of the grid. In the overviews also the networks of the scenarios overview (figure 2.4) are presented: the IoT network, the Intranet and the Internet.

The mapping has been done on base of the known scenarios: Some scenarios are related to smart home, so to manage the environment of the home like temperature and air quality, or trace the energy consumption of the house. Other scenarios are related to smart building and control the access to the building or a parking lot as well as manage the light bulbs. The last scenario consists of the charging of electric vehicles at a parking lot, including the payment process of the energy.

In most of the scenarios, the edge- or IoT devices are accessed directly by the user devices via wireless technologies. This requires physically being close the the edge. The logic of these scenarios are concentrated in the edge or fog layer without having a cloud component. Only one scenarios (home environment) does not give direct access to the edge and requires the access of cloud applications via the Internet. This might even be the case with being physically close to the edge devices.The cloud applications provide the only interface for the users and manage the communication and exchange of data with the edge devices. The logic is split between the cloud and the edge, moving as much as possible to the edge.

13.1 Scenarios that cover only edge- and fog-nodes

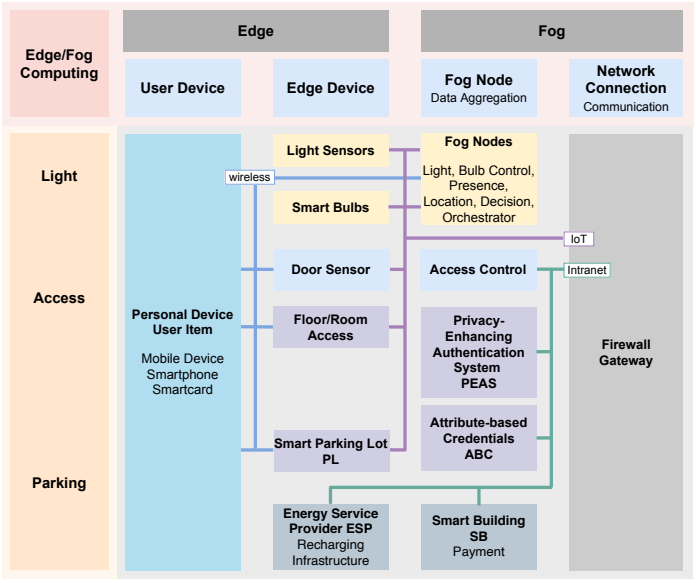

Figure 13.1: Scenarios in relation to edge- and fog-computing.

In the smart building scenarios, the IoT devices are accessed directly or via a fog nodes (Figure 13.1). In case of the Smart Bulbs, every communication stays in the IoT network between bulb, sensor and fog node, as well as between fog nodes. In the access scenarios, the sensors in the IoT network at the edge request information from fog nodes in the Intranet, which take the final decisions. Both networks are decoupled via a firewall. In the vehicles charging scenario the parking lot acts as the edge device that is responsible to retrieve the payment information. A second edge device is the recharging infrastructure that charges the car after the payments have been processed by the payment system, which is a fog node.

13.2 Scenarios that cover edge- and fog- and cloud-nodes

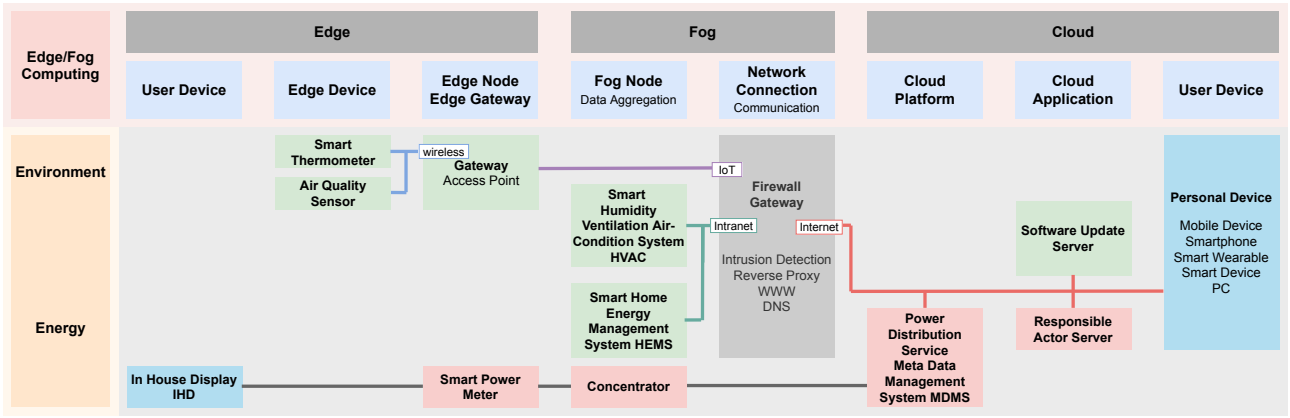

Figure 13.2**:** Scenarios in relation to edge-, fog- and cloud-computing.

In the smart home scenarios (Figure 13.2) the energy provision cannot fully be aligned to the edge-, fog, and cloud-computing grid, because the used communication networks between the in house display and the power distribution service depend on the infrastructure of the energy provider and does not necessarily use the internet connectivity of the house. In the environment control scenario, the user directly interacts at the edge-device, mainly to set the preferred temperature and to monitor the current temperature. The edge node in the IoT network acts as a wireless access point for the smart sensors as well as a gateway to the management systems in the intranet. A second type of users manage software updates for the smart sensors that are provided in the cloud and which can be downloaded from the sensors themselves.

Chapter 14 Conclusion

In this document we described the year 2 demonstrations of the HAII-T Program. In particular, the document presented a list of demonstrations, each showing one of the technologies developed in WP6. All the demonstrations were staged in a dedicated application scenario and all the application scenarios are based on a common use case replicating the infrastructure of a real smart building. Each application scenario shows how one of the proposed techniques deals with a specific security issue. Although preliminary, the results presented in this document highlight that actual intelligent infrastructures can benefit from the methodologies under development in this Program. Further research needs to be carried out for ensuring better applicability and integration of these methodologies. This activity will be reported in the year 3 demonstrator.

Chapter 15 Bibliography

[1] EU General Data Protection Regulation. URL https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L.2016.119.01.0001.01.ENG.

[2] Clusion library. https://github.com/encryptedsystems/Clusion.

[3] The Xen Hypervisor. https://wiki.xen.org/wiki/Main_Page.

[4] The gnu mp bignum library, 2014. URL https://gmplib.org/.

[5] Openssl: The open source toolkit for ssl/tls, 2014. URL https://www.openssl.org/docs/manmaster/crypto/crypto.html.

[6] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti. Control-flow integrity. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 340–353. ACM, 2005.

[7] Jean-Philippe Aumasson and Daniel J Bernstein. SipHash: a fast short-input PRF. 2012.

[8] Emmanuel Baccelli et al. RIOT: An open source operating system for low-end embedded devices in the IoT. *IEEE Internet of Things Journal*, 2018.

[9] M. Bollo, A. Carelli, S. Di Carlo, and P. Prinetto. Side-channel analysis of secube™ platform. In *2017 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–5, Sep. 2017. doi: 10.1109/EWDTS.2017.8110067.

[10] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, 47(2):1–51, 2014.

[11] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[12] Jan Camenisch, Manu Drijvers, and Jan Hajny. Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, WPES '16, pages 123–133, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4569-9. doi: 10.1145/2994620.2994625. URL http://doi.acm.org/10.1145/2994620.2994625.

[13] Jan Camenisch, Manu Drijvers, Petr Dzurenda, and Jan Hajny. Fast keyed-verification anonymous credentials on standard smart cards. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 286–298. Springer, 2019.

[14] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Annual cryptology conference*, pages 353–373. Springer, 2013.

[15] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In *NDSS*, volume 14, pages 23–26. Citeseer, 2014.

[16] Khadijah Chamili, Md. Jan Nordin, W. Ismail, and A. Radman. Searchable encryption: A review. *International journal of security and its applications*, 11:79–88, 2017.

[17] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1205–1216, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2957-6. doi: 10.1145/2660267.2660328. URL http://doi.acm.org/10.1145/2660267.2660328.

[18] Timothy Claeys et al. RIOT and OpenWSN 6TiSCH: Happy Together. *in Proceedings of IFIP/IEEE PEMWN*, December 2020.

[19] Gabriele Costa, Alessandro Armando, Joaquin Garcia-Alfaro, Jean-Max Dutertre, Jean-Luc Danger, Gianluca Roascio, Paolo Prinetto, Michel Hurfin, Ludovic Me, Giorgio Bernardinetti, Francesco Mancini, Sergej Proskurin, Claudia Eckert, Uwe Roth, Qiang Tang, Lukas Malina, Petr Dzurenda, Manon Knockaert, Jean-Marc Van Gyseghem, Raimundas Matulevicius, Abasi-Amefon O. Affia, Kaspar Kala, Branka Stojanovic, Katha-

rina Hofer-Schmitz, Marek Pawlicki, Tewodros Beyene, and Nerijus Morkevicius. *D6.1 Security-by-Design Framework for the Intelligent Infrastructure*. SPARTA, Feb 2020. URL https://www.sparta.eu/assets/deliverables/SPARTA-D6.1-Security-by-design-framework-for-the-intelligent-infrastructure-PU-M12.pdf.

[20] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *Public Key Cryptography - PKC 2005*, pages 416–431, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-30580-4.

[21] G. A. Farulla, A. J. Pane, P. Prinetto, and A. Varriale. An object-oriented open software architecture for security applications. In *2017 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–6, Sep. 2017. doi: 10.1109/EWDTS.2017.8110070.

[22] Heartbleed. The Heartbleed Bug. https://heartbleed.com/.

[23] K. Hofer-Schmitz and B. Stojanović. Towards formal methods of iot application layer protocols. In *2019 12th CMI Conference on Cybersecurity and Privacy (CMI)*, pages 1–6, 2019. doi: 10.1109/CMI48017.2019.8962139.

[24] Katharina Hofer-Schmitz and Branka Stojanović. Towards formal verification of iot protocols: A review. *Computer Networks*, 174:107233, 2020. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2020.107233. URL http://www.sciencedirect.com/science/article/pii/S1389128619317116.

[25] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual. https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf.

[26] Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. Outsourced symmetric private information retrieval. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 875–888, 2013.

[27] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 94–124. Springer, 2017.

[28] Tamas K Lengyel. Stealthy Monitoring with Xen altp2m. https://xenproject.org/2016/04/13/stealthy-monitoring-with-xen-altp2m/.

[29] Paul Lipton, Chris Lauwers, Matt Rutkowski, Claude Noshpitz, and Calin Curescu. TOSCA Simple Profile in YAML Version 1.3. Technical report, OASIS, February 2020. URL https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.pdf.

[30] Raimundas Matulevičius, Jake Tom, Kaspar Kala, and Eduard Sing. A method for managing gdpr compliance in business processes. In *International Conference on Advanced Information Systems Engineering*, pages 100–112. Springer, 2020.

[31] N. Maunero, P. Prinetto, G. Roascio, and A. Varriale. A fpga-based control-flow integrity solution for securing bare-metal embedded systems. In *2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–10, 2020. doi: 10.1109/DTIS48698.2020.9081314.

[32] Bredan Moran et al. A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest. *IETF Internet Draft draft-ietf-suit-manifest-09*, 2020.

[33] OpenSSL. OpenSSL. https://www.openssl.org/.

[34] Sergej Proskurin, Marius Momeu, Seyedhamed Ghavamnia, Vasileios P. Kemerlis, and Michalis Polychronakis. xmp: Selective memory protection for kernel and user space. In *IEEE Symposium on Security and Privacy (Oakland)*, May 2020. URL https://www.sec.in.tum.de/i20/publications/xmp-selective-memory-protection-for-kernel-and-user-space/@@download/file/2020-oakland-xmp.pdf.

[35] Gabriele Restuccia, Hannes Tschofenig, and Emmanuel Baccelli. Low-Power IoT Communica-

tion Security: On the Performance of DTLS and TLS 1.3. *in Proceedings of IFIP/IEEE PEMWN*, December 2020.

[36] RIOT. 2020.10 Release. *GitHub*, 2020. https://github.com/RIOT-OS/RIOT/releases/tag/2020.10.

[37] R. Roemer, E. Buchanan, H. Shacham, and S. Savage. Return-oriented programming: Systems, languages, and applications. *ACM Transactions on Information and System Security (TISSEC)*, 15(1):2, 2012.

[38] Iman Sharafaldin., Arash Habibi Lashkari., and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*, pages 108–116. INSTICC, SciTePress, 2018. ISBN 978-989-758-282-0. doi: 10.5220/0006639801080116.

[39] Mitsunari Shigeo. Mcl library. https://github.com/herumi/mcl, 2018.

[40] Sandro Skansi. *Introduction to Deep Learning: from logical calculus to artificial intelligence*. Springer, 2018.

[41] H. A. Sonawane and T. M. Pattewar. A comparative performance evaluation of intrusion detection based on neural network and pca. In *2015 International Conference on Communications and Signal Processing (ICCSP)*, pages 0841–0845, April 2015. doi: 10.1109/ICCSP.2015.7322612.

[42] Jake Tom, Raimundas Matulevičius, and Mari Seeba. Model-Driven DGPR Compliance Management in Business Processes. *Submitted for publication*, 2020 (submitted).

[43] Hannes Tschofenig and Emmanuel Baccelli. Cyberphysical security for the masses: A survey of the internet protocol suite for internet of things security. *IEEE Security & Privacy*, 2019.

[44] Algimantas Venčkauskas, Nerijus Morkevicius, Vaidas Jukavičius, Robertas Damaševičius, Jevgenijus Toldinas, and Šarūnas Grigaliūnas. An Edge-Fog Secure Self-Authenticable Data Transfer Protocol. *Sensors*, 19(16), 2019. ISSN 1424-8220. doi: 10.3390/s19163612. URL https://www.mdpi.com/1424-8220/19/16/3612.

[45] N. R. Weidler, D. Brown, S. A. Mitchel, J. Anderson, J. R. Williams, A. Costley, C. Kunz, C. Wilkinson, R. Wehbe, and R. Gerdes. Return-oriented programming on a cortex-m processor. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 823–832, Aug 2017. doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.318.

[46] N. R. Weidler, D. Brown, S. Mitchell, J. A. Anderson, J. R. Williams, A. Costley, C. Kunz, C. Wilkinson, R. Wehbe, and R. Gerdes. Return-oriented programming on a resource constrained device. *Sustainable Computing: Informatics and Systems*, 22:244–256, 2019.

[47] Koen Zandberg and Emmanuel Baccelli. Minimal Virtual Machines on IoT Microcontrollers: The Case of Berkeley Packet Filters with rBPF. *in Proceedings of IFIP/IEEE PEMWN*, December 2020.

[48] Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. Secure firmware updates for constrained IoT devices using open standards: A reality check. *IEEE Access*, 7:71907–71920, 2019.