SPARTA

D6.4

Final 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 / D6.4 / V1.0
Work package contributing to the deliverable	WP6
Due date	January 2022 – M36
Actual submission date	9th February, 2022

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

Abstract	This document describes the final 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, Alexandre Sanchez (Inria)
Gianluca Roascio, Paolo Prinetto, Gabriele Costa, Alessandro Armando (CINI)
Gabriele Restuccia (CNIT)
Nerijus Morkevičius, Algimantas Venčkauskas (KTU)
Uwe Roth, Qiang Tang (LIST)
Marek Pawlicki (ITTI)

Reviewers (ordered according to beneficiary numbers)

Katarzyna Kapusta (TCS)
Artsiom Yautsiukhin (CNR)

Disclaimer

Executive Summary

This document presents the final release of the demonstration of the HAII-T Program. The document focuses on the notion of security workflows, which we use to represent security tasks common in Security-by-Design methodologies. The main reason behind this approach is that security processes may significantly vary between different infrastructures. Thus, a general approach to Security-by-Design should provide customizable processes based on well established, existing methodologies. Our security workflows combine the methodologies that currently populate the HAII-T orchestrator. By defining a security workflow and executing it during a specific phase of the life cycle of an intelligent infrastructure, one can tackle security issues of interest.

For the sake of presentation, here we put forward three security workflows as well as other, standalone techniques, and we demonstrate them through an application to the case study virtual infrastructure introduced in [34]. The three workflows are discussed in Chapters 3, 4 and 5, respectively. There, we present the components forming each workflow. The demonstration of the HAII-T approach is presented in Chapter 6. Finally, Chapter 7 presents an effectiveness evaluation based on specific security issues involved in the demonstration scenarios.

Table of Content

List of Figures

List of Tables

Chapter 1 Introduction

Properly managing the security of a modern Intelligent Infrastructure (II) is both extremely complex and sensitive. On the one hand, complexity arise from the coexistence of many different paradigms, such as Internet of Things (IoT), Cloud and Fog computing, and many legacy technologies which all together collaborate for the business logic of the infrastructure. On the other hand, most II run critical services that cannot be discontinued without dramatic effects on our society.

In the context of SPARTA, the High Assurance Intelligent Infrastructure Toolkit (HAII-T) Program is responsible for the development of a Security-by-Design methodology which effectively improves the security of existing and next-generation II. In our previous deliverable [34], we presented and demonstrated the first version of our toolkit. Briefly, it consisted of an orchestration framework based on a standard Infrastructure-as-a-Service (IaaS) system, called TOSCA [59]. The orchestration framework was enriched with a number of security mechanisms and tools developed within the HAII-T Program. Each of them deals with a specific security issue that may affect a real II. To demonstrate the effectiveness of these methodologies we also developed a virtual replica of a real II which served as case study.

Following the Security-by-Design philosophy, security is not a mere technological problem and strong guarantees can only be provided by means of security processes. Security processes chaperon the entire life cycle of any II and they consist of tasks dealing with some specific security aspects. Processes and tasks must be defined by combining security and domain expertise. Although some of them are recurrent, others may significantly vary depending on technologies, context and other factors. Nonetheless, all of them must be well integrated in a long-term security strategy and continuously maintained over time.

In this document, we describe the year 3 demonstration of the HAII-T Program. The main novelty introduced in this version is the focus on the definition of *security workflows*. A workflow is a security-oriented activity which is obtained from the orchestration of automated tools and human beings. Intuitively, security workflows capture the notion of security task as they are designed and implemented for managing a specific security aspect of a given II. Ideally, the owner of an II can define her own security workflows as well as integrating existing ones, e.g., defined by external authorities. The result is a multifaceted and operational definition of security procedures driven by the HAII-T orchestration framework.

For the sake of presentation, here we put forward three security workflows relevant for the scenario of our case study. Each workflow is first described in detail and then demonstrated through an application scenario. Furthermore, this document includes a technical description and demonstrations of the methodologies that are not included in the selected workflows. These methodologies are the fundamental building blocks for the definition of further security workflows which one may want to implement.

Chapter 2 Integration strategy

In this section we present the integration methodology followed during the development of the final version of the Toolkit. The overall approach consists of an extension of the first integration presented in [34] and it follows the Security-by-Design strategy introduced in [33].

2.1 Secure orchestrator architecture

The cornerstone of the integration methodology is the secure orchestrator, i.e., a system responsible for managing all the phases of the life cycle of an intelligent infrastructure. As discussed in [34], the orchestration framework relies on TOSCA [59], which we extended with new elements for modeling, analyzing, deploying and monitoring intelligent infrastructures.

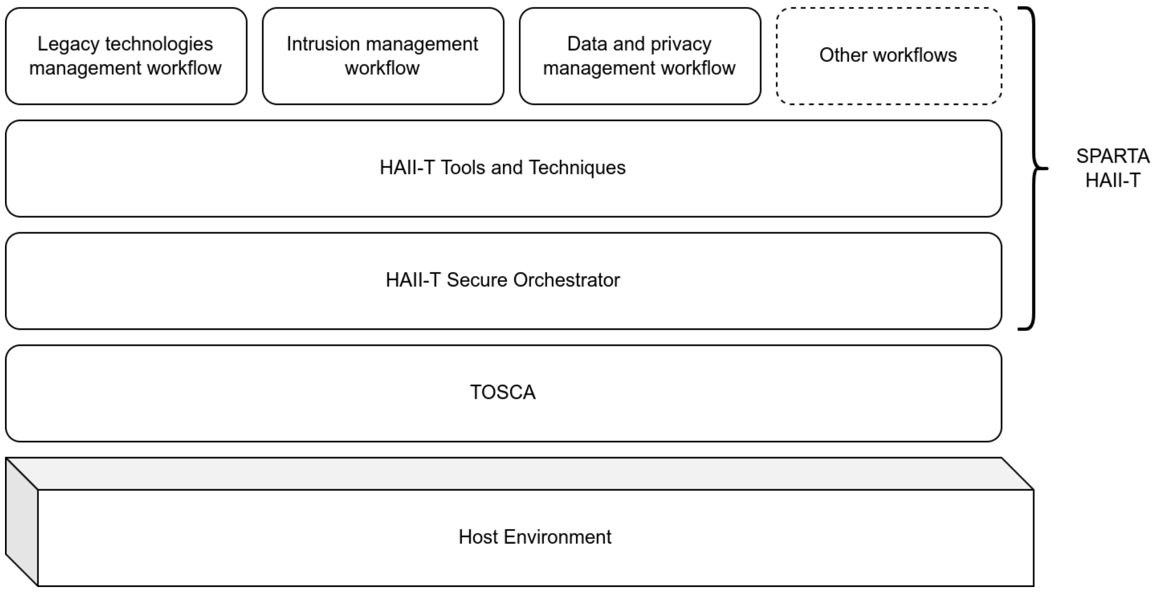

Legacy technologies management workflow
Intrusion management workflow
Data and privacy management workflow
Other workflows
HAII-T Tools and Techniques
HAII-T Secure Orchestrator
TOSCA
Host Environment
SPARTA HAII-T

Figure 2.1: HAII-T Deployment diagram

The overall architecture of the secure orchestrator is depicted in Figure 2.1. Briefly, all the tools and techniques developed in WP6 contribute to the extension of the TOSCA framework carried out by the secure orchestrator. These tools have been previously demonstrated in [34]. There we showed how they can deal with security issues which commonly affect a generic II. Nevertheless, real and fruitful integration occurs when different techniques cooperate for the execution of a *security task*. In Security-by-Design, security tasks populate the phases of the life cycle of a system and they are meant to assess specific security aspects related to phase they refer to. For instance, a threat modeling task typically occurs in the early stages, while penetration testing is carried out when the system is up and running. Although most security tasks are recurrent, they may vary depending on the adopted security framework. Even more problematic, their actual implementation is application-specific as it depends on the implementation details of the considered II.

For the reasons discussed above, although some can be reused, security tasks cannot be defined once for all, in advance. On the contrary, the definition of the security tasks of interest, as well as their maintenance over time, is left to the owner of the II. Still, the secure orchestrator must support this activity by providing a catalog of security technologies and support for combining them.

2.2 Security workflows

In TOSCA, the execution of a task associated to a specific phase in the life cycle of an infrastructure is called a *workflow*. TOSCA workflows are typically used for automating some process, e.g., configuration of some elements after their deployment. Here, we extend this notion to implement security-related workflows. Their role is to carry out a specific security task. In the following we introduce three security workflows that will take part in our demonstration.

2.2.1 Legacy technologies management workflow

The presence of multiple, legacy technologies is frequent in large scale II. These technologies are typically necessary to handle specific sub-processes and sub-systems which cooperate in the complex business logic of the II. Although this problem may occur in different contexts, it is more common for hardware devices. As a matter of fact, modern paradigms such as IoT favor the spreading of small, networked edge devices. These devices may suffer from a number of limitations and weaknesses which, in some cases, are known and documented. For instance, to keep its price affordable, the developer of a field sensor may prefer not to equip it with particular anti-tampering protection mechanisms. In this way, this burden is left to the II owner who has to carefully evaluate the context where the device is deployed.

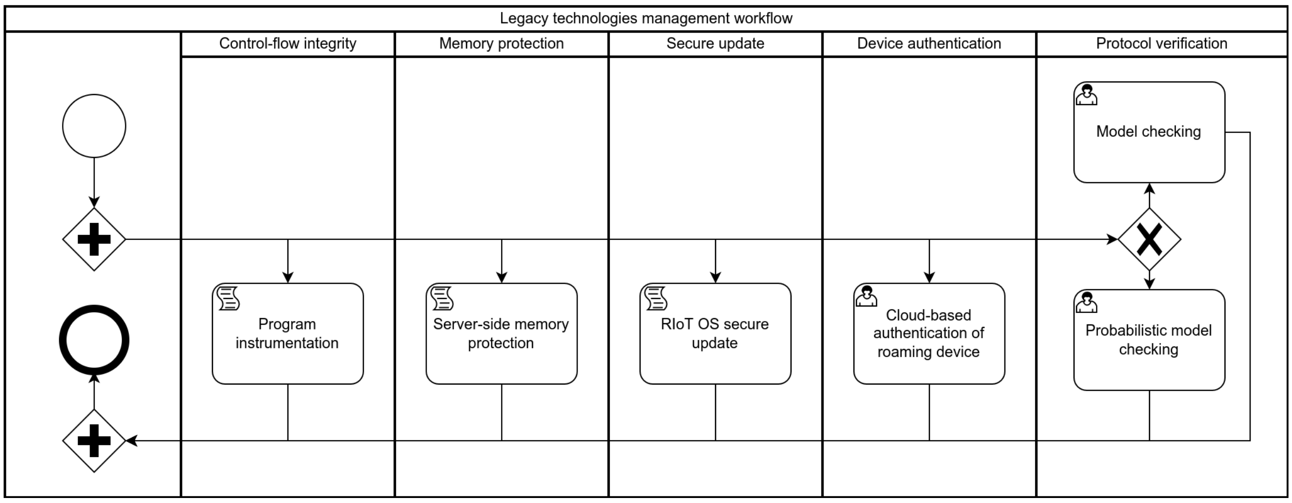


Figure 2.2: Legacy technologies management workflow

Figure 2.2 shows a possible implementation of a legacy technologies management workflow. The workflow is mainly oriented to IoT applications and it consists of a series of hardening and verification techniques. These techniques include:

- program instrumentation, for enabling control-flow integrity guarantees on edge devices;
- memory protection, for ensuring secure, server-side data storage;
- secure update, which permits to securely install new firmware on RIoT devices;
- cloud-based authentication, for managing and verifying the identity of roaming devices, and;
- formal protocol verification, for proving that IoT protocols do not suffer from security flaws.

A distinguished feature of the legacy technologies management workflow is its cross-phase nature. As a matter of fact, the techniques contribuiting to the workflow may operate during different phases of the II life-cycle. A detailed description of these methodologies and their role in the workflow will be given in Section 3.

2.2.2 Intrusion management workflow

Intrusions are a major threat for modern infrastructures. Although proper design and security mechanisms can decrease the likelihood of intrusions, new vulnerabilities and attack strategies which emerge over time cannot be avoided. Hence, assuming that intrusions can occur and implementing security tasks for handling them is always a good practice. Recently, security orchestration, automation and response (SOAR), are emerging for the definition of security processes triggered by critical events. This workflow highlights that the HAII-T orchestrator can implement the logic of a SOAR system.

An Intrusion Detection System (IDS) is the component responsible for identifying and reporting intrusion events. An IDS monitors the network traffic, as well as other sources of information, and applies some detection strategy, e.g., based on pattern matching or other heuristics. As a consequence of an intrusion, an alert is fired and an intrusion management process is triggered. For instance, this process may involve the intervention of a human analyst who revises the alert report and activates some emergency plan.

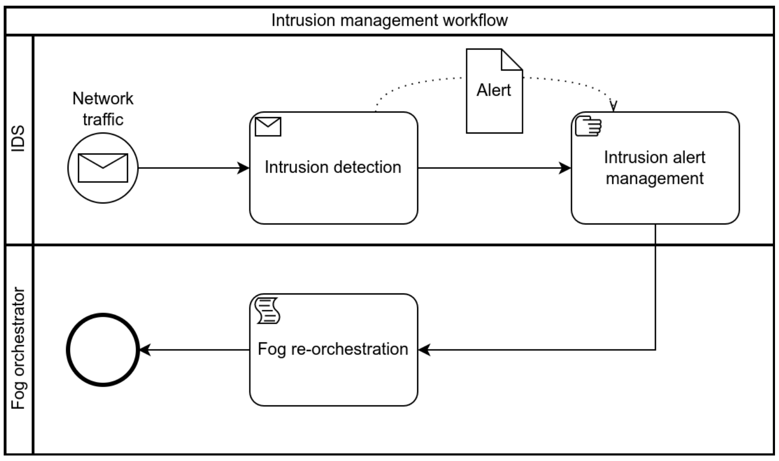


Figure 2.3: Intrusion management workflow

Figure 2.3 shows an instance of the intrusion management workflow where the human operator decides to re-orchestrate a Fog application. Such a decision is base, for example, on the specific nature of the alert, e.g., network traffic compatible with an attack to Fog nodes, or to ensure that the Fog application Quality of Service (QoS) is not compromised by the intrusion. A detailed description of this workflow and its core elements is given in Chapter 4.

2.2.3 Data and privacy management workflow.

Privacy protection is another fundamental issue for II. As a matter of fact, one can hardly think of a real infrastructure that does not involve collecting and processing sensitive data. Identifying appropriate mechanisms ensuring data protection must be carried out from the earliest stages of the design process. This approach goes under the name of Privacy-by-Design. Recently, lawmakers have devoted a significant effort in the development of rules which aim at favoring the adoption of Privacy-by-Design. Among them, the General Data Protection Regulation (GDPR) is a prominent example.

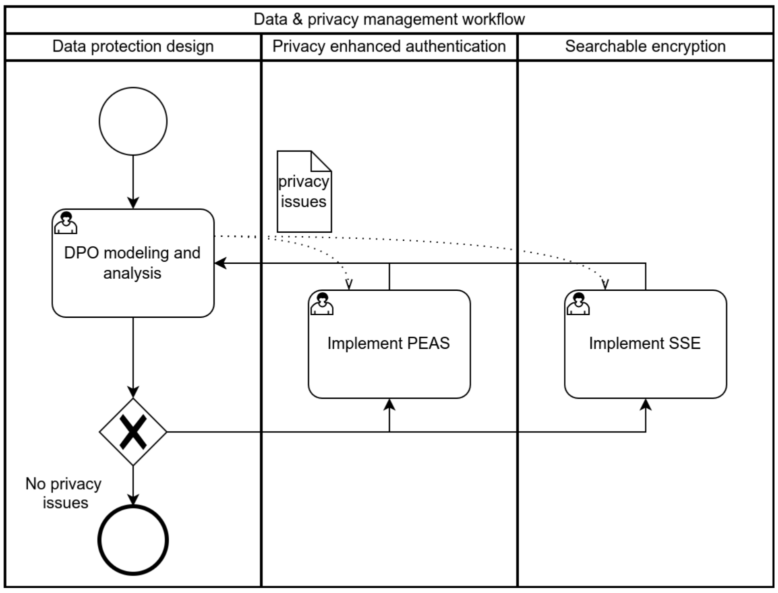


Figure 2.4: Data & privacy management workflow

Figure 2.4 shows an example of security workflow for designing and refining privacy management. Briefly, the workflow starts from a specification of the system behavior and its data management logic. An automatic analysis is then carried out to spot out potential issues in the way data is handled w.r.t., the GDPR definitions and rules. If some issues are found by a Data Protection Officer (a DPO tool), privacy-enhancing technologies can be adopted. For instance, a privacy enhancing authentication system (PEAS) can be considered to strengthen the authentication process. Similarly, searchable encryption (SSE) can be included to prevent data disclosure during the evaluation of database queries. Full details about this workflow and its components are given in Chapter 5, the demonstration of the developed tools is presented in Section 6.3 and its evaluation in Section 7.4.Full details about this workflow and its components are given in Section 5, the demonstration of the developed tools is presented in Section 6.3 and its evaluation in Section 7.4.

2.3 Workflow implementation

The implementation of the workflow engine is a non trivial task. As a matter of fact, a generic workflow may need to be integrated with any relevant aspect of the II life-cycle. For instance, consider again the legacy technology management workflow discussed above. Since its tasks may occur during different phases of the lyfe-cycle, its implementation may require particular attention. Below we briefly discuss two implementation strategies that allow for the implementation of security workflows in the HAII-T.

2.3.1 TOSCA workflows

For implementing the security workflows we rely on TOSCA built-in functionalities. In particular, TOSCA allows for the definition of two types workflows, i.e., *declarative* and *imperative* workflows.[1] Both of them are meant to manage and modify the TOSCA topologies. For instance, they allow for deploying, dynamically reconfiguring and deleating TOSCA nodes from a TOSCA topology. The main difference is that declarative workflows are automatically generated by the TOSCA orchestrator based on topology items (e.g., node, relationships and attributes), while imperative workflows are manually specified.

To provide a concrete example, consider the following declarative workflow.

```
topology_template:
  workflows:
```

[1] https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.html#_Toc26969478

```
    install:
      description: Installing and starting an application on a node
      steps:
        Install:
          target: compute
          activities:
            - delegate: install
          on-success:
            - Configure
        Configure:
          target: compute
          activities:
            - set_state: configuring
            - call_operation: tosca.interfaces.node.lifecycle.Standard.configure
            - set_state: configured
          on-success:
            - Initialize_backend
            - Initialize_frontend
        Initialize_backend:
          node: backend
            activities:
            - set_state: initializing_backend
            - call_operation: tosca.interfaces.node.lifecycle.Standard.create
            - set_state: starting_backend
            - call_operation: tosca.interfaces.node.lifecycle.Standard.start
            - set_state: backend_ready
        Initialize_frontend:
          node: frontend
            activities:
            - set_state: initializing_frontend
            - call_operation: tosca.interfaces.node.lifecycle.Standard.create
            - set_state: starting_frontend
            - call_operation: tosca.interfaces.node.lifecycle.Standard.start
            - set_state: frontend_ready
```

Briefly, the workflow consists of four tasks (steps), i.e., `Install`, `Configure`, `Initialize_backend` and `Initialize_frontend`. The workflow execution starts from the initial step which is responsible for instantiating the compute node where the application will reside. This is achieved by delegating the operation the the install workflow defined by the compute node. When this step is completed (`on-success`), the configuration task is launched. This task uses the `call_operation` directive to trigger the execution of an operation defined by a TOSCA interface. Then, two tasks occur in parallel, i.e., `Initialize_frontend` and `Initialize_backend`. Since none of these two declare successor tasks, the workflow terminates upon their completion.

In terms of HAII-T implementation, TOSCA workflow have several advantages. In particular, they are directly integrated in the II topology specificiation and they can combine, i.e., via delegation, different functionalities provided by different nodes. Nevertheless, they also have limitations, in particular, TOSCA workflows are bounded to a specific topology and its elements. However, in general we can expect that some security workflows also include tasks that cannot be mapped to some node, e.g., think of workflows having humans in the loop.

2.3.2 Custom workflows

Custom workflows extend the native TOSCA workflows in order to surpass the limittions discussed above. Custom workflows are implemented by means of external programs and scripts which rely on orchestrator APIs to interact with the TOSCA topology and the II.

To better highlight the behavior of custom workflows, consider the following (abstract) custom workflow implemented by means of makefile.

```
.PHONY: sparta-compile
sparta-compile:
  echo "Building␣and␣processing␣a␣model"
  cat /sparta/topology.yml
  | docker run -v /sparta/inputs.yml:/inputs.yml puccini compile --inputs=/inputs.yml
  | python /sparta/python/script.py -d /tmp/sparta/ output.db process
  echo "Done"
```

The makefile is responsible for carrying out two tasks, i.e., building a model by extracting and combining fragments as well as other details from a target topology. Such a structure is very common and many techniques share it. For instance, formal verification tools often rely on a formal specification, e.g., a behavioral model of the target system. The first task in the example above is carried out by means of docker container running *Puccini*,[2] i.e., a TOSCA processor meant to parse TOSCA topology specifications. Then, the second task applies a python script (`script.py`) to the output of the previous step and, eventually, returns an output file (`output.db`).

Both the implementation strategies described above, i.e., TOSCA workflows and custom workflows, have been applied during the implementation of the security workflows presented in this document.

[2] https://puccini.cloud/

Chapter 3 Legacy technologies management

3.1 Secure software updates for wearable low-power IoT with RIOT, SUIT, and femto-containers

With the emergence of COVID-19, the need for contact tracing in various places, in particular in the work environment, has become a necessity. Contact tracing solutions have been quickly designed, developed and massively deployed over the recent months leveraging smartphones applications and Bluetooth radio scanning.

In the context of this use case, we design, implement and validate a prototype for contact tracing using low-power IoT wearable devices, embedding open source software that is securely updatable over the network e.g. to fix bugs and vulnerabilities, or to adapt to modified sanitary rules and new precautions, based on new knowledge on a developing epidemics.

3.1.1 Related work on Contact Tracing

Such solutions include DP3T [5] (implemented in GAEN [45], the dominant solution), or ROBERT [27] (implemented in TousAntiCovid [79], used in France). As debates and controversies flared concerning the downsides of their design (centralized versus decentralized, impact on privacy etc. see for instance [45, 82, 83]) alternatives are being explored.

One avenue aims to offer better privacy guarantees by exploring alternative secure multi-party computation paradigms e.g. Private Set Intersection protocols [1].

Another avenue aims to modify the contact logging protocol. For example, DESIRE [28] is an alternative approach based on a hybrid mechanism logging cryptographically-generated encounter IDs instead of logging user IDs, which aims to garantee more privacy-by-design. Beyond designing and implementing algorithms, an important factor is the open/close source nature of this implementation and of the software platform it is based upon. So far no solution is fully open source, although this would improve transparency and may offer more opportunities for researchers to modify/optimize the embedded software and algorithms across the stack.

Yet another trend intends to modify the hardware used for contact tracing. For example, a cheap physical token could be used instead of a smartphone, such as experimented with in Singapore with TraceTogether [9]. Studies such as [58] also show that Bluetooth, as used in this context, tends to not provide distance estimations that are solid and reliable enough. Therefore complementary radio thechniques are explored.

We have contributed to this space by designing PEPPER, a prototype and a fully open source platform which enables experimenting with low-power IoT wearables and the full software stack in this context, using commodity (disposable-like) physical tokens for contact tracing. We implemented and demonstrated PEPPER as described below.

3.1.2 Architecture for Low-power Contact Tracing

Broadly speaking, contact tracing solutions are composed with the below building blocks and steps.

- 1. Enrollment: how a user initially registers his identity and his contact tracing device to the contact tracing system;
- 2. Proximity discovery: how and what information on potential contacts is collected;
- 3. Contact registration: how an encounter is deemed relevant and what needs to be logged;
- 4. Exposure evaluation: how logged information is used to determine exposure;
- 5. Exposure declaration: how and what information is upstreamed by the user's contact tracing device;

[1]OpenMined Covid Alert PSI code. Online: https://github.com/OpenMined/covid-alert

- 6. Exposure notification: how and what warning information reaches users of the contact tracing system;

In our preliminary work [35] we layed to ground for a fully open source platform based on RIOT to experiment with low-power IoT wearables, or for proximity detection combining Bluetooth Low-Energy (BLE) and Ultra-Wide Band (UWB) for above steps 2, 3 and 4. We use this base for our demonstration as described below.

3.1.3 Femto-Containers for Low-power Business Logic DevOps

Our preliminary observation was that firmware updates and typical DevOps tools are inadequate for the development, deployment and maintenance of low-power business logic, such as logic dedicated to Step 3 and 4 (see above 3.1.2). Indeed, firmware updates in these cases are both highly inefficient in terms of network transfer congestion and energy consumption.

We have thus designed Femto-Containers, a new architecture which enables containerization, virtualization and secure deployment of software modules embedded on microcontrollers over low-power networks.

As proof-of-concept, we implemented and evaluated Femto-Containers on popular microcontroller architectures (Arm Cortex-M, ESP32 and RISC-V), using eBPF virtualization [88], and RIOT as host operating system. We show in [89] that Femto-Containers can virtualize and isolate multiple software modules, executed concurrently, with very small memory footprint overhead (below 10%) and very small startup time (tens of microseconds) compared to native code execution.

We published Femto-containers open source implementations and tutorials [12]. We use femto-containers as building block for our demonstration as described below.

3.1.4 PEPPER Prototype: IoT Hardware and Software

We have designed PEPPER, a prototype for low-power IoT wearables running embedded software that can be securely updated over the network. We extended our prior work on SUIT-based secure firmware updates [8, 90] to support SUIT-based Femto-Containers secure updates over low-power networks.

From the embedded software and network point of view, we have implemented the PEPPER prototype using RIOT, extended and combined with

- hosting of a femto-container (as desribed in [89]),
- updatability of either the full firmware or only the femto-container over the low-power network, using SUIT end-to-end security,
- network stacks wich scan BLE neighborhood and simultaneously, upon detecting an encounter, perform UWB precise Two-Way Ranging (as described in [35]),
- accommodating various business logic for step 4 (see above 3.1.2) to evaluate exposure.

From the embedded hardware point of view, we have designed the PEPPER prototype based on a DWM1001 Decawave board shown in Fig. 3.1, to implement a low-power adaptation of DESIRE [28] contact tracing. The prototype includes wearable casing, as shown in Fig. 3.2, which we leveraged in the demo described below.

Note that while the current hardware prototypes are rather ”bulky”, the same functionality (software, network, business logic) could theoretically run on some much smaller form factor similar to the Apple AirTag for instance (i.e. roughly the size of coin).

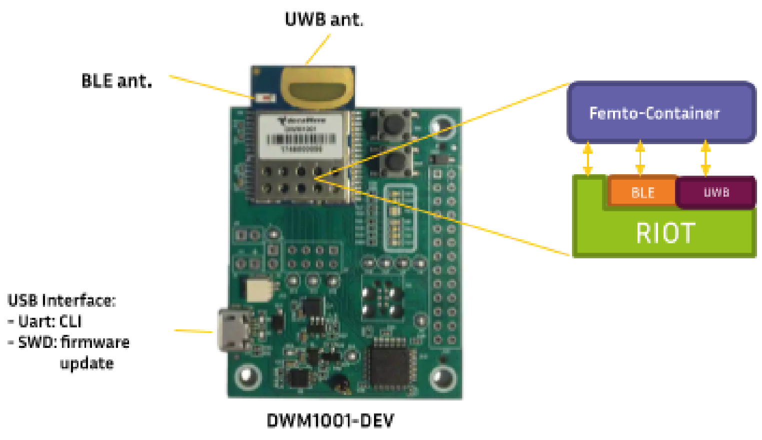


Figure 3.1: PEPPER contact tracing hardware and software design.

Figure 3.2: PEPPER prototype.

3.2 Software instrumentation for control-flow integrity on edge devices

To provide significant performance in terms of real-time execution, simplicity and low power consumption, IoT applications are developed over low-cost devices, where the code is often directly executed from the Flash memory with no middleware or supervisor layer. Furthermore, for optimization reasons, C or C++ languages are widely chosen to program these devices. Still in 2021, IEEE Spectrum rates them as the second and third most used languages in this domain [6]. C and C++ have the advantage of allowing direct memory management to optimize its usage. Although, right this feature may lead to common programming errors, causing problems such as memory leakage [11] or the famous buffer overflow [10].

Malicious opponents can exploit this kind of weaknesses to launch very powerful attacks at the binary level, such as Code-Reuse Attacks (CRA) [4] [73] [23], which are able to redirect the flow of execution through groups of instructions already present in memory, in order to obtain a malevolent execution with full expressiveness [80].

Control-Flow Integrity (CFI) has been investigated as the final defense solution [13]. The basic idea is to statically get the Control-Flow Graph (CFG) of the application before execution, and then instrument an online monitor able to guarantee compliance with the CFG at runtime, i.e., to prevent the program from performing different branches from those originally expected. The original idea to implement such a monitor is to insert, within the executable code itself, additional pieces of code at branches sites that, through a system of coupled unique labels, are able to perform an authenticate the branch with respect to the CFG [13].

Provided that any instrumentation or addition must have as little impact as possible from the point of view of the required time and/or energy due to the nature of the considered devices, a good compromise would be reached if the branch authentications were optimized and limited to the points of the program where the risk of control-flow corruption is really concrete. In fact, not all flow transfers require validation, but only those whose destination is computed with data that has passed through a data memory area at risk of corruption [64]. In addition, given the legacy scenario, it is good that such instrumentation can take place at the binary level, without requiring the software creator to recompile the executable.

The technique developed by us meets all the specified requirements. It is able to recompose the CFG of the application, to identify the flow transfers that match a predefined risk pattern, and to apply a binary instrumentation that is not fixed, but depending on the control-flow monitoring mechanism adopted [40].

3.2.1 Protection features

Within the binary application, the engine that we developed is able to distinguish 7 categories (*types*) of critical points for branch protection:

1. **Forward insecure branches with single target**: protected through the generation of 2 unique labels (hashes), one for the source and one for the destination of the branch, which are verified by the appropriate instrumentation code immediately before the execution of the branch and immediately after it, at the intended destination location;
2. **Backward insecure branches with single target**: same as above;
3. **Forward insecure branches with multiple targets**: same as the case of single target, but here all target locations are instrumented;
4. **Forward secure branches to a routine ending with a backward insecure branch with multiple targets**: this transfer is not to be protected, but the hash of the site to which the called routine must return is stacked in a proper protected region, depending on the adopted monitoring system;
5. **Backward insecure branches with multiple targets**: same as (2), but the hash of the target site must correspond to the hash stacked as described in (4);
6. **Forward insecure branches to a routine ending with a backward insecure branch with single target**: again, as in (4), the return site hash is stacked, but also the hash of the forward target site is verified, to ensure both caller identity at return time and validity of destination of the present call;
7. **Forward insecure branches to a routine ending with a backward insecure branch with multiple targets**: same as above, but here all possible return sites are instrumented.

Branch protection alone is not sufficient to cover all cases in which the execution flow of a control software running on endpoint devices can be maliciously hijacked. In fact, in undefined moments of the execution, a processor can jump to execute special routines to serve interrupt requests (Interrupt Service Routines, ISRs). Actually there is no static analysis that can forecast in which order (or where in the code) these routines will be called, so they can never be part of a statically-extracted CFG. Yet, the ISRs are full-fledged routines, which operate on data and registers and which preserve the current program status moving it into memory. Not considering also the protection of the context when switching to these routines results in hard risks [65].

For these reasons, our engine also identifies 2 additional categories of critical points for context protection:

1. **Entry points of an Interrupt Service Routine (ISR)**: all the Callee-Saved Registers (CSR), that are the registers stacked and then used by the routine to execute its task, plus all the registers automatically stacked by the architecture (e.g., in case of ARM, `R0`, `R1`, `R2`, `R3`, `R12`, `LR`, `PC` and the status register `xPSR`), must be suitably stacked in a dedicated portion under monitor control;
2. **Exit points of an Interrupt Service Routine (ISR)**: here, the instrumentation to be inserted concerns the integrity check of all the previously-mentioned registers, i.e., their content at the

end of the routine must be compliant with the content saved in the specific structure of the monitor.

3.2.2 Protection algorithm

In the following, it is described the formal rule followed to apply the protection.

Let P be the program to be instrumented, with C the set of its instruction locations and X the set of its data locations. X represents the union of X_k (set of constant memory locations), X_{nk} (set of non-constant memory locations), R (set of CPU registers), I (set of virtual input locations) and O (set of virtual output locations). The set $\Sigma_c \subset X \cup C$ is the set of source operands of the instruction contained in c[2].

The *destination function* is a function $\theta : X \setminus O \mapsto X \setminus (I \cup X_k)$ such that $y = \theta(c)$ is the destination operand of the instruction contained in c. A *control-flow transfer instruction* is an instruction $c_0 \in C$ such that $\theta(c_0)$ is the program counter register ($\mathtt{PC}$) of the CPU. A *basic block* BB is an ordered set of $n \in \mathbb{N}^+$ unique instruction locations $c_i \in C$, with $i \in [1, n]$, where c_n is a control-flow transfer instruction. The *entry point* of BB, (BB), is defined as c_1 and the *exit point* of BB, (BB), is defined as c_n.

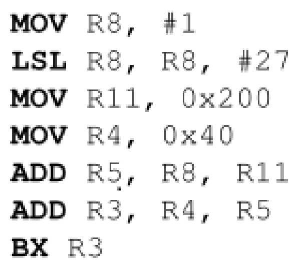
The Control-Flow Graph of P is formally defined as a directed graph $G = (V, E)$ where V is the set of basic blocks of the program P, and E is a set of ordered pairs $(BB_i, BB_j) = e_{ij}$, with $BB_i, BB_j \in V$, such that the location $\theta((BB_i))$ contains a value expressing the location (BB_j). In such a case, (BB_i) can be referred to as (e_{ij}), and (BB_j) can be referred to as (e_{ij}).

The edges of graph G can be further split into *direct edges* and *indirect edges*. A direct edge is an edge $e_{ij} = (BB_i, BB_j)$ such that the value expressing the location (BB_j) is stored in $X_k \cup C$, i.e., through a constant or an immediate. On the contrary, an indirect edge is such that the value expressing the location (BB_j) is stored in $R \cup X_{nk}$, i.e., in a register or data memory location.

If an edge represents a branch towards a constant location, then it is secure by definition, and there is no need to do any enforcement for it. The edges to be protected are therefore the indirect ones, but not all of them. In fact, even if the argument of the control-flow transfer instruction is variable, it can be composed of a combination of constants. The probability of this case certainly increases in embedded applications, where the code is usually all present in Flash memory since the beginning, and there is no dynamic linking. It is then reasonably assumed that *no input can contribute to forming a pointer to the code*: in the worst case, a value generated at runtime can at most act as a selector for a series of targets (e.g., function entry points), already predefined and constant. In light of this, the only indirect edges at risk of hijacking are the edges whose target is determined even only partially using data that has passed through the data memory, where some vulnerability may be present.

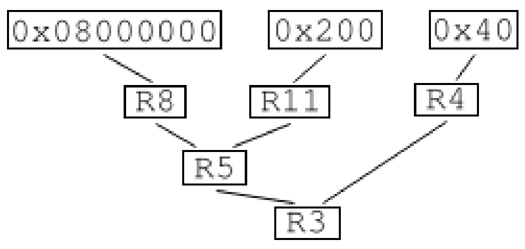
Therefore, to protect an indirect edge, it is necessary to reconstruct the origin of the value expressing its target. The value is found by plotting the *origin tree* relative to the branch target. The origin tree relative to a control-flow transfer instruction c originating an edge $e_{ij} = (BB_i, BB_j)$ is a tree $\Gamma_c = (X, \delta)$ defined over the set X, where the root is the location containing the value of (BB_j), and the child function is the function $\delta : (X \cup C) \setminus O \mapsto X \setminus (I \cup X_k)$ such that $y_0 = \delta(x_0)$ if $y_0 \in \Sigma_q$, for $q \in C : \theta(q) = x_0$. y_0 is a leaf for Γ_c if $y_0 \in I \cup X_k$.

[2]Σ_c contains c itself if an immediate operand is present.



```
MOV R8, #1
LSL R8, R8, #27
MOV R11, 0x200
MOV R4, 0x40
ADD R5, R8, R11
ADD R3, R4, R5
BX R3
```

(a)



(b)

Figure 3.3: A snippet of ARM-like Assembly language (a) and the origin tree Γ_c of $c =$ BX R3 (b) [40].

Figure 3.3 shows an origin tree example, relative to the ARM Assembly instruction BX R3. The content of R3 is determined by the sum of R4 and R5. In turn, R4 is an immediate. R5 is instead computed as the sum of R8 and R11; while R11 is an immediate too, R8 is computed as the left shift of the immediate 1, of 27 positions, i.e., leading to the hexadecimal value 0x08000000. In conclusion, R3 hosts the value 0x08000240. As it is possible to notice, the complete history of R3 has been reconstructed, and there is no value contributing to R3 which comes from data memory regions at risk of corruption. In other words, even if BX R3 is an indirect edge, which can be considered as insecure, it is completely secure, as no corruption is possible on the value finally stored in R3.

In light of this, using the formalisms defined so far, an indirect edge $e_{ij} = (BB_i, BB_j)$ originating from an instruction c is said to be secure iff

$$\nexists x \in \Gamma_c : x \in X_{nk} \tag{3.1}$$

All other edges are considered conservatively insecure, and therefore instrumented in light of what listed in the previous subsection.

As final view, the developed engine takes as input:

1. an executable binary file containing the application to be protected;
2. the group of instructions to be added before the edge source instruction, possibly parametric with respect to the position;
3. the group of instructions to be added before the edge target instruction, possibly parametric with respect to the position.

The returned outputs are:

1. a binary executable instrumented in insecure sites;
2. a complete list of insecure sites, to be supplied to the control-flow monitor to implement the protection.

Parsing
P
Extraction
E
Reconstruction
R
Recognition
R
Instrumentation
I

Figure 3.4**:** The 5 stages of the CFI instrumentation engine.

After a complete disassembly of the source binary, the engine develops through 5 stages (Figure 3.4):

1. *Parsing*: it groups all disassembled instructions of all application functions by translating it into an overall big Assembly source file;
2. *Extraction*: it outlines the general program stream exploring all the performed function calls and retrieves the CFG of those functions that include indirect edges;
3. *Reconstruction*: it is responsible for determining the storing locations where indirect branch operands transited (registers, non-constant data memory locations), tracing their history, and therefore collecting the instructions that contributed to the value;
4. *Recognition*: it identifies the edges and classifies them based on specifications;
5. *Instrumentation*: it applies the instrumentation statements based on the discovered edge type.

3.3 Protocol verification

This chapter presents a methodology of IoT Protocols formal verification, as part of Legacy technologies workflow. The demonstrator described below presents an update of demonstrator described in SPARTA Deliverable 6.3 (phase 1), and includes additional scenarios as part of demonstrator phase 2. This document, in order to present all aspects and contributions of the protocol verification demonstrator, will include a description of methodology and both phase 1 and phase 2 demonstrator scenarios description.

Protocol verification demonstration consists of two parts, with two phases in each of them:

1. **Model checking** – formal modeling of IoT protocols and security relevant parts of protocols
    - Phase 1 – Formal modeling of the unidirectional teach-in procedure in EnOcean protocol
    - Phase 2 – Formal modeling of the high security authentication process in EnOcean protocol
2. **Probabilistic model checking** – IoT network risk analysis based on probabilistic model checking and threat modeling

- Phase 1 – Risk analysis of hijacking of smart HVAC system in smart home environment
- Phase 2 – Risk analysis of hijacking of smart meter in smart home environment.

As stated in previous version of this deliverable, the general goal of this research is to review both the methods of formal verification and their practical applications within IoT and Intelligent Infrastructures domains.

A smart home environment, as part of SPARTA WP6 common use case, is selected in order to demonstrate an application of different security-by-design, and especially formal modeling, approaches and tools. Modeled environment consists of 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 (using EnOcean protocol), smart meter, access point (gateway), smart/wearable user devices, power distributor service, responsible actor/s.

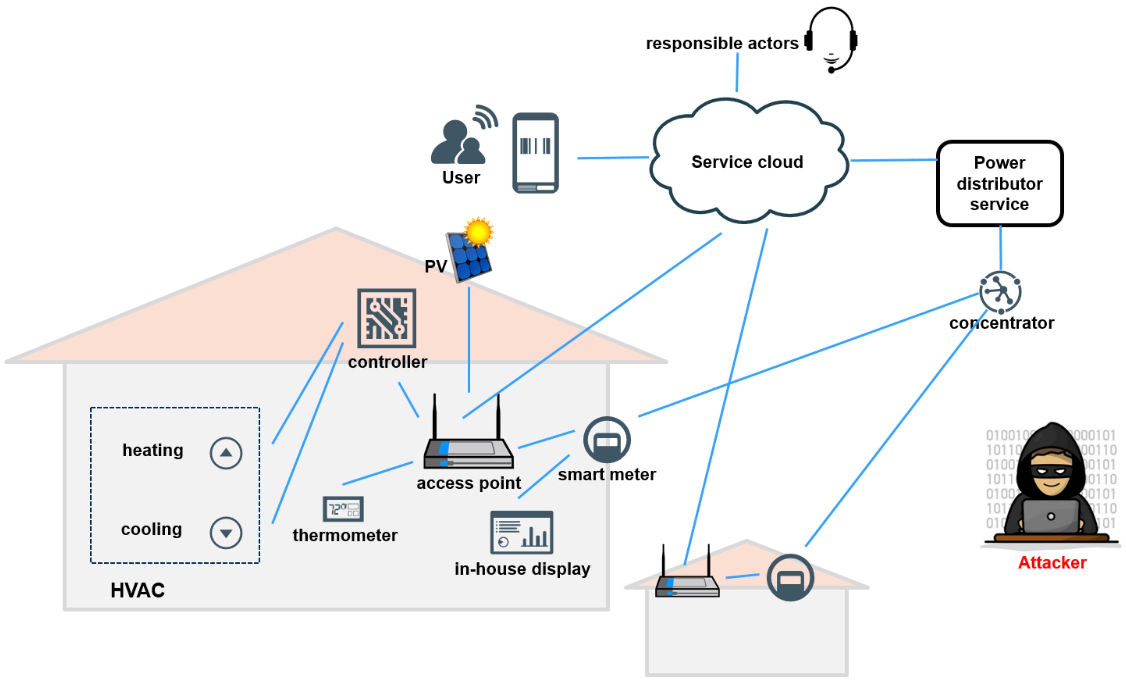


Figure 3.5: Protocol verification smart home environment

Model checking, and more specifically formal verification of protocols, presents 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 commonly used model checkers that focus on security protocols checking are AVISPA, ProVerif, Scyther and Tamarin. In our analysis described in this document, the tool ProVerif[3] is used, which is widely used in literature, e.g. for the Needham-Schröder public-key protocol and corrected versions modeling [60, 69], for studying TLS [19], for the verification of Signal [53] and the JFK protocol [15], etc. It is also used for the investigation of IoT protocols, especially for Bluetooth [30] and for 5G [32, 91].

Probabilistic model checking is one of the methods to detect weaknesses and possible vulnerabilities at an early stage of system architecture design. Most commonly used probabilistic formal verification tools, that found their application in risk analysis, are PRISM and UPPAAL, with the STORM as the most recent one. Our analysis is based on PRISM[4], the most commonly used probabilistic formal modeling tool, with a wide area of application domains including wireless communication protocols,

[3]https://prosecco.gforge.inria.fr/personal/bblanche/proverif/
[4]https://www.prismmodelchecker.org/

quantum cryptography and systems biology.

3.3.1 Model checking – formal modeling of IoT protocols

First demonstrator presents a formal modeling of IoT protocols based on model checking. Security aspects of IoT protocols used in the Smart Home Domain (Bluetooth Low Energy, ZigBee, Z-Wave, KNX-RF, Thread and EnOcean) are covered in several publications [51, 62]. While there are many publications focusing on different security aspects for the protocols Bluetooth, ZigBee, Z-Wave and Threat, the situation is different for EnOcean. The authors in [62] explicitly state that the security of EnOcean protocol was not analyzed in the scientific literature. Our state-of-the-art analysis at the beginning of the Sparta project confirmed that conclusion. Considering that EnOcean protocol is a part of numerous Smart Home applications world wide, Sparta HAII-T program demonstrator includes formal analysis of this protocol, in order to discover potential design-embedded weaknesses. This demonstrator is implemented in two phases – phase 1 that includes EnOcean unidirectional teach-in formal modeling, and phase 2 that includes bidirectional teach-in and authentication.

3.3.1.1 Methodology

A methodology for formal modeling of IoT protocols is presented on the Figure 3.6, in the form of sequential flow chart. 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). The output of the formal verification process is most commonly 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.

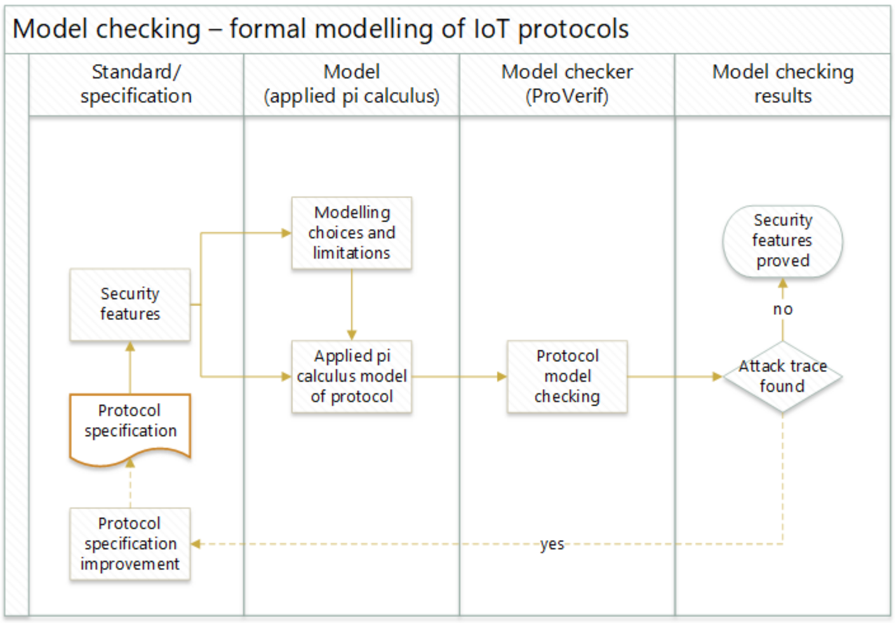

Model checking – formal modelling of IoT protocols
Standard/ specification
Model (applied pi calculus)
Model checker (ProVerif)
Model checking results
Modelling choices and limitations
Security features
Applied pi calculus model of protocol
Protocol model checking
Security features proved
no
Attack trace found
Protocol specification
Protocol specification improvement
yes

Figure 3.6: Model checking flow chart

3.3.1.2 Protocol specification

EnOcean is an IoT protocol developed by the EnOcean Alliance. Although it was patented in 2001, it became an international standard in 2012, namely $ISO/IEC 14543-3-10$. The standard has been revised in March 2020 [48]. It's signal range is specified as up to 100 meters in free-field and up to 30 meters inside a building. It is optimized for battery-less ultra-low power devices and energy harvesting applications in building and home automation, and it uses a wireless power supply. The EnOcean standard involves the two lower layers of the TCP/IP model, the *Network Access* and the *Internet*.

In order to initiate a communication between wireless devices in a radio network, a *teach-in process* has to be performed in order to define which transmitter/s have to listen to a receiver [48, 51]. The teach-in message can be sent in plain text or by using a pre-shared key.

EnOcean provides encryption, authentication, integrity checking and replay protection [38, 62]. The protocol- and implementation-related security issues of EnOcean protocol are summarized bellow:

- **Encryption:** There are two options, namely AES-CBC and variable AES (VAES). Due to the constant initialization vector used in the EnOcean specification, which is regarded as insecure, using AES-CBC is not recommended.
- **Authentication and integrity checking:** AES-CMAC is used for this. In case there are no all-zero payloads and the AES key is changed after at most 2^{48} messages this is considered secure.
- **Replay protection:** The RLC can be used in conjunction with the CMAC for a replay protection

Marksteiner et al. [62], based on previous findings, recommend to use a nonce-starting RLC, VAES for encryption and CMAC for authentication and integrity checking.

3.3.1.3 Protocol modeling

We used applied pi calculus, a language introduced in [14], for the formal modeling of EnOcean. This language is especially useful for the analysis of security protocols. Applied pi calculus are very useful for describing concurrent processes and their interactions. A presentation of the language in a tutorial style is given in [74].

For modeling choices and limitations the Dolev-Yao Model [36] is used. It assumes that a protocol interacts with public communication channels (internet), where a potential attacker might have full control over the communication channel. Therefore, an attacker might be able to read, modify, delete or inject messages. In case the attacker is able to intercepts the key, the attacker is also able to decrypt a message. However, an attacker is not able to decrypt any message without obtaining the corresponding key, since cryptography primitives are assumed to be perfect. Our model focuses on EnOcean High Security [38] which aims to enable a protection against special man-in-the-middle attacks and a more flexible handling for devices. Due to the selected scenario the focus is on a pre-shared key exchange in a secure way (and not via plain text). In this scenario, the key and the RLC are sent encrypted. However, several parameters are sent in plain text. According to [38] the nonce for the authentication represents the challenge and is therefore exchanged via air interface.

According to the security specification [38], the unidirectional authentication with high security settings is specifically intended for (unidirectional) data flow applications, where one device is energy autonomic (device A) and the other one is line-powered (as device B, acting as gateway).

On the other hand, the security specification states that use cases for bidirectional communication include bidirectional data flow where both devices are line-powered and such where one device is energy autonomic and the other one line-powered.

More details about this part of the formal modeling can be found in our paper [46].

3.3.2 Probabilistic model checking – risk analysis in IoT environment

Second demonstrator presents a risk analysis of smart home IoT network based on probabilistic model checking. As previously stated probabilistic model checking is one of the methods to detect weaknesses and possible vulnerabilities at an early stage of system architecture design. The general goal of this demonstrator is to test a practical application of probabilistic model checking methodology within IoT and Intelligent Infrastructures domains, by analyzing parts of Sparta HAII-T common use case. The obtained results provide insight into potential threats and the likeliness of successful attacks.

This use scenario focuses on a probabilistic risk analysis of two different smart home system configurations through threat modeling and model checking.

3.3.2.1 Methodology

The methodology for the application of formal methods to risk analysis with PRISM model checker is shown in the Figure 3.7, in the form of sequential processes. The first step is the definition of the use case sub-system architecture, its components and communication channels. The system architecture is used for the threat-modeling step, which as a result provides a list of threats. The vulnerabilities of the system and the attack possibilities are identified based on the threat list. Next, an efficient means of calculating the exploitation probabilities for the identified vulnerabilities is implemented and the attacker's behavior thus modeled. The formal system model is created based on the system architecture, the identified vulnerabilities with exploitation probabilities, and the modeled attacks. The formal properties of the attacks are identified next, and the model is checked against the identified properties using the model checker. This finally results in risk exposure scores.

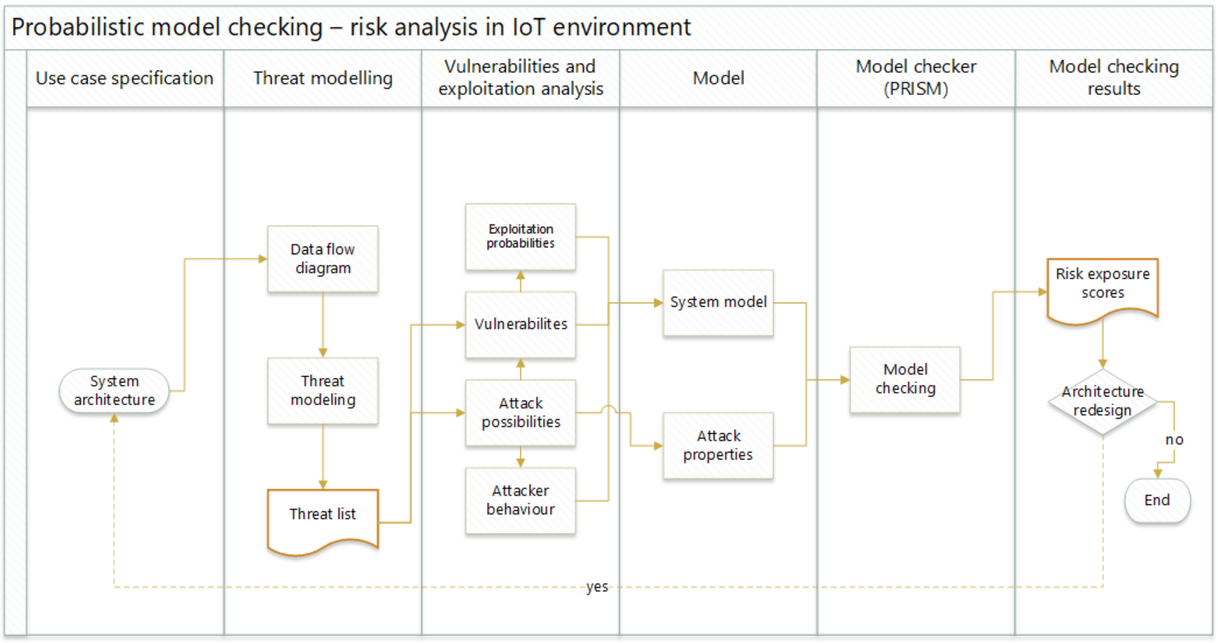


Figure 3.7: Probabilistic model checking flow chart

3.3.2.2 Use case specification

As previously stated, this demo was implemented in two phases – phase 1 includes modeling of smart HVAC system hijacking attack within smart home environment, and phase 2 includes modeling of smart meter hijacking 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.

3.3.2.3 Threat modeling

A purpose of threat modeling is the identification of threats and vulnerabilities within IT-related system architectures [77, 81]. It also helps to put security and privacy by design into practice. As part of risk analysis process presented in this chapter, a threat modeling approach is meant to secure a project setup with a systematic security analysis. The model was created by using the Microsoft Threat Modeling Tool [3], which works on data flow diagrams (DFD) that describes data stores, processes and communication lines and provides threats based on the STRIDE model [47].

The Microsoft Threat Modeling Tool is not limited to a set of threats but offers the possibility to create individual templates for a given domain. Furthermore, we rely on the Azure cloud and IoT templates from Microsoft for the smart home area. We combine these templates with our own, which are based on our research in the smart energy domain. In the model itself, different trust zones were identified according to our use case. First, the trust zone of components is identified within the smart home, then in the outsourced cloud area and the immediate personal area. It has to be emphasized that modeled DFD of smart home environment is modeled as a part of a large smart grid environment, and takes into consideration also back-end components of a smart grid system in detail, which results in a high number of discovered threats. As an illustration Figure3.8 depicts a part of our DFD that includes smart home environment components.

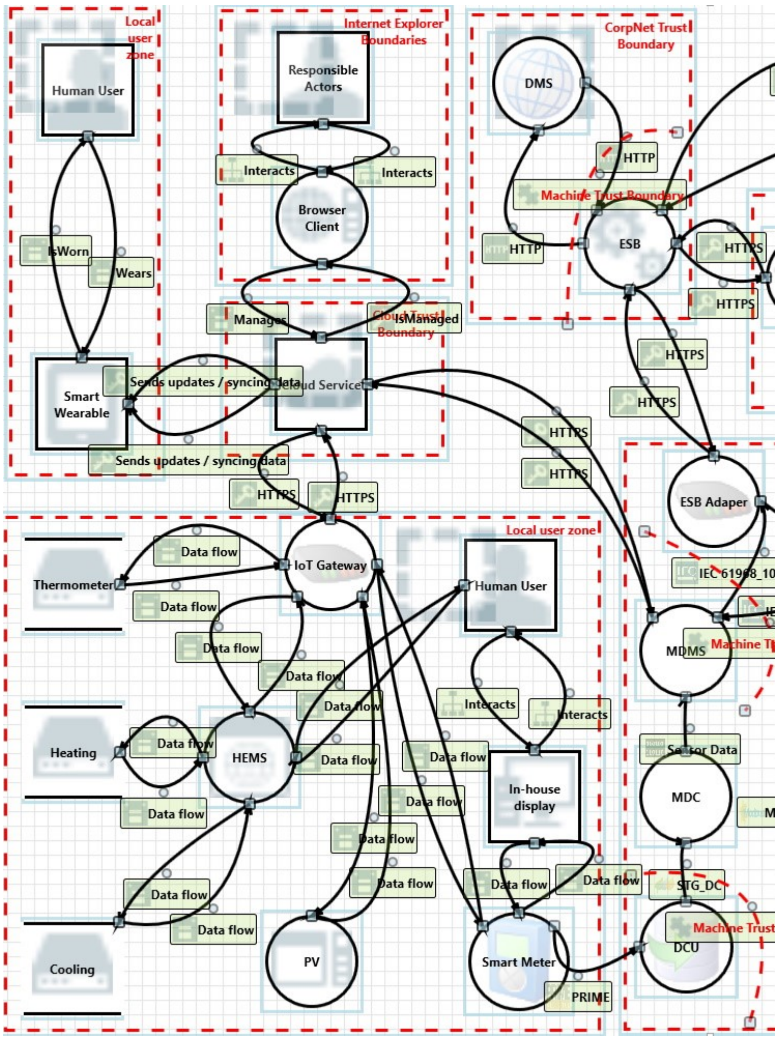


Figure 3.8: Threat model – Smart Home environment DFD [81]

In total, the modeling approach resulted in the identification of 1137 threats. These are classified according to STRIDE, with additional one that describes threats in smart home components. The latter also cover physical threats, which contribute to the overall number of threats. A detailed threat model based on our demonstration use case and resulting threat list are given in our paper Vallant et al. [81]. However, for the conducted assessment we rely just on the threats listed bellow, manually selected in accordance with considered scenarios and devices:

- Threat ID: 857 – Elevation by Changing the Execution Flow in DCU
  - STRIDE Category: Elevation Of Privilege
  - An attacker may pass data into DCU in order to change the flow of program execution within DCU to the attacker's choosing.
- Threat ID: 880 – Spoofing the In-house display External Entity
  - STRIDE Category: Spoofing
  - In-house display may be spoofed by an attacker and this may lead to unauthorized access to Smart Meter.
- Threat ID: 903 – An adversary may block access to the application or API hosted on In-house display through a denial of service attack
  - STRIDE Category: Denial Of Service
  - An adversary may block access to the application or API hosted on In-house display through a denial of service attack.
- Threat ID: 1005 – An adversary may gain elevated privileges and execute malicious code on HEMS host
  - STRIDE Category: Elevation Of Privilege
  - If an application runs under a high-privileged account, it may provide an opportunity for an adversary to gain elevated privileges and execute malicious code on host machines.
- Threat ID: 1013 – An adversary may execute unknown code on Heating/Cooling
  - STRIDE Category: Tampering
  - An adversary may launch malicious code into Heating and execute it.
- Threat ID: 1028 – Potential Excessive Resource Consumption for IoT Gateway or Thermometer
  - STRIDE Category: Denial Of Service
  - Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job.
- Threat ID: 1111 – Elevation by Changing the Execution Flow in IoT Gateway
  - STRIDE Category: Elevation Of Privilege
  - An attacker may pass data into IoT Gateway in order to change the flow of program execution within IoT Gateway to the attacker's choosing.

3.3.2.4 Vulnerabilities and exploitation analysis

Next step in risk analysis process is modeling of each attack scenario as a sequence of attack steps, which leads from an attacker to the exploitation of the system. Usually, every connection in the system is implemented using different technology and communication protocols. Because of this, each device in the network represents a distinctive security challenge for the attacker. From the outside, the system can be accessed either physically or via a web interface. It should be noted that advanced hacking skills and domain knowledge are mandatory in order to carry out the described attacks.

In the hijacking of smart HVAC system scenario the attacker successfully exploits the vulnerabilities in temperature regulation systems and their sensors. The ultimate target represents the HVAC system, which represents the last link in a chain of devices. The attacker must gain control over the internet system of the smart home. Then, the attacker must find a way to compromise the gateway of a single smart home. Afterwards, the attacker injects a payload to obtain operational access in the corresponding HEMS system. Finally, by controlling HEMS, instructions can be sent to the targeted HVAC system. The overall attack steps are presented in Figure 3.9. More details are available in our paper[81].

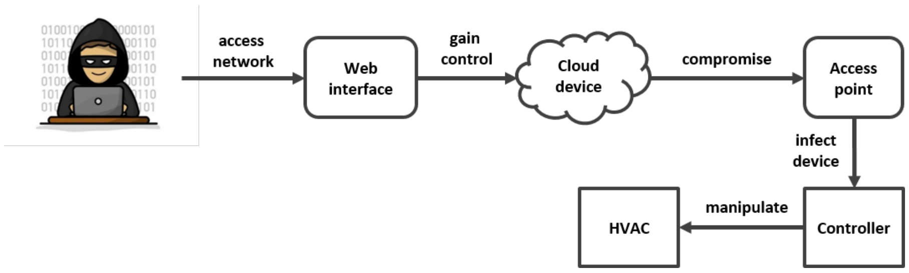


Figure 3.9: Attack scenario for hijacking of smart HVAC system[81]

In the hijacking of smart meter scenario an attacker exploits vulnerabilities in a household's smart meters. In our scenario, the attacker gains via the access point access to the smart meter. Afterwards, an attacker is able to interfere with the smart homes' energy consumption in different ways. The use case for this attack is shown in Figure 3.10.

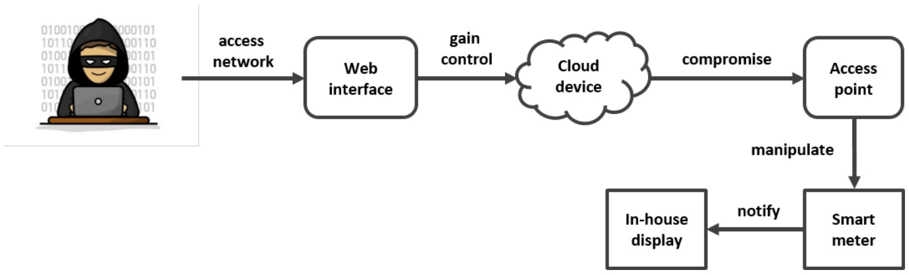


Figure 3.10: Attack scenario for hijacking of smart meter system[81]

An important step in this process is an assessment of the system components from the exploitation probability point of view. While exploitation probabilities in similar approaches are most commonly estimated using an extensive literature survey [55, 67], we attempted to formalize this process by using the CVSS tool [5][81].

In our approach, the CVSS tool is adopted for the calculation of exploitation probabilities, with certain methodology modifications compared to the existing approaches. The first step is a mapping of the components in use and their interrelation at the evaluated environment, based on the threats obtained by the threat modeling process of the specified use cases.

The CVSS tool have several modules available, where the base score module, according to Wadhawan et al. [85], provides a good basis for the calculation of the probability of an attack. Base score module have three sub-modules that can be used for calculating probabilities: exploitability, scope and impact sub-modules. Exploitability sub-module reflects how easily a vulnerability can be exploited, while the scope and impact sub-modules quantify the consequences of a successful exploit. In contrast to Wadhawan et al., our work takes into account, apart from exploitability of the submodule, also the core and impact sub-modules, because in penetrating the smart grid related systems, these impact-related parameters are also in the focus of an attacker and, therefore, should be part of the calculation. The exploitability sub-module takes into account:

- The attack vector – reflects the context by which a vulnerability can be exploited;
- The attack complexity – outlines how much effort in the preparation or execution of the attack against a vulnerable component the attacker have to invest;

[5]https://www.first.org/cvss/calculator/3.1

- Privileges required – denote the level of privileges an attacker must obtain in order to successfully attack a component;
- User interaction – if another human user must be involved for a successful attack.

The scope sub-module rates if a vulnerability in one asset affects other assets, which are outside of this security authority.

The impact sub-module takes into account the CIA Triade:

- Confidentiality – to what extent confidentiality is affected;
- Integrity – refers to the trustworthiness and correctness;
- Availability – rates the impact of a successful attack to the availability of the affected component.

For less complex attacks, the base metric score is higher because such an attack have a higher likelihood. The score obtained from the CVSS system in range 1–10, and is normalised to the range 0–1 for further calculations.

Selected threats to be modeled, and resulting exploitation probabilities are presented in Tables 3.1 and 3.2.

Table 3.1: Hijacking of smart HVAC system scenario – threats and exploitation probabilites [81]

Threat ID	Threat	Expl. Prob.
1005	An adversary may gain elevated privileges and execute malicious code on HEMS host	0,57
1013	An adversary may execute unknown code on Heating/Cooling	0,59
1028	Potential Excessive Resource Consumption for IoT Gateway or Thermometer	0,65
1111	Elevation by Changing the Execution Flow in IoT Gateway	0,59

Table 3.2: Hijacking of smart meter system scenario – threats and exploitation probabilities [81]

Threat ID	Threat	Expl. Prob.
857	Elevation by Changing the Execution Flow in DCU	0,57
880	Spoofing the In-house display External Entity	0,68
903	An adversary may block access to the application or API hosted on In-house display through a denial of service attack	0,65

3.3.2.5 Formal Modeling

In order to perform formal risk analysis and to obtain an indication on how safety and security requirements can be fulfilled within a given environment, selected model checker, PRISM, is applied on the two described use scenarios, and several example attack scenarios are modeled. It has to be noted that PRISM model checker requires manual modeling of the system under consideration. The system is modeled in the PRISM model checker as MDP (Markov Decision Process) because of the non-deterministic nature of cyber-attacks. In the modeling process it is assumed that an attacker actively attacks the system, and has skills and means to perform certain attacks by exploiting existing vulnerabilities. An attacker's skills are measured through the maximum number of vulnerabilities that he can try to exploit in one attack scenario—the cost value. For example, the attacker's skill level in range one to five (cost = (1 : 5)) means that a less skilled attacker is able to exploit only one vulnerability in one attack scenario, while a more skilled attacker can exploit up to five vulnerabilities at the same time in one attack scenario [81].

After the system modeling is completed, the next step is the definition of the attack properties. These properties are the formal definition and precondition of the successful attempt of an attack within the modeled system. The properties are defined by using the Probabilistic Computation Tree Logic (PCTL) [43], embedded in the PRISM model checker. The formal verification of the defined attack properties results in the maximum likelihoods of successful attack attempt – risk exposure scores. Modeled attack properties are described bellow.

Hijacking of smart HVAC system scenario:

- Attack scenario 1.1: Hijack heating – An attacker takes control over heating, high or optimal temperature is detected, heating is switched on, resulting in damage;
- Attack scenario 1.2: Hijack HVAC – An attacker takes control over HVAC, optimal temperature is detected, both cooling and heating are switched on, resulting in damage and high power consumption.

Hijacking of smart meter system scenario:

- Attack scenario 2.1: Fraud – A user takes control over his and neighbor's smart meter or concentrator, decreases his own power consumption, increases his neighbor's power consumption, responsible actors are not alerted;
- Attack scenario 2.2: Decrease bill – An attacker takes control over his own smart meter or concentrator, and decreases the power consumption;
- Attack scenario 2.3: Increase bill – An attacker takes control over the user's smart meter or concentrator, and increases the power consumption;
- Attack scenario 2.4: Increase bill, no alarm – An attacker takes control over the user's smart meter or concentrator, increases the power consumption, the user is not alerted.

3.4 RIOT-AKA: cellular-like authentication over IoT devices

In the Internet of Things (IoT) context, because of the devices' persistently connected status, secure communication and authentication take a crucial role. The ”usual” risks, attacks and threats are amplified and/or call for tailored defense and security measures [54]. When compared with the security of traditional devices, IoT security comes along with an extra caveat: things (might) move not only across environments but also across different owners. Note that such ”roaming” may occur also when things are static or even buried in the environment itself - imagine, for instance, a smart house which changes ownership: all smart objects deployed in the house are called to ”forget” the past data history, and be controlled by the new owner.

We concentrate on one specific aspect: authentication and key agreement. An authentication protocol devised to permit devices to roam through different domains, and to change domain ownership, must obviously guarantee that no long term secrets are deployed in the entity/agent which controls the smart environment.

3.4.1 Cellular Networks inspiration

The roaming problem has been widely addressed in cellular networks since their initial digital incarnation (i.e. the 2G/GSM system) and the current authentication mechanisms can be considered an incremental evolution of the original protocol. Despite the technical improvements the authentication and key agreement (AKA)[92] paradigm employed in cellular network has conceptually remained the same. It clearly decouples [41] i) a ”home” entity ultimately responsible of maintaining long term secrets and provide means to verify authenticity, from ii) a ”serving” entity (the visited domain) which is instead called to run the actual real-time authentication process, *without the need* to access long term secrets - it uses a bulk of authentication credentials opportunistically provided offline by the home server.

3.4.2 RIOT-AKA contribution

We design a prototype as a preliminary proof-of-concept implementation and demonstration in order to perform an experimental evaluation of our proposed authentication and key agreement protocol called ”RIOT-AKA”, on the RIOT IoT secure Operating System. Our protocol is designed to mimic as closely as possible the Authentication and Key Agreement (AKA) mechanisms implemented in 3G and beyond cellular networks, but using instead more common cryptography primitives (i.e. the off-the-shelf HMAC-SHA256 construction already available in RIOT) and different communication protocols (i.e. CoAP), since we do not have any backward compatibility requirement. As a side contribution, we further explore the possibility to manage long term secrets generated via a Physical Unclonable Function (PUF) exploiting the unique randomness of SRAM cells installed in IoT devices running RIOT OS as their Operating System, thus achieving the possibility to deploy and manage secrets which are never stored in the device memory. Since the premature state of our prototype, we remark that it is a preliminary version; some extra details (such as sequence number anonymization and generation of ephemeral encryption keys) are not yet supported and left to future implementation work.

3.4.3 RIOT-AKA Authentication Framework

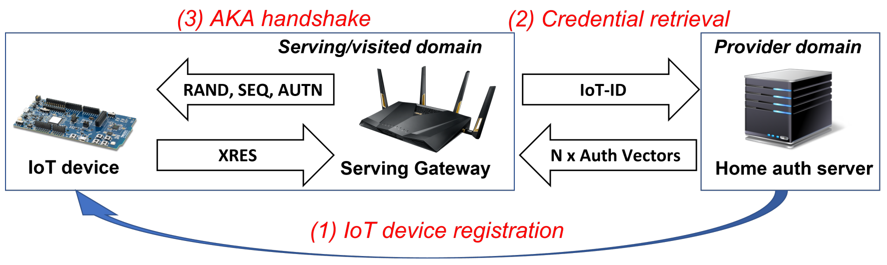


Figure 3.11: RIOT-AKA deployment framework

The RIOT-AKA authentication framework involves three entities: the IoT device, the ”serving” domain in which the device is deployed, and the ”provider” cloud domain. A gateway placed in the service domain acts as local authentication server using credentials retrieved offline from a provider's home authentication server. More specifically, our proposed scheme involves three phases (Fig. 3.11):
1) IoT device registration - the device is initially provided with a secret key K_0. Such secret key is normally issued by the provider's home authentication server, and therein registered along with the unique device identifier. As an alternative option, we further investigated the feasibility of issuing the secret K_0 via an SRAM Physically Unclonable Function (SRAM-PUF) made available by the RIOT OS.
2) Retrieval of authentication credentials by the serving gateway - this operation is *conceptually identical* to what happens in cellular networks, a part from the technical caveats concerning how such authentication credentials are specifically computed. Specifically, once the IoT device roams into - or is deployed - in an (eventually temporary) serving domain, the serving gateway retrieves a set of N authentication ”vectors”, being N a configuration parameter which defines the (maximum) number of authentications that can be performed by the IoT device before a new set of credentials shall be retrieved. This operation is repeated every time all the previously retrieved authentication credentials are consumed.
3) Run-time authentication and key agreement handshake - This is the online phase which occurs whenever the IoT device performs the actual authentication (or re-authentication). As in the case of cellular systems, also in our case the handshake performed over the air interface permits mutual authentication using only two messages, as illustrated in Figure 3.11. First, the gateway sends a message comprising i) a random challenge, ii) the sequence number of the current authentication, and ii) an authentication token (AUTN) whose role is to permit the IoT device to verify that the gateway is authentic (i.e. that it has received proper authentication credentials from the home authentication server). Then, the IoT device replies with a message XRES which is a proof of its authenticity.

3.4.3.1 Protocol design details

In details, every authentication vector retrieved by the serving Gateway contains five quantities:

- **RAND** = random challenge;
- **SEQ** = sequence number, used by the IoT device as implicit challenge for the network authentication (see next discussion), and used by the home authentication server to control the retrieval process;
- **AUTN = HMAC**$_{K_0}$**(RAND** | **SEQ** | **"AUTN")**
  the network authentication token that permits the IoT device to verify authenticity of the serving domain - note the usage of the context string "AUTN" in the computation;
- **XRES = HMAC**$_{K_0}$**(RAND** | **SEQ** | **"XRES")**
  the response of the IoT device to the gateway authentication challenge - the usage of a different context string (namely, "XRES") guarantees an independent HMAC output;
- **MKEY = HMAC**$_{K_0}$**(RAND** | **SEQ** | **"MKEY")**
  the ephemeral master key used to derive the protocol-specific session keys.

Note that an interesting caveat of our approach (actually an improvement with respect to the approach used in the cellular networks) is the usage of both challenges (RAND and SEQ) in *all* the computations - for backward compatibility reasons this is not done for XRES in 4G-AKA.

3.4.3.2 Key Agreement

Another notable difference of our proposed approach with respect to 4G AKA is the possibility to combine RIOT-AKA with a legacy security protocol. For this purpose, we propose to derive a single ephemeral master key (MKEY), and then integrate this key into a different protocol supported by RIOT. We specifically see two interesting extensions along this line. The first one consists in using the ephemeral master key MKEY derived during the RIOT-AKA handshake as pre-master key in the (Datagram) Transport-Layer-Security (TLS/DTLS) stack. Normally (D)TLS derives [71] the pre-master key using an Ephemeral Diffie-Hellman (DHE) or an Elliptic Curve Diffie-Hellman (ECDHE) exchange, which is extremely demanding [72] in terms of computational overhead and memory usage. (D)TLS also supports the option PSK (Pre-Shared Key) but this would yield a static secret key. With our RIOT-AKA protocol, the two parties can easily produce a fresh Key at every new authentication, and provide that as PSK in a TLS handshake. Since we are already relying on CoAP-exchanged messages, a second interesting approach consists in plugging the ephemeral MKEY in the OSCORE [76] protocol, in order to protect the application layer request/response messages between the endpoints.

3.4.4 Implementation and Extensions

We provide a working prototype on real hardware in order to demonstrate our scheme. The code is open-source, and it is published at [56] and [57] for reproducibility. Our experiments were carried out on **nRF52840-DK**, Arm Cortex M-based IoT device with 1MB of Flash memory and 256kB of RAM. For reliable measurements of the communication we used 6LoWPAN over Ethernet over serial. For our setup, we used RIOT as the real-time operating system in version 2021.01. Our setup uses one RIOT node communicating to a virtual Python-implemented server "Gateway" node via a 6LoWPAN/-CoAP protocol stack. The Gateway sends the request through HTTP(S) to the "Provider" (which is Python-implemented too) in order to receive all the hashing material needed for the Authentication of the IoT device. We confront the two implementations for both PSK and SRAM-PUF key generation modes. They are not intended as a mere performance comparation between the two since they are almost entirely the same code-base, but as a general overview of the two modes and how our scheme impacts on real hardware. For the PSK mode we use a 32 bytes long key. In the SRAM-PUF mode we can obtain and use a 10 bytes long key from the PUF. The key-provisioning part is not included in the testing phase. The measurement captures the entire protocol exchange, including network communication and hash generation. We start the measurements when the first packet is sent out to initiate the authentication and stop them when the last packet is sent.

Table 3.3: Energy consumption

	PSK	SRAM-PUF	Diff
AVG (mA)	1.02	1.06	0.04
MAX (mA)	4.64	4.63	-0.01

Table 3.4: Bandwidth consumption

	PSK	SRAM-PUF	Diff
Linux CoAP (bytes)	149	149	0
RIOT-OS (bytes)	136	136	0
Time (s)	0.041	0.040	-0.001

Table 3.5: Memory consumption

	PSK	SRAM-PUF	Diff
Text (bytes)	70976	71152	176
Data (bytes)	832	832	0
Bss (bytes)	28196	28212	16
Tot (bytes)	100004	100196	192

Table 3.6: ROM and RAM consumption

	PSK	SRAM-PUF	Diff
ROM (bytes)	71808	71984	176
RAM (bytes)	29028	29044	16

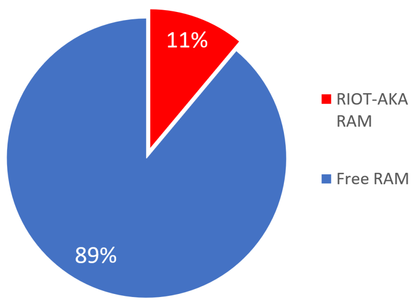


Figure 3.12: RAM portions used by our implementation.

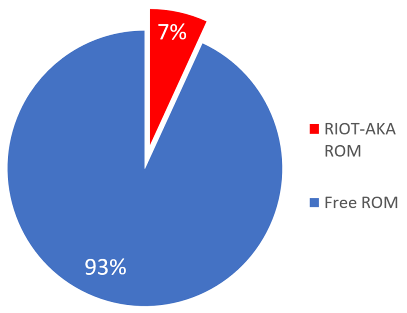


Figure 3.13: ROM portions used by our implementation.

3.4.4.1 Energy consumption impact

We observe that (Table 3.3) there is a 3.9% increase in the average energy consumption going from PSK to SRAM-PUF key generation. However, we can safely assume that it is not a relevant difference caused by the two modes but rather a normal slight fluctuation during execution in such a small window of time. We can conclude the same for the peak value that sees a 0.22% decrease from PSK to SRAM-PUF.

3.4.4.2 Bandwidth consumption impact

We observe that (Table 3.4) there is no difference at all on the bytes sent over-the-wire in the two implementations. That is because the two behave the same, they just use a different key for the HMACs. Time spent is basically identical (2% decrease from PSK to SRAM-PUF) for the same reason.

3.4.4.3 Memory Consumption impact

We see (Table 3.5 and 3.6) a minimal difference in the two implementations on the memory consumption. That is caused by the fact that we include (or exclude) the Modules of the SRAM-PUF from the compilation by changing a flag in our Makefile. ’Text’, ’Data’ and ’Bss’ refers to the objects that divide the .ELF files of a compiled application: ‘Text’ is what ends up in flash memory (ROM). ‘Data’ refers to initialized data, which is not constant, so in ends up in both ROM and RAM. ’Bss’ refers to all unintalized data and ends up entirely in RAM. Specifically, we can see in a 0.25% ROM usage increase (176 bytes) and a 0.06% RAM usage increase (16 bytes) going from PSK to SRAM-PUF. Fig. 3.12 and 3.13 graphically highlight our implementation’s used ROM and RAM memory portions in our nrf board (differences between PSK and SRAM-PUF being negligible). Overall, memory consumption of our prototype is totally suitable and feasible in a IoT usage environment especially since we still have room for a lot of optimization.

Chapter 4 Intrusion management

Under reasonable assumptions, intrusion events should always be considered possible in any infrastructure. Managing these events requires a proper design as well as runtime strategies for minimizing risk and impact. Intrusion management workflow demonstrates how Intelligent Infrastructure is capable to mitigate intrusion risks by reorchestrating mobile services in Fog architecture. At runtime the security of the infrastructure is monitored using IDS systems. To avoid false positive events the Human-in-the-loop approach is used (see Figure 4.1). The possible intrusion events are detected by IDS systems, and reported to the human, which can mitigate the increased risk levels by using means available in the Intelligent Infrastructure.

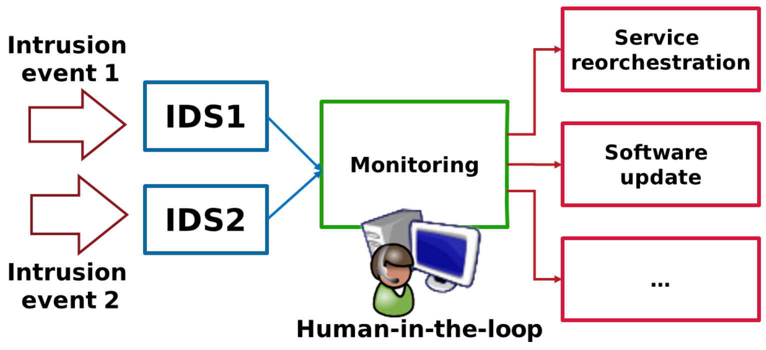


Figure 4.1: Intrusion management workflow.

Intrusion management demonstration workflow is following:

1. One of the IDS systems detects that security of one of Fog nodes is compromised and shows corresponding warning to the monitoring specialist using GUI;
2. Human decides to mitigate the security risk by informing the Intelligent Infrastructure that the security level of the affected Fog node changed from “High” to “Low”. He/she uses GUI and informs the orchestrator of the corresponding Fog node about the reduced security level;
3. Orchestrator evaluates new situation and decides to reorchestrate services running on the compromised Fog node by placing them in the remaining Fog nodes, as their security is still “High” and available resources are sufficient to run all services;
4. All services from the compromised Fog node migrate to the remaining Fog nodes and overall security of the intelligent infrastructure is restored.

The remaining of this chapter is organized in the following way: first, the technologies used in two Intrusion Detection Systems developed by ITTI and Inria are presented, then the method for dynamic service orchestration in the Fog Computing proposed by KTU is described, and finally the results are summarized.

4.1 ML-based Network Intrusion Detection System

Network Intrusion Detection Component

The tool responsible for detecting cyberattacks is built upon the Apache Spark. The choice of this particular framework is motivated by the fact that it has established a great, unrivalled reputation as being fit for dealing with vast volumes of information, and capable of distributing tasks to multiple servers. The tool is of modular character and leverages languages such as Python or Scala. This feature makes it possible to include additional components, and the architecture remains scalable, as it is able to employ further spark workers and administer work among servers. The pipeline of the Component has been presented in Figure 4.2.

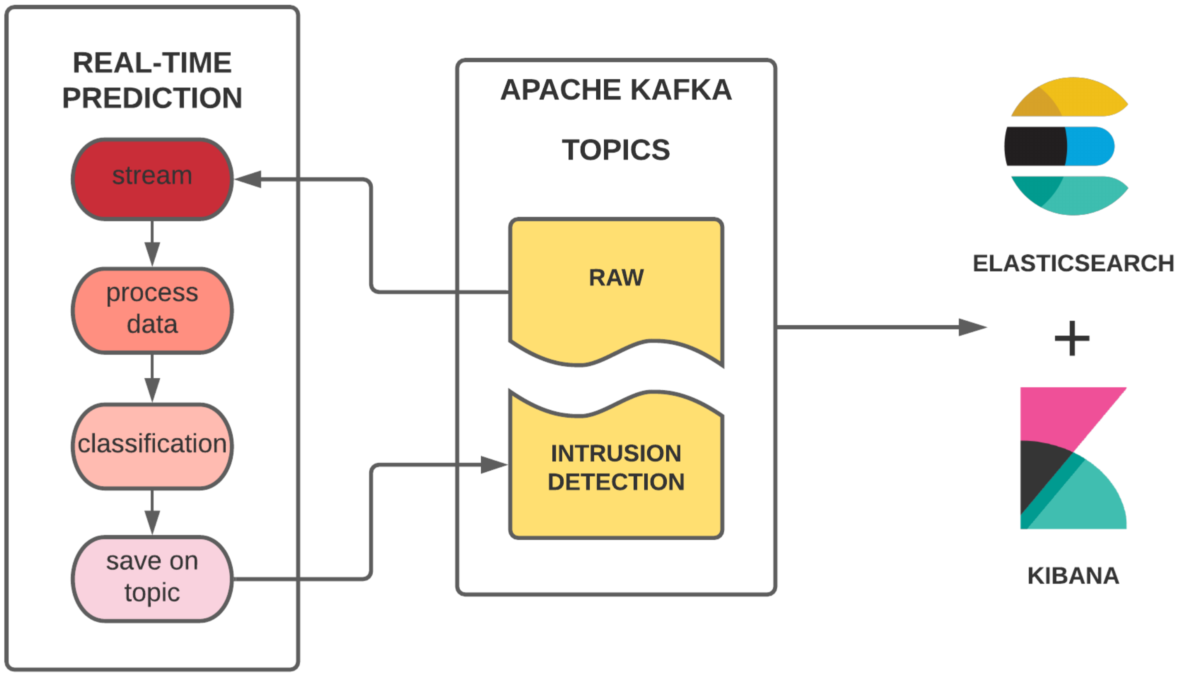


Figure 4.2: The pipeline of the Network Intrusion Detection Component

The intrusion detection tool receives the data for processing via the distributed streaming platform. As it is done by means of Apache Kafka, it enables receiving, validating, transforming and sending messages among applications. The process begins with a mode responsible for sourcing data from a stream and inspects the data on the network traffic. The data is provided in the form of frames on one of the Kafka topics by the collector module; thus, the data can be collected either from miscellaneous devices connected to the network, or inspect the live network flow. As soon as the Netflow frames have been transformed into an exact Kafka topic, the information is then cleaned and scaled, and the features get extracted. After pre-processing, the data is sent to the classifier module, based on machine learning. There, the classification of specific frames takes place, by means of a trained model, such as an artificial neural network. Based on this, a specific label is assigned. The outcomes are then published to the Kafka bus. The architecture enables employing an infinite number of models and processes, which contribute to the creation of a machine learning environment which is complex and convoluted. The entirety of the environment is integrated into Elasticsearch. In turn, thanks to the application of the Kibana solution, the outcomes are visualized in the form of comprehensive, easily-understandable dashboards. In this architecture, the network traffic from the configured devices is analysed and monitored in real time. Delivered in the NetFlow format, the traffic goes through the machine-learning-based anomaly detection algorithms. The Kafka message on a predefined topic is sent to the service in the json format (See: Table 4.1); it is also the format of the system's outcome (Presented in Table 4.2).

Table 4.1: INPUT. Schema: NetFlow. Input data format: JSON. Documentation NetFlow Collector: https://www.ntop.org/guides/nprobe/cli_options.html#usage-examples

#	Parameter Name	Description
1	IN_BYTES	Incoming flow bytes (src->dst)
2	IN_PKTS	Incoming flow packets (src->dst)
3	PROTOCOL	IP protocol byte
4	SRC_TOS	TOS/DSCP (src->dst)
5	TCP_FLAGS	Cumulative of all flow TCP flags
6	L4_SRC_PORT_MAP	Layer 4 source port symbolic name

7	L4_DST_PORT_MAP	Layer 4 destination port symbolic name
8	SRC_AS	Source BGP AS
9	DST_AS	Destination BGP AS
10	LAST_SWITCHED	SysUptime (msec) of the last flow pkt
11	FIRST_SWITCHED	SysUptime (msec) of the first flow pkt
12	OUT_BYTES	Outgoing flow bytes (dst->src)
13	OUT_PKTS	Outgoing flow packets (dst->src)
14	MIN_IP_PKT_LEN	Len of the smallest flow IP packet observed
15	MAX_IP_PKT_LEN	Len of the largest flow IP packet observed
16	ICMP_TYPE	ICMP Type * 256 + ICMP code
17	TOTAL_FLOWS_EXP	Total number of exported flows
18	MIN_TTL	Min flow TTL
19	MAX_TTL	Max flow TTL
20	DST_TOS	TOS/DSCP (dst->src)
21	FLOW_START_SEC	Seconds (epoch) of the first flow packet
22	FLOW_END_SEC	Seconds (epoch) of the last flow packet
23	FLOW_DURATION_MILLISECONDS	Flow duration (msec)
25	ICMP_IPV4_TYPE	ICMP Type
26	ICMP_IPV4_CODE	ICMP Code
27	APPL_LATENCY_MS	Application latency (msec), a.k.a. server response time
28	SRC_TO_DST_AVG_THROUGHPUT	Src to dst average thpt (bps)
29	DST_TO_SRC_MAX_THROUGHPUT	Dst to src max thpt (bps)
30	DST_TO_SRC_MIN_THROUGHPUT	Dst to src min thpt (bps)
31	DST_TO_SRC_AVG_THROUGHPUT	Dst to src average thpt (bps)
32	CUMULATIVE_ICMP_TYPE	Cumulative OR of ICMP type packets
33	FLOW_PROTO_PORT	L7 port that identifies the flow protocol or 0 if unknown
34	LONGEST_FLOW_PKT	Longest packet (bytes) of the flow
35	SHORTEST_FLOW_PKT	Longest packet (bytes) of the flow
36	RETRANSMITTED_IN_BYTES	Number of retransmitted TCP flow bytes (src->dst)
37	RETRANSMITTED_IN_PKTS	Number of retransmitted TCP flow packets (src->dst)
38	RETRANSMITTED_OUT_BYTES	Number of retransmitted TCP flow bytes (dst->src)
39	RETRANSMITTED_OUT_PKTS	Number of retransmitted TCP flow packets (dst->src)
40	L7_PROTO	Layer 7 protocol (numeric)
41	DURATION_IN	Client to Server stream duration (msec)
42	DURATION_OUT	Client to Server stream duration (msec)
43	TCP_WIN_MIN_IN	Min TCP Window (src->dst)
44	TCP_WIN_MAX_IN	Max TCP Window (src->dst)
45	TCP_WIN_MSS_IN	TCP Max Segment Size (src->dst)
46	TCP_WIN_SCALE_IN	TCP Window Scale (src->dst)
47	TCP_WIN_MIN_OUT	Min TCP Window (dst->src)
48	TCP_WIN_MAX_OUT	Max TCP Window (dst->src)
49	TCP_WIN_MSS_OUT	TCP Max Segment Size (dst->src)
50	TCP_WIN_SCALE_OUT	TCP Window Scale (dst->src)
51	PAYLOAD_HASH	Initial flow payload hash
52	SRC_AS_MAP	Organization name for SRC_AS
53	DST_AS_MAP	Organization name for DST_AS

Table 4.2: Output. Data format: JSON

#	Parameter Name	Description
1	ID	UniqueId
2	version	Version of detector
3	partner	IP protocol byte
4	timestamp	Alert Timestamp
5	alert_type	Category of Alert
6	src_add	Source address IP
7	dst_add	Destination address IP
8	protocol	Protocol

4.2 Anomaly-based Intrusion Detection System

The second IDS used in the Intrusion management workflow is an anomaly-based IDS. During a preliminary phase called ”the learning phase”, a model which characterizes the normal behavior of the system is built. The normal learned behaviors are observed during executions that are not disturbed by malicious actions or misuses. As only safe scenarios are learned, the anomaly-based IDS does not rely on labeled data. The learning phase ends when the knowledge of the set of possible normal behaviors seems fairly complete. Indeed multiple observations are needed to capture the diversity of correct behaviors produced by authorized interactions between the various entities of the system. In a second phase called ”the detection phase”, the built model is used to monitor the progress of the system: any deviation from the behavior expected by the model is interpreted as the manifestation of an attack or a misuse that falls out of normal system operation.

4.2.1 Supervision of a Partially Ordered Set of Events

Initially the proposed solution was designed to supervise the execution of distributed applications. In this context, the observation and the interpretation of a behavior are carried out at the highest level (source code level). At runtime, thanks to an instrumentation of the application's source code, the occurrences of significant actions (corresponding to instructions previously identified within the code) are captured. Each occurrence is called an event and the whole set of observed events is called a trace. In the proposed solution, a trace is exploited to build different representations of the monitored computation (lattice, automaton, list of invariants, ...). All these possible models are based on the same basic principle: the gathered trace is interpreted as a partially ordered set of events (POSet). In the case of a distributed computation, the order relation is naturally linked to the notions of dependency and concurrency between events occurring on different machines. Through two sub-models used in parallel (one based on an automaton and the other based on invariants), the proposed IDS checks whether the order of occurence of events is acceptable or not with regard to the constraints defined by the automaton and the list of invariants. To reduce their sizes and also to obtain a more general characterization of acceptable behaviors, the two sub-models are based on the definition of event types. An event is unique and appears only once in the trace while its event type (more generic) may be present several times. In the case of a distributed computation, the definition of such event classes can be done in a simple and natural way because, at a minimum, the events can be grouped according to the instruction of the code which triggered them. In the sub-models, the notion of event type is central. The set of learned event types is the alphabet of the automaton and each transition is labelled with a particular event type. An invariant is a temporel property (satisfied during all the learned executions) that refers to two different event types. For example, the invariant ”an event of type a *is never followed by* an event of type b” focuses on two event types, namely a and b.

In [34], we describe how this IDS strategy can be applied to detect attacks against a particular distributed application, namely a distributed file system called XtreemFS (`http://www.xtreemfs.org/`).

4.2.2 Analysis of a Network Traffic

During the second part of the SPARTA project, we have investigated the possibility of using the same approach to analyse network traffic. In this new context, the traffic is observed in a particular location of the network by a packet sniffer in charge of capturing packets and presenting them to the IDS (either on the fly or within a PCAP file). In order to be able to use similar analysis techniques, we assume that each observed packet is corresponding to an event and the sequence of packets is interpreted as a totally ordered set of events.

Figure 4.3 describes the four main components of the IDS and their interactions.

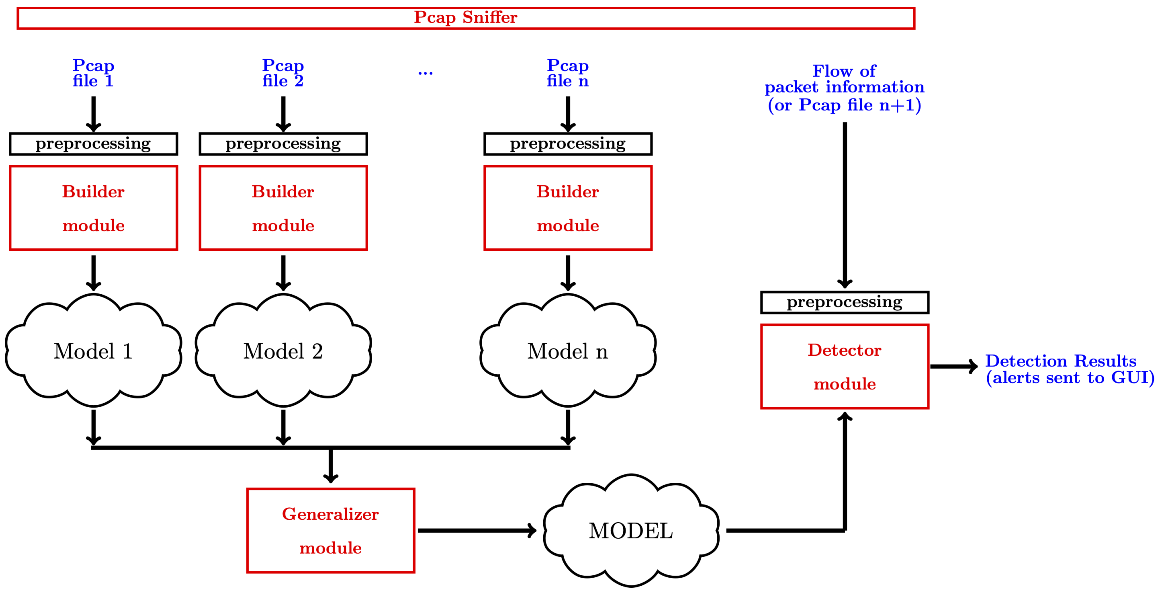


Figure 4.3: General View of the Architecture of the Second IDS

- A component ”Pcap Sniffer” is in charge of the capture of the traffic. This component can either immediately transfer information about each observed packet (online detection phase) or store this data in a file that will be used later during an offline processing (learning phase or detection phase). The information delivered includes the main fields of the packet’s header namely the source and destination IP adresses, the source and destination ports, a frame number, a timestamp and a label which may identify the used protocol.
- A component ”Builder” is repeatedly executed to create one model (*i.e.* an automaton plus a list of invariants) for each captured traffic. For example, in Figure 4.3, the ”Builder” component is executed n times. During each of its execution, the input is either a classical Pcap file or a file produced by the ”Pcap Sniffer” component.
- A component ”Generalizer” is in charge of merging the different models that have been created separately by the previous component. This distribution of roles between the two main components used during the learning phase allows to update the model incrementally by learning new executions when necessary. While merging the models, the component can also used various techniques to transform the model so that it can now accept more behaviors close to those that have been learned.
- The last component ”Detector” is used during the detection phase to check if a particular network traffic is compliant or not with the model constructed previously. Detection can be done online or offline. Packets are analyzed in the order in which they have been observed. An alert is generated when an event (*i.e.* a packet) is rejected for at least one reason. An event can be rejected by the automaton when its event type does not belong to the automaton’s alphabet or when no transition labeled with this event type allows a progression from the current state. An event can be rejected by an invariant which refers to its event type if its occurrence at this stage of the execution violates this invariant. So when an alert is raised, the offending packet may

have been rejected for several reasons.

Analyzing a totally ordered set of events rather than a partially ordered set slightly decreases the interest of some features provided in the initial solution. Adaptations to the tools were necessary to take into account the fact that the order relation between events (corresponding now to network packets) is much less significant than the dependencies between high level events of a distributed application. Indeed, the order in which packets of various origins are routed and pass through a point of the network is often meaningless. Consequently, the use of generalization techniques which allow the model to accept behaviors relatively close to the learned ones has been reinforced. Finally, as the two submodels are based on the definition of event types, it is necessary to associate a generic type to each network packet. Due to the absence of semantic information, the event types that can be defined are rather artificial. Moreover, several choices are possible and the final choice has a real impact on the size of the constructed model, the time required for the detection and the accuracy of the detection (false negative and false positive). Table 4.3 summarizes different definitions of event types that can be selected by the IDS user when the learning phase starts (the user indicates its choice using a configuration file). The event types contained in this table are specified using only four fields namely the source and destination ID (or two keywords ”external” for external addresses and ”internnode” for internal adresses) and the source and destination ports (or a modulo applied on the port number or a focus on registered special ports). These possible choices have been evaluated using the CIC-IDS-2017 dataset (https://www.unb.ca/cic/datasets/ids-2017.html) to determine their impacts on the IDS. To quickly give an idea of this impact, when the trace corresponding to the first day of the CIC-IDS-2017 dataset is used to build a model, the number of discovered invariants is comprised between 488 (if choice number 31 is selected) and 3.461.509 (if choice number 34 is selected). A similar gap appears also when the size of the automaton is considered. Thus the definition adopted to specify the event types has to be carefully selected depending on the context, the expected accuracy and the ressource capacities. To provide more flexibility, the two event type definitions used by the two sub-models (automaton and list of invariants) can be different.

No	Event Type Definition	Example
20	Source Keyword : Port	external:80
21	Destination Keyword : Port	internnode:5325
22	Source Keyword : Special Port	internnode:21
23	Destination Keyword : Special Port	external:neg
30	Source IP	192.168.10.12
31	Destination IP	external
32	Source IP : Port	192.168.10.9:80
33	Destination IP : Port	external:5325
34	Source IP : Port % Modulo	192.168.10.1:3
35	Destination IP : Port % Modulo	192.168.10.2:4
36	Source IP : Special Port	192.168.10.1:22
37	Destination IP : Special Port	192.168.10.14:neg
40	Source Keyword : Port - Destination Keyword	internnode:80-external
41	Source Keyword - Destination Keyword : Port	internnode-external:5235
42	Source Keyword : Port - Destination Keyword : Port	external:80-internnode:5642
43	Source Keyword : Special Port - Destination Keyword	internnode:80-external
44	Source Keyword - Destination Keyword : Special Port	internnode-external:neg
50	Destination IP : Port \external	external
51	Source IP - Destination IP	192.168.10.1-external
52	Source IP : Port - Destination IP	192.168.10.1:80-192.168.10.2:6423
53	Source IP - Destination IP : Port	192.168.10.1-192.168.10.2.80
54	Destination IP : Port % Modulo \external	external
55	Source IP : Special Port - Destination IP	external:80-192.168.10.12
56	Source IP - Destination IP : Special Port	192.168.10.12-external:neg

Table 4.3**:** Some Definitions of Event Types Accepted by the IDS

4.3 Method for Dynamic Service Orchestration in the Fog Computing

The need for dynamic service orchestration rises from the nature of the Fog architecture which includes various heterogeneous constrained Edge devices. Moreover the situation near the Edge of the infrastructure is dynamic and changes rapidly during the runtime of the system. The changes may include: security of the Fog node is compromised and some critical services need to be reallocated; security of the Edge sensor or actuator is compromised and depending services should no longer trust them; Edge device moves in space and corresponding services need to be reallocated to meet the requirements of communication protocols; energy, power, computational resources of the Fog node are exhausted and services need to be redistributed among all available Fog nodes to achieve best possible running time of the whole solution; etc. Dynamic service orchestration may be applied to address these challenges. A more detailed analysis of these issues and proposed novel method for Dynamic Service Orchestration in Fog Computing is published in paper [68].

The simplified architecture of such approach is presented in Figure 4.4.

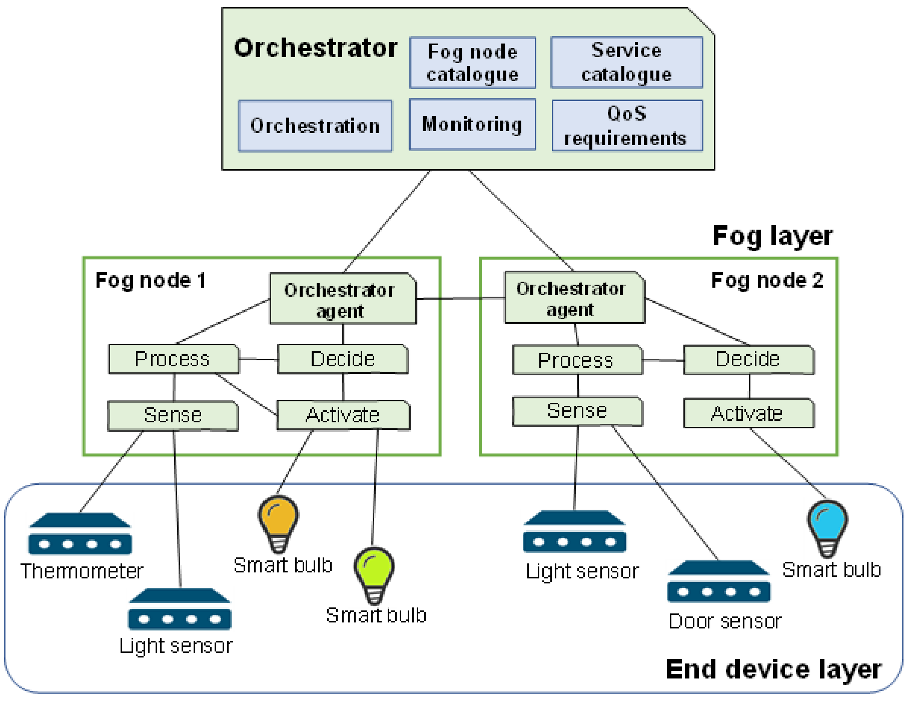


Figure 4.4: Architecture of Fog orchestrators.

The main parts here are the orchestrators, which are running in all Fog nodes and collect information about the infrastructure and the services. Orchestrators communicate with each other and exchange the information about the capabilities of Fog nodes, current status of the software and hardware, as well as maintain the service catalogue. All services in such architecture are mobile and may migrate, stop, start, suspend and resume their operation upon request of corresponding orchestrator. To monitor the situation changes during the runtime orchestrators execute the control loop presented in Figure 4.5. The control loop allows orchestrators to react to the changing situation and make corresponding decisions. The main processes in the control loop are the following:

- Monitoring. Collect dynamic information about all Fog nodes and running services and detect if the situation is changed compared to the previous state.
- Optimization. Find the best service distribution among all available Fog nodes according to the current situation in the Intelligent Infrastructure while satisfying multiple constraints and optimising QoS, including security level, available power, available energy, bandwidth limitations, RAM and CPU utilization, etc.
- Execution. If the new optimal service placement is different from the actual, initiate services migration from one Fog node to another to achieve optimal distribution of services while preserving the running state.

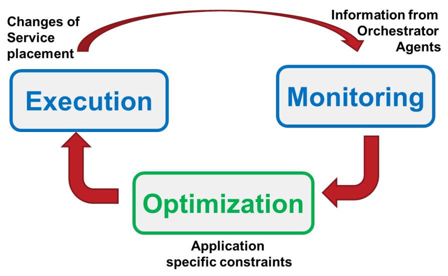

Changes of Service placement
Information from Orchestrator Agents
Execution
Monitoring
Optimization
Application specific constraints

Figure 4.5: Fog orchestration control loop.

To implement the control loop two solutions described in the next subchapters are used. Multi-agent based architecture allows to implement mobile services needed for the execution phase. Communication and collaboration oriented nature of the Agents also helps to implement the monitoring phase. Method for finding optimal placement of the services provides a solution for the optimization phase.

4.3.1 Method for Finding Optimal Placement of the Services

An optimization method must help to distribute the available services among Fog nodes while ensuring the required level of QoS and using minimal resources from Fog and End devices, thus providing the longest lifecycle of the whole Intelligent Infrastructure. The objective of this method is to find which placement of n available services in k Fog nodes is the best according to a given multiple constraints and conditions. To simplify the problem we assume that different Fog nodes have different characteristics and capabilities, but are able to host all the services. The goal of an optimization is to place the services in such a way, that a set of QoS parameters is optimal. QoS parameters of the i-th possible service placement X_i are expressed by the values of the objective functions $f_j\left(X_i\right)$, $j=1,2,\ldots,m$ (where m is a number of objective functions used for optimization) and constraints (Equation 2). The objective of the optimization is to find the best service placement X_{opt} which minimizes all the objective functions f_j:

$$X_{opt} = \text{arg}\underset{i}{min} F\left(X_i\right) \tag{1}$$

where $F\left(x\right) = \left\{f_1\left(x\right),\ f_2\left(x\right),\ \ldots,\ f_m\left(x\right)\right\}$ is a set of the objective functions, and $x \in \left\{X_i\right\}$ is a member of the set with all the possible service distributions.

Constraints are given by equations:

$$\begin{cases} g_j\left(X_i\right) \geq 0, j=1,2,\ldots,n_g \\ h_k\left(X_i\right) = 0, k=1,2,\ldots,n_h \end{cases} \tag{2}$$

Some examples of the objective functions considered in this method are following: A security of the whole system $f_{sec}\left(.\right)$ is defined by the lowest security of all the services. Security levels are expressed in security bits, according to NIST [17] recommendations. CPU usage $f_{CPU}\left(.\right)$ and RAM usage $f_{RAM}\left(.\right)$ evaluate how evenly hardware resources of available Fog nodes are used. Power usage $f_{pw}\left(.\right)$ is evaluated using the average power requirements of each service and the available power of Fog nodes. Criterion of maximum range $f_{rng}\left(.\right)$ accounts for the preference of local communications which helps to minimize energy consumption and bandwidth limitations. Other criteria such as permanent storage capabilities, communication channel bandwidth and latency, etc. may also be used by defining corresponding objective functions, which must have following characteristics:

- A return value of the function must be a positive real number.
- Better values of the criterion must be represented by smaller numbers.

As one can see, the objective functions are contradicting to each other so there is no single solution to this multi-objective optimization problem that optimizes all the objective functions at the same time. So the two stage optimization method, summarized in Figure 4.6 is used.

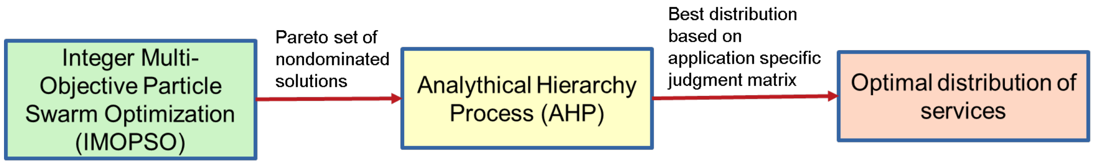

Integer Multi-Objective Particle Swarm Optimization (IMOPSO)
Pareto set of nondominated solutions
Analythical Hierarchy Process (AHP)
Best distribution based on application specific judgment matrix
Optimal distribution of services

Figure 4.6: Main steps of service distribution optimization process.

Modified Integer Multi Objective Particle Swarm Optimization (IMOPSO) [68] method is used to find a set of Pareto optimal solutions. All service placements in this set are non-dominated (Pareto optimal), which means that each of them is better than all the other ones by at least one criterion. The second step is to choose best solution from the Pareto optimal set by using the Analytical Hierarchy Process (AHP) [52, 75]. AHP uses only a pairwise comparisons of all alternatives by all objective functions, is easy to implement and gives consistent results. An example of three-level AHP process structure is presented in Figure 4.7.

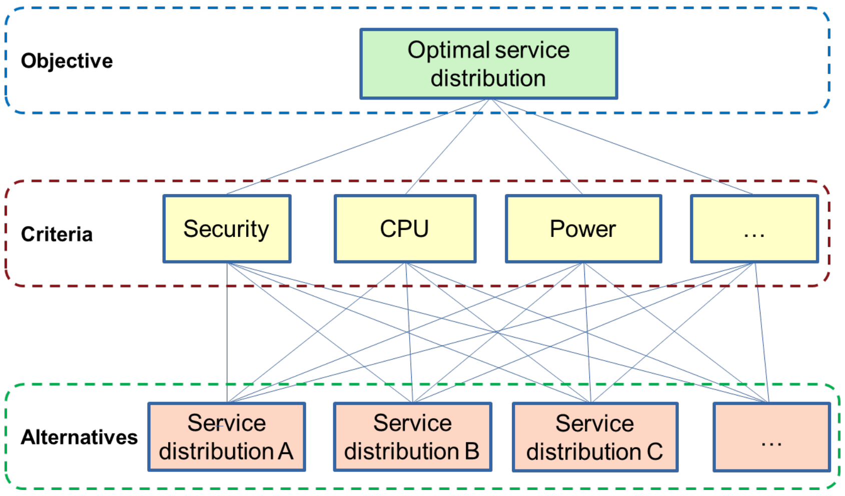


Figure 4.7: Hierarchical structure of AHP [68].

On the top level is an objective of the process – optimal distribution of the services among Fog nodes. The bottom level is alternatives – all Pareto optimal solutions which were found using the IMOPSO method in the previous step. All objective functions used in the IMOPSO part of the process are used as criteria in the middle level of AHP process. In order to get one final answer all criteria must be assigned a weight. This is done by using a pairwise comparison of the criteria and forming the so called judgment matrix. This step must be done manually by human, as only the human can compare two criteria and decide which one is more important in the given application area. Thankfully, judgment matrix can be formed once, deployed into the orchestrator, and after that the AHP process is fully automatic and is able to choose one best alternative.

4.3.2 Prototype Architecture for Hosting of Mobile Services

The prototype IoT system (see Figure 4.8) was developed to demonstrate the feasibility of the monitoring and execution stages of the dynamic Fog service orchestration.

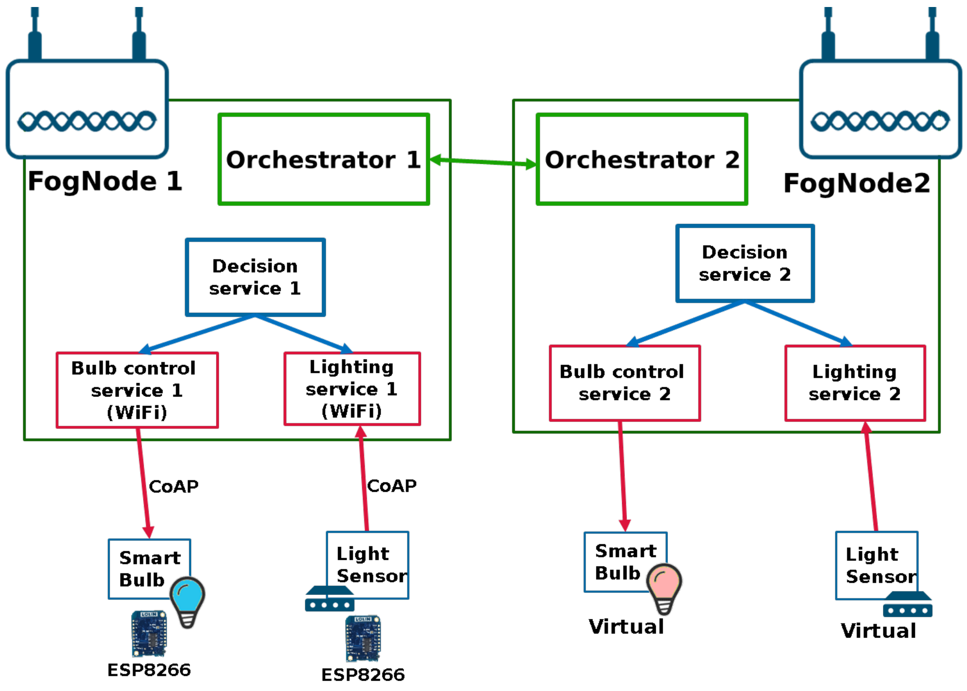


Figure 4.8: Prototype system architecture.

The prototype was implemented as Multi-Agent system [37, 87] using Jade platform [18]. Multi-Agent architecture is suitable for development of loosely coupled resilient IoT systems [31, 70] and allows to implement all Fog layer services as independent mobile agents which are able to communicate between each other, make decisions and migrate between different Agent platforms as needed.

Each Fog node runs an Agent platform (Jade platform), and all services are implemented as Agents. All agents may be divided into three different classes: sensor and actuator agents, decision agents and orchestrator agents. Sensor and actuator agents run inside the Fog nodes and are responsible for monitoring and controlling physical devices placed in the Edge layer. In prototype implementation we have lighting agents which communicate with lighting sensors using CoAP protocol and are able to respond to the queries from the other agents providing them the current state of the lighting. Bulb control agents use CoAP protocol to activate (or deactivate) the physical bulb in the Edge layer. These agents are able to respond to messages from other agents and according to them control the bulb they are responsible for. The prototype Edge layer devices were build using ESP8266 microcontrollers, which implement CoAP protocol for communications with Fog node services. BH1750 sensor was used for measurement of lighting and simple LED for emulating the bulbs. "Virtual" versions of sensor and actuator agents were also implemented. Theses agent act as normal agents of the corresponding type but they do not communicate with real hardware in the Edge layer. These agents allow to easily increase the number of Fog nodes and services without needing to use additional hardware.

Decision agents are the agent which provide the "useful" service. They are software agents which are responsible for lighting control in the rooms. To make things simple very straightforward decision algorithm was used. Each decision agent continually communicates with lighting agents and gets the current lighting conditions. If the lighting is less than the predefined threshold level, then the decision agent contacts the bulb control agent ant asks it to activate the bulb. The process is inverted if the lighting conditions are better than predefined threshold.

Orchestrator agent is the agent responsible for implementing platform status monitoring and execution of changes. Each Fog node (or Jade platform) has one orchestrator agent, which communicates with all other local agents and collects basic information about them: the list of available agents, the current state of the agents (running, suspended, stopped), etc. Each orchestrator is also responsible for monitoring the local network and detection of the new Fog nodes if they emerge during the runtime. Each orchestrator communicates with all other known orchestrators and collects the information about all services available in all know Fog nodes.

If some changes occur in such dynamic system then all orchestrators will collect information about them. After the changes are detected the orchestrator must decide if the current situation is optimal according to the application specific QoS requirements. This is done by using the method for finding optimal placement of the service described in the previous subsection. If the new optimal placement of services differs from the current actual state, then the process of service migration is started. Orchestrator communicates with all affected agents and tells them to move to the new Fog nodes as required by optimal placement. To make things simpler in the prototype implementation all services are provided using WiFi protocol, which has sufficient range to place all services in all available Fog nodes. Moreover, as Fog nodes are implemented as Docker containers, they have sufficient resources to host all services without degrading the overall performance of the whole Intelligent Infrastructure. The main goal of demonstration in such conditions is to ensure ”High” security level of the whole infrastructure, which means, that all services must be placed only in those Fog nodes which can provide ”High” security levels. If some security problems are detected (e. g. by IDS system) then the services from the affected Fog node must be removed and placed into more secure Fog nodes.

The method for finding optimal placement of the services was implemented using Matlab. The method validation and performance evaluation was performed using the simulation data. This approach makes it easier to scale the solution and to reproduce the results. The main objective was to show how the optimization method performs in different situations. Detailed experimental results are presented in [68]. The implementation of Agent-Based Intelligent Infrastructure was tested using two, three and four Fog nodes running mixed set of “real” and “virtual” sensor and actuator agents as well as decision and orchestrator agents. Experimental results show that the system is able to adapt to situations when one of even two (out of three or four) Fog nodes are compromised.

4.4 Remarks

The main goal of Intrusion management workflow is to show how Intelligent Infrastructure adapt to environment changes and reorganize itself to maximize compliance to the QoS and security requirements. Machine learning and anomaly based intrusion detection systems complement each other to enable detection of possible security compromising attacks targeted to the devices and/or services of Intelligent Infrastructure. Human-in-the-loop approach allows to mitigate the risks of false positive alarms affecting the system and degrading its performance. Multi-agent based implementation allows to add resilience and ability to easily reorchestrate services according to the changes in the environment. The orchestrators running in each Fog node enable easy collection of information about the state of the dynamic infrastructure and provide needed information for decision making process, which (augmented with mobility of agents) enables to easily mitigate various risks (including security ones).

All the technologies used in Intrusion management workflow were implemented and tested independently by their developers. The integration testing was performed using the virtual infrastructure and the results are presented in Chapter 6 of this document.

Chapter 5 Data & privacy management

Intelligent Infrastructures (IIs) interconnect a variety of Internet of Things (IoT) applications and services in order to capture and analyze data and also invoke autonomic responses. Some services that are operating with personal data may unintentionally cause privacy leakages. Therefore, applying the privacy-by-design approach is essential while designing and deploying digital services in IIs. The correctly designed data and privacy management, which aggregates deployed Privacy-Enhancing Technologies (PETs) and correct business model settings, may provide reasonable user privacy and compliance with legislation rules such as GDPR.

The PETs usually try to provide the subset from these security and privacy features:

- attribute-based user authentication,
- anonymity (or pseudonymity),
- unlinkability,
- revocation,
- data privacy,
- privacy-preserving post-processing,

The various PETs and their state of the art are described in the deliverable D6.1 or in our recent survey paper [61].

The GDPR imposes various obligations for the data controller[1] to ensure the protection of personal data[2]. The core of data protection revolves around transparency, lawfulness, purpose limitation, minimization of personal data, accuracy, limitation of retention of personal data, and security[3]. A scenario and tools that take these requirements into account from the design stage can ensure effective implementation of the data protection principles[4]. In Figure 5.1, we focus our analysis of compliance with the regulation governing the processing of personal data on the technical aspects to be implemented in the intelligent infrastructure tools. In addition, Figure 5.2 lists the security requirements for compliance with the principle of privacy by design. The data controller must also put in place a real internal personal data governance policy. For this, we refer in particular to the guidelines of the European Data Protection Board[5] and the deliverable D6.1.

[1]According to Article 4.7 of the GDPR, the data controller is the natural or legal person "which, alone or jointly with others, determines the purposes and means of the processing of personal data".

[2]We recall the broadness of the definition of personal data: "any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person" (Article 4.1 of the GDPR).

[3]See Articles 5 and 6 of the GDPR.

[4]The principle of Privacy-by-Design is a core element for the protection of personal data. Article 25.1 of the GDPR states that: "Taking into account the state of the art, the cost of implementation and the nature, scope, context, and purposes of processing as well as the risks of varying likelihood and severity for rights and freedoms of natural persons posed by the processing, the controller shall, both at the time of the determination of the means for processing and at the time of the processing itself, implement appropriate technical and organizational measures, such as pseudonymization, which are designed to implement data-protection principles, such as data minimization, in an effective manner and to integrate the necessary safeguards into the processing in order to meet the requirements of this Regulation and protect the rights of data subjects".

[5]EDPB, "Guidelines 04/2019 on Article 25 Data Protection by Design and by Default", 20 October 2020. Available at: https://edpb.europa.eu/our-work-tools/our-documents/guidelines/guidelines-42019-article-25-data-protection-design-and_en

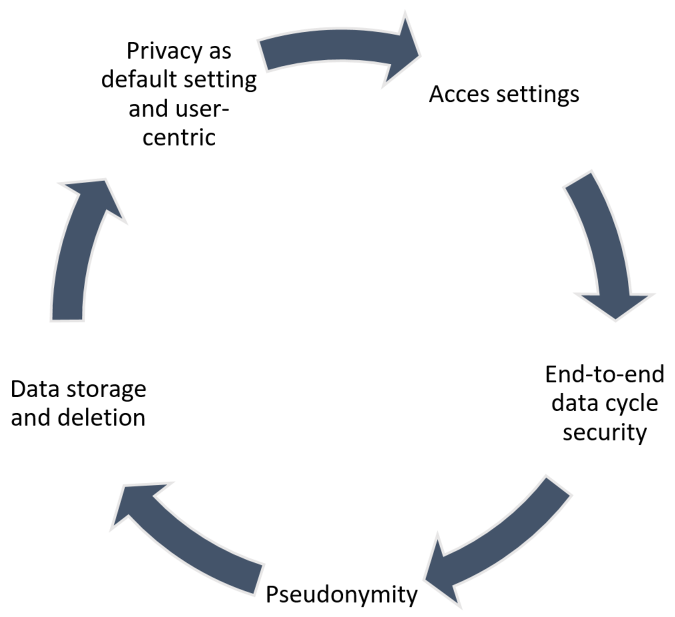

Privacy as default setting and user-centric
Acces settings
End-to-end data cycle security
Pseudonymity
Data storage and deletion

Figure 5.1: Technical settings for Privacy-by-Design implementation.

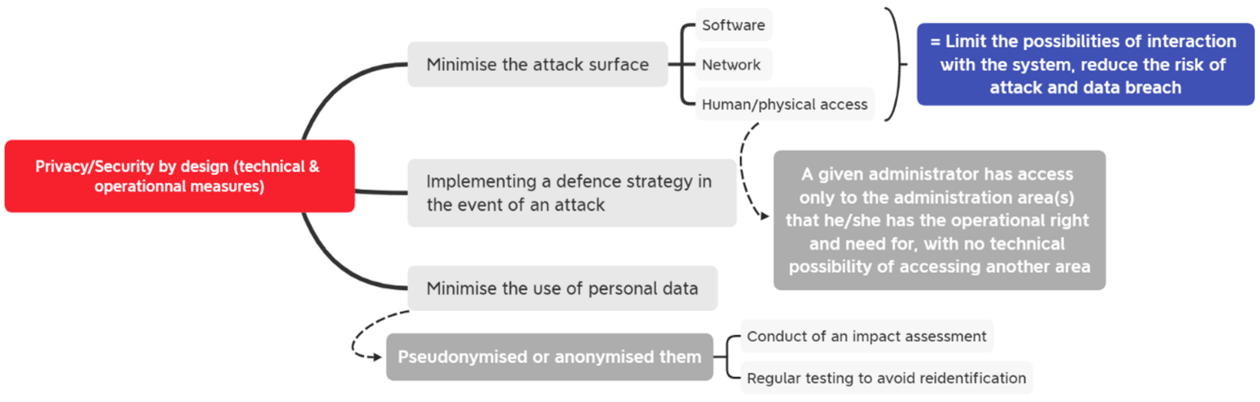

Privacy/Security by design (technical & operationnal measures)
Minimise the attack surface
Software
Network
Human/physical access
= Limit the possibilities of interaction with the system, reduce the risk of attack and data breach
Implementing a defence strategy in the event of an attack
A given administrator has access only to the administration area(s) that he/she has the operational right and need for, with no technical possibility of accessing another area
Minimise the use of personal data
Pseudonymised or anonymised them
Conduct of an impact assessment
Regular testing to avoid reidentification

Figure 5.2: A GDPR perspective for a security by design implementation.

This chapter presents our developed privacy-preserving technologies concretely: PEAS, GDPR checker, and MC-SSE. In particular, we describe the basic principle of each tool/technology and their final extensions from the initial state described in report D6.3 to their final release. How these technologies may serve in the concrete scenario is described in Section 6.3.

5.1 Privacy-Enhancing Authentication System

This section presents the final release of the Privacy-Enhancing Authentication System (PEAS) and how PEAS can support privacy in chosen intelligent infrastructure scenarios, i.e., smart campus and smart building. PEAS is a privacy-preserving authentication system that does not disclose the whole user identity to a verifier. Only necessary pieces of the user identity (e.g., age, gender, membership, access ticket, etc.) are provided during the verification phase. The authentication sessions of PEAS are mutually unlinkable. Therefore, the protocol protects user identity and avoid profiling and trucking users. The core of PEAS is based on an Attribute-based Credentials (ABC) cryptography protocol, namely, Keyed-Verification Anonymous attribute-based Credentials (KVAC), published in [24]. The PEAS expands the KVAC protocol by the revocation process to Revocable Keyed-Verification Anonymous attribute-based Credentials (RKVAC), [42]. PEAS is also practical 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 their personal data, and they decide which 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.

PEAS can be deployed in these following scenarios:

- Smart parking access control — PEAS can provide the secure and privacy-enhancing access of drivers (users) into campus/institution parking lots and parking areas.
- Smart building access control – PEAS can provide the secure and privacy-enhancing access of persons (users) to smart buildings, rooms, departments, and various areas of the campus/institution.

Figure 5.3 depicts the basic PEAS topology with the entities such as Issuer, Revocation Authority, User with a personal device, and Verifier with a reader device.

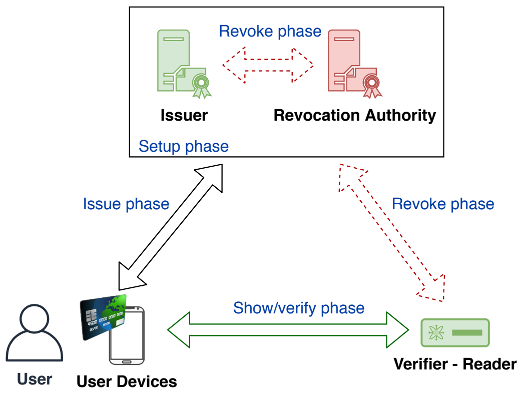


Figure 5.3: Highlevel Topology of PEAS.

The full description of PEAS can be found in report D6.3 and in the paper [42]. The following subsections describe the technical extensions of PEAS (from the version from D6.3 to D6.4) and testing results.

5.1.1 PEAS Extensions and Implementation Details

The implementation of PEAS is highly modular. PEAS consists of the front-end parts that are realized as web-based or mobile-based applications that can be separately enhanced by new specific features. PEAS uses the cryptography core part that supports all basic and cryptography operations. The cryptography core part is realized as a standalone C-library that is then used by front-end parts. All main PEAS parts can be easily updated and fixed as separated layers. The PEAS core part uses several third party libraries such as pcsc-lite and ccid libraries for smartcard connections, libcjson and libwebsockets for web server connections, openssl, gmp, zlib, mcl for the cryptographic support, and the libpeas library for the RKVAC operation support.

The PEAS has been extended (from version D6.3 to D6.4) as follows:

- upgrading the core PEAS C-library,
- upgrading PEAS demonstrator by graphical user interfaces by using web-based applications,
- creating a Android-based PEAS user application for smartphones,
- integrating the mobile and web-based applications,
- adding the Bluetooth communication interface.

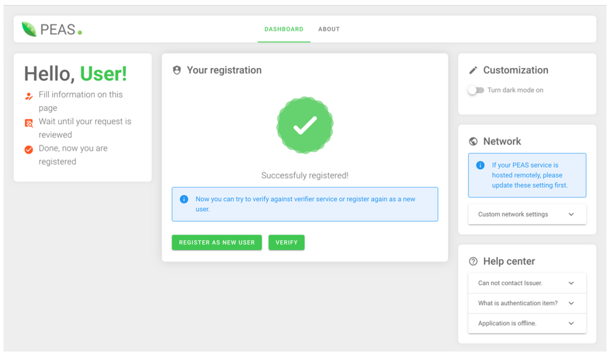
All source codes are stored in the private gitlab repository (`gitlab.com/brno-axe/peas/`) and each PEAS software part is developed as the separated project, e.g. `peas-web-gui` for PEAS web application, `peas-app-android` for PEAS Android application, `peas-app` and `libpeas` for PEAS core application and library, and `peas-documentation` for PEAS documentation. The installed demonstration of all web-base parts is available via docker containers and stored in the `peas-docker` project `gitlab.com/brno-axe/peas/peas-docker`.

5.1.2 Web-based PEAS

The web-based demonstrator of PEAS contains 4 applications for each system entity, i.e., Issuer, Verifier, User and RA. The applications have been implemented by using JavaScript, Node.js, HTML, CSS and Vue web technologies. The third-party library, Vuetify (2.4.5), has been used for setting GUI components. For a simple deployment, the docker and YAML language have been used. The web-based PEAS Issuer provides issuing attributes, users' management and their revocation. The web-based PEAS Verifier provides users' verification, a selection of necessary attributes, and setting epochs. The web-based PEAS User sends registration and verification requests, and provides a user device and network management. The web-based PEAS RA aggregates logs of revoked users. The web-based PEAS implementation supports these communication protocols/interfaces between basic parts:

- REST API - for communication between the GUI and the web server when the communication is initiating by the GUI.
- WEB SOCKET - for communication between the GUI and the web server in real time, or when the communication is initiating by the web server.
- STDIO - for entering commands by web servers into console applications.

Figures 5.4 and 5.5 show dashboards of web-based PEAS Verifier and web-based PEAS Users. The user can set and register own personal attributes including various memberships, e.g., for parking lots or for an access to a building/room, via the web-based User application. Then, the user can connect to the verifier and can prove its attributes for getting the access. The verifier via the web-based PEAS Verifier can set which attributes should be disclosed (verified) and is able to log successful or faulty access attempts by various anonymous users.



- Fill information on this page
- Wait until your request is reviewed
- Done, now you are registered

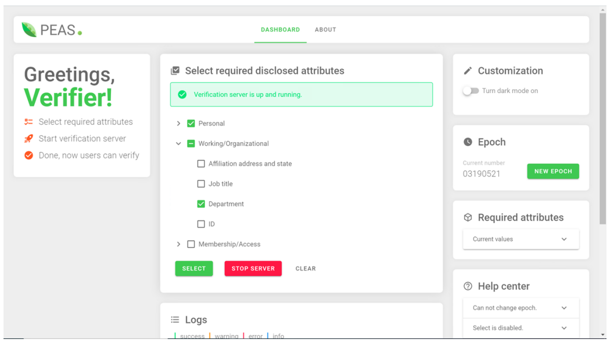
Successfuly registered!

Now you can try to verify against verifier service or register again as a new user.

If your PEAS service is hosted remotely, please update these setting first.

- Can not contact Issuer.
- What is authentication item?
- Application is offline.

Figure 5.4: Dashboard of web-based PEAS User.



- Select required attributes
- Start verification server
- Done, now users can verify

Verification server is up and running.

- ☑ Personal
- ▣ Working/Organizational
  - ☐ Affiliation address and state
  - ☐ Job title
  - ☑ Department
  - ☐ ID
- ☐ Membership/Access

Current number: 03190521

- Can not change epoch.
- Select is disabled.

Figure 5.5: Dashboard of web-based PEAS Verifier.

5.1.3 Android-based PEAS

To enhance PEAS for more real use cases, the Android-based PEAS application has been developed. The application represents a user side, i.e., Android-based PEAS user, and supports the communication with other PEAS entities such as Issuer, Verifier and RA. The Android application reflects current trends in usage of mobiles as personal items and enables users to replace smart cards. The application is based on the Java programming language version 14.0.1 (Android Studio, Arctic Fox 2020.3.1), and employs C++ functions and classes. Besides standard Android and Java libraries, the application deploys the com.herumi.mcl library. The Android-based PEAS works with up to 9 attributes that can be issued repeatedly. The application performs all necessary cryptographic operations in order to perform verification of holding the attributes. The application also keeps records of past communication in the history of events and offers the reset of settings. The communication with other system entities is based on NFC and Bluetooth technologies. Further, the personal access to the application can be secured by a 4-digit pin code or by a user fingerprint. Issuing the attributes can be disabled from security reasons. The application cannot be re-personalized repeatedly without the reset of all settings and personalized data.

5.2 Model-Driven GDPR Compliance Management

The goal of the Data Protection Officer tool (DPO tool, https://dpotool.cs.ut.ee/) is to support the model-driven GDPR compliance management approach (see activity DPO modeling and analysis in Fig.2.4). The approach consists of a few components: the GDPR method [63], the method to apply the GDPR model [78] for compliance checking, and the modelling language to annotate the business process model with the GDPR-related information.

In this section, we specifically describe the modelling language to annotate the business process model with the GDPR-related information. This modelling notation allow the analyst (e.g., DPO) to express explicitly the GDPR concepts in the business processes modelled using business process model and notation (BPMN)[6]. The BPMN language is extended following the language engineering principles [44], which includes extension of the language semantics, abstract syntax and concrete syntax. The language acronym is BPMN2GDPR. Next, we illustrate how BPMN2GDPR is used in the business processes for vehicle charge process.

5.2.1 BPMN2GDPR: BPMN extension to capture process compliance to GDPR regulation

Semantics of the BPMN2GDPR is based on the GDPR model, which presents the structural perspective of the regulation and provides the conceptual grounding for the modelling language extension. The GDPR model is discussed in [78].

Abstract Syntax of BPMN2GDPR is build on the BPMN metamodel as presented in Fig. 5.6 expressed as the UML class diagram. Here, DataHandler (e.g., Controller, Processor, ThirdParty, or Recipient) is the subclass of the BPMN Pool. DataSubject, ProcessingSystem, and FilingSystem are also subclasses of the BPMN Pool. The ProcessingSystem attributes (e.g., confidentiality, availability and others) express the security features of the processing features. Similarly, FilingSystem attributes chacraterise the type of storage.

SecurityMeasures and ProcessingTask measures are the subclass of the BPMN task. Hence, artifacts RecordOdProcessing, PersonalData,
textsfConsent and PrivacyPolicy are extensions of the BPMN DataObjects.

Each abstract syntax extension has their corresponding semantics mapped to the GDPR model concept. And similarly, each abstract syntax construct has its concrete syntax (visual) construct as discussed below.

[6]https://www.bpmn.org

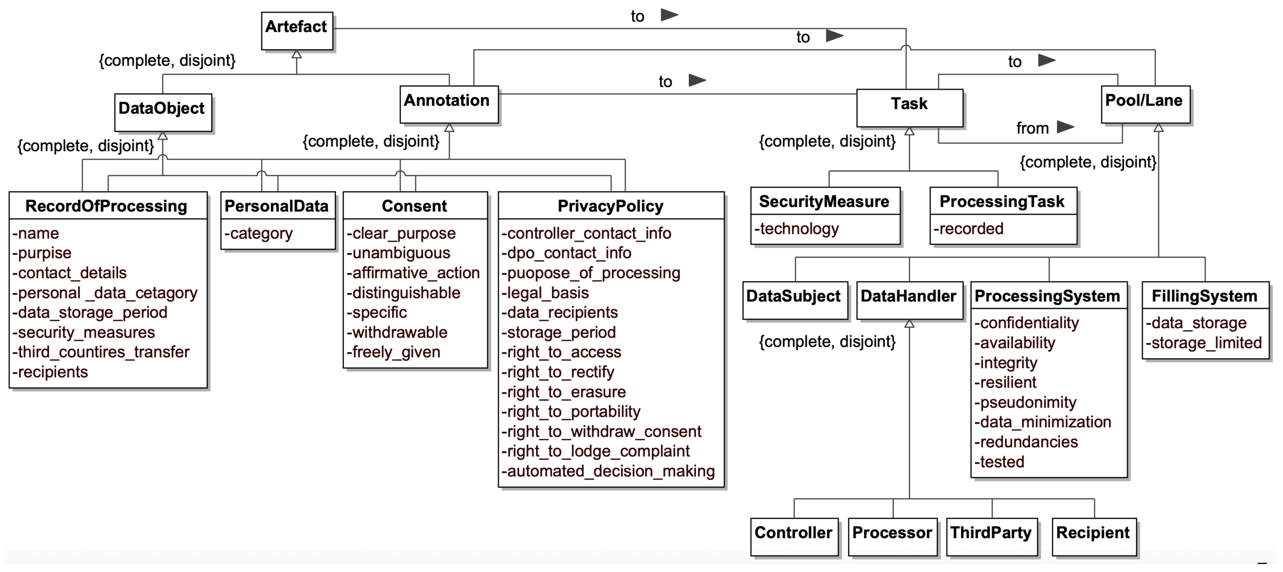


Figure 5.6**:** BPMN2GDPR Abstract Syntax

Concrete Syntax. The GDPR-BPMN modeling syntax is based on the GDPR Compliance Model and captures all the classes and their corresponding attributes using BPMN annotations attached to appropriate BPMN elements. The annotations are described using square brackets:

- Actors such as the controller are described [Controller], e.g., SmartBuilding [Controller] in Fig. 5.7.
- Artifacts are described using [Artifact] Consent, [Artifact] PrivacyPolicy or [Artifact] RecordsOfProcessing as illustrated in Fig. 5.7.
- Attributes of artifacts are described by annotating the appropriate artifact with labels corresponding to the attributes. Multiple attributes are separated by a space. For example, [clear_purpose] [unambiguous] [affirmative_action] [distinguishable] [specific] [withdrawable] [free_give] are attributes of [Artifact] Consent as illustrated in Fig. 5.7.
- Personal data is assigned by prefixing the appropriate data object label with the prefix [personal_data], as it is set [personal_data] PaymentDetails in Fig. 5.7.
- Data category is assigned by annotating the personal data object with the appropriate label, e.g., see [general] as annotation [personal_data] PaymentDetails in Fig. 5.7.
- Technical measures and processing task are described by prefixing the task label with the technical measure and the label [processing_task] in that order: [PKComputation][processing_task] 6. Check payment details as shown in Fig. 5.7.
- Attributes of the processing system, filing system and miscellaneous attributes are annotated on the controller's pool, e.g., see [confidentiality] [integrity] [availability [resilient] [pseudonimity] [data_minimization] [redundancies] [tested] [isms_standard] annotation to SmartBuilding in Fig. 5.7.
- Legal ground is described by annotating the controller's pool, e.g. [consent] annotation to SmartBuilding in Fig. 5.7.

5.2.2 Application of BPMN2GDPR in Intelligent Infrastructures

Let's consider the vehicle charge process in Fig. 5.7. It is in BPMN2GDPR prepared using the *bpmn.io* modelling tool. The process starts when the SmartBuilding displays privacy policy (see activity C1. Display privacy policy). Privacy policy is an artefact and it has a number of properties as illustrated by the data object PrivacyPolicy. The DriverPersonalDevice provides the consent (see activity C2. Provide consent). Here again the consent is a an artifact equipped with a number of properties as illustrated by the data object [Artifact] Consent. The process then continues with the encryption of the payment details (see C3. Encrypt payment details). This way illustrates the technical countermeasures (see also 6. Check payment details and C4. Decrypt payment details).

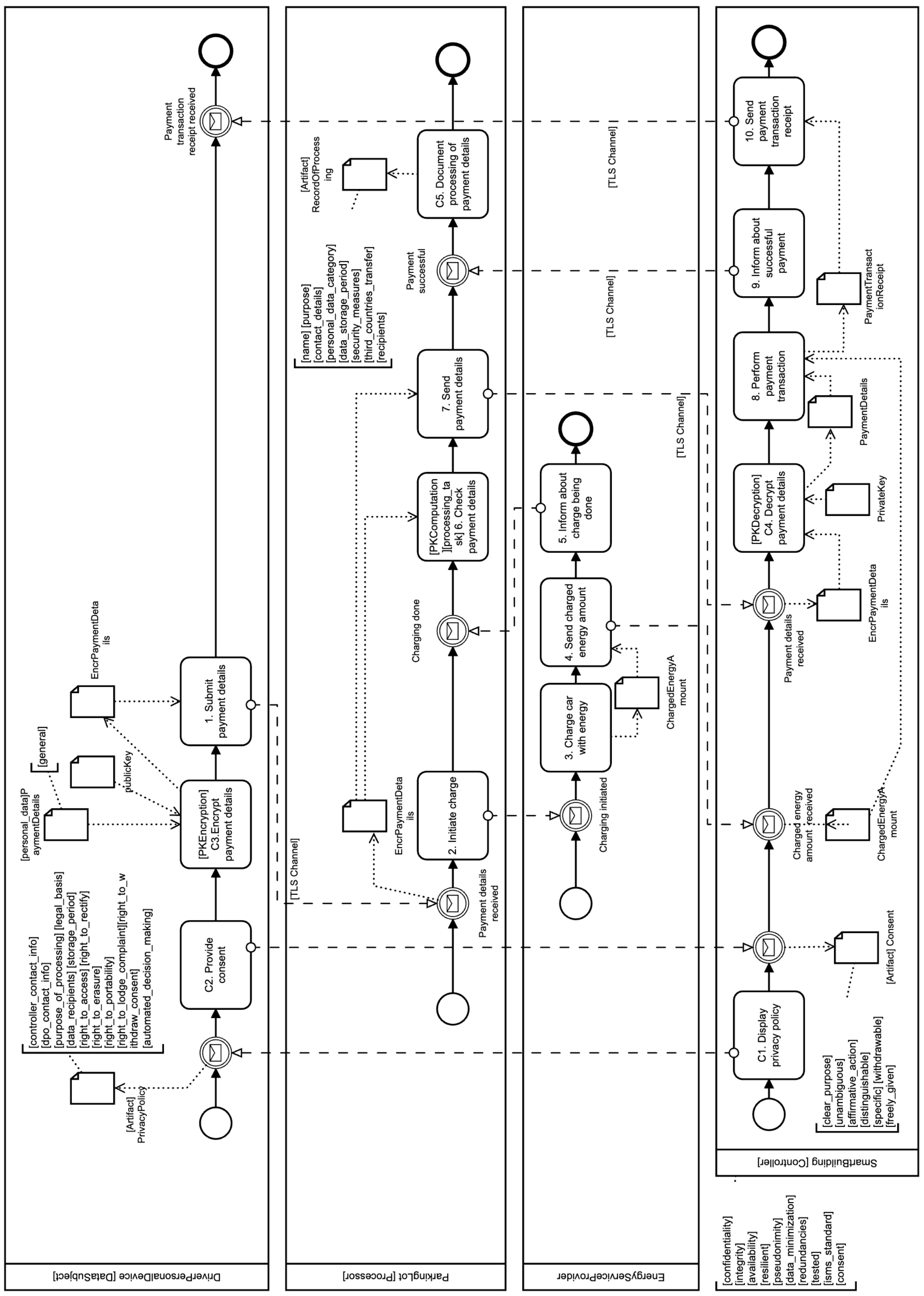

DriverPersonalDevice [DataSubject]
ParkingLot [Processor]
EnergyServiceProvider
SmartBuilding [Controller]
C1. Display privacy policy
C2. Provide consent
[PKEncryption] C3.Encrypt payment details
1. Submit payment details
2. Initiate charge
3. Charge car with energy
4. Send charged energy amount
5. Inform about charge being done
[PKComputation_task][processing_task] 6. Check payment details
7. Send payment details
[PKDecryption] C4. Decrypt payment details
8. Perform payment transaction
9. Inform about successful payment
10. Send payment transaction receipt
C5. Document processing of payment details
[Artifact] PrivacyPolicy
[Artifact] Consent
[Artifact] RecordOfProcessing
[personal_data]PaymentDetails
publicKey
EncrPaymentDetails
ChargedEnergyAmount
PaymentDetails
PrivateKey
PaymentTransactionReceipt
Payment details received
Charging initiated
Charging done
Charged energy amount received
Payment successful
Payment transaction receipt received
[TLS Channel]
[general]
[controller_contact_info] [dpo_contact_info] [purpose_of_processing] [legal_basis] [data_recipients] [storage_period] [right_to_access] [right_to_rectify] [right_to_erasure] [right_to_portability] [right_to_lodge_complaint][right_to_withdraw_consent] [automated_decision_making]
[name] [purpose] [contact_details] [personal_data_category] [data_storage_period] [security_measures] [third_countries_transfer] [recipients]
[clear_purpose] [unambiguous] [affirmative_action] [distinguishable] [specific] [withdrawable] [freely_given]
[confidentiality] [integrity] [availability] [resilient] [pseudonimity] [data_minimization] [redundancies] [tested] [isms_standard] [consent]

Figure 5.7: Vehicle Charge Process Annotated using BPMN2GDPR

After submitting the payment details, the ParkingLot initialise the charge. Charge is performed by the Energy Service Provider. The process is then continued with the 6. Check payment details, which is declared as the processing task in this model. After execution of processing task, Parking lot documents processing of parking details (see task C5. Document processing of payment details). Here the artifact of processing (see RecordOfProcessing) is created including its respective properties.

The process is completed when Smart Building performs payment transaction (see 8. Perform payment transaction), informs Parking Lot about the successful payment (see 9. Inform about successful payment), and sends payment transaction receipt to the Driver Personal Device (send 10. Send payment transaction receipt).

Hence the Smart Building could also be treated as the filing system where properties such as confidentiality, integrity, availability, resilience and etc. should be guaranteed. In addition, to the PKEncryption technical countermeasure, the process communication is secured using the [TLS Channel].

5.3 Privacy-preserving data processing: MC-SSE for privacy-preserving data processing

5.3.1 Introduction and motivation

The use of electronic services today results in the collection and storage of large amounts of data, whose handling can incur important privacy risks. The post-processing of the collected data though, can offer valuable insights on the service usage and drive decisions on functionality configurations. Privacy protection during this processing is important to ensure protection both from data breaches and unauthorized access and processing inside the system.

A privacy-preserving way of managing the collected data is required, in order to retain the benefits of the data processing, without compromising the data protection.

To achieve this goal we design and implement a searchable encryption based solution, that allows storage of the collected data in encrypted form, while allowing queries to be performed on the encrypted dataset.

In particular in this work, using as a building block the BIEX searchable encryption scheme [50], that achieves high efficiency and applicability, we propose an extension for the multi-client setting, while remaining practical and efficient. The proposed solution is implemented by extending the BIEX-based SSE library Clusion and tested for the parking use case data.

5.3.2 Searchable Encryption Background

SE, as illustrated in figure 5.8 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.

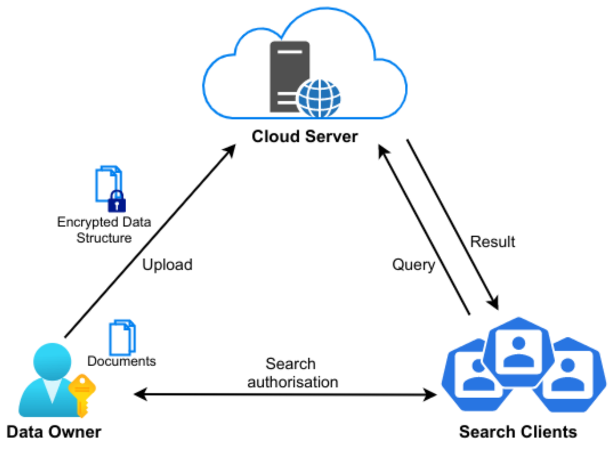


Figure 5.8: Searchable Encryption

Searchable encryption has been a very active research topic over the last decade, producing increasingly effective and applicable solutions [21, 29]. 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 category 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 main structure can be further enhanced to enable multi-client functionality and dynamic record additions to the encrypted dataset [25, 26, 49].

5.3.3 Multi-client BIEX

To design the proposed solution we use the BIEX SSE scheme [50], which achieves secure and efficient functionality with boolean query support. This scheme is implemented in the Clusion open source library [2].

In our system the goal to extend this single-user functionality to the multi-client setting, while retaining the properties of the BIEX scheme. This way authorised third parties can perform queries on the encrypted dataset.

The entities interacting in the system are the following:

- The Data owner (D), that creates an encrypted dataset and outsources it.
- The Storage and query server (E), that stores the encrypted dataset and performs queries on the encrypted dataset.
- Search clients (C) authorised to search on the encrypted dataset.

In the original BIEX SSE scheme, the following algorithms exist:

- Setup: taking as input a security parameter k and an index DB, it outputs the encrypted database EDB.
- Token generation: using as input a key and a vector of keywords w=$(w_1, ..., w_q)$, it creates a token consisting of sub-tokens for each keyword.
- Search: using as input the EDB and a search token tk, it outputs the set of document tags T, corresponding to the search query.

To extend the BIEX functionality to the multi-client setting, the BIEX token generation algorithm is modified and split into two stages: the Data owner authorization token creation phase and the Client search token customisation. During the first phase, D creates an authorisation token for the keywords they wish to allow C to search for. Given this authorisation token, C is able to construct an BIEX search token and submit a query to E, containing a combination of the authorised keywords.

With this extension the functionality and properties of the BIEX SSE scheme are preserved, with support for boolean queries and sub-linear search performance, while enabling multi-client search.

5.3.3.1 MC-BIEX library implementation

The MC-BIEX library has been implemented in Java, as an extension to the Clusion open source library [2]. The source code is available at: `https://github.com/atasidou/MC-Clusion`. The dataset parsing mechanisms has been modified for the needs of the application scenario. The library can process datasets consisting of one text file per record and keywords separated by line breaks. Lines can be exluded from keyword parsing by beginning with an exclamation mark. The evaluation of the implementation is presented in section 7.4.3.

5.3.4 Utilisation within the Intelligent Infrastructure

The proposed privacy-preserving data processing solution can be utilized in the HAII-T infrastructure, to manage collected data items of any service in the infrastructure and produce statistics on that data. At the moment the solution is 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 mechanism. For example, this data can be processed to produce statistics on the parking and charging spots demand, peak hours and availability, to determine appropriate policies for the service.

Chapter 6 Final demonstration

In this chapter we present the final demonstration of the HAII-T. The demonstration is organized according to the three workflows presented above in this document.

6.1 Legacy technologies management demonstration

This section presents the demonstration of the modules involved in the legacy technology management workflow described in Chapter 3.

6.1.1 PEPPER Demonstration

The demonstration scenario is show in Fig. 6.1. Two tokens are enrolled to a software update and contact tracing server, securely over the network. The last hop of which is low-power wireless (BLE) via a commodity wireless access point (Bluetooth), and in particular security bootstrap uses EDHOC {vucinic2021edhoc. The tokens are used to trace encounters between co-workers, and to evaluate exposure status, as well as notify exposure, depending on the interaction and the femto-container logic currently operating on the token. The latter is securely updated, over the network, to demonstrate system behaviour change, via the remote (update and tracing) server to which the tokens were initially enrolled.

A video showing the demo is available online [7].

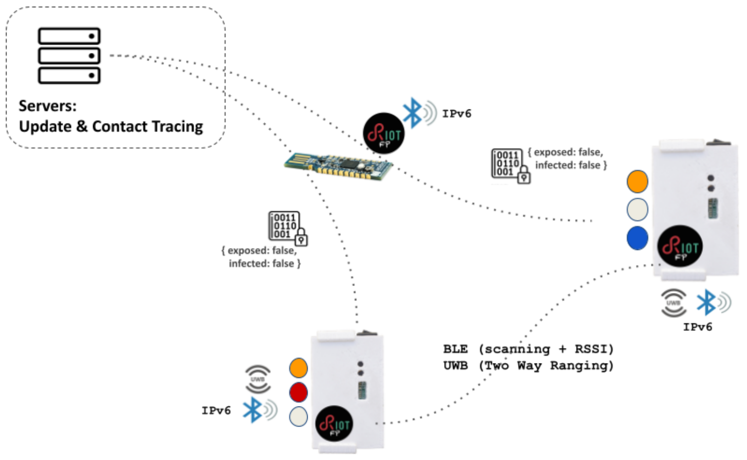


Figure 6.1: PEPPER demo scenario.

6.1.2 Perspectives regarding security-enhanced embedded IoT system software

With PEPPER we have demonstrated low-power contact tracing in a workplace environment with evolving sanitary regulation, using a secure software update workflow, and cheap hardware token prototypes. This base can be used to explore alternatives to the currently dominant contact tracing solution which is restricted to what Google/Apple provide as API.

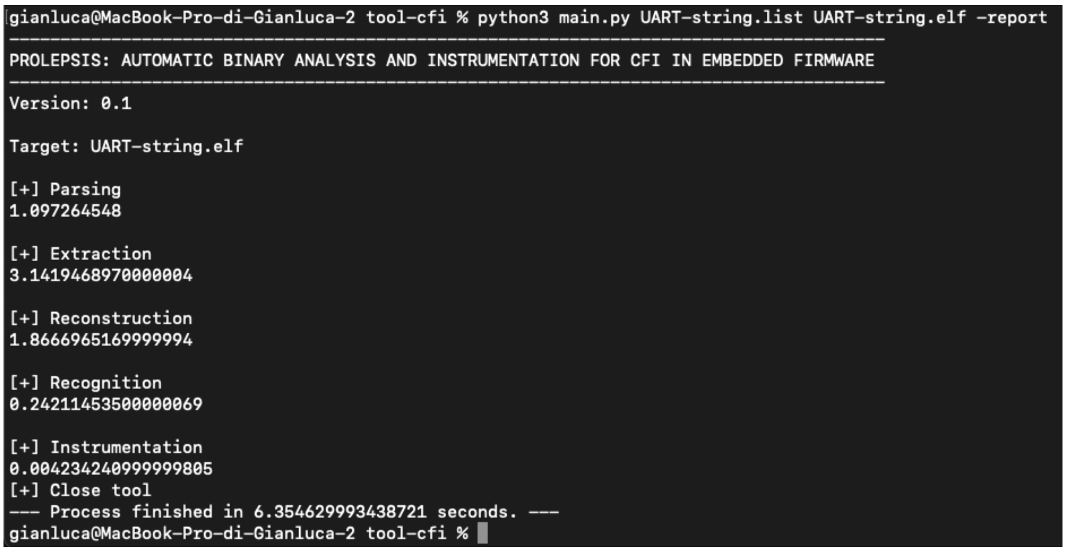
The open source platform we have developped for PEPPER has furthermore a wider applicability beyond this use case. Based on RIOT, SUIT end-to-end security, and femto-containers and standard low-power network protocols, our platform can provide security-enhanced embedded system software and secure updates over the network for general-purpose IoT OS firmware update or for more lightweight and more specific IoT software modules hosted and isolated in femto-containers.

6.1.3 Edge-devices control-flow integrity demonstration

This section presents some passages from a practical demonstration of how the analysis and instrumentation technique for edge devices described in 3.2 works.

In principle, the technique can be applied to all embedded software run within the infrastructure. For reasons of simplicity, here are presented the results obtained by applying the technique on a small-sized benchmarking firmware, `UART-string`, adapted from the `stringsearch` program, available online at the MiBench website[1]. The application performs a stupid task, i.e., counts the occurrences of a substring over a text.

The application was compiled for an ST Microelectronics STM32F429 processor (with ARM ISA version 7). An application consisting of 12960 machine instructions was obtained (see the results obtained in [40]). Figure 6.2 shows the execution screen of the tool, artistically called *PROLEPSIS* by the authors. The script takes as parameter the application disassembled listing (.list) as input, as well as the ELF executable file itself. The screen shows the execution times of the 5 phases described in Figure 3.4 (in seconds), and the total execution time. The `-report` option allows to generate a detailed report of the result in a file.

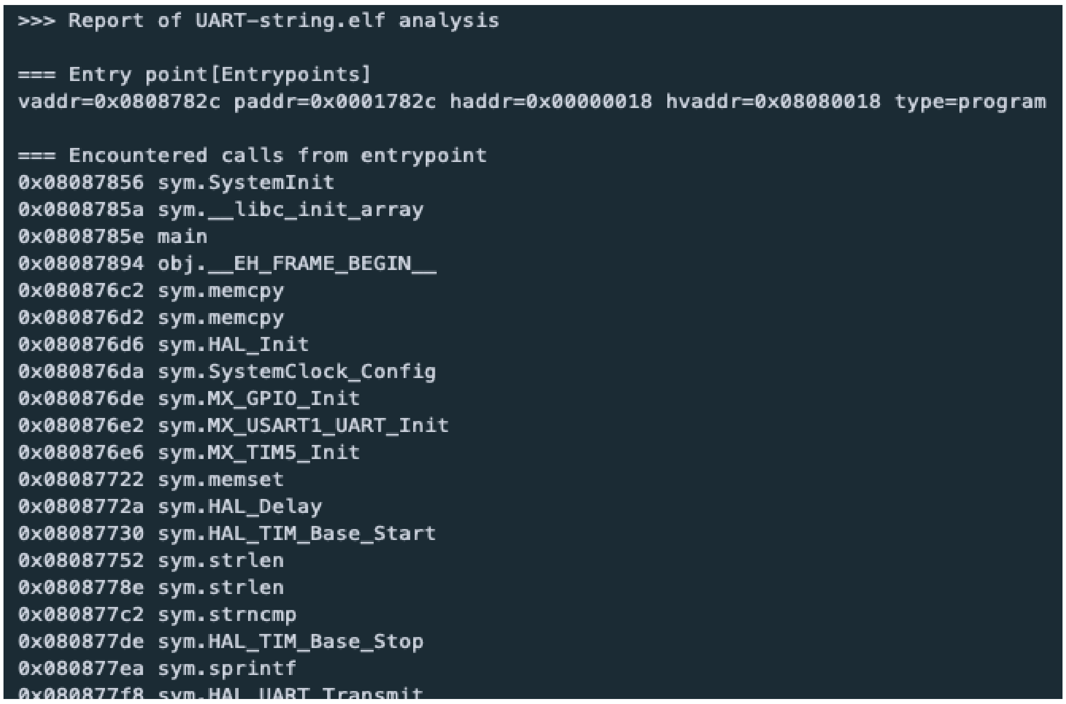


```
gianluca@MacBook-Pro-di-Gianluca-2 tool-cfi % python3 main.py UART-string.list UART-string.elf -report
-------------------------------------------------------------------------------------
PROLEPSIS: AUTOMATIC BINARY ANALYSIS AND INSTRUMENTATION FOR CFI IN EMBEDDED FIRMWARE
-------------------------------------------------------------------------------------
Version: 0.1

Target: UART-string.elf

[+] Parsing
1.097264548

[+] Extraction
3.1419468970000004

[+] Reconstruction
1.8666965169999994

[+] Recognition
0.24211453500000069

[+] Instrumentation
0.004234240999999805
[+] Close tool
--- Process finished in 6.354629993438721 seconds. ---
gianluca@MacBook-Pro-di-Gianluca-2 tool-cfi %
```

Figure 6.2: Execution screen of the analysis and instrumentation tool for control-flow integrity.

Figure 6.3 shows the beginning of this report file (deliberately truncated for its length), which contains information on the entry point of the code, and on the functions that the Extraction algorithm traces from that point on. For each call, the address at which it takes place is reported.

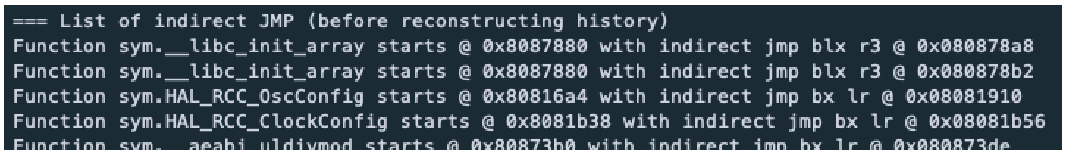
[1] https://vhosts.eecs.umich.edu/mibench/office.tar.gz. Mibench is a free representative embedded benchmark suite.



```
>>> Report of UART-string.elf analysis

=== Entry point[Entrypoints]
vaddr=0x0808782c paddr=0x0001782c haddr=0x00000018 hvaddr=0x08080018 type=program

=== Encountered calls from entrypoint
0x08087856 sym.SystemInit
0x0808785a sym.__libc_init_array
0x0808785e main
0x08087894 obj.__EH_FRAME_BEGIN__
0x080876c2 sym.memcpy
0x080876d2 sym.memcpy
0x080876d6 sym.HAL_Init
0x080876da sym.SystemClock_Config
0x080876de sym.MX_GPIO_Init
0x080876e2 sym.MX_USART1_UART_Init
0x080876e6 sym.MX_TIM5_Init
0x08087722 sym.memset
0x0808772a sym.HAL_Delay
0x08087730 sym.HAL_TIM_Base_Start
0x08087752 sym.strlen
0x0808778e sym.strlen
0x080877c2 sym.strncmp
0x080877de sym.HAL_TIM_Base_Stop
0x080877ea sym.sprintf
0x080877f8 sym.HAL_UART_Transmit
```

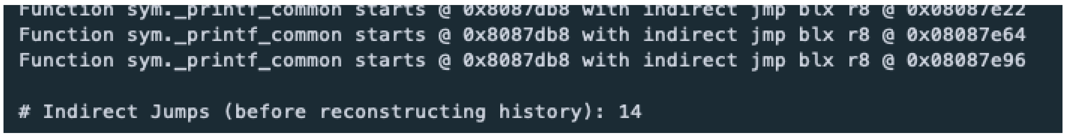
Figure 6.3: Report on the application entry point and cascaded calls from it.

Figures 6.4 and 6.5 show the beginning and end of the list of indirect edges found within the application, with primary reference to the function they belong to, and reporting the instruction itself that originates the edge and its address in memory. This result is obtained before the Reconstruction phase, so it is still to be determined how many of these edges are actually to be protected.



```
=== List of indirect JMP (before reconstructing history)
Function sym.__libc_init_array starts @ 0x8087880 with indirect jmp blx r3 @ 0x080878a8
Function sym.__libc_init_array starts @ 0x8087880 with indirect jmp blx r3 @ 0x080878b2
Function sym.HAL_RCC_OscConfig starts @ 0x80816a4 with indirect jmp bx lr @ 0x08081910
Function sym.HAL_RCC_ClockConfig starts @ 0x8081b38 with indirect jmp bx lr @ 0x08081b56
Function sym.__aeabi_uldivmod starts @ 0x80873b0 with indirect jmp bx lr @ 0x080873de
```

Figure 6.4: Detail from the beginning of the indirect edge list within the report.

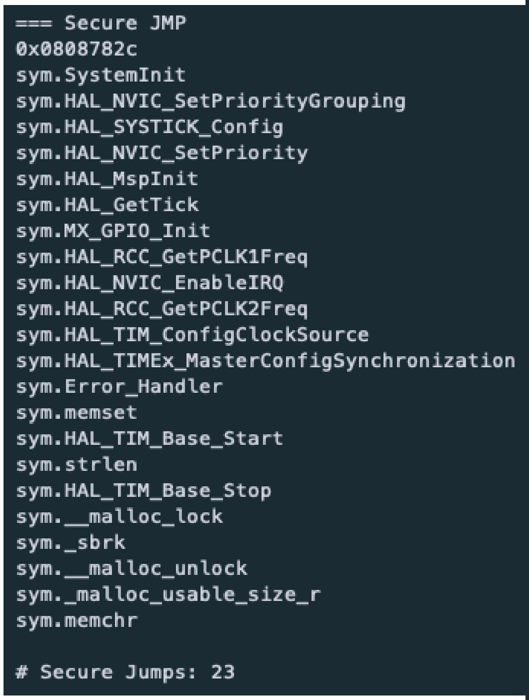


```
Function sym._printf_common starts @ 0x8087db8 with indirect jmp blx r8 @ 0x08087e22
Function sym._printf_common starts @ 0x8087db8 with indirect jmp blx r8 @ 0x08087e64
Function sym._printf_common starts @ 0x8087db8 with indirect jmp blx r8 @ 0x08087e96

# Indirect Jumps (before reconstructing history): 14
```

Figure 6.5: Detail from the beginning of the indirect edge list within the report.

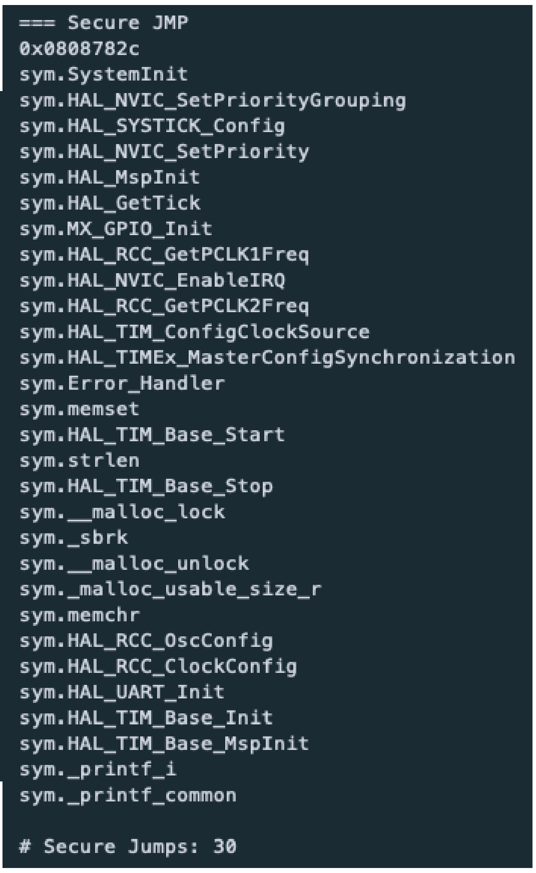
Figure 6.6 instead shows the result of the Reconstruction phase. The left subfigure shows the status of the edges identified as secure before this step. The subfigure on the right, on the other hand, shows a bigger list, populated by the indirect edges that have been labeled as secure after the reconstruction of their origin tree. Therefore, there are only 7 remaining insecure edges requiring instrumentation. This instrumentation causes an overhead in terms of code area which, added to that necessary for the protection of ISRs (see 3.2.1), reaches 1.98% (cfr. [40])[2].

[2]Table I contained in the aforementioned paper erroneously reports "Insec. Edges" instead of "Indir. Edges" due to a typo.



```
=== Secure JMP
0x0808782c
sym.SystemInit
sym.HAL_NVIC_SetPriorityGrouping
sym.HAL_SYSTICK_Config
sym.HAL_NVIC_SetPriority
sym.HAL_MspInit
sym.HAL_GetTick
sym.MX_GPIO_Init
sym.HAL_RCC_GetPCLK1Freq
sym.HAL_NVIC_EnableIRQ
sym.HAL_RCC_GetPCLK2Freq
sym.HAL_TIM_ConfigClockSource
sym.HAL_TIMEx_MasterConfigSynchronization
sym.Error_Handler
sym.memset
sym.HAL_TIM_Base_Start
sym.strlen
sym.HAL_TIM_Base_Stop
sym.__malloc_lock
sym._sbrk
sym.__malloc_unlock
sym._malloc_usable_size_r
sym.memchr

# Secure Jumps: 23
```

(a)

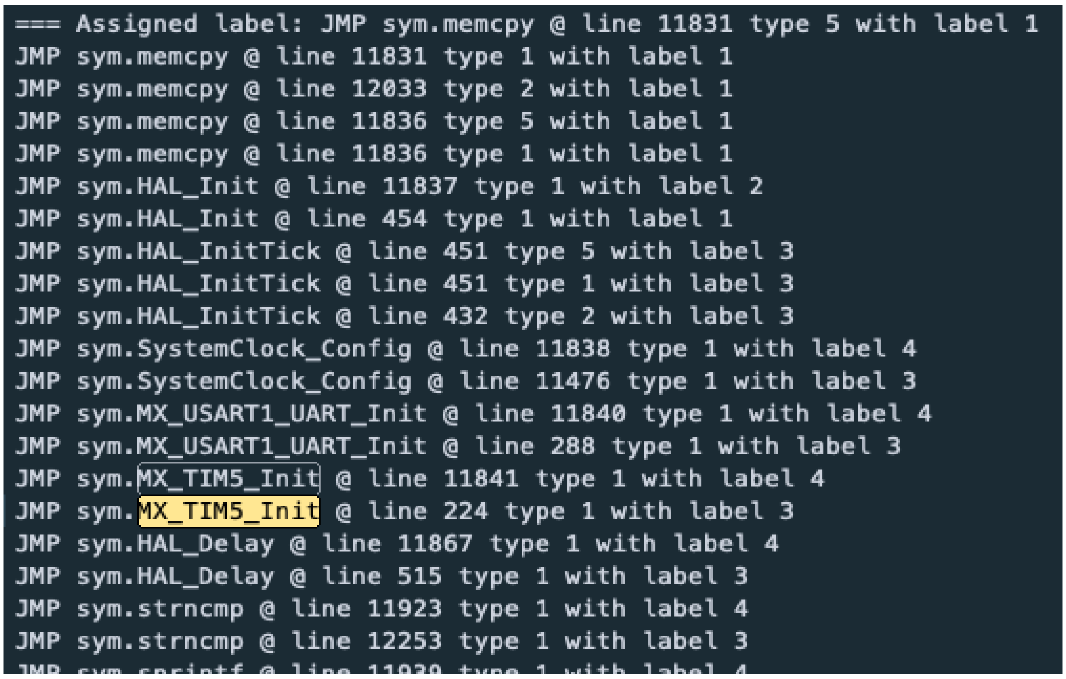


```
=== Secure JMP
0x0808782c
sym.SystemInit
sym.HAL_NVIC_SetPriorityGrouping
sym.HAL_SYSTICK_Config
sym.HAL_NVIC_SetPriority
sym.HAL_MspInit
sym.HAL_GetTick
sym.MX_GPIO_Init
sym.HAL_RCC_GetPCLK1Freq
sym.HAL_NVIC_EnableIRQ
sym.HAL_RCC_GetPCLK2Freq
sym.HAL_TIM_ConfigClockSource
sym.HAL_TIMEx_MasterConfigSynchronization
sym.Error_Handler
sym.memset
sym.HAL_TIM_Base_Start
sym.strlen
sym.HAL_TIM_Base_Stop
sym.__malloc_lock
sym._sbrk
sym.__malloc_unlock
sym._malloc_usable_size_r
sym.memchr
sym.HAL_RCC_OscConfig
sym.HAL_RCC_ClockConfig
sym.HAL_UART_Init
sym.HAL_TIM_Base_Init
sym.HAL_TIM_Base_MspInit
sym._printf_i
sym._printf_common

# Secure Jumps: 30
```

(b)

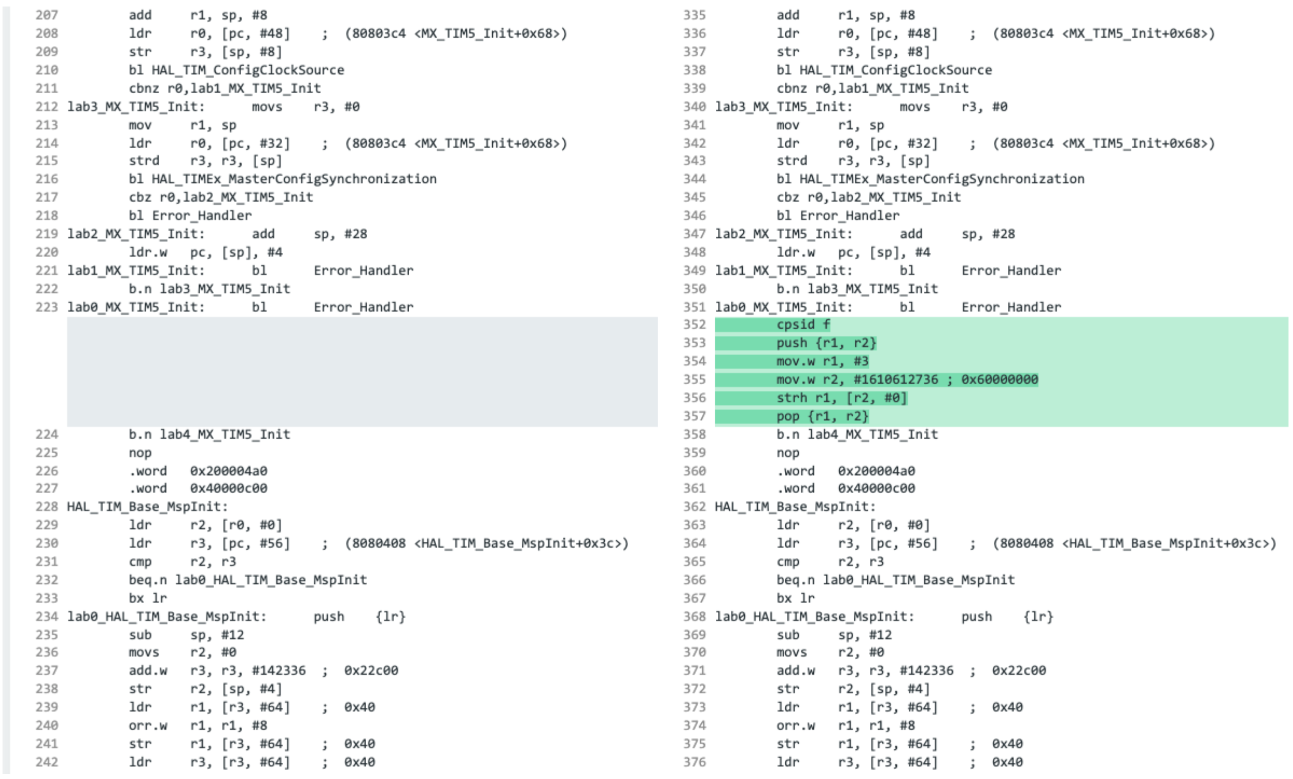
Figure 6.6: Secure control-flow transfers before and after the Reconstruction phase.

Finally, Figure 6.7 shows the detail on the Instrumentation phase. The highlighted line shows that inside the `MX_TIM5_Init` function, at line 224 of the original disassembled executable, an instrumentation relating to edge type 1 (see 3.2.1) has been inserted, with the assignment of a random label (in this case, 3). Figure 6.8 shows the difference between the disassembled of the original executable file and the disassembled of the instrumented one, according to the technique already described in [64], whereby this label is passed to the monitor on a standard shared address. In particular, this instrumentation here is part of a more complex one which is needed to ensure that the `Error_Handler` function returns exactly to `MX_TIM5_Init` if called from there.



```
=== Assigned label: JMP sym.memcpy @ line 11831 type 5 with label 1
JMP sym.memcpy @ line 11831 type 1 with label 1
JMP sym.memcpy @ line 12033 type 2 with label 1
JMP sym.memcpy @ line 11836 type 5 with label 1
JMP sym.memcpy @ line 11836 type 1 with label 1
JMP sym.HAL_Init @ line 11837 type 1 with label 2
JMP sym.HAL_Init @ line 454 type 1 with label 1
JMP sym.HAL_InitTick @ line 451 type 5 with label 3
JMP sym.HAL_InitTick @ line 451 type 1 with label 3
JMP sym.HAL_InitTick @ line 432 type 2 with label 3
JMP sym.SystemClock_Config @ line 11838 type 1 with label 4
JMP sym.SystemClock_Config @ line 11476 type 1 with label 3
JMP sym.MX_USART1_UART_Init @ line 11840 type 1 with label 4
JMP sym.MX_USART1_UART_Init @ line 288 type 1 with label 3
JMP sym.MX_TIM5_Init @ line 11841 type 1 with label 4
JMP sym.MX_TIM5_Init @ line 224 type 1 with label 3
JMP sym.HAL_Delay @ line 11867 type 1 with label 4
JMP sym.HAL_Delay @ line 515 type 1 with label 3
JMP sym.strncmp @ line 11923 type 1 with label 4
JMP sym.strncmp @ line 12253 type 1 with label 3
JMP sym.sprintf @ line 11939 type 1 with label 4
```

Figure 6.7: Details on the instrumentation sites with edge type and assigned label.



```
        add     r1, sp, #8
        ldr     r0, [pc, #48]    ;  (80803c4 <MX_TIM5_Init+0x68>)
        str     r3, [sp, #8]
        bl HAL_TIM_ConfigClockSource
        cbnz r0,lab1_MX_TIM5_Init
lab3_MX_TIM5_Init:      movs    r3, #0
        mov     r1, sp
        ldr     r0, [pc, #32]    ;  (80803c4 <MX_TIM5_Init+0x68>)
        strd    r3, r3, [sp]
        bl HAL_TIMEx_MasterConfigSynchronization
        cbz r0,lab2_MX_TIM5_Init
        bl Error_Handler
lab2_MX_TIM5_Init:      add     sp, #28
        ldr.w   pc, [sp], #4
lab1_MX_TIM5_Init:      bl      Error_Handler
        b.n lab3_MX_TIM5_Init
lab0_MX_TIM5_Init:      bl      Error_Handler
        b.n lab4_MX_TIM5_Init
        nop
        .word   0x200004a0
        .word   0x40000c00
HAL_TIM_Base_MspInit:
        ldr     r2, [r0, #0]
        ldr     r3, [pc, #56]    ;  (8080408 <HAL_TIM_Base_MspInit+0x3c>)
        cmp     r2, r3
        beq.n lab0_HAL_TIM_Base_MspInit
        bx lr
lab0_HAL_TIM_Base_MspInit:      push    {lr}
        sub     sp, #12
        movs    r2, #0
        add.w   r3, r3, #142336  ;  0x22c00
        str     r2, [sp, #4]
        ldr     r1, [r3, #64]    ;  0x40
        orr.w   r1, r1, #8
        str     r1, [r3, #64]    ;  0x40
        ldr     r3, [r3, #64]    ;  0x40
```

```
        add     r1, sp, #8
        ldr     r0, [pc, #48]    ;  (80803c4 <MX_TIM5_Init+0x68>)
        str     r3, [sp, #8]
        bl HAL_TIM_ConfigClockSource
        cbnz r0,lab1_MX_TIM5_Init
lab3_MX_TIM5_Init:      movs    r3, #0
        mov     r1, sp
        ldr     r0, [pc, #32]    ;  (80803c4 <MX_TIM5_Init+0x68>)
        strd    r3, r3, [sp]
        bl HAL_TIMEx_MasterConfigSynchronization
        cbz r0,lab2_MX_TIM5_Init
        bl Error_Handler
lab2_MX_TIM5_Init:      add     sp, #28
        ldr.w   pc, [sp], #4
lab1_MX_TIM5_Init:      bl      Error_Handler
        b.n lab3_MX_TIM5_Init
lab0_MX_TIM5_Init:      bl      Error_Handler
        cpsid f
        push {r1, r2}
        mov.w r1, #3
        mov.w r2, #1610612736 ; 0x60000000
        strh r1, [r2, #0]
        pop {r1, r2}
        b.n lab4_MX_TIM5_Init
        nop
        .word   0x200004a0
        .word   0x40000c00
HAL_TIM_Base_MspInit:
        ldr     r2, [r0, #0]
        ldr     r3, [pc, #56]    ;  (8080408 <HAL_TIM_Base_MspInit+0x3c>)
        cmp     r2, r3
        beq.n lab0_HAL_TIM_Base_MspInit
        bx lr
lab0_HAL_TIM_Base_MspInit:      push    {lr}
        sub     sp, #12
        movs    r2, #0
        add.w   r3, r3, #142336  ;  0x22c00
        str     r2, [sp, #4]
        ldr     r1, [r3, #64]    ;  0x40
        orr.w   r1, r1, #8
        str     r1, [r3, #64]    ;  0x40
        ldr     r3, [r3, #64]    ;  0x40
```

Figure 6.8: Comparison between the original and instrumented application.

6.1.4 Protocol verification demonstration

6.1.4.1 Model checking – formal modeling of IoT protocols

A demonstration of a formal modeling of IoT protocols based on model checking is presented in this section. Demonstration components of phase 1 and phase 2 demonstrators are presented on Figure 6.9.

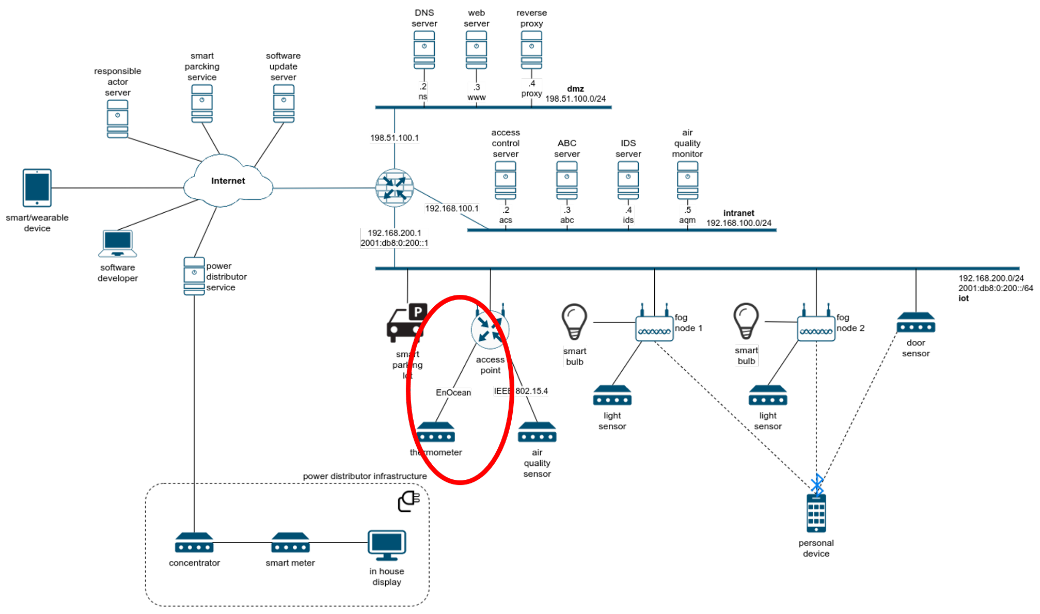


Figure 6.9: Model checking demo architecture

Visualisation of modeled unidirectional and bidirectional teach-in procedures, as well as one resulting console output example are presented on Figures 6.10, 6.11 and 6.12.

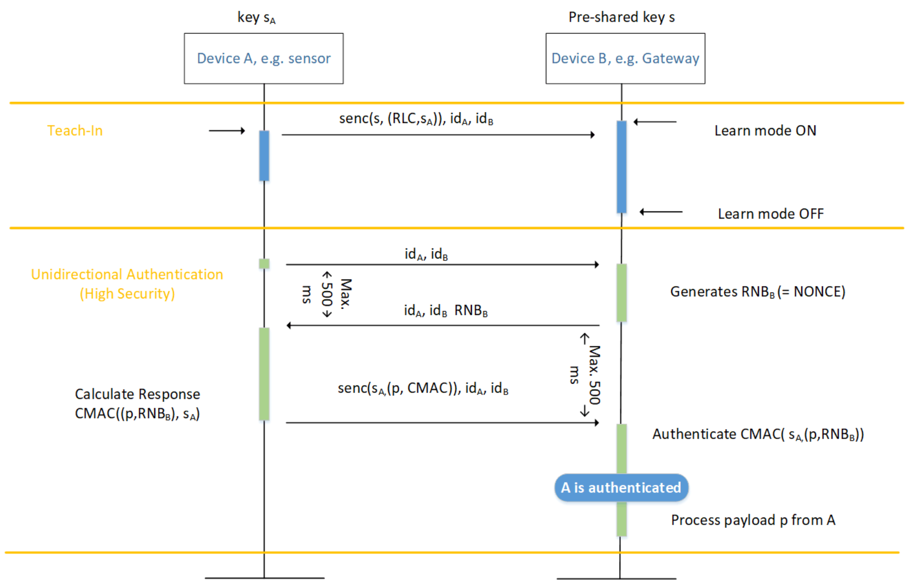

key s_A
Pre-shared key s
Device A, e.g. sensor
Device B, e.g. Gateway
Teach-In
senc(s, (RLC,s_A)), id_A, id_B
Learn mode ON
Learn mode OFF
Unidirectional Authentication (High Security)
id_A, id_B
Max. 500 ms
Generates RNB_B (= NONCE)
id_A, id_B RNB_B
Calculate Response CMAC((p,RNB_B), s_A)
senc(s_A,(p, CMAC)), id_A, id_B
Max. 500 ms
Authenticate CMAC(s_A(p,RNB_B))
A is authenticated
Process payload p from A

Figure 6.10: Unidirectional Teach-in and authentication model in EnOcean protocol[46]

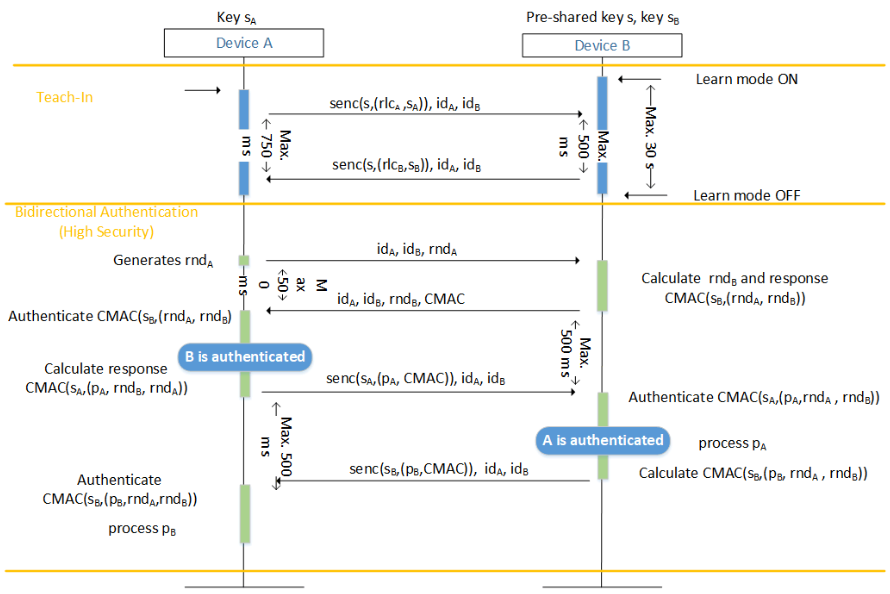
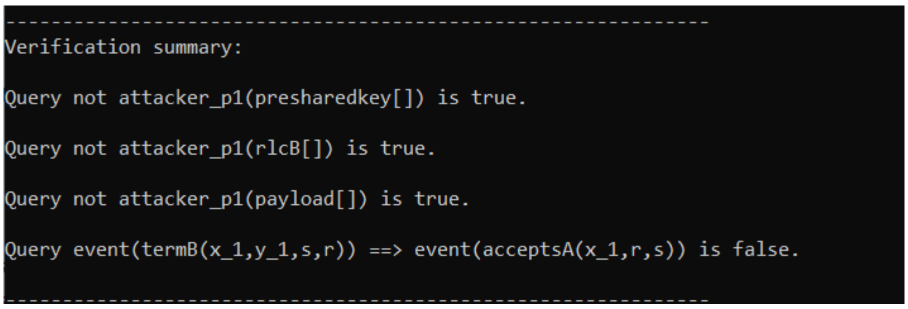

Key s_A
Device A
Pre-shared key s, key s_B
Device B
Teach-In
Learn mode ON
$senc(s,(rlc_A,s_A))$, id_A, id_B
Max. 750 ms
Max. 500 ms
Max. 30 s
$senc(s,(rlc_B,s_B))$, id_A, id_B
Learn mode OFF
Bidirectional Authentication (High Security)
Generates rnd_A
id_A, id_B, rnd_A
Calculate rnd_B and response $CMAC(s_B,(rnd_A, rnd_B))$
Max. 500 ms
id_A, id_B, rnd_B, CMAC
Authenticate $CMAC(s_B,(rnd_A, rnd_B)$
Max. 500 ms
B is authenticated
Calculate response $CMAC(s_A,(p_A, rnd_B, rnd_A))$
$senc(s_A,(p_A, CMAC))$, id_A, id_B
Authenticate $CMAC(s_A,(p_A,rnd_A, rnd_B))$
Max. 500 ms
A is authenticated
process p_A
$senc(s_B,(p_B,CMAC))$, id_A, id_B
Calculate $CMAC(s_B,(p_B, rnd_A, rnd_B))$
Authenticate $CMAC(s_B,(p_B,rnd_A,rnd_B))$
process p_B

Figure 6.11: Bidirectional Teach-in and authentication model in EnOcean protocol[46]



```
----------------------------------------------------------------
Verification summary:

Query not attacker_p1(presharedkey[]) is true.

Query not attacker_p1(rlcB[]) is true.

Query not attacker_p1(payload[]) is true.

Query event(termB(x_1,y_1,s,r)) ==> event(acceptsA(x_1,r,s)) is false.

----------------------------------------------------------------
```

Figure 6.12: Unidirectional teach-in procedure in EnOcean protocol - console output example

Our obtained results shows that in case of a secure key exchange, the communication by using the high security level is secure. If the teach-in with a pre-shared key is used, there seems to be a potential weakness in the security specification. We double checked the security specification, which states: “The security mechanism may transform the DATA and R-ORG field of the non-secure message. Other fields like message sender ID, receiver ID, repeater counter are not affected or altered. The RLC and CMAC should be added. Not modified fields like sender ID, receiver ID or repeater counter are not depicted through the chapter when a message is represented.” The security specification also states which parts of the message are encrypted in case of wireless teach-in namely: “For the encryption pre-shared key is used. Encrypted are the RLC and KEY”.

We can therefore conclude that the prevention of replay attacks is based solely on the time limit and the obligation to (manually) set a device in learning mode and press the teach-in trigger. As we have seen, in case of a teach-in with pre-shared key, there is a small time window in which a potential attacker can interact and disturb a proper authentication. Nevertheless, the time limit raises the effort and an attacker needs to be aware of the ongoing teach-in procedure or automate the process of actively listening to teach-in messages continuously. Furthermore even in this case, although the attacker can interfere with a proper authentication, the (strong) secrecy of the payload and the RLC’s can still be kept.

We further want to emphasize that our model confirms that a proper authentication could also be disturbed unintentionally, i.e. a set-up of different pairs in neighboring rooms in a house.

6.1.4.2 Probabilistic model checking – risk analysis in IoT environment

A demonstration of a risk analysis of smart home IoT network based on probabilistic model checking is presented in this section. As previously stated, this use scenario focuses on a probabilistic risk analysis of two different smart home system configurations through threat modeling and model checking. Demonstration components of phase 1 and phase 2 demonstrators are presented on Figures 6.13 and 6.14, respectively.

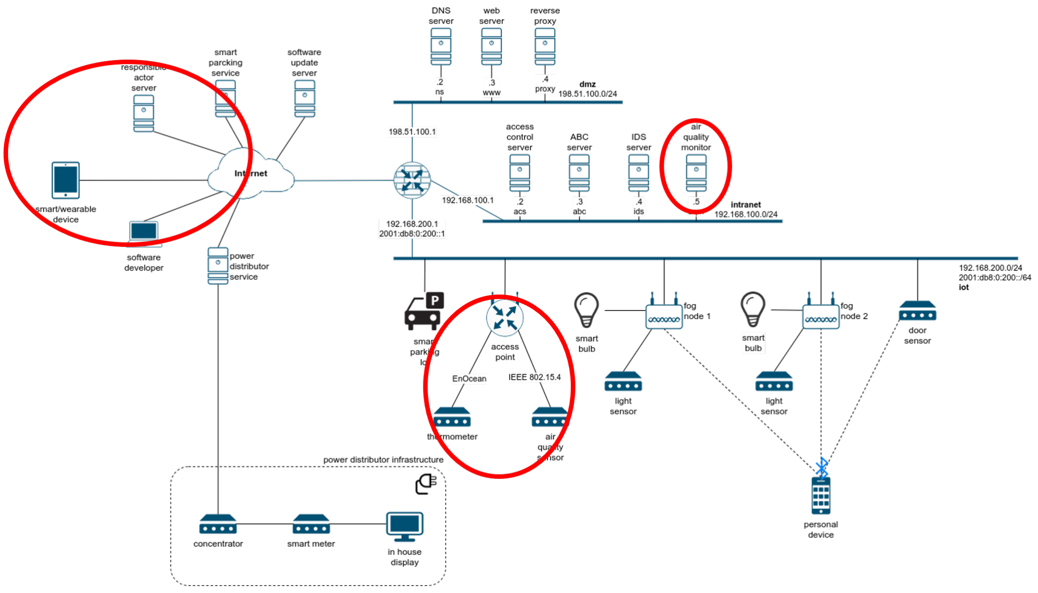


Figure 6.13: Probabilistic model checking demo architecture - phase 1

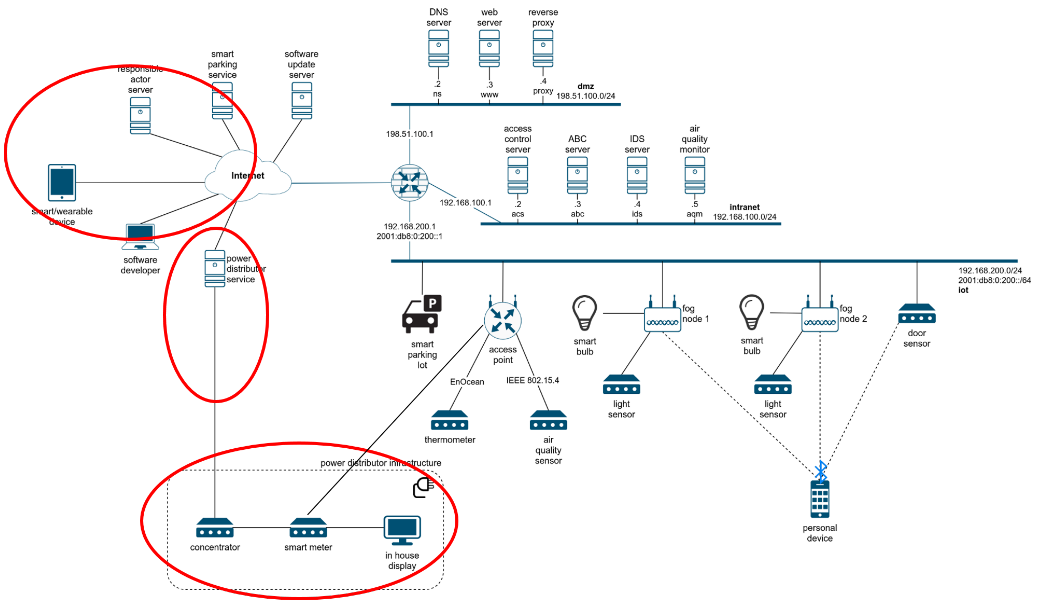
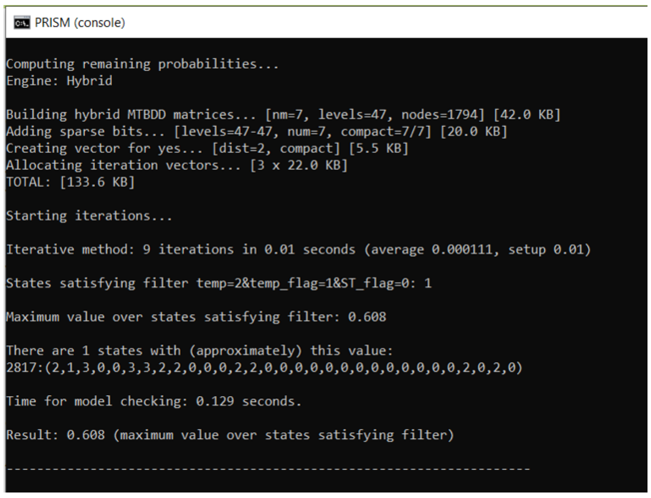


Figure 6.14: Probabilistic model checking demo architecture - phase 2

Previously described preconditions and input data (Section 3.3.2) are modeled in PRISM model checker in form of a sequential flow of events. The output of this process is probability of successful attack within given assumptions. All tests are conducted for different cost values – 1-5. Resulting console output example and risk probabilities for two scenarios are presented on figures below.



PRISM (console)

```
Computing remaining probabilities...
Engine: Hybrid

Building hybrid MTBDD matrices... [nm=7, levels=47, nodes=1794] [42.0 KB]
Adding sparse bits... [levels=47-47, num=7, compact=7/7] [20.0 KB]
Creating vector for yes... [dist=2, compact] [5.5 KB]
Allocating iteration vectors... [3 x 22.0 KB]
TOTAL: [133.6 KB]

Starting iterations...

Iterative method: 9 iterations in 0.01 seconds (average 0.000111, setup 0.01)

States satisfying filter temp=2&temp_flag=1&ST_flag=0: 1

Maximum value over states satisfying filter: 0.608

There are 1 states with (approximately) this value:
2817:(2,1,3,0,0,3,3,2,2,0,0,0,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,2,0)

Time for model checking: 0.129 seconds.

Result: 0.608 (maximum value over states satisfying filter)

---------------------------------------------------------------------
```

Figure 6.15: Hijacking of smart HVAC risk analysis - console output example

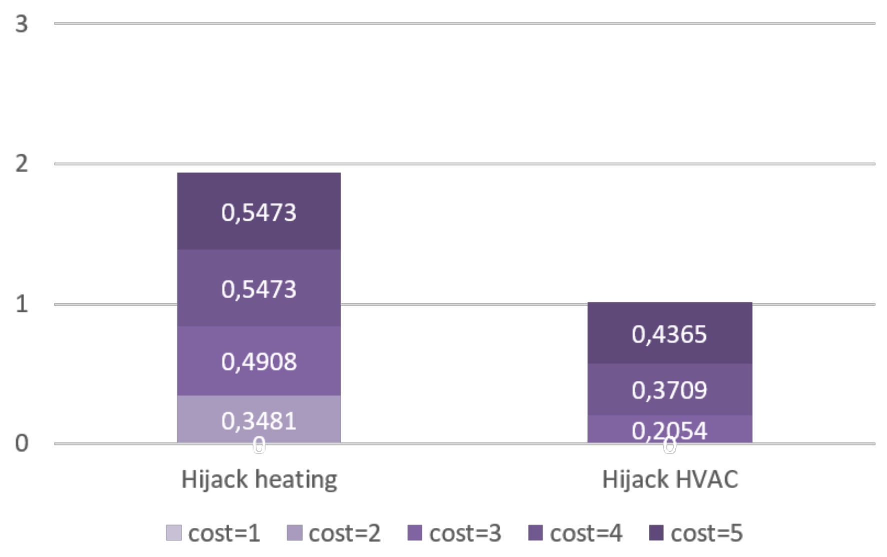


Figure 6.16: Hijacking of smart HVAC risk analysis - resulting risk probabilities; cost: maximum number of

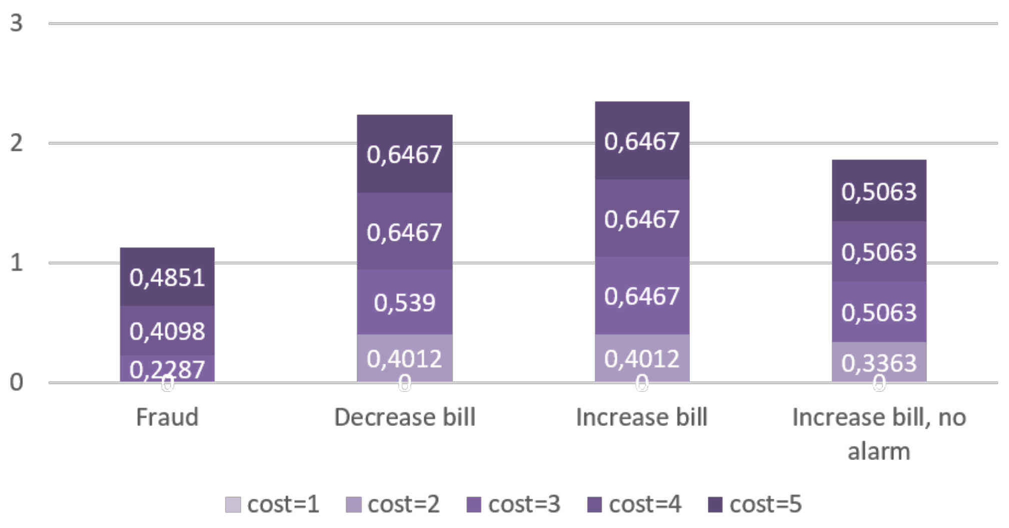


Figure 6.17: Hijacking of smart meter risk analysis - resulting risk probabilities; cost: maximum number of

The results of the Hijacking of smart HVAC risk analysis show that the attack hijack HVAC (attack 1.2) is less likely to be successful comparing to the attack hijack heating (attack 1.1). This is due to the fact that in order to successfully conduct attack 1.2 more vulnerabilities need to be exploited. The attack 1.1 requires exploitation of two vulnerabilities, while the attack 1.2 requires exploitation of at least three vulnerabilities to be successful.

The results of the Hijacking of smart meter risk analysis show that the attack fraud (attack 2.1) has the least likelihood to be successful, under given assumptions. Attack 2.1 requires the exploitation of at least three vulnerabilities, while the other three scenarios considered in this use case can be conducted by exploiting only two vulnerabilities. Attack 2.1 requires the exploitation of at least five vulnerabilities for maximum likelihood, attack decrease bill (attack 2.2) requires exploitation of three, while attacks increase bill (attack 2.3) and increase bill, no alarm (attack 2.4) only require the exploitation of two vulnerabilities for the maximum likelihood. It has to be emphasized that although both attacks 2.3 (or 2.2) and 2.4 require the exploitation of a minimum of two vulnerabilities to be successful, the risk exposure scores of 2.4 are lower because this attack requires the exploitation of parts of the system that have a higher security level (reflected in the lower exploitation probability).

Our results show that in a smart home environment, an attacker needs to exploit at least two vulnerabilities to successfully perform an attack, and that more complex attack scenarios requires successful exploitation of at least three vulnerabilities. These results clearly indicate that all use cases could benefit from a layered security approach, that includes several protection mechanisms in place. An additional conclusion is that this methodology provides guidance to address bigger and more complex scenarios. For example, it can be applied in order to address security in smart cities.

6.1.5 RIOT-AKA Authentication demonstration

We have to remind that ”RIOT-AKA” is a protocol implemented in order to authenticate IoT roaming devices into federated networks. We recommend to read Section 3.4 to recapitulate all of the phases and details of the exchanged messages between the three parties. In this demonstration we show only a generic, virtualized environment in order to represent a real world deployment scenario of the scheme. The placement of all involved elements inside the WP6 common use case is presented in Figure 6.18.

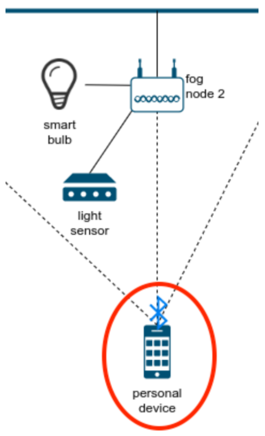


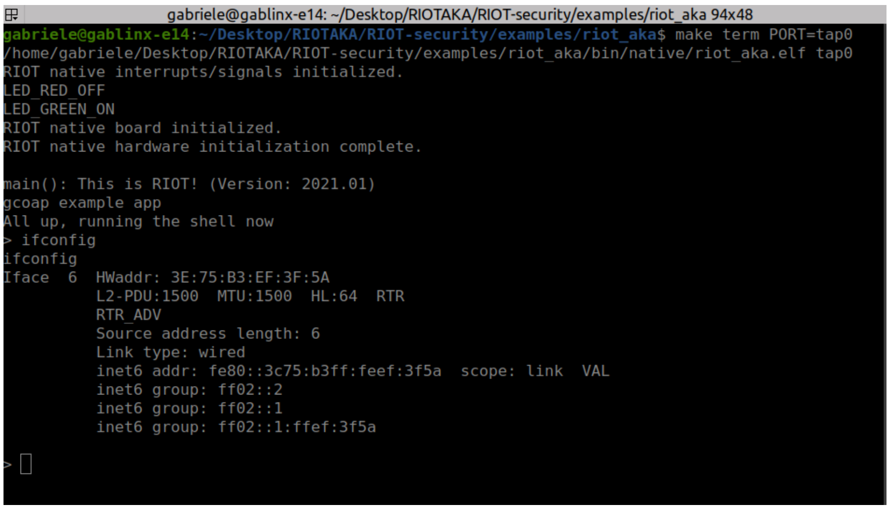
Figure 6.18: RIOT-AKA represented in the common WP6 use case infrastructure

6.1.5.1 Test setup

We showed in Section 3.4.4 a detailed evaluation of the scheme’s performances on real working IoT hardware. Here, we demonstrate it only using the `native` mode of RIOT OS which is an abstraction offered by the OS’s tools in order to simulate a physical IoT board by relying on internal system calls of the host machine to simulate hardware access. The `native` board will represent the UE in our scheme. We also provide HN and SN representations by developing simple virtual python-based servers. Finally, we clarify that by using the `native` implementation we cannot exploit the SRAM-PUF mode for root key generation on the devices since it is not supported.

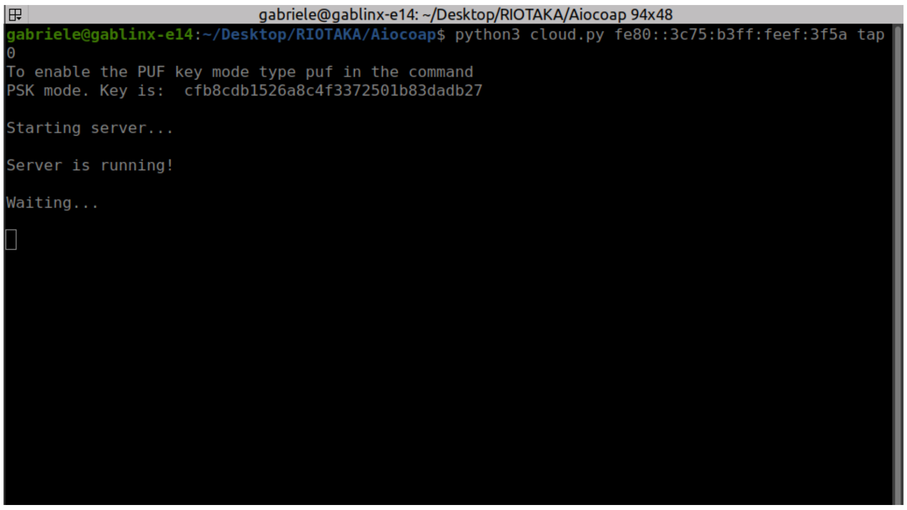
6.1.5.2 Execution flow

We start the simulation by triggering the registration of the device.



```
gabriele@gablinx-e14: ~/Desktop/RIOTAKA/RIOT-security/examples/riot_aka 94x48
gabriele@gablinx-e14:~/Desktop/RIOTAKA/RIOT-security/examples/riot_aka$ make term PORT=tap0
/home/gabriele/Desktop/RIOTAKA/RIOT-security/examples/riot_aka/bin/native/riot_aka.elf tap0
RIOT native interrupts/signals initialized.
LED_RED_OFF
LED_GREEN_ON
RIOT native board initialized.
RIOT native hardware initialization complete.

main(): This is RIOT! (Version: 2021.01)
gcoap example app
All up, running the shell now
> ifconfig
ifconfig
Iface  6  HWaddr: 3E:75:B3:EF:3F:5A
          L2-PDU:1500  MTU:1500  HL:64  RTR
          RTR_ADV
          Source address length: 6
          Link type: wired
          inet6 addr: fe80::3c75:b3ff:feef:3f5a  scope: link  VAL
          inet6 group: ff02::2
          inet6 group: ff02::1
          inet6 group: ff02::1:ffef:3f5a

>
```

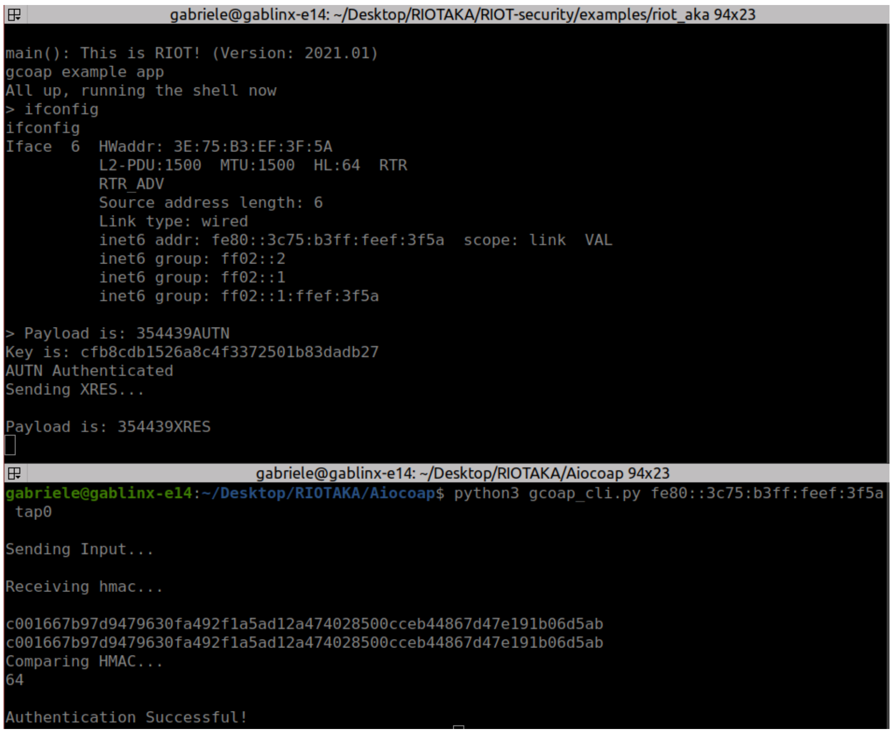
Figure 6.19: RIOT-AKA IoT IP address showup



```
gabriele@gablinx-e14: ~/Desktop/RIOTAKA/Aiocoap 94x48
gabriele@gablinx-e14:~/Desktop/RIOTAKA/Aiocoap$ python3 cloud.py fe80::3c75:b3ff:feef:3f5a tap
0
To enable the PUF key mode type puf in the command
PSK mode. Key is:  cfb8cdb1526a8c4f3372501b83dadb27

Starting server...

Server is running!

Waiting...
```

Figure 6.20: RIOT-AKA IoT registration to the HN

First we compile the IoT application developed specifically for our `native` board and launch it in a terminal. We grab its IPv6 address and paste that in another terminal in which we have to start the HN server and let it run. This phase initiates all the registration requirements between these two actors (Fig. 6.19, 6.20).

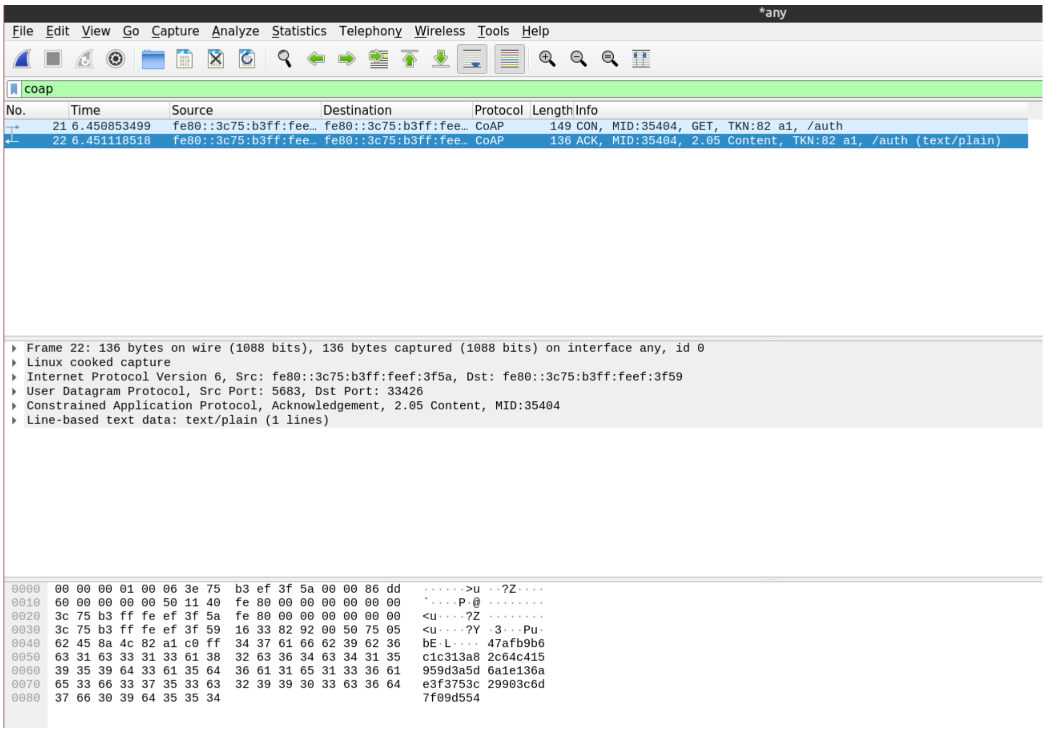


```
gabriele@gablinx-e14: ~/Desktop/RIOTAKA/RIOT-security/examples/riot_aka 94x23

main(): This is RIOT! (Version: 2021.01)
gcoap example app
All up, running the shell now
> ifconfig
ifconfig
Iface  6  HWaddr: 3E:75:B3:EF:3F:5A
          L2-PDU:1500  MTU:1500  HL:64  RTR
          RTR_ADV
          Source address length: 6
          Link type: wired
          inet6 addr: fe80::3c75:b3ff:feef:3f5a  scope: link  VAL
          inet6 group: ff02::2
          inet6 group: ff02::1
          inet6 group: ff02::1:ffef:3f5a

> Payload is: 354439AUTN
Key is: cfb8cdb1526a8c4f3372501b83dadb27
AUTN Authenticated
Sending XRES...

Payload is: 354439XRES
```

```
gabriele@gablinx-e14: ~/Desktop/RIOTAKA/Aiocoap 94x23
gabriele@gablinx-e14:~/Desktop/RIOTAKA/Aiocoap$ python3 gcoap_cli.py fe80::3c75:b3ff:feef:3f5a
 tap0

Sending Input...

Receiving hmac...

c001667b97d9479630fa492f1a5ad12a474028500cceb44867d47e191b06d5ab
c001667b97d9479630fa492f1a5ad12a474028500cceb44867d47e191b06d5ab
Comparing HMAC...
64

Authentication Successful!
```

Figure 6.21: RIOT-AKA IoT registration to the HN

Then we can simulate an authentication exchange between the IoT board and a SN. We still have to paste the IoT's address to another terminal in which we have to start the SN server. By doing so, we trigger an authentication request in which the two actors perform their handshake and, eventually, authenticate (Fig. 6.21).



```
No.  Time         Source                Destination           Protocol Length Info
  21 6.450853499  fe80::3c75:b3ff:fee…  fe80::3c75:b3ff:fee…  CoAP     149 CON, MID:35404, GET, TKN:82 a1, /auth
  22 6.451118518  fe80::3c75:b3ff:fee…  fe80::3c75:b3ff:fee…  CoAP     136 ACK, MID:35404, 2.05 Content, TKN:82 a1, /auth (text/plain)
```

```
▸ Frame 22: 136 bytes on wire (1088 bits), 136 bytes captured (1088 bits) on interface any, id 0
▸ Linux cooked capture
▸ Internet Protocol Version 6, Src: fe80::3c75:b3ff:feef:3f5a, Dst: fe80::3c75:b3ff:feef:3f59
▸ User Datagram Protocol, Src Port: 5683, Dst Port: 33426
▸ Constrained Application Protocol, Acknowledgement, 2.05 Content, MID:35404
▸ Line-based text data: text/plain (1 lines)
```

```
0000  00 00 00 01 00 06 3e 75  b3 ef 3f 5a 00 00 86 dd   ······>u ··?Z····
0010  60 00 00 00 00 50 11 40  fe 80 00 00 00 00 00 00   `····P·@ ········
0020  3c 75 b3 ff fe ef 3f 5a  fe 80 00 00 00 00 00 00   <u····?Z ········
0030  3c 75 b3 ff fe ef 3f 59  16 33 82 92 00 50 75 05   <u····?Y ·3···Pu·
0040  62 45 8a 4c 82 a1 c0 ff  34 37 61 66 62 39 62 36   bE·L···· 47afb9b6
0050  63 31 63 33 31 33 61 38  32 63 36 34 63 34 31 35   c1c313a8 2c64c415
0060  39 35 39 64 33 61 35 64  36 61 31 65 31 33 36 61   959d3a5d 6a1e136a
0070  65 33 66 33 37 35 33 63  32 39 39 30 33 63 36 64   e3f3753c 29903c6d
0080  37 66 30 39 64 35 35 34                            7f09d554
```

Figure 6.22: RIOT-AKA packets capture

In Fig. 6.22 we can see the network capture of the CoAP exchange-only between the IoT device and the SN.

6.2 Intrusion management workflow demonstration

To demonstrate the Intrusion management workflow the WP6 common virtual infrastructure was used. The placement of all involved elements inside the WP6 common use case is presented in Figure 6.23.

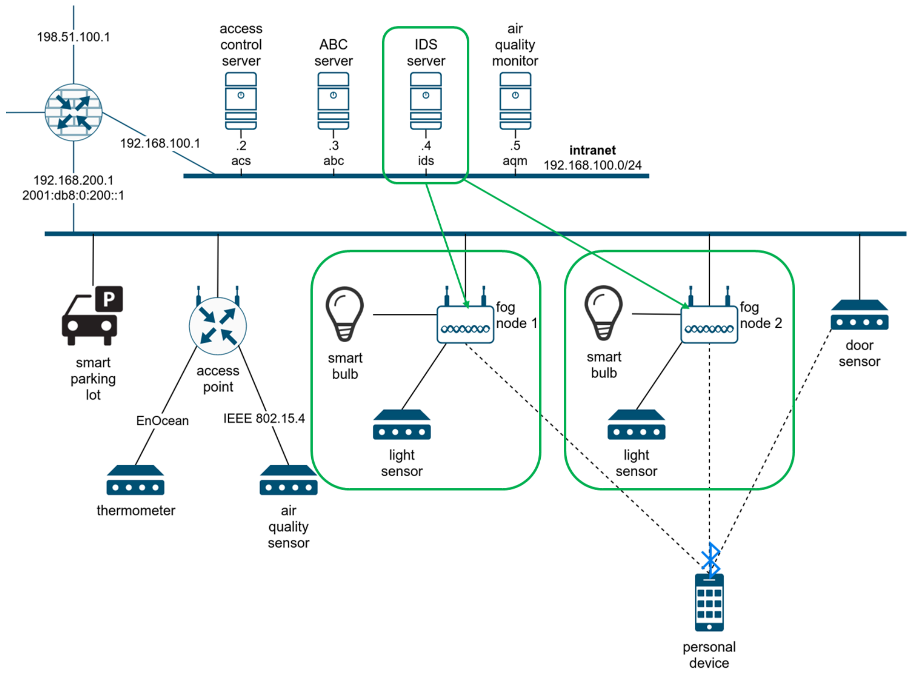


Figure 6.23: Intrusion management workflow components in the common WP6 use case infrastructure.

The two fog nodes were implemented as Docker containers and are running the corresponding services. One set of smart bulb and light sensor is implemented using the real hardware prototypes which are composed from ESP8266 micro controllers, BH1750 light sensor and LED. The services (implemented as agents) are running inside the Fog node 1 and monitor the sensors and actuators using CoAP protocol. The second set of sensors is virtual, implemented only as corresponding agents running inside Fog node 2. The IDS servers are running in virtual infrastructure and are monitoring the whole IoT network using Machine learning and Anomaly based intrusion detection approaches. The main phases of demonstration are these:

- The first security event which compromises the Fog node 1 is manually triggered and the Machine learning based IDS system detects it and displays the corresponding event using GUI;
- The second security event which compromises the Fog node 1 is manually triggered and the Anomaly based IDS system detects it and displays the corresponding event using GUI;
- The human operator sees the security events on the GUI, realizes that security level of Fog node 1 is compromised, and changes it using the provided GUI. The orchestrator of Fog node 1 detects the situation changes and reorchestrates the services.

All three main steps of the demonstration flow are described in more details in the following three sub chapters.

6.2.1 Machine Learning-based Network Intrusion Detection component

At test time, the Machine Learning-based Network Intrusion Detection component collects the NetFlow frames published on a specific Apache Kafka topic. The tool applies the necessary preprocessing procedures to the collected data and passes it through to the pre-trained ML components. After ML classification, the results are forwarded to a target Apache Kafka topic. This allows for real-time network intrusion detection. The classified frames are indexed in ElasticSearch and pushed to an informative Kibana dashboard (see. Fig.6.24). The use of Apache Kafka allows for easy integration with any tool capable of connecting to the broker.

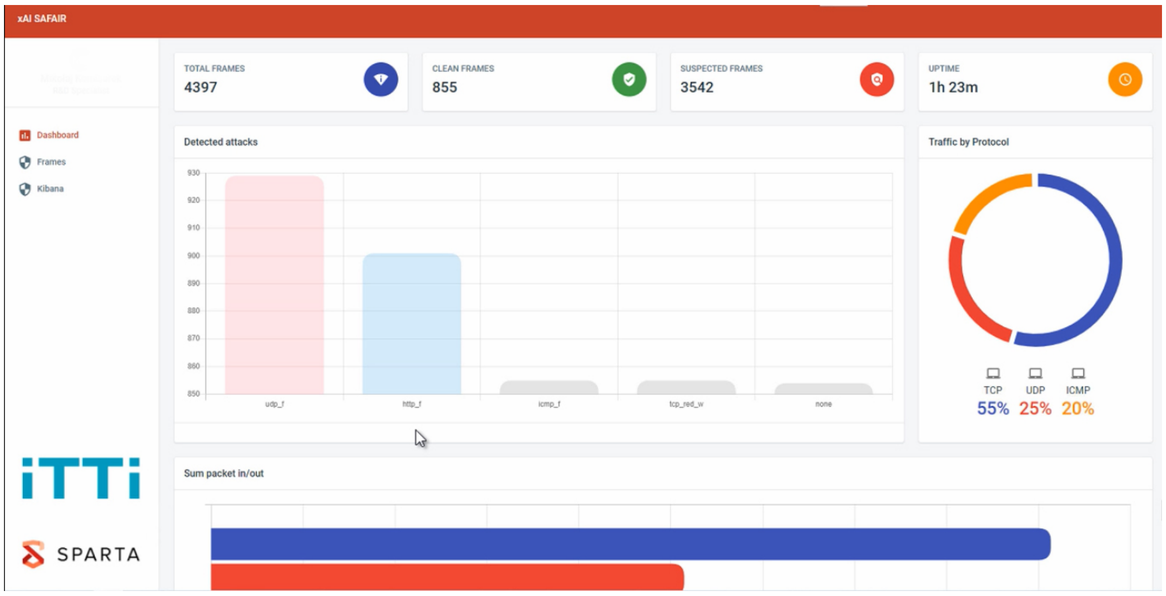
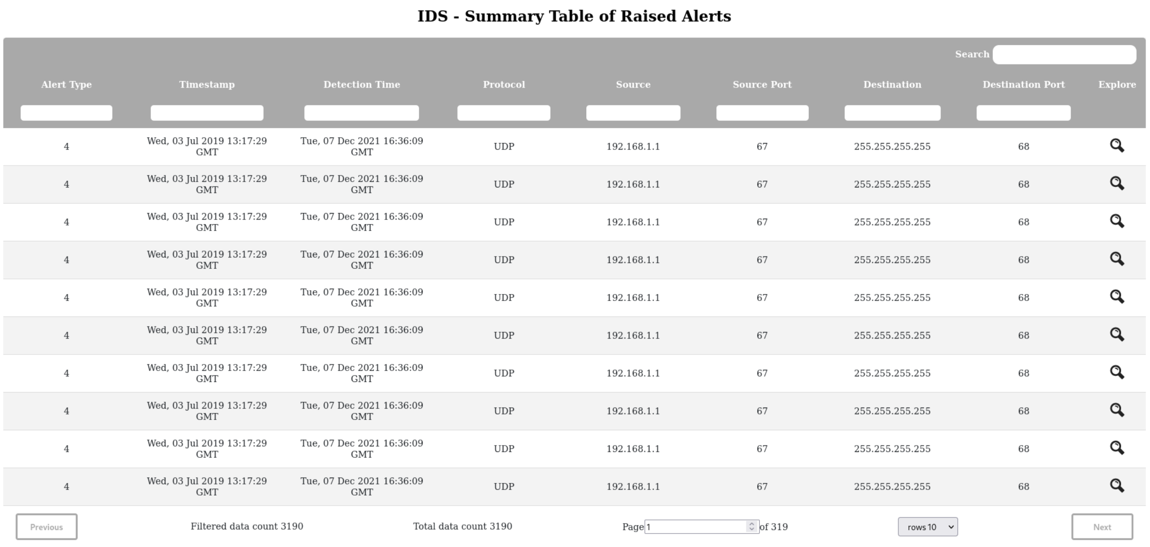


Figure 6.24: The component dashboard

6.2.2 Anomaly-based Detection of Network Intrusion Event

The demonstration focuses on the use of the IDS tool during the detection phase. Indeed the model used during this detection phase has been produced previously during a learning phase (whose duration is too long to be executed during the demonstration). Before this construction of the model, execution traces (pcap files) corresponding to normal system behaviors have been collected in contexts and conditions of use similar to those adopted on the day of the demonstration. In particular, the normal interactions between the main monitored entities (Fog node 1 and 2) and the other entities of the system (the orchestration mechanism for example) must have been observed in order to be learned. As the learning phase is not part of the actual demonstrations, the pcap files used to build the model as well as the representations of the intermediate models generated by the successive executions of the ”Builder” component will be made available. Let us recall that it is during the learning phase that the crucial choice of the definitions of event types is made.

During the demonstration, two main components of the IDS are used, namely the ”Detector” component and the ”Pcap Sniffer” component. The latter has to capture the network traffic concerning the Fog nodes. Information on observed packets are transmitted via a socket to the ”Detector” component which will be located on the IDS Server. The ”Detector” component is responsible for analyzing the received informations about packets and checking their compliance with the model according to the principles described in Section 4.2. The raised alerts are accessible via a graphical interface developed in React (See Figure 6.25).



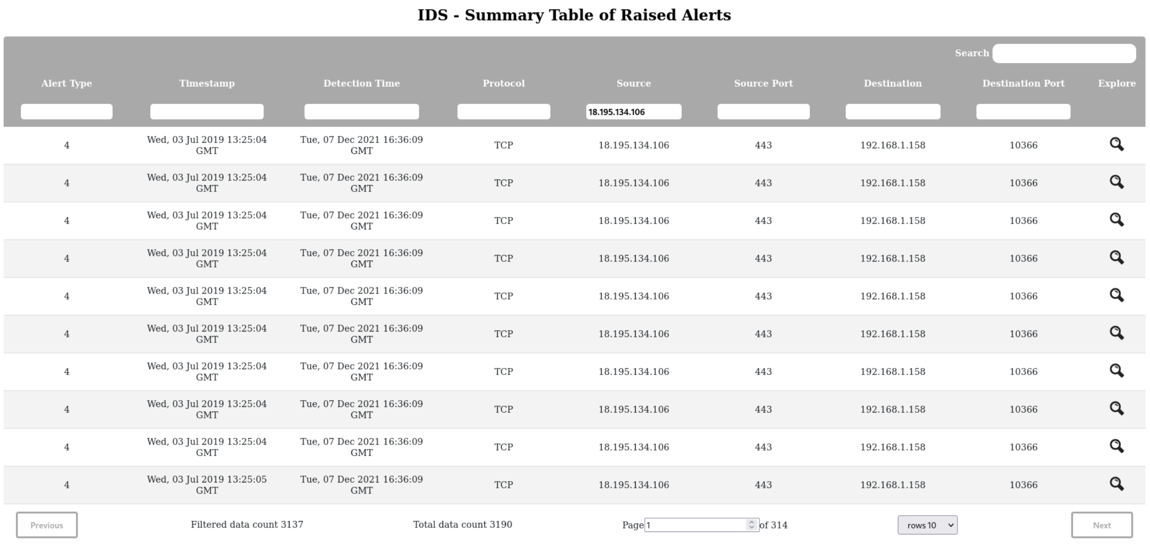
IDS - Summary Table of Raised Alerts

Search

Alert Type	Timestamp	Detection Time	Protocol	Source	Source Port	Destination	Destination Port	Explore
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍
4	Wed, 03 Jul 2019 13:17:29 GMT	Tue, 07 Dec 2021 16:36:09 GMT	UDP	192.168.1.1	67	255.255.255.255	68	🔍

Previous | Filtered data count 3190 | Total data count 3190 | Page 1 of 319 | rows 10 | Next

Figure 6.25: Display of all the Alerts



IDS - Summary Table of Raised Alerts

Search

Alert Type	Timestamp	Detection Time	Protocol	Source	Source Port	Destination	Destination Port	Explore
				18.195.134.106				
4	Wed, 03 Jul 2019 13:25:04 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍
4	Wed, 03 Jul 2019 13:25:04 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍
4	Wed, 03 Jul 2019 13:25:04 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍
4	Wed, 03 Jul 2019 13:25:04 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍
4	Wed, 03 Jul 2019 13:25:04 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍
4	Wed, 03 Jul 2019 13:25:04 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍
4	Wed, 03 Jul 2019 13:25:04 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍
4	Wed, 03 Jul 2019 13:25:04 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍
4	Wed, 03 Jul 2019 13:25:04 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍
4	Wed, 03 Jul 2019 13:25:05 GMT	Tue, 07 Dec 2021 16:36:09 GMT	TCP	18.195.134.106	443	192.168.1.158	10366	🔍

Previous | Filtered data count 3137 | Total data count 3190 | Page 1 of 314 | rows 10 | Next

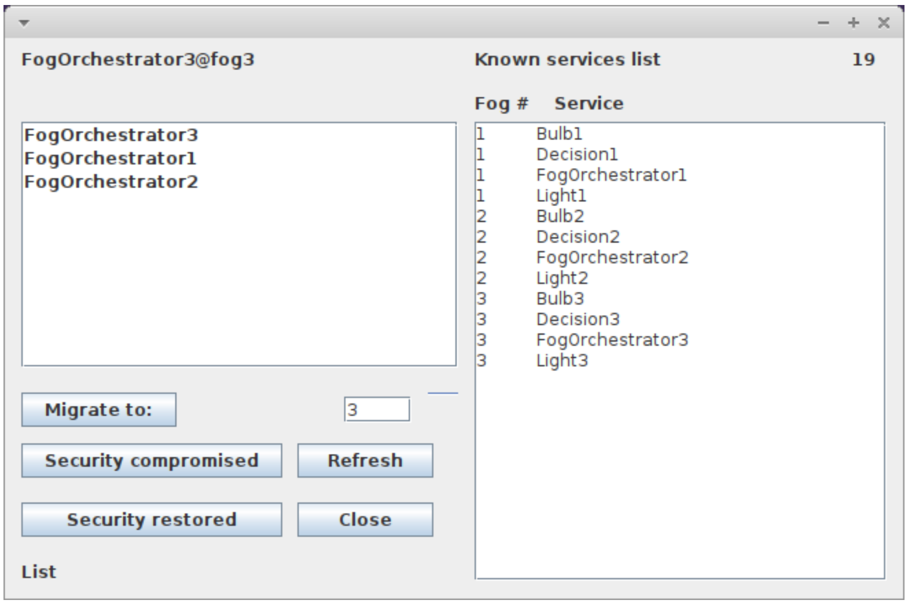
Figure 6.26: Display of a Sub-set of Alerts using a Filter on the Source IP

In particular, it is possible to access the list of alerts produced and to focus on subsets of this list using filters applied on the displayed attributes (See Figure 6.26 where a filter on the source IP is used).

The security event triggered to disrupt Fog node 1 (inspired from the Mirai attack) has an impact on the observed traffic, which should allow anomalies to be detected. Depending on the alerts displayed and their relevance, a human operator decides to re-orchestrate the services if necessary. To make this judgment, the operator may consider the frequency of the alerts and the reasons for the rejection of the packets. For example, in Figure 6.25, the alert type attribute indicates that the alerts have been raised because the corresponding packets were not accepted by the automaton.

6.2.3 Fog service reorchestration

To be able to communicate with orchestrators running in Fog nodes 1 and 2 we need a GUI. The easiest way to achieve this is to add third Fog node which is running on computer with GUI capabilities. In such case, the situation before the reorchestration of services is presented in Figure 6.27.


FogOrchestrator3@fog3

FogOrchestrator3
FogOrchestrator1
FogOrchestrator2

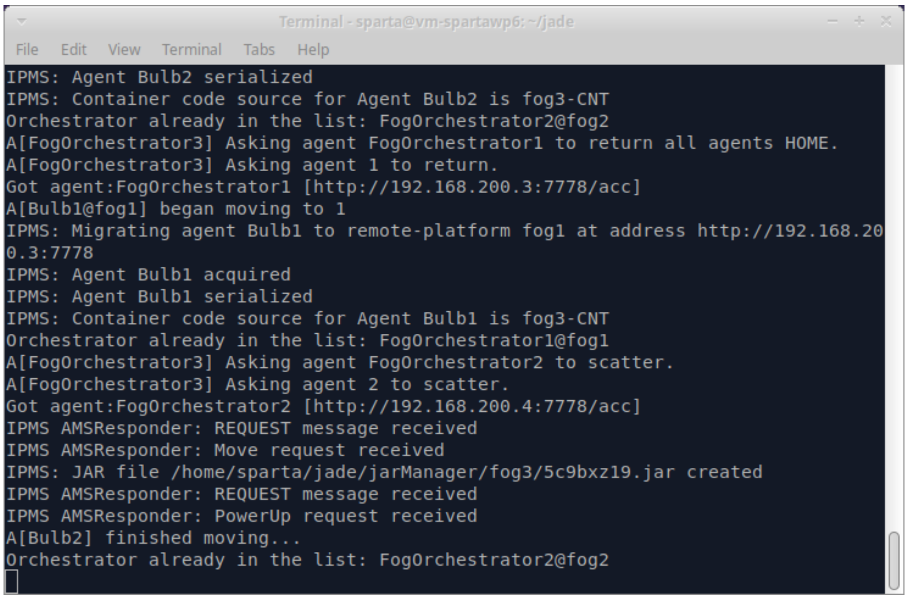
Known services list 19

Fog #	Service
1	Bulb1
1	Decision1
1	FogOrchestrator1
1	Light1
2	Bulb2
2	Decision2
2	FogOrchestrator2
2	Light2
3	Bulb3
3	Decision3
3	FogOrchestrator3
3	Light3

Figure 6.27: Initial situation in the IoT infrastructure.

In Figure 6.27 we can see the whole IoT system from the point of view of orchestrator 3, which is running inside the Fog node 3 (top left corner of the window). The left side of the window diplays a list of all currently known orchestrators. A list of all running services (or agents) along with the numbers of the Fog nodes in which they are currently residing is presented on the right side of the window. Before the security event all the Fog nodes have “High” security and all services are distributed evenly. Each Fog node hosts one set of services, which are initially started by the corresponding orchestrators. The number in the name of the services indicates the initial Fog node and also acts as a mean to ensure that all names of the services are unique inside the whole infrastructure.

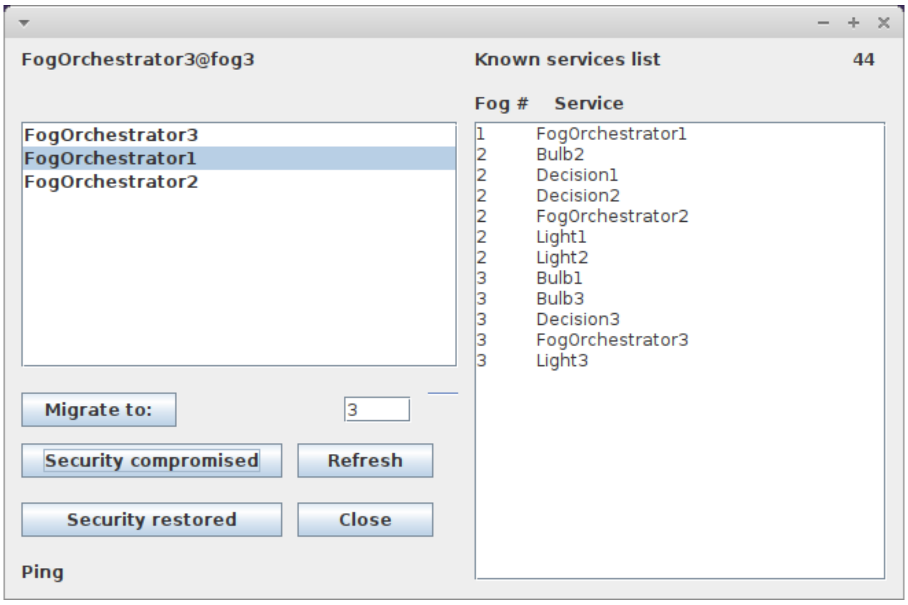
After the detection of security event, the human operator must inform the corresponding orchestrator about the reduced security level. To do this, the orchestrator 1 must be selected in the left part of the window and button “Security compromised” must be pressed. This action sends a special message to the orchestrator 1 informing it about compromised security level. Then the orchestrator must make decision on mitigating the security risks. As another two Fog nodes are available, and the resources on them are sufficient to host all services, it decides to reallocate all service running in Fog node 1 to other two available Fog nodes. The affected agents (or services) are asked to migrate to another Fog nodes, see Figure 6.28. After several seconds, all orchestrators exchange the information about the new service placement and the orchestrator 3 (which we are monitoring) shows the situation summarized in Figure 6.29.



```
IPMS: Agent Bulb2 serialized
IPMS: Container code source for Agent Bulb2 is fog3-CNT
Orchestrator already in the list: FogOrchestrator2@fog2
A[FogOrchestrator3] Asking agent FogOrchestrator1 to return all agents HOME.
A[FogOrchestrator3] Asking agent 1 to return.
Got agent:FogOrchestrator1 [http://192.168.200.3:7778/acc]
A[Bulb1@fog1] began moving to 1
IPMS: Migrating agent Bulb1 to remote-platform fog1 at address http://192.168.20
0.3:7778
IPMS: Agent Bulb1 acquired
IPMS: Agent Bulb1 serialized
IPMS: Container code source for Agent Bulb1 is fog3-CNT
Orchestrator already in the list: FogOrchestrator1@fog1
A[FogOrchestrator3] Asking agent FogOrchestrator2 to scatter.
A[FogOrchestrator3] Asking agent 2 to scatter.
Got agent:FogOrchestrator2 [http://192.168.200.4:7778/acc]
IPMS AMSResponder: REQUEST message received
IPMS AMSResponder: Move request received
IPMS: JAR file /home/sparta/jade/jarManager/fog3/5c9bxz19.jar created
IPMS AMSResponder: REQUEST message received
IPMS AMSResponder: PowerUp request received
A[Bulb2] finished moving...
Orchestrator already in the list: FogOrchestrator2@fog2
```

Terminal - sparta@vm-spartawp6: ~/jade
File Edit View Terminal Tabs Help

Figure 6.28: The process of agent migrations as viewed from Fog node 3.



FogOrchestrator3@fog3

FogOrchestrator3
FogOrchestrator1
FogOrchestrator2

Known services list 44

Fog #	Service
1	FogOrchestrator1
2	Bulb2
2	Decision1
2	Decision2
2	FogOrchestrator2
2	Light1
2	Light2
3	Bulb1
3	Bulb3
3	Decision3
3	FogOrchestrator3
3	Light3

Figure 6.29: The situation in the IoT infrastructure after the reorchestration.

Now all the services from Fog node 1 are distributed between Fog nodes 2 and 3, which still remain secure. In such way the Intelligent IoT infrastructure adapted to the changed conditions during the runtime and ensured that the overall security of the system is still “High”.

6.3 Privacy and Data Management Demonstration

This section presents how the privacy and data management can be supported by our developed PETs tools (the PEAS system, the MC-SSE system and the DPO tool) in the chosen scenario that is focused on the privacy-preserving verification of user access inside the smart campus, privacy-preserving data processing and checking services’ GDPR compliance. Figure 6.30 shows the privacy-preserving access and data management scenario and explains how the developed PETs tools can be deployed. The high-level description of this scenario is as follows:

1. users with vehicles can show and prove their attributes, i.e., memberships (parking permit attributes), for getting access to a campus parking lot that is equipped by parking lot machine/ter-

minal (a verifier). This process can be realized by the **PEAS** system that provides the privacy-preserving authentication of users. Details are in 6.3.1.

2. the compliance of the system business model to GDPR regulation management can be directly checked by the application of the **Data Protection Officer** (DPO) tool.
3. users who require access to buildings (areas) can prove their possession of the memberships (area attributes) also by the **PEAS** system, more details can be found in 6.3.1.
4. all usage data from the system containing user access logs, to parking spots and buildings can be used to produce privacy-preserving post-processing statistics. This is achieved by using the **MC-SSE** data storage and processing solution. This is a multi-client searchable encryption based solution, that supports boolean queries. Collected data is stored in encrypted form and multiple search clients can submit queries to be performed on the encrypted dataset. Using encrypted keywords that act as trapdoors for encrypted documents that contain them, the encrypted dataset can be searched to produce system usage statistics without revealing any sensitive information about the searched documents.

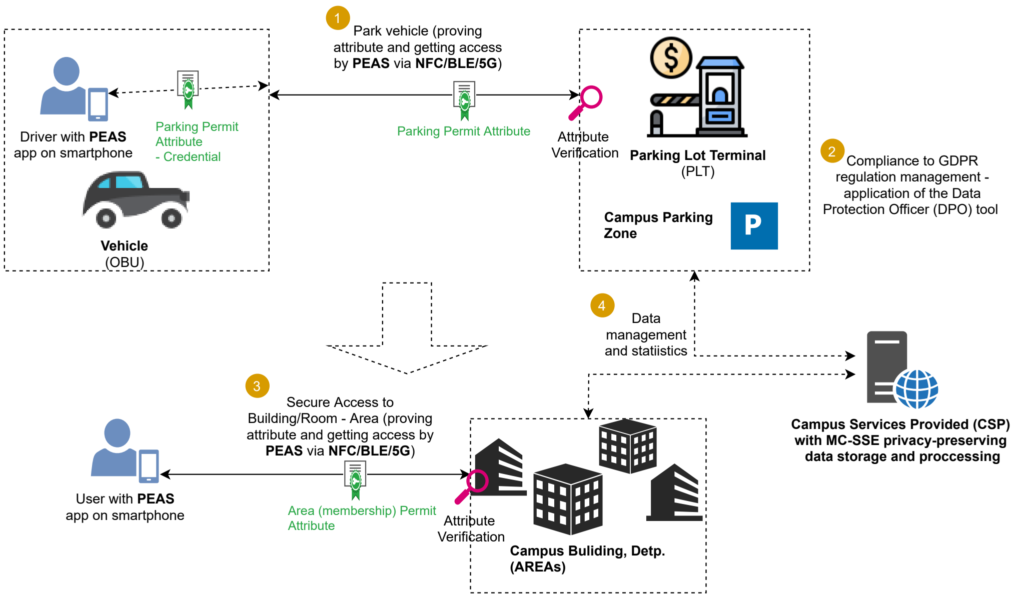

1 Park vehicle (proving attribute and getting access by PEAS via NFC/BLE/5G)
Driver with PEAS app on smartphone
Parking Permit Attribute - Credential
Parking Permit Attribute
Attribute Verification
Parking Lot Terminal (PLT)
2 Compliance to GDPR regulation management - application of the Data Protection Officer (DPO) tool
Vehicle (OBU)
Campus Parking Zone
P
4 Data management and statiistics
3 Secure Access to Building/Room - Area (proving attribute and getting access by PEAS via NFC/BLE/5G)
User with PEAS app on smartphone
Area (membership) Permit Attribute
Attribute Verification
Campus Buliding, Detp. (AREAs)
Campus Services Provided (CSP) with MC-SSE privacy-preserving data storage and proccessing

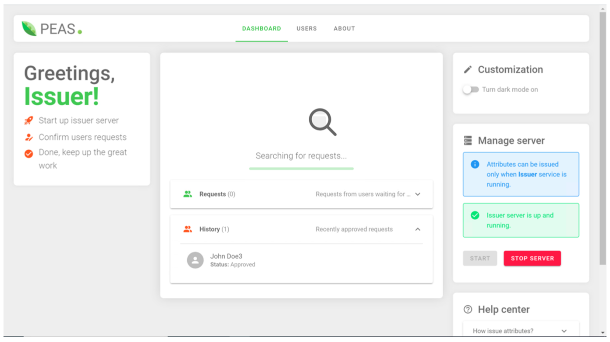
Figure 6.30**:** The demonstration scenario of privacy-preserving access and data management in smart campus.

6.3.1 Privacy-preserving access supported by PEAS

The privacy-preserving access scenarios can be supported by PEAS. The scenarios considers access to parking lots and access to protected areas (buildings/departments/rooms).

Privacy-preserving access to parking lots

Parking credentials/attributes are firstly issued by the internal trusted entity (Issuer) in the campus and can be released to various users such as visitors (a short term membership), students or campus staff (a long term membership). The issuer part can be realized by the web-based PEAS Issuer application that enables a manager to set all system parameters and watch log events. The graphical interface of Issuer is depicted in Figure 6.31. Then, the users can simply use the Android-based PEAS user applications in their smartphones to provide their parking credentials. The Android-based user application can communicate with a verifier-parking lot terminal via the Bluetooth or the NFC (Near Field Communication) interface. Figures 6.32 and 6.33 show the graphical activities of Android-based PEAS User application. The verifier part can be realized by the web-based PEAS Verifier application (depicted in Figure 5.5) which can run at the embedded PC and can be accessed through a web interface using a smartphone or a tablet controlled by a security staff who can check the results from the authentication process. The core application of the verifier can automatically verify parking credentials/attributes provided by users and several other attributes. Further, the issuer can revoke users who break rules or stop using this service. The attributes are verified under specific epoch time periods to make the user revocation possible. Furthermore, the user can run only a certain number (in our implementation max. 100) of authentications within one epoch. Exceeding this value would impact user privacy, and therefore, it is controlled and prevented by the system. The length of the epoch is configurable by the system administrator.



Greetings, Issuer!

Start up issuer server
Confirm users requests
Done, keep up the great work

Searching for requests...

Requests (0) Requests from users waiting for ...

History (1) Recently approved requests

John Doe3
Status: Approved

Attributes can be issued only when **Issuer** service is running.

Issuer server is up and running.

Figure 6.31: Dashboard of web-based PEAS Issuer.

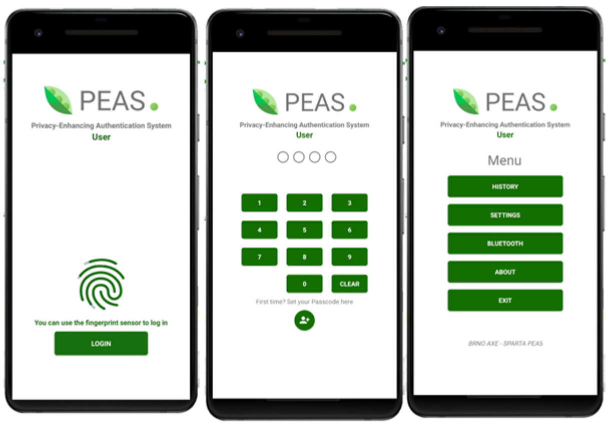
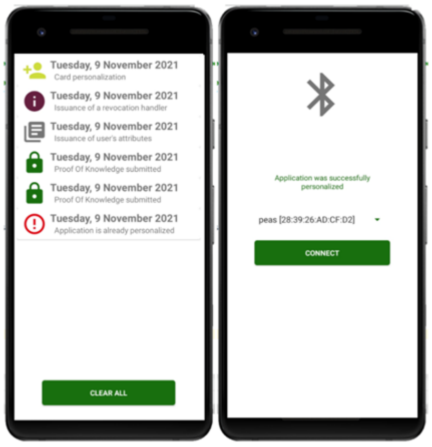

PEAS
Privacy-Enhancing Authentication System
User
You can use the fingerprint sensor to log in
LOGIN
PEAS
Privacy-Enhancing Authentication System
User
1
2
3
4
5
6
7
8
9
0
CLEAR
First time? Set your Passcode here
PEAS
Privacy-Enhancing Authentication System
User
Menu
HISTORY
SETTINGS
BLUETOOTH
ABOUT
EXIT
BRNO AXE - SPARTA PEAS

Figure 6.32: GUI of Android-based PEAS User - login and main menu.



Tuesday, 9 November 2021
Card personalization

Tuesday, 9 November 2021
Issuance of a revocation handler

Tuesday, 9 November 2021
Issuance of user's attributes

Tuesday, 9 November 2021
Proof Of Knowledge submitted

Tuesday, 9 November 2021
Proof Of Knowledge submitted

Tuesday, 9 November 2021
Application is already personalized

Application was successfully personalized

peas [28:39:26:AD:CF:D2]

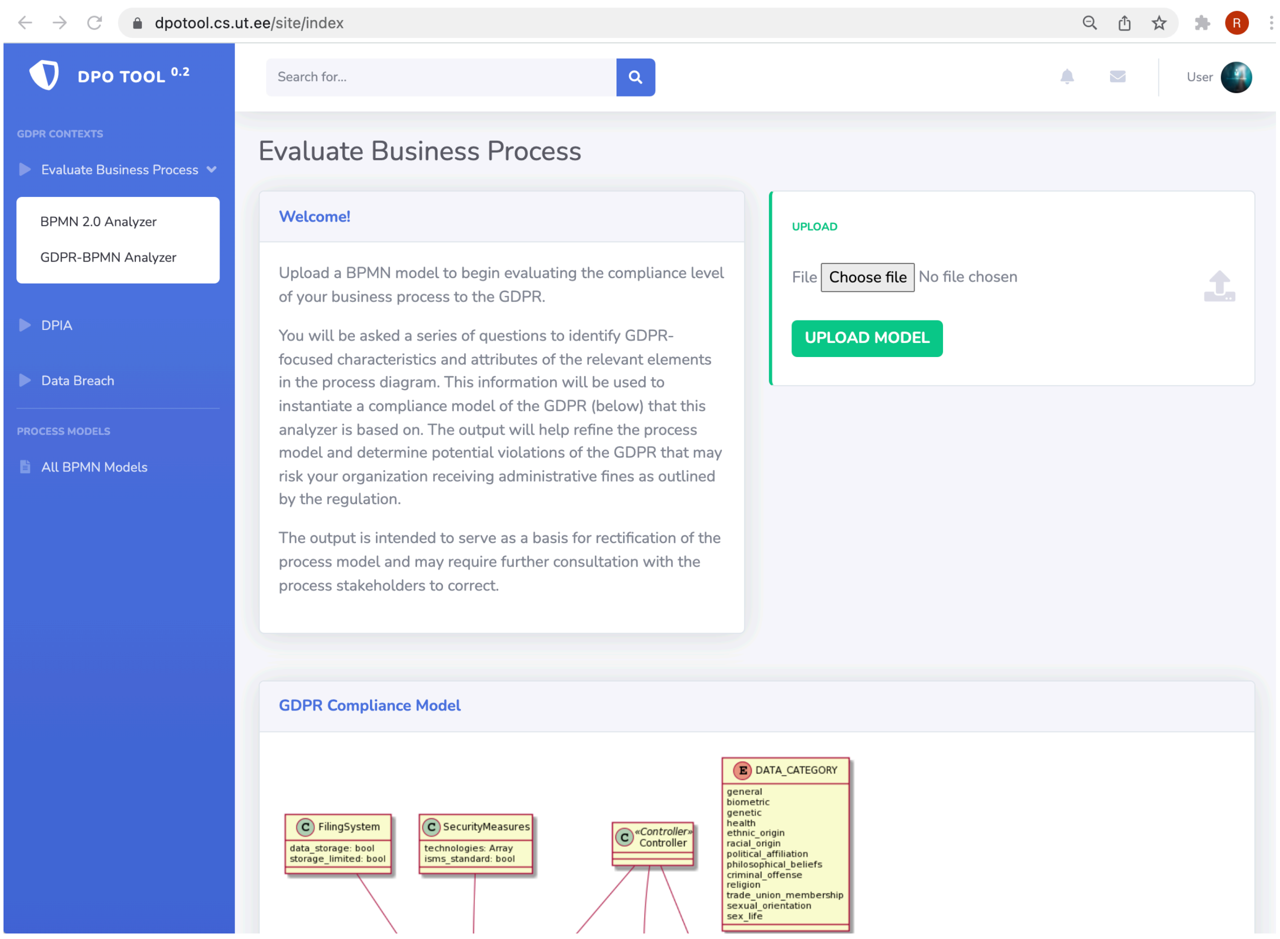
Figure 6.33: GUI of Android-based PEAS User - logs and communication.

Privacy-preserving access to protected areas

The user can use the same Android-based PEAS user application and the web-based PEAS User application (depicted in Figure 5.4) as in privacy-preserving access to parking lots. However, he/she shows a different attribute, i.e., the area attribute membership. The verifier with a terminal (e.g. embedded computer unit close to the door) controlling an entrance can check that users have valid access to the concrete building/department/area again via the Verifier application. The web-based PEAS Verifier application can be used for setting which attributes should be checked and disclosed from users. The verifier is able to log successful or faulty access attempts. The PEAS can be set for both application scenarios (access to parking lots, access to areas) by the campus manager with the web-based PEAS Issuer application. The complete video demonstration of web-based applications for all parties (User, Issuer, Verifier) for setting, issuing, proving the department attributes, verification and revocation can be downloaded at `https://www.vut.cz/www_base/vutdisk.php?i=280744ab24`. All PEAS source codes including the web-based demo in docker containers are stored in the private GitLab repository (`gitlab.com/brno-axe/peas/`).

6.3.2 DPO tool

DPO tool is a Web-based prototype tool (`https://dpotool.cs.ut.ee/`) to support the model-driven GDPR compliance management approach at the business process level. The PDO source code is stored in the github repository at `https://github.com/motekaj/gdpr-analyzer`. The tools interface is given in Fig. 6.34. Currently the tool's main functionality includes upload of the business process model for compliance analysis, evaluation of of the business process compliance, viewing the model, editing the model and deleting the model. In this section we will shortly discuss the function for compliance checking.



Evaluate Business Process

Welcome!

Upload a BPMN model to begin evaluating the compliance level of your business process to the GDPR.

You will be asked a series of questions to identify GDPR-focused characteristics and attributes of the relevant elements in the process diagram. This information will be used to instantiate a compliance model of the GDPR (below) that this analyzer is based on. The output will help refine the process model and determine potential violations of the GDPR that may risk your organization receiving administrative fines as outlined by the regulation.

The output is intended to serve as a basis for rectification of the process model and may require further consultation with the process stakeholders to correct.

Figure 6.34: DPO tool interface

A method of checking compliance is described in [63] and shown in Fig. 6.35. In the first step (i.e., 1. Extract AS-IS compliance model), the BPMN2GDPR process model is considered. Once the model is annotated with the GDPR concepts (as discussed in Section 5.2.2, see Fig. 5.7), no additional user input is required in the first step.

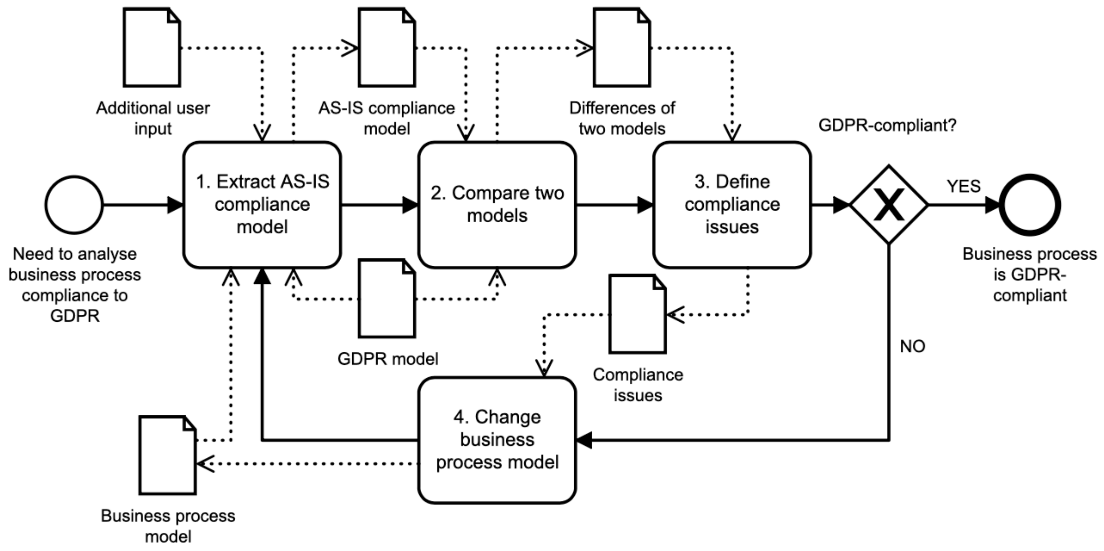

Additional user input
AS-IS compliance model
Differences of two models
GDPR-compliant?
1. Extract AS-IS compliance model
2. Compare two models
3. Define compliance issues
YES
Need to analyse business process compliance to GDPR
Business process is GDPR-compliant
GDPR model
Compliance issues
NO
4. Change business process model
Business process model

Figure 6.35: Method for achieving regulation compliance, adapted from [63]

In the second step and the third steps of the method, the DPO tool automatically compares the AS-IS compliance model and the GDPR model. Based on the comparison results, the compliance issues are defined. If compliance issues are found, the analyst needs then to refine the business process model (i.e., step 4. Change business process model). This can be done using the DPO tool (e.g., function for editing) or the external `bpmn.io` tool. In our example (Fig. 5.7), there is no compliance issues as indicated in the DPO-generated AS-IS compliance model, see Fig. 6.36. Thus, in the given case, the process is finished after the third step.

As the result, compliance of the BPMN2GDPR model is checked using the DPO tool which supports the model-driven compliance analysis of the business processes. In our example, we define the business model in a way that no compliance issues are shown in the outcome. The resulting model AS-IS compliance model (see Fig. 6.36) can be exported in the PlantUML format, which can be integrated to other analysis (see `https://plantuml.com/running`).

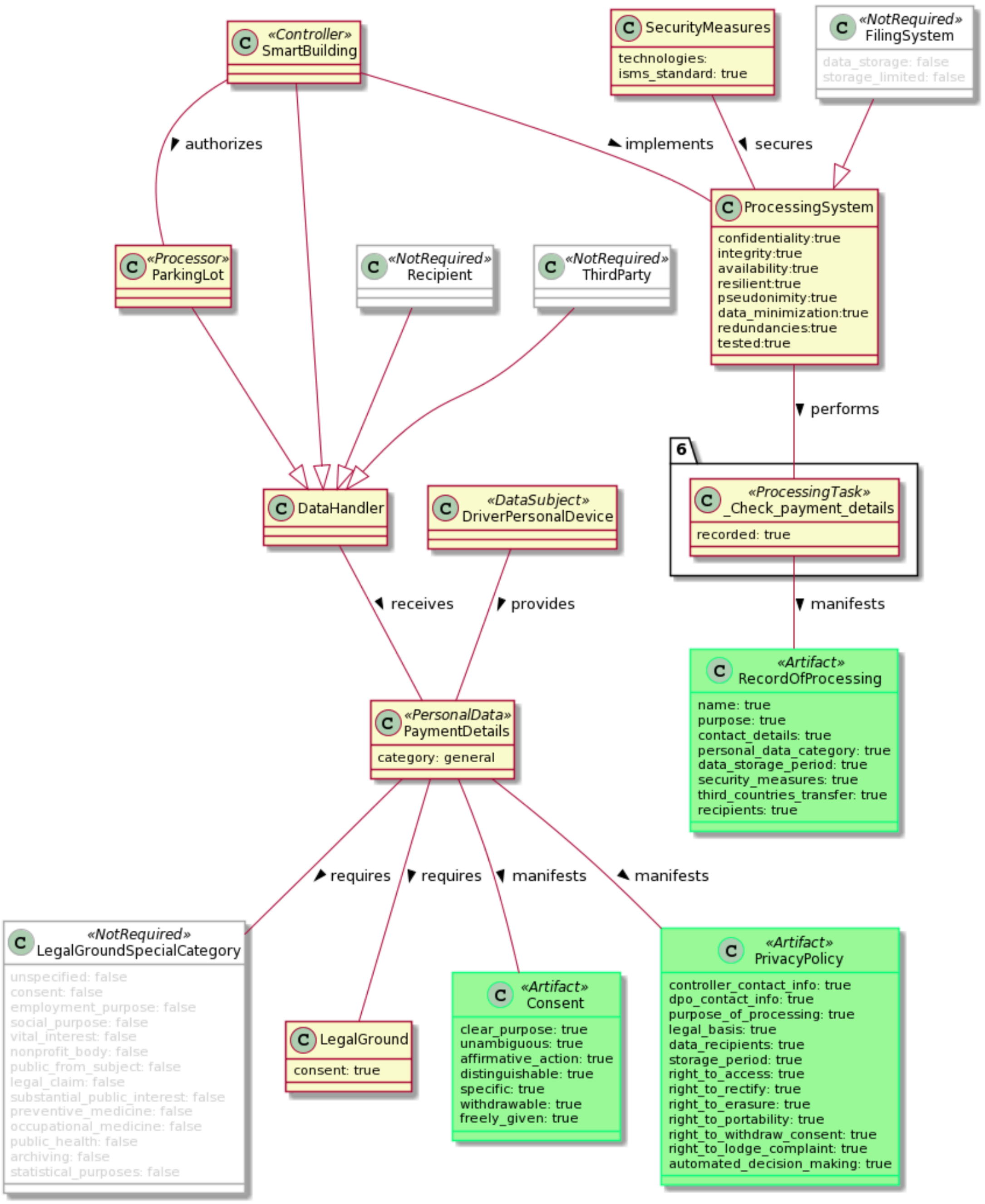
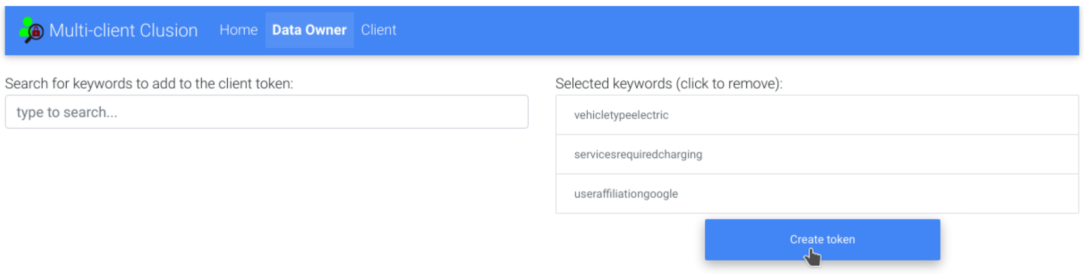


Figure 6.36**:** Compliance Check of Vehicle Charge Process Annotated using BPMN Extension for GDPR

6.3.3 MC-SSE demonstration

The MC-SSE demo tool implementation consists mainly of two interfaces, one for the Data Owner, where they can select the keywords to be included in the authorisation token and export it in a .json file format, as illustrated in Figure 6.37 A live demo of the tool is accessible at: https://www-public.imtbs-tsp.eu/~lauren_m/SPARTA_MC-SSE/ where the tool can be tested with a dataset of 10000 documents, containing synthetic parking transactions with 21 keywords per document, resulting in 210K document-keyword pairs. A docker container of the MC-Clusion library and the demo tool is also available for downloading at https://hub.docker.com/r/atasidou/multi-client_clusion. By replacing the documents in the data folder, the tool can be used with any appropriately formatted dataset (as described in section 5.3.3).



Search for keywords to add to the client token:

type to search...

Selected keywords (click to remove):

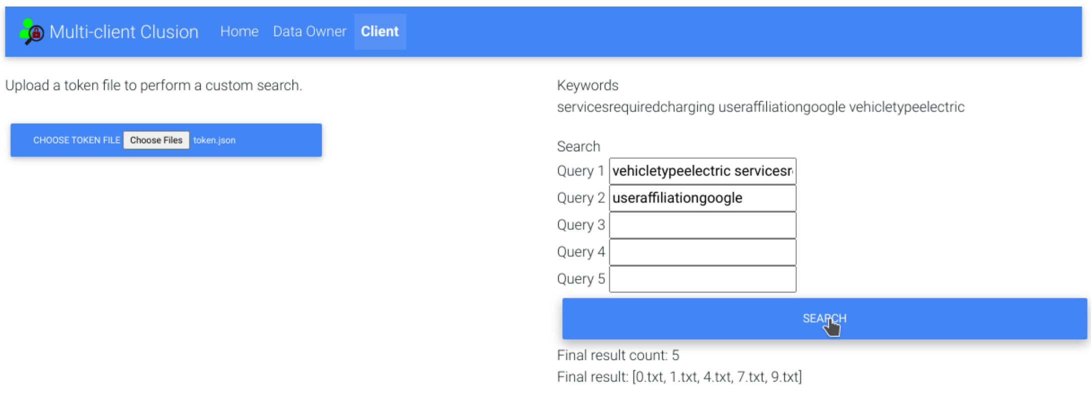
vehicletypeelectric

servicesrequiredcharging

useraffiliationgoogle

Figure 6.37: Data Owner interface for authorisation token creation

In the Client search interface, the authorisation token created by the Data Owner can be loaded, displaying the authorised keywords available for the search query, as illustrated in Figure 6.38. The search query is formulated by adding keywords separated by spaces in each subquery line, to formulate the final query. Keywords within the same subquery are disjuncted, while subqueries are conjuncted. The query result displayed under the search button contains the total number of documents and the document ids relevant to the submitted query.



Upload a token file to perform a custom search.

token.json

Keywords
servicesrequiredcharging useraffiliationgoogle vehicletypeelectric

Search

Query 1	vehicletypeelectric servicesr
Query 2	useraffiliationgoogle
Query 3	
Query 4	
Query 5	

Final result count: 5
Final result: [0.txt, 1.txt, 4.txt, 7.txt, 9.txt]

Figure 6.38: Client search interface for authorisation token loading and query execution

Chapter 7 Relevant security aspects and evaluation

In this Chapter, we asseess the effectiveness of the technical results produced by the HAII-T program. Initially, we survey the main security concerns relevant for the HAII-T demonstration. Then, we discuss on the effectiveness of the proposed methodologies in relationship with the security aspects introduced above.

7.1 Vulnerabilities

In the context of Intelligent Infrastructures, Edge Computing, and especially IoT environments, certain types of vulnerabilities are relevant. In general, they can be classified by the potential **attack surface**, their **class of attack** as well as their **root causes**. In [86], an in-depth description of these three elements is given. Also visit [22, 84] for further details.

After laying down the foundations, we will analyse the relevance of the root causes in regard to the three workflows that are described in this deliverable. We identify the most relevant vulnerabilities and provide the the codes of the Common Weaknesses Enumerations (CWE) database [66].

7.1.1 Attack surface

Weak Computation Power and Poorly Secured Devices

The computation power at the edge is weaker compared to cloud servers. This might lead to a poorly configured secured edge device that leads to weaker defence mechanisms and attacks that do not work on cloud servers still might work at the edge infrastructure.

Attack Unawareness

Users have limited knowledge about the state of IoT devices, nor is it likely to realize ongoing attacks.

OS and Protocol Heterogeneities

Different types of operating systems and communication protocols or (open source) implementations makes it difficult to unify the defence mechanisms.

Coarse-Grained Access Control

Managing access to IoT devices requires more fine-grained definition, than currently exists.

Misalignment across device edge and service provider edge

Involvement of third-party service providers might lead to shared responsibilities which are not clearly defined.

7.1.2 Attack classification

DDoS attacks

- **Flooding-based attacks:**
  Security has been ignored in the design phase and resulted in flaws at the protocol level.
  Countermeasures: Real-time based detection per-packet; detection based on statistics
- **Zero-day DDoS attacks:**
  Vulnerabilities on code-level that can cause memory corruptions.
  Countermeasures: Pointer taintedness detection; ECC-memory

Side-channel attacks

- **Attacks exploiting communication channels:**
  Conclusions made on base of side-channel information that is publicly available and an unknown hidden correlation with the sensitive data.
  Countermeasures: Data perturbation; Restricting accesses to side channel
- **Attacks exploiting power consumption**

Malware injection attacks

- **Server-side injections:**
  Design flaws on protocol-level, e.g., SQL injection, Cross-site request forgery, server-side request forgery, XML signature wrapping
  Countermeasures: Detection filter
- **Device-side injections:**
  Design flaws at the device, e.g. Firmware modification attacks, Design flaws on code-level
  Countermeasures: Analysis on code-level for malicious behaviour; Coarse-grained access control model adaption; Fine-grained access control

Authentication and authorisation attacks

- **Dictionary attacks:**
  Weak credentials in authentication protocols.
  Countermeasures: Adding an additional authentication layer with stronger properties; Hardening the process of password verification
- **Attacks exploiting vulnerabilities in authentication and authorisation protocols:**
  Design flaws on protocol-level or flaws on implementation-level.
  Countermeasures: Patching and strengthening the current protocols; Conducting code-level analyses
- **Over-privileged attacks:**
  Design flaws on protocol-level or flaws on implementation-level.
  Countermeasures: Strengthening the current permission models; Patching and strengthening the current protocols; Conducting code-level analyses

Routing Information Attacks

- **Manipulation of data paths:**
  Deleting incoming/outgoing network data, replaying that data in different networks, or creating routing confusion by broadcasting wrong messages.

Man-in-the-middle attacks

- **Intercept and change data:**
  Flaws on protocol-level design or implementation-level might allow to interfere the communication link between two parties and intercept and change the exchanged data.

Bad-data injection attacks

- **Compromises Sensor Readings**:
  An attacker compromises sensor readings to introduce undetected errors in the calculations.

Malicious Hardware/Software Injections

- **Unauthorised injection**
  Inject unauthorised hardware components as well as unauthorised software into the edge network.
- **Node replication:**
  A node is added to the network with an existing ID.
- **Hardware trojan injection:**
  Control integrated circuits of a node already at build time

Physical Tampering and Attacks

- **Change or modification:**
  Node software or operating system

- **Manipulation:**
  Node circuits
- **Extraction:**
  Sensitive cryptographic information
- **Destruction:**
  Nodes

7.1.3 Root causes

Protocol-Level Design Flaws The vulnerability is already part of the design of the protocol and not a bug in the implementation.

Implementation-Level Flaws The protocol is secure, but there are logic flaws in the implementation. This might come from a misunderstanding of the protocol or by inconsistencies due to the migration of the protocol from other contexts into the edge-computing environment.

Code-Level Vulnerabilities The protocol is secure and there are no logic flaws. But the programming introduces the risk for cause memory failures and corruption, e.g., format string vulnerability, heap overflow, stack overflow, use-after-free with dangling pointer.

Lacking Fine-Grained Access Controls Due to limitations of the edge-computing environment, the barriers of gaining access rights that are higher than foreseen can open the door for different types of attacks (man-in-the-middle, authorisation attacks).

Data Correlations There might be a relation between the insensitive data that is produced in the edge computing environment and other sensitive data, that is not obvious. If someone can identify that relation, it opens the path to some types of attacks.

7.1.4 CWE Codes

CWE manages a list of common vulnerabilities, that can be referenced by a number or code. Numbered categories combine a set of vulnerabilities, in case these vulnerabilities belong to the same group of problems.

Code-Level Vulnerabilities
The descriptions of the workflows are referring to programming errors that causes problems like memory leakage and buffer overflows, which fall under the root cause of this section.

- CWE-118: Incorrect Access of Indexable Resource ('Range Error')
- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer
- CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
- CWE-121: Stack-based Buffer Overflow
- CWE-122: Heap-based Buffer Overflow
- CWE-125: Out-of-bounds Read Before Release ('Heap Inspection')
- CWE-134: Use of Externally-Controlled Format String
- CWE-244: Improper Clearing of Heap Memory
- CWE-401: Missing Release of Memory after Effective Lifetime
- CWE-415: Double Free
- CWE-416: Use After Free
- CWE-465: Pointer Issues

- CWE-787: Out-of-bounds Write
- CWE-788: Access of Memory Location After End of Buffer
- CWE-825: Expired Pointer Dereference
- Category-465: Pointer Issues

Protocol-Level Design Flaws & Lacking Fine-Grained Access Controls
Another important elements of the workflows are authentication and key agreement. Flaws in that area are part of the flaws in the protocol design as well as flaws specific to access control.

- CWE-284: Improper Access Control
- CWE-287: Improper Authentication
- CWE-295: Improper Certificate Validation
- CWE-303: Incorrect Implementation of Authentication Algorithm
- CWE-322: Key Exchange without Entity Authentication
- CWE-670: Always-Incorrect Control Flow Implementation
- Category-320: Key Management Errors
- Category-957: SFP Secondary Cluster: Protocol Error

Data Correlation
In regard to data privacy as described in the workflows, flaws that enable the disclosure of sensitive data is important.

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
- CWE-213: Exposure of Sensitive Information Due to Incompatible Policies
- CWE-311: Missing Encryption of Sensitive Data
- CWE-312: Cleartext Storage of Sensitive Information
- CWE-319: Cleartext Transmission of Sensitive Information
- CWE-359: Exposure of Private Personal Information to an Unauthorized Actor
- CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere
- CWE-668: Exposure of Resource to Wrong Sphere
- CWE-693: Protection Mechanism Failure
- CWE-921: Storage of Sensitive Data in a Mechanism without Access Control
- CWE-922: Insecure Storage of Sensitive Information
- Category-934: OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure
- Category-1029: OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure

OWASP Top Ten 2021 Categories
It is worth to mention that some vulnerabilities, which are relevant for the described workflows, fall into the top ten 2012 categories of OWASP. These categories combine vulnerabilities of the different root causes.

- Category-1345: OWASP Top Ten 2021 Category A01:2021 - Broken Access Control
- Category-1347: OWASP Top Ten 2021 Category A03:2021 - Injection
- Category-1348: OWASP Top Ten 2021 Category A04:2021 - Insecure Design
- Category-1352: OWASP Top Ten 2021 Category A06:2021 - Vulnerable and Outdated Components
- Category-1353: OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures
- Category-1354: OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures
- Category-1356: OWASP Top Ten 2021 Category A10:2021 - Server-Side Request Forgery (SSRF)

7.2 Legacy technologies management evaluation

The goal of this workflow is hardening legacy components. As such, the effectiveness of this workflow breaks down to the effectiveness of its constituents, which we discuss in detail below.

7.2.1 Control-flow integrity for edge devices evaluation

As regards the control-flow integrity solution described in 3.2 and demonstrated in 6.1.3, the following parameters described in 7.1 can be taken into consideration:

- **Attack surface**: Weak Computation Power and Poorly Secured Devices, Attack Unawareness
- **Attack classification**: Malware injection attacks (Device-side injections)
- **Root causes**: Code-Level Vulnerabilities
- **CWE Codes**: All numbers related to Code-Level Vulnerabilities as reported in 7.1.4

The solution proved to be 100% effective in avoiding control-flow redirection attacks in the specific application scenario within the smart building demonstrator (Figure 7.1, also cfr. [34]).

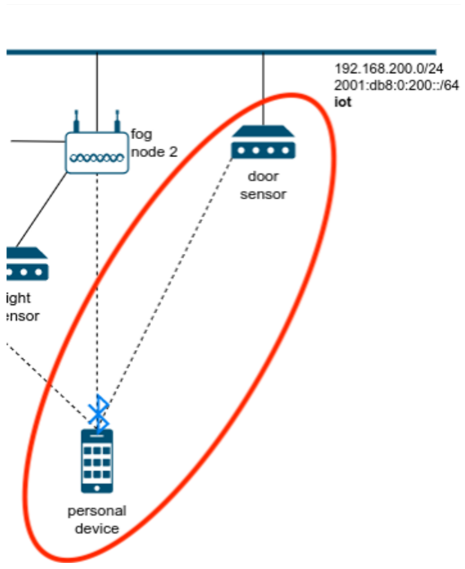


Figure 7.1: The door sensor component inside Legacy technologies management workflow in the common WP6 use case infrastructure.

Here, a physical access monitoring application within the facility was built using the GATT protocol over BLE. The smartphone, acting as a GATT server, communicates to the GATT client (i.e., the door sensor), which asks for a unique identification key. The smartphone instead acts as a malicious agent by exploiting a buffer overflow vulnerability that is inside a `readChar()` function, which lies in the only connection to the user outside the network and which internally uses the unsafe `strcpy()` instead of the `memcpy()`. The attacker thus manages to send more characters than the ones required for the identifier, corrupting the stack and inserting the address of a single gadget, i.e., the address of the `recordSet()` function, plus some parameters necessary for the attack to work. Through this, the attacker is successful in changing the admin access PIN, allowing a subsequent successful login from the admin side during a connection to the door sensor attempted from the internal network. More details on the attack experiments can be found at [39].

7.2.2 RIOT-AKA evaluation

RIOT-AKA aims to cover multiple aforementioned vulnerabilities as described in 3.4 and demonstrated in 6.1.5. The following parameters described in 7.1 can be taken into consideration:

- **Attack surface**: Weak Computation Power and Poorly Secured Devices, Attack Unawareness

- **Attack classification**: Authentication and authorisation attacks
- **Root causes**: Protocol-Level Design Flaws and Lacking Fine-Grained Access Controls
- **CWE Codes**: All numbers related to Protocol-Level Design Flaws & Lacking Fine-Grained Access Controls as reported in 7.1.4. Data correlation codes will be also taken into account in future iterations.

The detailed security analysis of the robustness of the protocol is reported in [20]. We remind that RIOT-AKA implementations is still at a premature prototype state so deep evaluation of the practical effectiveness of its solution will be showed in future releases.

7.2.3 Evaluation of secure software updates for wearable low-power IoT with RIOT, SUIT, and femto-containers

The open source platform we have developped for the PEPPER demo (low-power IoT wearables, see Section 6.1.1) has wider applicability beyond this use case. Based on RIOT, SUIT end-to-end security, and femto-containers and standard low-power network protocols, our platform can provide security-enhanced embedded system software and secure updates over the network for general-purpose IoT OS firmware update or for more lightweight and more specific IoT software modules hosted and isolated in femto-containers.

Our software (firmware or femto-container) update security guarantees include:

- 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.
- Unauthorized Firmware 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.
- 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.
- 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.

Moreover, we evaluate the performance of femto-containers as general-purpose building block for secure DevOps and Function-as-a-Service use cases in more details in the report we published in [89]. In this report we analyze that femto-containers incur very small memory footprint overhead (below 10%) and very small startup time (tens of microseconds) compared to native code execution. We also show that Femto-Containers can satisfy the constraints of both low-level debug logic inserted in a hot code path, and high-level business logic coded in a variety of common programming languages. Compared to prior work, femto-containers thus offer an attractive new trade-off in terms of memory footprint, energy consumption, agility and security.

7.3 Intrusion management workflow evaluation

The effectiveness of this workflow depends on the following two aspects.

1. The effectiveness of intrusion detection methodologies in identifying dangerous activities.
2. The effectiveness of the Fog reorchestration mechanisms in reacting to critical, intrusion events.

Below we discuss these two idems in detail.

7.3.1 Evaluation of the Anomaly-based Intrusion Detection System

The anomaly-based intrusion detection system aims at identifying deviations from a learned model of normal behaviors. Referring to the classification defined in Section 7.1, the proposed IDS can be characterized as follows:

- **Attack surface**: different attack surfaces can be considered but the main ones are obviously ≪ Weak Computation Power and Poorly Secured Devices ≫ and ≪ Attack Unawareness ≫.
- **Attack classification**: Any class of attack can be detected as long as it violates the behavior model used by the IDS. When the analysis is based on an observation of the network traffic, DDos attacks which have often an high impact on the communication pattern are among the important targets of the IDS.
- **Root causes**: any. Indeed, the tool focuses on the consequences of an attack rather than on its causes.
- **CWE Codes**: mainly those that are related to ≪ Code-Level Vulnerabilities ≫.

Note that, during the learning phase, the tool requires only unlabeled traces corresponding to normal behaviors. Consequently, during the detection phase, when an alert is raised, the class of attack that has been conducted cannot be identified. Only an analysis of the reasons that led to raise an alert can provide hints about the followed attack scenario.

Evaluation of the IDS tool have been performed using a distributed application, namely a distributed file system called XtreemFS (http://www.xtreemfs.org/). Main results appear in the previous delivered document [34]. The adaptation of the tool to cope with network traffic has also been evaluated using a well-known dataset that includes several attack scenarios (CIC-IDS-2017 dataset: https://www.unb.ca/cic/datasets/ids-2017.html). The results show that the detection is clearly better when the events are corresponding to high level actions of a distributed application rather than to packets captured at a single point of the network. All the evaluations aim to analyse the quality of the detection (false positive, false negative, ...) but also the complementarities between the different sub-models that are used.

7.3.2 Evaluation of dynamic service orchestration in the Fog computing

Method of dynamic service orchestration in the Fog Computing as described in 4.3 and demonstrated in 6.2.3 is actually more related to the security countermeasures than to the attack detection. If used with external intrusion detection technologies it may be considered as a part of intrusion prevention system. In such context the method has such security characteristics:

- **Attack surface**: Weak Computation Power and Poorly Secured Devices
- **Attack classification**: DDoS attacks, Attacks exploiting power consumption
- **Root causes**: may vary

The method allows Intelligent Infrastructure to react to the changes and reorganize itself according to the preferences of the area of application. In such way the method may protect the infrastructure from some attacks exploiting power consumption by reorchestrating services to Fog nodes with bigger energy reserves. The services also may be dynamically moved from compromised devices or devices under DDoS attack to other, more secure devices, thus mitigating the risks of potential service availability problems. The implementation of the method was performed using prototype system. The experimental result are published in [68]. All practical implementations of the infrastructure still are at a prototype state so more detailed experimental evaluation will be performed in the future research.

7.4 Privacy and data management evaluation

The effectiveness of this workflow is related to two distinct aspects, i.e., (i) the effectiveness of the DPO tool in detecting design flaws and (ii) the effectiveness of PEAS and MC-SSE tool in dealing with such flaws. In the following, we discusse these aspects in detail.

7.4.1 Evaluation of DPO tool

There are several initiatives to evaluate and validate the Data Protection Officer tool (DPO tool). Firstly, in the deliverable D6.3 we have discussed how DPO tool corresponds to the requirements of the ISO27701:2019 standard. The overview presented the standard controls included in the DPO tool, controls partially included in the DPO tool, and control not included in the DPO tool.

Secondly, despite the vehicle charge process, which is discussed in this document, DPO tool is also applied for a few other cases. For instance, in [63] the Tollgate scenario is considered. Elsewhere in [16], the DPO tool is applied to explain how personal passenger's data could be protected in the interaction with the autonomous vehicle. In the later study, an integration of the DPO results to the privacy leakage analysis is discussed, too. These DPO tool applications show the feasibility and usefulness to explain how business processes are compliant to the GDPR regulation.

7.4.2 Evaluation of PEAS

PEAS provides the security properties typical for privacy-preserving authentication schemes such as correctness, soundness, anonymity, and revocation. The detailed security analysis of PEAS can be found in the previous report D6.3 and the papers [24] and [42]. The first performance results of PEAS have been published in D6.3. To prove the practicality of the design, PEAS has been also implemented at the off-the-shelf card platforms (smart cards). The benchmarks and a comparison with existing solutions can be found in [42].

In addition, the PEAS implementation has been enhanced by a web-based GUI and the Android applications. Figure 7.2 shows how a smart phone (Nokia 7.2) with the Android PEAS application speeds up the PEAS authentication protocol in comparison with the smart card (Multos ML4). Users using smart cards for storing their attributes need for the authentication process around 2.6 seconds. Users with the Android-based PEAS application can prove their attributes within less than 461 ms.

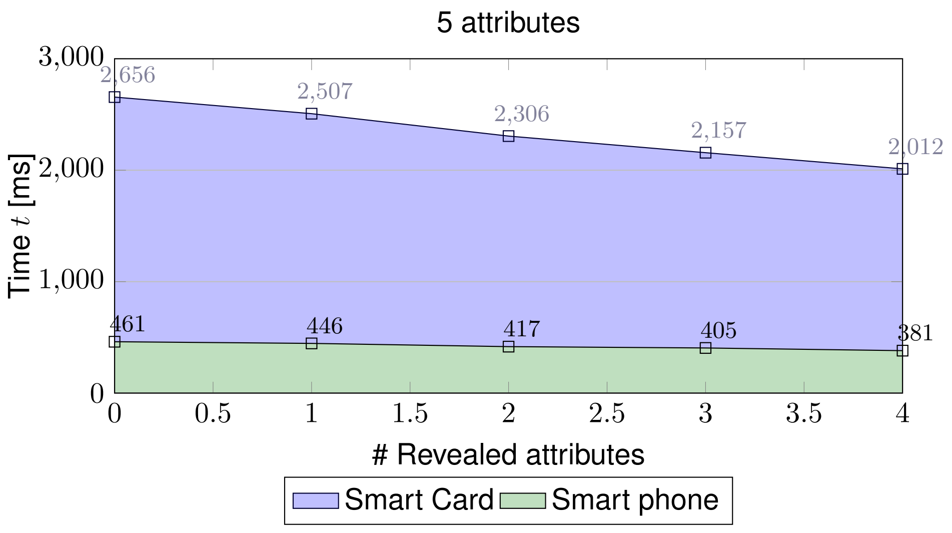


Figure 7.2: Performance test of PEAS authentication phase (Smart Card MultOS ML 4 and Smartphone Nokia 7.2).

7.4.3 Evaluation of the MC-SSE tool

The MC-Clusion library implementation was evaluated by performing experiments with up to 1M documents, containing synthetic parking transactions, resulting in 21M document-keyword pairs. Experiments were executed on the Grid'5000 testbed [1] with Intel Xeon Gold 6130 (Skylake, 2.10GHz, 4 CPUs/node, 16 cores/CPU) processors and 60GB of RAM, running Debian 11 (64-bit) OS. The experimental results confirm the correct functionality of the multi-client extension of the Clusion library. The performance evaluation of the library implementation shows that the properties of the original BIEX SSE scheme algorithm are retained, offering practical and efficient boolean search functionality. Finally, the MC-SSE demonstration implementation also verifies the usability of the solution, illustrated through the parking use case scenario.

Chapter 8 Conclusion

In this document we presented the final demonstration of the HAII-T. Our integration strategy leveraged on the HAII-T orchestrator for combining all the methodologies and tools developed in WP6. Also, to better highlight how HAII-T supports the Security-by-Design approach, we put forward three security workflows dealing with common issues which affect many II. Below, we briefly discuss some conclusive remarks for each of them.

Legacy technologies management

We remind that this conclusions are derived from work showed in Section 3.4 and 6.1.5. We confirm that widely established authentication means, exploited since decades in the cellular networking domain, can be also considered for IoT devices. Their simple and consolidated roaming model fits very well with the need for IoT deployments to permit relocation of objects or change of smart space ownership. We still require to address many other aspects which our preliminary proof-of-concept work has so far neglected. Indeed, our current work in progress revolves around two main goals: i) hardening and extension of the protocol, with more thorough registration and management procedures and handling of desynchronization events/attacks, and ii) integration with security protocols, specifically (D)TLS and OSCORE, already available in the RIOT OS.

Intrusion management

The Intrusion management workflow shows how Intelligent Infrastructure is able to adapt itself to the changing environment and reorganize services in order to maximize security or comply with other important QoS requirements. It was demonstrated that Intrusion detection systems, based on Machine learning and Anomaly detection approaches, are able to detect possible security risks affecting devices comprising the Intelligent Infrastructure. The demonstrator also shows that multi-agent based implementation of Fog layer services allows to implement orchestrator services, which are able to constantly collect current information about dynamic infrastructure as well as to perform service migrations from security compromised Fog nodes to more secure ones. We used Human-in-the-loop approach to mitigate the risks of false positive alarms affecting the system and degrading its performance. Integration of multiple Intrusion detection systems using different technologies and dynamic service orchestration results in an interesting system which could be called Intrusion detection and prevention solution.

Data and privacy management

The data and privacy management workflow focused on practical aspects of the privacy-by-design approach and presented how various PETs tools can be deployed in HAII-T infrastructure, i.e., smart campus, smart building etc. Privacy-Enhancing Authentication System (PEAS) has been presented and demonstrated on common smart campus services, i.e., checking parking permits and access to protected areas. Using the Android-based PEAS user application and multi-platform web-based PEAS applications for Verifier and Issuer may provide an efficient and privacy-preserving authentication system in IIs. Then, the Multi-client - Searchable Symmetric Encryption (MC-SSE) tool has been proposed to support privacy-preserving data processing for the intelligent infrastructure services. Finally, we also presented the DPO tool - Data Protection Officer tool for checking business process compliance to the GDPR regulation. The three tools can be used in combination to construct a practical common application scenario, presented in Section 6.3. In conclusion, each of the three PET tools covers a different privacy aspect (regulation compliance, authentication and data processing) and all three together provide a practical toolset that can offer privacy support for intelligent infrastructure services.

Chapter 9 Bibliography

[1] Grid'5000 testbed. https://www.grid5000.fr/.

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

[3] Microsoft threat modeling tool. https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool. Accessed at 2021-04-26.

[4] Getting around non-executable stack (and fix). https://seclists.org/bugtraq/1997/Aug/63, 1997. [Online; accessed 07-June-2021].

[5] DP3T Decentralized Privacy-Preserving Proximity Tracing. https://github.com/DP-3T/documents, 2020. [Online; accessed 01-Dec-2021].

[6] Top Programming Languages 2021 - IEEE Spectrum. https://spectrum.ieee.org/top-programming-languages/, 2021. [Online; aAccessed December 07, 2021].

[7] PEPPER Demo Video. https://github.com/future-proof-iot/PEPPER, 2021. [Online; accessed 13-Dec-2021].

[8] SUIT-based Security-Enhanced IoT Operating System Software. https://github.com/future-proof-iot/RIOT/tree/H2020-Sparta-Deliverable-D6-2, 2021. [Online; accessed 01-Dec-2021].

[9] Singapore TraceTogether Token. https://token.gowhere.gov.sg/, 2021. [Online; accessed 01-Dec-2021].

[10] CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer. https://cwe.mitre.org/data/definitions/119.html, 2021. [Online; Accessed December 07, 2021].

[11] CWE-401: Missing Release of Memory after Effective Lifetime. https://cwe.mitre.org/data/definitions/401.html, 2021. [Online; accessed 07-June-2021].

[12] Femto-Containers Tutorials. https://github.com/future-proof-iot/Femto-Container_tutorials, 2021. [Online; accessed 13-Dec-2021].

[13] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information and System Security (TISSEC)*, 13 (1):1–40, 2009.

[14] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. *Acm Sigplan Notices*, 36(3):104–115, 2001.

[15] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just fast keying in the pi calculus. *ACM Transactions on Information and System Security (TISSEC)*, 10(3):9–es, 2007.

[16] M. Bakhtina. Securing Passenger's Data in Autonomous Vehicles. Master's thesis, University of Tartu, 2021.

[17] Elaine Barker. Nist special publication 800-57 part 1 revision 4, recommendation for key management part 1: General. *NIST*, 2016.

[18] Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. Jade — A Java Agent Development Framework. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 125–147. Springer US, Boston, MA, 2005. ISBN 978-0-387-26350-2. doi: 10.1007/0-387-26350-0_5.

[19] Karthikeyan Bhargavan, Cédric Fournet, Ricardo Corin, and Eugen Zalinescu. Cryptographically verified implementations for tls. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 459–468, 2008.

[20] Giuseppe Bianchi, Alberto La Rosa, and Gabriele Restuccia. Riot-aka: cellular-like authentication over iot devices. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–6, 2021. doi: 10.1109/ICNP52444.2021.9651952.

[21] 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.

[22] Daniel Brett. Is edge computing secure? here are 4 security risks to be aware of. Online (accessed 2021-12-13), December 2020. URL https://www.trentonsystems.com/blog/is-edge-computing-secure.

[23] E. Buchanan, R. Roemer, H. Shacham, and S. Savage. When good instructions go bad: Generalizing return-oriented programming to risc. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 27–38. ACM, 2008.

[24] 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.

[25] 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.

[26] 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.

[27] Claude Castelluccia, Nataliia Bielova, Antoine Boutet, Mathieu Cunche, Cédric Lauradoux, Daniel Le Métayer, and Vincent Roca. Robert: Robust and privacy-preserving proximity tracing. 2020.

[28] Claude Castelluccia, Nataliia Bielova, Antoine Boutet, Mathieu Cunche, Cédric Lauradoux, Daniel Le Métayer, and Vincent Roca. Desire: A third way for a european exposure notification system leveraging the best of centralized and decentralized systems. *arXiv preprint arXiv:2008.01621*, 2020.

[29] 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.

[30] Richard Chang and Vitaly Shmatikov. Formal Analysis of Authentication in Bluetooth Device Pairing. FCS-ARSPA07, 2007.

[31] Ionuț-Adrian Cojoacă, Constantin Bulac, and Claudiu-Ionuț Popîrlan. A proposed multi-agent based platform for monitoring and control of Active Power Distribution Systems. In *2021 3rd Global Power, Energy and Communication Conference (GPECOM)*, pages 214–219. IEEE, 2021.

[32] F. Conceicao, N. Oualha, and D. Zeghlache. Security Establishment for IoT Environments in 5G: Direct MTC-UE Communications. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–5, Oct 2017. doi: 10.1109/PIMRC.2017.8292693.

[33] 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, Katharina 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.

[34] 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, Katharina Hofer-Schmitz, Marek Pawlicki,

Tewodros Beyene, and Nerijus Morkevicius. *D6.3 First Release of Demonstration*. SPARTA, Feb 2021.

[35] Roudy Dagher, Francois-Xavier Molina, Alexandre Abadie, Nathalie Mitton, and Emmanuel Baccelli. An open experimental platform for ranging, proximity and contact event tracking using ultra-wide-band and bluetooth low-energy. In *CNERT 2021-IEEE INFOCOM Workshop on Computer and Networking Experimental Research using Testbeds*, 2021.

[36] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[37] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Multi-Agent Systems: A Survey. *IEEE Access*, 6: 28573–28593, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2831228.

[38] EnOceanAlliance. Security of EnOcean Radio networks v2.5. URL https://www.enocean-alliance.org/wp-content/uploads/2019/04/Security-of-EnOcean-Radio-Networks-v2_5.pdf.

[39] Antonio Ettore Epifani. *Control-Flow Integrity for Embedded Systems: Study Case of an FPGA-Based Solution*. PhD thesis, Politecnico di Torino, 2021.

[40] Valentina Forte, Nicolò Maunero, Paolo Prinetto, and Gianluca Roascio. Prolepsis: Binary analysis and instrumentation of iot software for control-flow integrity. In *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, pages 1–6, 2021. doi: 10.1109/ICECCME52200.2021.9591080.

[41] Vangelis Gazis, Nikos Housos, Athanasia Alonistioti, and Lazaros Merakos. Generic system architecture for 4g mobile communications. In *The 57th IEEE Semiannual Vehicular Technology Conference, 2003. VTC 2003-Spring.*, volume 3, pages 1512–1516. IEEE, 2003.

[42] Jan Hajny, Petr Dzurenda, Raul Casanova-Marques, and Lukas Malina. Privacy abcs: Now ready for your wallets! In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 686–691, 2021. doi: 10.1109/PerComWorkshops51409.2021.9431139.

[43] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6, 02 1995. doi: 10.1007/BF01211866.

[44] D. Harel and B. Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"? *Computer*, 37(10):64–72, 2004. doi: 10.1109/MC.2004.172.

[45] Jaap-Henk Hoepman. A critique of the google apple exposure notification (gaen) framework. *arXiv preprint arXiv:2012.05097*, 2020.

[46] Katharina Hofer-Schmitz. A formal analysis of enocean's teach-in and authentication. In *The 16th International Conference on Availability, Reliability and Security*, pages 1–8, 2021.

[47] M. Howard and LeBlanc D. *Writing secure code*. Redmond: Microsoft Press, 2014.

[48] ISO/IEC. Iso/iec 14543-3-10:2012. URL http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/98/59865.html.

[49] 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.

[50] 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.

[51] Georgios Kambourakis, Constantinos Kolias, Dimitrios Geneiatakis, Georgios Karopoulos, Georgios Michail Makrakis, and Ioannis Kounelis. A state-of-the-art review on the security of mainstream iot wireless pan protocol stacks. *Symmetry*, 12(4):579, 2020.

[52] Ashish Khaira and R. K. Dwivedi. A State of the Art Review of Analytical Hierarchy Process. *Materials Today: Proceedings*, 5(2, Part 1):4029–4035, 2018. ISSN 2214-7853. doi: https://doi.org/10.1016/j.matpr.2017.11.663.

[53] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 435–450. IEEE, 2017.

[54] David Kotz and Travis Peters. Challenges to ensuring human safety throughout the life-cycle of smart environments. In *Proceedings of the 1st ACM Workshop on the Internet of Safe Things*, pages 1–7, 2017.

[55] S. Krivokuća, B. Stojanović, K. Hofer-Schmitz, N. Nešković, and A. Nešković. Smart water distribution system communication architecture risk analysis using formal methods. In *2020 28th Telecommunications Forum (TELFOR)*, pages 1–4, 2020. doi: 10.1109/TELFOR51502.2020.9306654.

[56] Alberto La Rosa. Riot-security/examples/riot_aka at sparta-dev deus-ex-mortis/riot-security. GitHub, . URL https://github.com/Deus-Ex-Mortis/RIOT-security/tree/SPARTA-dev/examples/riot_aka.

[57] Alberto La Rosa. Deus-ex-mortis/aiocoap. GitHub, . URL https://github.com/Deus-Ex-Mortis/Aiocoap.

[58] Douglas J Leith and Stephen Farrell. Measurement-based evaluation of google/apple exposure notification api for proximity detection in a light-rail tram. *Plos one*, 15(9):e0239943, 2020.

[59] 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.

[60] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166. Springer, 1996.

[61] Lukas Malina, Petr Dzurenda, Sara Ricci, Jan Hajny, Gautam Srivastava, Raimundas Matulevičius, Abasi-Amefon O. Affia, Maryline Laurent, Nazatul Haque Sultan, and Qiang Tang. Post-quantum era privacy protection for intelligent infrastructures. *IEEE Access*, 9:36038–36077, 2021. doi: 10.1109/ACCESS.2021.3062201.

[62] S. Marksteiner, V. J. E. Jimenez, H. Valiant, and H. Zeiner. An Overview of Wireless IoT Protocol Security in the Smart Home Domain. In *2017 Internet of Things Business Models, Users, and Networks*, pages 1–8, Nov 2017. doi: 10.1109/CTTE.2017.8260940.

[63] R. Matulevičius, J. Tom, K. Kala, and E. Sing. a method for managing gdpr compliance in business processes. In *CAiSE Forum 2020: 100-112 CAiSE Forum 2020: 100-112 CAiSE 2020 Forum*, pages 100–112.

[64] 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. IEEE, 2020.

[65] Nicoló Maunero, Paolo Prinetto, and Gianluca Roascio. Cfi: Control flow integrity or control flow interruption? In *2019 IEEE East-West Design Test Symposium (EWDTS)*, pages 1–6, 2019. doi: 10.1109/EWDTS.2019.8884464.

[66] MITRE. CWE Version 4.6. Online (accessed 2021-12-13), October 2021. URL https://cwe.mitre.org/data/published/cwe_v4.6.pdf.

[67] M. Mohsin, M. U. Sardar, O. Hasan, and Z. Anwar. IoTRiskAnalyzer: A probabilistic model checking based framework for formal risk analytics of the internet of things. *IEEE Access*, 5: 5494–5505, 2017. ISSN 2169-3536. doi: 10.1109/ACCESS.2017.2696031.

[68] Nerijus Morkevicius, Algimantas Venčkauskas, Nerijus Šatkauskas, and Jevgenijus Toldinas. Method for Dynamic Service Orchestration in Fog Computing. *Electronics*, 10(15), 2021. ISSN 2079-9292. doi: 10.3390/electronics10151796.

[69] Roger M Needham and Michael D Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[70] Agnė Paulauskaite-Taraseviciene, Egidijus Kazanavicius, Nerijus Morkevicius, Vaidas Jukavicius, and Laura Kizauskiene. Agent-Based System Architecture for Intelligent Lighting Control Based on Resident's Behavior. *International Journal of Modeling and Optimization*, 5(1):48–54, 2015. ISSN 2010-3697. doi: 10.7763/ijmo.2015.v5.435.

[71] Eric Rescorla and Tim Dierks. The transport layer security (tls) protocol version 1.3. 2018.

[72] Gabriele Restuccia, Hannes Tschofenig, and Emmanuel Baccelli. Low-Power IoT Communication Security: On the Performance of DTLS and TLS 1.3. *in Proceedings of IFIP/IEEE PEMWN*, December 2020.

[73] 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.

[74] Mark Dermot Ryan and Ben Smyth. Applied pi calculus. *Formal Models and Techniques for Analyzing Security Protocols*, 5:112–142, 2011.

[75] Thomas Saaty and Luis Vargas. The Seven Pillars of the Analytic Hierarchy Process. In *Int. Ser. Oper. Res. Manage. Sci.*, volume 175, pages 27–46. July 2011. doi: 10.1007/978-1-4615-1665-1_2. Journal Abbreviation: Int. Ser. Oper. Res. Manage. Sci.

[76] Göran Selander, John Mattsson, Francesca Palombini, and Ludwig Seitz. Object security for constrained restful environments (oscore). *Work in Progress*, 2019.

[77] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[78] J. Tom, E. Sing, and R. Matulevičius. Conceptual Representation of the GDPR: Model and Application Directions. In *BIR 2018*, pages 18–28, 2018.

[79] Rajae Touzani, Emilien Schultz, Seth M Holmes, Stéphanie Vandentorren, Pierre Arwidson, Francis Guillemin, Dominique Rey, Alexandra Rouquette, Anne-Déborah Bouhnik, Julien Mancini, et al. Early acceptability of a mobile app for contact tracing during the covid-19 pandemic in france: national web-based survey. *JMIR mHealth and uHealth*, 9(7):e27768, 2021.

[80] M. Tran, M. Etheridge, T. Bletsch, X. Jiang, V. Freeh, and P. Ning. On the expressiveness of return-into-libc attacks. In *International Workshop on Recent Advances in Intrusion Detection*, pages 121–141. Springer, 2011.

[81] Heribert Vallant, Branka Stojanović, Josip Božić, and Katharina Hofer-Schmitz. Threat modelling and beyond-novel approaches to cyber secure the smart energy system. *Applied Sciences*, 11 (11):5149, 2021.

[82] Serge Vaudenay. Analysis of dp3t. Cryptology ePrint Archive, Report 2020/399, 2020. https://ia.cr/2020/399.

[83] Serge Vaudenay. Centralized or decentralized? the contact tracing dilemma. Technical report, 2020.

[84] Jaikumar Vijayan. 4 ways edge computing changes your threat model. Online (accessed 2021-12-13), May 2020. URL https://www.csoonline.com/article/3543191/4-ways-edge-computing-changes-your-threat-model.html.

[85] Yatin Wadhawan, Anas AlMajali, and Clifford Neuman. A comprehensive analysis of smart grid systems against cyber-physical attacks. *Electronics*, 7(10):249, 2018.

[86] Yinhao Xiao, Yizhen Jia, Chunchi Liu, Xiuzhen Cheng, Jiguo Yu, and Weifeng Lv. Edge computing security: State of the art and challenges. *Proceedings of the IEEE*, 107(8):1608–1631, 2019. doi: 10.1109/JPROC.2019.2918437.

[87] Jing Xie and Chen-Ching Liu. Multi-agent systems and their applications. *Journal of International Council on Electrical Engineering*, 7(1):188–197, 2017. doi: 10.1080/22348972.2017.1348890.

[88] Koen Zandberg and Emmanuel Baccelli. Minimal virtual machines on iot microcontrollers: The case of berkeley packet filters with rbpf. In *2020 9th IFIP International Conference on Performance Evaluation and Modeling in Wireless Networks (PEMWN)*, pages 1–6. IEEE, 2020.

[89] Koen Zandberg and Emmanuel Baccelli. Femto-containers: Devops on microcontrollers with lightweight virtualization & isolation for iot software modules. 2021.

[90] 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.

[91] J. Zhang, L. Yang, W. Cao, and Q. Wang. Formal Analysis of 5G EAP-TLS Authentication Protocol Using Proverif. *IEEE Access*, 8:23674–23688, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2969474.

[92] Yu Zheng, Dake He, Xiaohu Tang, and Hongxia Wang. Aka and authorization scheme for 4g mobile networks based on trusted mobile platform. In *2005 5th International Conference on Information Communications & Signal Processing*, pages 976–980. IEEE, 2005.