## Contents

1. Introduction 1
   1.1. Motivation 1
   1.2. Methodology 2
   1.3. Document Organization 3

2. Architecture Overview 4
   2.1. Privacy-enforceable Runtime Architecture 4
       2.1.1. Technical Enforcement of Privacy Policies 4
       2.1.2. PeRA Components Summary 6
   2.2. Runtime Use Case Examples 8
       2.2.1. Collision Warning 10
       2.2.2. Floating Car Data 15
       2.2.3. Online Navigation 21
       2.2.4. Hotel Booking 24

3. Policies and Queries 26
   3.1. Query Language PPQL 28
       3.1.1. Operation Requirements 28
       3.1.2. Supported operations 29
       3.1.3. SQL-Light 30
       3.1.4. PeRA components realizing supported operations 31
   3.2. Privacy Metadata 33
       3.2.1. Policy Language P3L 35

4. Mandatory Privacy Control 42
   4.1. Privacy Control Monitor 43
       4.1.1. Purpose 44
       4.1.2. Interfaces and Services 44
       4.1.3. Description 46
       4.1.4. Platform Considerations 51
   4.2. Privacy Policy Manager 51
       4.2.1. Purpose 51
       4.2.2. Interfaces and Services 52
       4.2.3. Description 52
       4.2.4. Discussion 53
   4.3. Controlled Application Environment 53
       4.3.1. Purpose 53
       4.3.2. Dependencies and Prerequisites 55
4.3.3. Interfaces and Services ........................................ 55
4.3.4. Description ...................................................... 59
4.3.5. Platform Considerations ....................................... 62
4.3.6. Discussion ....................................................... 62

5. MPC Integrity Protection ................................. 63
  5.1. Trust Manager .................................................. 64
    5.1.1. Purpose ...................................................... 64
    5.1.2. Dependencies and Prerequisites .......................... 64
    5.1.3. Interfaces and Services .................................. 64
    5.1.4. Description .................................................. 67
    5.1.5. Platform Considerations ................................ 75
    5.1.6. Additional Material ...................................... 76
    5.1.7. Discussion .................................................. 76
  5.2. Communication Endpoints .................................... 78
    5.2.1. Importer ..................................................... 80
    5.2.2. Exporter ..................................................... 84
  5.3. Non-interactive Remote Attestation ...................... 88

6. Secure Data and Metadata Repository .............. 90
  6.1. Data storage capabilities .................................... 90
  6.2. Data model ..................................................... 90
    6.2.1. Policies ..................................................... 90
    6.2.2. Ontology ................................................... 91
    6.2.3. Application data ......................................... 93
    6.2.4. Audit data ................................................ 93
    6.2.5. Capabilities for data storage in PeRA ................ 94
  6.3. Platform considerations ................................... 94
  6.4. Discussion ..................................................... 95

7. Additional Components ................................. 96
  7.1. Query IDS ...................................................... 96
  7.2. Privacy Maintenance ......................................... 98
  7.3. Privacy Trail Manager ....................................... 99

8. Communication Privacy ................................. 100
  8.1. Requirements on Communication Privacy .............. 100
    8.1.1. Review of use cases .................................... 100
    8.1.2. Communication patterns ................................. 101
    8.1.3. Privacy requirements ................................... 102
  8.2. Components in Public Communication API ........... 103
    8.2.1. Components overview ................................... 103
    8.2.2. Security manager ....................................... 103
    8.2.3. Identification Manager ................................. 104
    8.2.4. Trust Manager ............................................ 105
    8.2.5. Pseudonym Manager .................................... 107
8.2.6. Pseudonym Application .............................................. 110
8.2.7. Secure Communication Module .................................... 111
8.2.8. Hardware Security Module and Crypto Support .................. 112
8.3. Enhancing Conditional Pseudonymity ................................. 112
  8.3.1. Privacy-preserving Pseudonym Issuance .......................... 114
  8.3.2. Collaborative Identity Resolution .................................. 115

9. Conclusion ........................................................................ 117
  9.1. Compliance to the principles for hippocratic cITS .................... 117
  9.2. Outlook ....................................................................... 120

A. Sequence Diagrams .......................................................... 121

Bibliography ......................................................................... 124
# List of Figures

1.1. Relations between deliverable D10 and other deliverables. . . . . . . . . . . . 2

2.1. The protection chain established by the PeRA. . . . . . . . . . . . . . . . . 5
2.2. An overview of the Privacy-enforcing Runtime Architecture. . . . . . . . . 6
2.3. A detailed look on the Privacy-enforcing Runtime Architecture. . . . . . . . 9
2.4. CW: Information flow to send beacon messages. . . . . . . . . . . . . . . . . 12
2.5. CW: Information flow upon reception of a beacon message. . . . . . . . . . . 13
2.6. CW: Information flow for the continuous analysis of stored status information. 14
2.7. FCD: Information flow inside the vehicle. . . . . . . . . . . . . . . . . . . . . 17
2.8. FCD: Information flow into the server. . . . . . . . . . . . . . . . . . . . . . 19
2.9. FCD: Information flow to the TraVis application. . . . . . . . . . . . . . . . . 20
2.10. Online navigation: Information flow in the TMC system . . . . . . . . . . . . 23

3.1. Traditional query execution environment. . . . . . . . . . . . . . . . . . . . . 26
3.2. Extended query execution environments according privacy requirements. . 27
3.3. Executing PPQL requests. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
3.4. PeRA components performing PPQL operations. . . . . . . . . . . . . . . . . 32
3.5. PeRA components providing privacy effect of PPQL operations. . . . . . . . 33
3.6. Assign policies with data and schemata or ontologies. . . . . . . . . . . . . 35
3.7. Exemplary ontology for a cITS scenario . . . . . . . . . . . . . . . . . . . . . 36
3.8. P3L user policy for collision detection . . . . . . . . . . . . . . . . . . . . . . 39
3.9. Primary elements of P3L . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
4.1. Interfaces between PCM and CAE . . . . . . . . . . . . . . . . . . . . . . . 45
4.2. Interfaces between PCM and Importer/Exporter . . . . . . . . . . . . . . . . 45
4.3. Interfaces between PCM and PPM . . . . . . . . . . . . . . . . . . . . . . . . 46
4.4. Interfaces between PCM and TM . . . . . . . . . . . . . . . . . . . . . . . . 47
4.5. Query execution including the evaluation of privacy information . . . . . . . 50
4.6. Interfaces of the Privacy Policy Manager . . . . . . . . . . . . . . . . . . . . 52
4.7. Information flow using tokens. . . . . . . . . . . . . . . . . . . . . . . . . . . 61

5.1. Establishing transitive trust with integrity measurement. . . . . . . . . . . . 68
5.2. Internal structure of the Trust Manager. . . . . . . . . . . . . . . . . . . . . 70
5.3. The Policy Enforcement Perimeter . . . . . . . . . . . . . . . . . . . . . . . 80
5.4. An Example for non-interactive remote attestation. . . . . . . . . . . . . . . 89

6.1. Triple in Directed Graph Data Model . . . . . . . . . . . . . . . . . . . . . . 92
6.2. Triple ontology representation in database . . . . . . . . . . . . . . . . . . . 92
6.3. Application data organisation in datastore . . . . . . . . . . . . . . . . . . . 93
8.1. Overview of components in Public COMM-API ........................................ 104
8.2. Enhanced pseudonym issuance. .......................................................... 113

A.1. A sensor provides data to the PERA .................................................... 121
A.2. A controlled application requests data from the DBMS ...................... 121
A.3. A controlled application exports data to an uncontrolled application .. 122
A.4. A controlled application requests data from a sensor ....................... 122
A.5. An uncontrolled application requests data from the DBMS ............... 123
A.6. An uncontrolled application invokes a method of an controlled application 123
List of Tables

3.1. SQL-Light examples .................................................. 31
3.2. Components and their implemented operations .......................... 31
3.3. TCC Data ................................................................. 38
3.4. CD Data ................................................................. 38
3.5. Predefined Elements of P3L ............................................. 40

5.1. TPM locality levels and anticipated usage .................................. 71

6.1. data model for policies .................................................. 91
6.2. data model for policies .................................................. 91
6.3. RDF ontology representation ............................................. 92
6.4. data model for ontologies ............................................... 93
6.5. data model for audit data ............................................... 94

8.1. Communication pattern analysis ....................................... 102
8.2. Module and component for public communication ..................... 104
1. Introduction

This deliverable (D10) provides a detailed specification of the PRECIOSA Privacy-enforcable Runtime Architecture (PeRA) and its components. While deliverable D7 takes a conceptual view on the architecture and gives a principal description, this deliverable describes mechanisms, components, and interfaces as a basis for architecture instantiation and future deployment of privacy-aware cooperative ITS.

1.1. Motivation

The aim of PRECIOSA is threefold. The development of a privacy-aware design process for cooperative ITS applications to facilitate privacy by design, the development of privacy guidelines for ITS applications, and the development of a privacy aware and verifiable architecture for cooperative ITS. Deliverable D7 [1] outlined the concepts of, both, the PRECIOSA cITS design process and the PRECIOSA runtime architecture, called Privacy-enforceable Runtime Architecture (PeRA). While D7 first defined a generic architecture framework and then a reference architecture, i.e., the PeRA. This deliverable is the direct continuation of the second part. While deliverable D7 described the PeRA on a conceptual level, we will now specify architecture components, mechanisms, and interfaces, in detail.

The aim is a structured and comprehensive specification of the PRECIOSA PeRA. In this course, decisions on specific mechanisms will be made to realize the PeRA concepts introduced in deliverable D7. This document will serve as a basis for the prototypical architecture implementation as part of the PRECIOSA project, but also as a reference for instantiation and deployment of privacy-aware cooperative ITS, in general.

Deliverable D10 is embedded in a set of other PRECIOSA deliverables. While deliverable D7 [1] introduces architecture concepts, it also provides ideas on the design of privacy policies, which is further extended in deliverable D6 Model and privacy ontology for V2X [2]. The specifications of the PRECIOSA privacy policy language and the query language in Chapter 3 draw from those deliverables. Deliverable D11 will provide guidelines for the design process of privacy-aware cooperative ITS applications. Deliverable D13 will describe privacy verification tool support during design-time and run-time and integration with the architecture. Finally, deliverable D14 will report and analyze how the PRECIOSA architecture has been instantiated and applied to an application based on the design methodology and mechanisms described in previous deliverables. Figure 1.1 illustrates the relationships between D10 and other deliverables. Note, that further relations exist between deliverables, but only those concerning D10 are shown.
1.2. Methodology

First, to validate the Privacy-enforceable Runtime Architecture (PeRA) as defined in D7, we utilize a use case driven verification process. Representative cooperative ITS use cases or applications are modeled and applied to the PeRA. This process ensures that components and interfaces between them are suitable and fitting. At the same time, due to the selection of different use cases that each are representative for a group of applications, the architecture can be validated for a large application base.

Each component and its subcomponents are then specified in detail in a coherent and structured manner. For this purpose, we define a component template, which is used to structure component descriptions in the rest of the document:

**Purpose.** Description of the component’s purpose. Why is it necessary? What are the component’s tasks?

**Prerequisites and Dependencies.** Description of any internal and external assumptions and requirements the component has towards the system. This may include assumptions on system software, hardware platform, external libraries, APIs, or other services. Anything required for the proper operation of the component should be listed here. Therefore, all other internal components the current component depends on should be listed as well.

**Interfaces and Services.** Description of any interfaces, data structures, and services that this component exports or provides to other components. Here, specifications should be as detailed as necessary in the component’s context.

**Description.** A detailed description of this component and its functionality, including all aspects that are relevant to understand the component’s role as part of the overall
architecture. Furthermore, employed mechanisms, technologies, and general approach taken should be outlined and detailed.

**Platform Considerations.** While some components may be implemented in the same way for vehicles and backend servers, other components may require special consideration on certain platforms. Platform dependent design and mechanism considerations are discussed here. Performance and scalability considerations may be discussed as well.

**Additional Material.** References to additional documents, specifications, or documentation, which are helpful for understand this components purpose, realization, or implementation.

**Discussion.** Extended discussion on design decisions and the chosen approach for the component. Advantages, disadvantages, and alternative designs can also be discussed to outline limitations as well as extension points.

This template is the base for the specification of each component. All component descriptions will be structured accordingly. Additionally, each component description starts with a table that summarizes the main characteristics of this component. The importance of the component for the overall system is also rated in that table as the component’s nature. We distinguish *core components*, which are integral to the architecture, *mandatory components*, which provide essential services and features, and *optional components* that provide enhancing services which the architecture does not rely on.

### 1.3. Document Organization

The remainder of this document is structured as follows. Chapter 2 provides a short overview of the PeRA architecture as defined in deliverable D7 [1] and discusses how a set of typical cITS use cases can be mapped to the architecture. Chapter 3 provides specification of the query language to interact with the PeRA, the PRECIOUSA privacy policy language, and the metadata format. Chapter 4 outlines components for mandatory privacy control. Chapter 5 defines components for privacy enforcement. Chapter 6 describes the data and metadata repository and its integration with a DBMS. Additional architecture components are described in Chapter 7. The employed communication privacy system, which ensures privacy on a communication and network level is outlined in Chapter 8. Chapter 9 concludes the deliverable with a summary and outlook.
2. Architecture Overview

2.1. Privacy-enforcable Runtime Architecture

In Deliverable D7 [1], we have discussed a comprehensive methodology for implementing ITS applications. Our methodology is comprised of elements that are applied at design time of an application and a runtime architecture that supports an ITS application. Both phases are necessary to take privacy into account at early stages of a project. After a privacy friendly application design has been selected during design time, it is the duty of the runtime architecture framework to protect and enforce privacy when the system is deployed and in use. The founding principles and main components of the PeRA will now be summarized for the readers’ convenience.

2.1.1. Technical Enforcement of Privacy Policies

In current IT systems, privacy regulations are only mandated on a legal level. That is, a service provider will show a certain privacy policy to the user, which is formulated as a contract, and the user then has the option to either accept or deny the policy. However, the latter will result in the user not being able to use the desired application. In this common scenario, it is not at the user’s discretion to govern how personal information is used by the service provider. Moreover, even if the user is fine with the privacy policy presented, there is no way for the user to technically verify whether the service provider actually adheres to the policy that has been stated during contract agreement.

Thus, two main principles of the PeRA are:

1. The user as the data subject has the power to specify policies which are then immutably coupled to any information about the user.
2. The PeRA then mandates policy compliance whenever information about a user is accessed. This enforcement can be technically verified.

To achieve these principles, a chain of trust needs to be established that is rooted in every system where data is stored, communicated, or processed and can be tracked back to individual user policies (see Figure 2.1).

At the user level, tight coupling of policies with any data item that users send out to other entities needs to be ensured. These policies describe the exact conditions, under which access to personal data is allowed. After data is stored in the backend, data access is granted to the core system that mediates between applications and stored data. Thus,
applications may never directly access data but always have to interact with the core PeRA components. These components in turn mandate that privacy policies set by the data subjects are adhered to. All access that would violate privacy policies is denied. Thus, policy compliance can be assured at all times. We call this concept of mandatory policy enforcement during data access Mandatory Privacy Control (MPC).

At a low level, we install a trust anchor by employing trusted computing components. In each entity, i.e., cars, intermediate nodes, and backend servers, hardware security modules are installed that permanently ensure that the generic runtime architecture framework provided by PRECiosa is in a correct and verified state. At any time, data in transit or stored in the database is encrypted against keys that are managed by the trusted components.

The use of trusted computing allows us to protect information about a data subject even against misuse by system operators in the backend. Without trusted computing as a trust anchor, information could be well-protected by encryption during transit, but, once it is in the storage repository, it would be possible for system administrators to circumvent the MPC and use data for purposes not allowed by policies. Moreover, trusted computing gives us a higher protection against attackers in the back end without administrator capabilities. Even if an attacker manages to gain physical access to servers, the data storage is still encrypted and the keys managed by the trusted components.

Thus, data received can only be decrypted by these trusted components. If the system is in an untrusted state, decryption will be denied. Similarly, data that is stored in the database can only be decrypted by the trusted components, and decryption is refused if the system is in an untrusted state. Therefore, the trusted computing components serve as a means for vehicles to be able to trust entities that they communicate with. We call this concept MPC Integrity Protection (MIP).

Both concepts together establish what we call the Policy Enforcement Perimeter (PEP). When data is handled inside the PEP, a user can rest assured that custom privacy policies are always coupled with data. Moreover, the MPC components ensure that all access to the stored data is policy-compliant. Because the stored data is protected by the MPC components, no direct access to data that would bypass MPC and possibly violate policies is possible. Thus, policy compliance is mandatory and verified by the system.
We will now give an overview of the main PeRA components. Section 2.2 will then take an orthogonal viewpoint, exemplifying the workflow of the PeRA following the information flow in a representative use case.

2.1.2. PeRA Components Summary

Figure 2.2 shows an overview of the PeRA system. We will now give a brief overview of the mandatory core components. Section 2.2 will outline details as necessary for specific use cases. For full description of all subcomponents, please refer to Chapters 4–7.

Components for Mandatory Privacy Control

The two major components of the Mandatory Privacy Control are the Secure Data/Metadata Repository and the Privacy Control Monitor. The purpose of the former is to provide encrypted storage for both data and policies. This is required to ensure that only authorized access to data is allowed, given that the whole core architecture is in a verified and secure state. Furthermore, the Data/Metadata Repository provides mechanisms to store...
policies in an efficient way, i.e., using references instead of redundant storage. Also, the repository provides a tamper-proof way to store audit data that cannot be modified once written.

Whenever applications want to access data stored in the repository, this access is mediated by the Privacy Control Monitor (PCM). The PCM is composed of several subcomponents and ensures that all access to data is policy compliant. Note, that the PCM not only mediates access to already gathered data in the repository, but also serves as a gateway to any data provided by in-vehicle sensors to attach policies and, thereby, ensure policy compliance upon data collection. The functionality of the PCM is mainly divided in query analyzing and policy management.

Required system functionality beyond that offered by the core PeRA needs to be divided between two types of applications: Uncontrolled applications and controlled applications. Uncontrolled applications have the advantage that they can be deployed with minor modifications to existing code, because they run outside the PEP. They access data only through the query API provided by the generic core. Because all data access of external applications is mediated by the PeRA components, these external applications do not need to be verified manually. All data they can access is necessarily policy compliant due to the checks done by the PeRA. This allows for easier deployment and upgrading of applications. However, access to unaggregated personal information will usually be limited for uncontrolled applications. This is due to the fact that data returned to uncontrolled applications leaves the PEP and further compliance to privacy policies is on a best effort level.

If applications need broader access to personal information, they need to be implemented as a Controlled Application. These Controlled Applications will then run inside a Controlled Application Environment that oversees data flow to and from these applications. While access to personal information can be granted to controlled applications on a broader level (given that corresponding policies exist), communication with other components is tightly controlled by the application environment. The choice to use Uncontrolled Applications, Controlled Applications, or a combination of both can only be made on a per-application basis.

Components for MPC Integrity Protection

In the previous section, we presented components for mandatory enforcement of privacy policies. Now, we need to establish a trust anchor that ensures that the MPC components are in a verified state, have not been altered maliciously, and no components other than the trusted MPC can directly access data, thereby bypassing the policy enforcement mechanisms. We call this mechanism the MPC Integrity Protection (MIP). The main component for such trust establishment is the Trust Manager. At deployment time, the state of the core framework is verified by an external trusted party. If the verification is positive and the PeRA is untampered, the verified state is stored in a tamper-proof hardware device. Further, an asymmetric key pair is generated that is signed by the verification authority, i.e., a government organization or other trusted party, and stored in a tamper-proof device.
as well. It must be ensured by hardware that the keys stored inside the Trust Manager cannot be accessed by outside components. When the system state and all components are verified, the keys can be accessed. In case of any security problems, the keys are deleted and the Trust Manager shuts down the PeRA.

When a user wants to submit personal information to a remote entity, Confidential Communication can be used to transfer data in a safe way. First, the signed public key of the remote entity is obtained and the signature of the verification authority is verified. These mechanisms are transparently provided by the Trust Manager. Then, the Exporter is configured by the PCM to use confidential communication for the transmission of data. The Exporter then obtains the remote public key from the Trust Manager and uses the Confidential Communication API to encrypt the outgoing data with said public key. Upon reception, the remote entity then asks its Trust Manager to decrypt the received data. Only if the local PeRA is in a verified state, will the Trust Manager internally decrypt the data and pass it to the core components. After processing by the Importer, the data is again encrypted with a key managed by the Trust Manager and stored in the Secure Data/Metadata Repository.

The hardware-backed Trust Manager will ensure correct operation of the PeRA, thereby protecting the Policy Enforcement Perimeter, which is operating across entity borders. Inside the PEP, policies are technically guaranteed to be immutably bound to data and adhered to by all applications processing the data.

2.2. Runtime Use Case Examples

The previous Section summarized the architecture and presented an overview of its components. Now, we will discuss selected runtime use case examples and how they could be implemented within the architecture. Where necessary for exemplification, we also include subcomponents in the description. See Figure 2.3 for a detailed architecture view. The goal of this use case driven approach is to gain a better understanding of the interplay and interaction of components in representative scenarios. Later chapters will then provide a component-centric description of the architecture.

The selection of the use cases follows the application chosen in deliverable D8 [3] and the discussions in deliverable D1 [4]. Four representative use cases have been selected that are also subject in other research projects.

1. Collision Warning. A basic cooperative awareness scenario using V2V communication. Warnings are generated using a local dynamic map (LDM). The LDM is populated by periodic beacon broadcasts of neighboring vehicles. The aim of this use case is to show how privacy can be maintained in typical V2V cooperative applications, like the vehicle-based applications of the SAFESPOT project [5].

2. Floating Car Data. Representative for a class of applications where a vehicle submits information to a centralized server via roadside units or cellular communication. For example, traffic information submitted to a traffic management center (TMC). The
TMC can use the gathered information to evaluate the current traffic situation and provide up-to-date road status information, on a website or by other means. This is a basic application employing one-way V2I communication, similar to the COoperative MOnitoring (COMO) application of the CVIS project [6].

3. Online Navigation. A vehicle requests a route from its current position to a destination from an online navigation service. The traffic situation as determined in use case 2 is considered for route calculation at the server. This entails bidirectional communication between vehicle and server in a request/response style. Requests and reply messages may both contain privacy sensitive information.

4. Hotel Booking. A vehicle requests booking of accommodation along a route and submits payment information. This can be regarded as an extension of use case 3. Multiple services are involved, and the use case requires use of the driver’s identity as well as payment credentials.

First, we give a general description of each use case to outline what information is exchanged between which entities. In particular, we focus on the exchange of personal information. We define exemplary privacy policies for messages and information items.
The use case is then mapped to the architecture and the application’s information flow through the architecture is discussed in detail, as well as how the user’s privacy policies are enforced by the PeRA.

2.2.1. Collision Warning

The collision warning application involves several steps:

1. Every vehicle $V$ periodically gathers information on its current status, such as position $pos_V$, speed $vel_V$, heading $h_V$, brake status $b_V$, and more. The data is considered to be primary, that is, no interpreted or aggregated information is computed.

2. A beacon packet containing these data elements and the vehicles identifier $id_V$ is generated. The packet is then sent via broadcast and every wireless unit in communication range receives the packet.

3. Receivers $R$ store the information in a local storage, sometimes also called local dynamic map. Thus, every vehicle obtains continuous information about the status of every other nearby vehicle. Together with older data, a vehicle is aware of the status trace of other vehicles. This can be used to interpolate the trajectory of vehicles.

4. The current situation is continuously analyzed based on the information in the local dynamic map. In case that a critical situation evolves, the collision warning informs the driver about the potential hazard.

Privacy implications

Because vehicles transmit beacon messages via broadcast, contained information can be eavesdropped and processed in any imaginable way. Note that the decision to broadcast unencrypted information is taken by the application: When a vehicle sends a beacon message, any other vehicle in transmission range should receive it. Because of the high dynamics of vehicular networks, it is not possible to establish closed groups of vehicles, so that beacon messages could be encrypted.

The data contained in messages is sensitive in two ways. First, messages contain vehicle identifiers, which are required to map consecutively received status data to individual vehicles. If no identifiers were included, it would be a hassle to match status data to their respective senders. Second, message data also contains intrinsically personal information such as current position. If the vehicle identifier allows deduction of the vehicle owner, authorities could use this for immediate prosecution of speeding.

Another aspect affecting privacy is that vehicles store the data received in beacon messages in the local dynamic map for a while, in order to be able to predict collision danger.
Privacy policies

The danger of tracking vehicles mainly results from the identifier contained in beacon messages. If the identifier is permanent, location profiles are feasible by collecting beacon messages at various locations and matching identifiers in the collected data. Therefore, a policy should require that identifiers in public communication must be changed from time to time.

A second aspect that might be covered by a privacy policy regards storing data at other vehicles. The user could define how long his status samples may be stored at other vehicles. However, enforcement of such a policy is not possible since the data is exposed to the wireless channel and any receiver may record the data.

Information flow

In the collision warning use case, only vehicles and road side units are involved. The logic in road side units is similar to the one implemented in every vehicle. The only difference is that RSUs issue warnings to other vehicles explicitly. In the following, we show the information flow through the PRECIOSA architecture. Because messages are public, they are transmitted outside the architecture, which means that policies cannot be enforced any more. We can distinguish three types of involved processes: message compilation at the sender, data insertion processing after arrival at the receiver, and continuous analysis of stored data at the receiver to detect potential hazardous situations.

Sender

1. The application to generate beacon messages runs as a controlled application in the vehicle’s local controlled application environment (CAE). It queries the PCM for the current position $pos_V$, heading $h_V$, speed $vel_V$, and more status data (like brake status $b_V$) of the vehicle. The application also requests the identity of the vehicle $id_V$. The query is accompanied by the application’s role and purpose, as well as an identifier to authenticate the application.

2. The Trust Manager grants access to the data repository and sensors, if and only if the system is in a trusted state. This process is transparent for the other components unless a component is in an untrusted state, which would result in the trust manger denying access to sensor streams and the secure (encrypted) repository.

3. The PCM evaluates the query request. $pos_V$, $h_V$, $vel_V$, etc. depend on the output of respective sensors. The PCM retrieves the current sensor values from the Importer, which provides a direct interface to sensor output for the PCM, and reads $id_V$ from the secure data repository.

4. Next, the PCM asks the privacy policy manager (PPM) for the policies for each of the retrieved values. The PCM evaluates the policies for the role and purpose of the beaconing application. We assume that the policy allows accurate sensor data
for this application. Thus, the data elements are left unchanged. The policy for $id_V$ should state that it should be anonymized for usage in the beaconing application. Thus, not the actual $id_V$ is returned, but the current pseudonym. All data items together with their policies are returned to the application environment. The application environment component then passes the results to the beaconing application, and caches the policies.

The detailed process how a controlled application retrieves sensor data is depicted in Figure A.4.

5. With the requested data, the beaconing application can assemble a beacon message. When complete, the beaconing application issues a send request to the PCM. The application environment attaches the cached policies before passing the request and policies to the PCM.

6. Using the policies, the role and purpose of the source application, and the current context, the PCM determines further processing of the data. In case of the beaconing application, the receiver of the data cannot be determined, as messages are sent via broadcast. Thus, no explicit receiver is set. Instead, the PCM marks the data to be sent via public communication. After processing related policies the PCM passes the message to the Exporter, along with configuration values.
7. Based on the PCM-provided configuration, the exporter dispatches the message to the public communication component and configures the component accordingly.

8. The public communication API passes the message to the communication stack, which sends out the message.

**Receiver - New data arrival** The following describes how data enters the PRECIOUSA architecture after reception by the public communication API (depicted in Figure 2.5).

1. The message arrives at the receiver over the public communication API. The public communication API validates that the message originates from a valid vehicle by verifying the attached signature.

2. The public communication API passes the message data to the importer component, which determines how to handle the data. In case of beacon messages, the Importer passes the data directly to the PCM for further processing by the controlled beaconing application.

3. Before passing it to the application environment, the PCM selects appropriate (pre-stored) collision warning policies for the data. Corresponding policies for the applications have to be preloaded.
4. Both message data and policies are passed to the application environment. Before passing the data to the controlled application, the application environment caches the policies.

5. The controlled beaconing application processes the data and places a storage request to the PCM. The PCM decides if this is acceptable based on the data policies, the application, its role, and its purpose.

6. In case of a positive decision, data is inserted into the database. For monitoring purposes, the privacy trail is updated.

Receiver - Data analysis As the core of the collision warning application, status data received from nearby vehicles must be continuously evaluated to find out potential dangers. For that, processing of accurate data from the neighbors is required. At the same time, the short-term traces should not be sent out by vehicles – although malicious nodes could gather the same data if they continuously monitor the (public) communication. The procedure as shown in Figure 2.6 involves the following steps:

1. The beaconing analysis application places a query to the PCM. The query includes retrieval of vehicle status data as stored earlier.
2. The application environment attaches appropriate role and purpose for the application.

3. The PCM retrieves policies for the request. Policies may include e.g. that only data within the last thirty seconds may be retrieved.

4. If the request complies to the policies, the PCM queries the database and returns the result to the application.

5. The application processes the data. In case that a problem is found, the application places a send request to the query API, which is intended to inform the external HMI (human machine interface) application. The PCM ensures that only the required information (e.g. anonymized data) is passed to the HMI.

6. The Exporter passes the notification to the appropriate HMI component.

In summary, it is clear that PeRA can only contribute to the protection of privacy in collision warning. In particular, sensor data can only be requested as defined in policies. However, it is also an open question if the fully functional PeRA can be applied to an application with time-critical performance requirements, such as Collision Warning.

### 2.2.2. Floating Car Data

In the Floating Car Data (FCD) use case, vehicles submit road information to the server of a traffic management center (TMC). The use case consists of three steps:

1. A vehicle $V$ sends information about its current traffic situation to server $TMC$ as a FCD record. FCD records contain a TMC-specific vehicle identifier $id_V$, the vehicle’s current position $pos_V$, a time stamp $t_V$, the current speed $vel_V$, and the heading of the vehicle $h_V$. Further information (e.g., road surface temperature) and derived events (e.g., icy road, traffic jam) could be included, but are not explicitly considered here.

2. Server $TMC$ receives the information from $V$ and stores it locally.

3. A traffic visualization application, hereafter called TraVis, utilizes the information on server $TMC$ to provide a map of the current traffic situation to end users (e.g., as a Website).

The communication between vehicle and server is unidirectional. The vehicle submits information, but the server does not actively query or contact any vehicles. Vehicles may use vehicle-to-infrastructure (V2I) communication via roadside units (RSU) or cellular communication.
Privacy implications

As discussed in deliverable D1 [4], the privacy implications of FCD can be summarized as follows. An attacker could eavesdrop on vehicles submitting FCD records if communication is not confidential or the relevant records could be extracted from an unsecured database at the server side. These threats facilitate privacy attacks like high-resolution tracking of individual vehicles based on $id_V$ and consecutive position samples. Driving patterns could be inferred and fused with data from other sources once enough data has been collected to identify a vehicle’s driver.

Hence, not only the vehicle identifier is sensitive, but also position samples and vehicle information, because they may indirectly convey personal information.

Privacy policies

A user’s privacy policy for this use case should address these issues. First, the policy could mandate that the vehicle identifier $id_V$ should never be exposed to applications making use of the data collected by TMC, because the purpose of $id_V$ is to detect FCD records of the same vehicle when entering them into the database to keep the vehicle count consistent with the real world situation.\(^1\)

Further, the policy could mandate that the submitted FCD records can only be used for the purpose *traffic visualization* and that an FCD record of vehicle $V$ may only be included if it is averaged with at least 20 other vehicles. If that requirement cannot be fulfilled, an alternative policy could mandate that the information can still be used if the accurate position $pos_V$ is generalized to a road segment with a minimum length of 5 km.

Additionally, the user could specify retention requirements. Because the purpose of the information is the visualization of current traffic, the policy could mandate that after 5 hours $(t_V + 5h)$ $id_V$ has to be deleted, and that $pos_V$ needs to be generalized to a road segment of 5 km. Another option could be that the record has to be deleted entirely after 5 hours - however, longer traffic history would be lost then.

Information flow

The information flow of three processes is of interest for this use case. First, the generation and transmission of FCD records by the vehicle. Second, receiving and storing of FCD records on the server-side. Third, the TraVis application querying road information.

Vehicle. We start with the generation of an FCD record in a vehicle by the *FCDgen* component. Figure 2.7 illustrates the steps:

\(^1\)Without $id_V$, TMC would not be able to decide if two submitted FCD records belong to the same or different vehicles, thus rendering traffic density analysis difficult.
1. FCDgen runs as a controlled application in the vehicle's local controlled application environment. FCDgen queries the PCM for the current position $pos_V$, heading $h_V$, and speed $vel_V$ of the vehicle, the current time $t_V$, and the identifier $id_V$ assigned to this vehicle by TMC. This query is accompanied by the application's role and purpose, as well as an identifier to authenticate the application.

2. The Trust Manager grants access to the data repository and sensors, if and only if the system is a trusted state. This process is transparent for the other components unless a component is in an untrusted state, which would result in the trust manager denying access to sensor streams and the secure (encrypted) repository.

3. The PCM evaluates the query request. $pos_V$, $h_V$, $vel_V$, and $t_V$ depend on output of respective sensors, while $id_V$ is stored in the secure data repository. The PCM retrieves the current sensor values from the Importer, which provides a direct interface to sensor output for the PCM, and reads $id_V$ from the secure data repository. See Figure A.4 for details on the retrieval sensor data and Figure A.2 for the process to access the DBMS.
4. Next, the PCM asks the privacy policy manager (PPM) for the policies for each of the retrieved values. The PCM evaluates the policies for the role and purpose of application FCDgen. We assume that the policy permits access to accurate sensor data for this application. The policy for \( \text{id}_V \) could state that usage is only possible for FCDgen. The data items together with their policies are returned to the application environment. Here, the application environment passes the result on to FCDgen, and caches the policies.

5. Now, the application FCDgen can assemble the FCD record with the received sensor data. Afterwards, FCDgen poses a send request to the PCM. The application environment attaches the cached policies before passing request and policies to the PCM.

6. The PCM first determines the receiver of the message, based on FCDgen’s application identifier and other context information. In our example, \( TMC \) is determined as the receiver. The PCM further evaluates the accompanying policies to determine if the data items are allowed to leave the local node and selects the policy that applies to the receiver and needs to be enforced by the PeRA on the receiver side. Then, the PCM passes the message to the Exporter, along with configuration values. That is, the message has to be encrypted for \( TMC \) and pseudonyms should be used on the network and MAC layer.

7. Based on the PCM-provided configuration, the exporter dispatches the message to the confidential communication component, and configures it accordingly.

8. The confidential communication API handles encryption with \( TMC \)’s public key, configures the underlying communication subsystem, and sends out the message. \( TMC \)’s public key is taken from a certificate issued by a trusted privacy agency that verified the proper operation of the PeRA instance on \( TMC \)’s server. The Trust Manager manages all locally stored key material.

Server. In this step, the server receives the vehicle’s FCD record as an encrypted message. Figure 2.8 illustrates the steps:

1. The Importer component receives an encrypted message authenticated by a valid vehicle pseudonym. The importer requests decryption of the message from the Trust Manager, which safeguards the corresponding private key.

2. Before decrypting the message, the Trust Manager verifies the integrity and trustworthiness of components that would have access to the received data. In this case, these would be the Importer, the local PCM, and the secure data/metadata repository. If the platform is in a trustworthy state, the Trust Manager decrypts the message and returns the plaintext to the Importer. If the platform is in an untrusted state, e.g., components have been tampered with, the Trust Manager would refuse decryption.
3. Once decrypted, the message and policies are passed to the application environment, where a controlled application handles storing the data. The use of a controlled application has the advantage that the Importer does not need to know message formats of different applications. The controlled application poses a storage request to the PCM, in which the controlled application environment injects the policies. The request is again accompanied with a role and purpose.

4. The PCM evaluates the policies, and decides if data can be stored. In case of a positive decision, data is inserted into the database, the privacy policy manager ensures the coupling of data and policies, and the privacy trail is updated accordingly.

5. Finally, the PCM triggers the retention manager to add new maintenance events to its schedule.

**Application.** The TraVis application runs as an unrestricted application outside of the PeRA on TMC’s server. It uses the submitted FCD records to generate a visualization of the real-time traffic situation. Figure 2.9 illustrates the following steps:
1. TraVis uses the Query API to pose a query for the average speed and the number of cars on a certain road segment (e.g., Highway 7, road kilometers 100-110). The application needs to specify a purpose, and may specify a minimum requirement for the data quality. The latter is important when certain data items are excluded from the result due to strict policies.

2. The Importer receives the query and passes it on to the Query and Policy Analyzer.

3. The Trust Manager unlocks the secure data repository, granting the PCM access to it, if and only if the system is in a trusted state.

4. The Query and Policy Analyzer executes the query. The policies of affected data items are retrieved from the Privacy Policy Manager. Policies are evaluated simultaneously with query execution, and data transformations are applied to achieve policy compliance. For example, an exact position sample may be generalized to a road segment. If the policy of a given data item cannot be matched it is excluded from the result set.
5. The Query IDS raises an alarm if a potential intrusion is detected. For example, if an application tries to spider through the database, an entry is added to the privacy trail.

6. Once the Query and Policy Analyzer has received the result, set it determines the outgoing policy to be attached to the final result set. The result set and policy are handed to the Exporter.

7. The Exporter dispatches the result set and policy to the unrestricted application TraVis. At this point, the data leaves the policy enforcement perimeter and policies can no longer be enforced. Nevertheless, a privacy policy is attached, which can be used by applications to comply on a voluntary basis.

8. The TraVis application processes the result set and visualizes the current traffic situation for the road segment specified in the original query.

The entire process of data retrieval by an uncontrolled application is modeled in detail in Figure A.5.

2.2.3. Online Navigation

In contrast to offline navigation, where all required data such as maps and traffic statistics is stored onboard a vehicle, online navigation involves a connection to a backend infrastructure. This is typically used to include some up-to-date map changes or the most recent traffic situation. In some cases, online navigation systems also comprise the delivery of data such as current position and speed, which we consider in the separate FCD use case. The fundamental difference of online navigation compared to FCD is that personal information e.g. on the destination has to be transmitted, and the service provider answers route requests with personalized information on the route.

Thus, online navigation basically requires two steps to which the use case can be broken down.

1. **Route request.** The driver of a vehicle $V$ initiates a route calculation to a certain destination $D$. We assume that map data is available onboard the vehicle, but the information on the most recent traffic situation is not available. Therefore, $V$ requests the TMC for the best route to $D$. The request includes $id_V$, the current position $pos_V$, and the destination $D$.

2. **Route response.** When the TMC receives the request, it calculates a route from $pos_V$ to $D$, incorporating the current traffic situation. The route information is then returned to $V$. 
Privacy implications

Privacy risks arise from the transmission and evaluation of the current position of the vehicle and its destination. First, this data could be eavesdropped, so that foreigners may learn about the destinations of vehicles. Second, if stored at the TMC, extraction of destinations of single vehicles from the database may reveal a lot about the life of their drivers.

Privacy policies

In order to address these problems, several privacy policies are advisable. One prerequisite is that route requests should only be sent to an authorized TMC, and the communication must be encrypted to ensure confidentiality on the way. Another policy may forbid storing of any route requests. If this is not feasible for example because of accounting, a policy may allow storage for a certain time, and allow queries only for billing applications.

Information flow

Vehicle: Route request creation. The information flow of creation and sending a route request is very similar to the one in FCD as described in Section 2.2.2. The flow only differs in the attached data and policies.

TMC: Request reception and response. Figure 2.10 shows the flow of information in the TMC server to process route requests. The numbers have the following meanings:

1. The Importer component receives an encrypted message authenticated by a valid vehicle pseudonym. The importer requests decryption of the message from the Trust Manager, which safeguards the corresponding private key.

2. Before decrypting the message, the Trust Manager verifies the integrity and trustworthiness of components that would have access to the received data. In this case, these would be the Importer, the local PCM, and the secure data/metadata repository. If the platform is in a trustworthy state, the Trust Manager decrypts the message and returns the plaintext to the Importer. If the platform is in an untrusted state, e.g., components have been tampered with, the Trust Manager would refuse decryption.

3. Once decrypted, the message and policies are passed to the application environment, where a controlled application handles the data. The application is capable to calculate routes on the base of map data and current traffic data. The current traffic information is queried from the database. Therefore, the application places a query to the PCM, accompanied by role and purpose, in order to retrieve information on those road segments on the planned route. In case that a certain stretch of road is congested, the application may adapt the planned route. But in general, only road
Figure 2.10.: Online navigation: Information flow in the TMC system (arrival of requests, processing of routes, and sending of responses.

segments are queried; current position and destination never leave the controlled application environment. $id_V$ may be stored by the application for billing purposes.

4. The PCM evaluates the policies, and decides if queries can be fulfilled for the application. If yes, it returns the results to the application (see Figure A.2 for details on the whole process up to now).

5. When finished, the controlled application assembles a package with route information and places a send query to the PCM.

6. Together with the controlled application environment, the PCM ensures that the application can return the (route) data only to the original sender vehicle. The PCM determines appropriate policies for the route data and passes it to the Exporter.

7. The Exporter dispatches the data set to the confidential communication API and configures it to encrypt the result with the public key of the requesting vehicle.

8. The confidential communication API encrypts the data accordingly and sends it back to the vehicle.
2.2.4. Hotel Booking

The hotel booking use case once more increases the complexity. One aspect is that more personal information is revealed during the communication, because the user defines special room preferences for the hotel search. When booking a certain hotel, we also have to deal with critical payment information, for example a credit card number. Moreover, the use case is more complex because it comprises different request/response cycles: first, the user requests hotels along his route, and gets a list of available ones. Second, the user selects a certain hotel he would like to stay with, and confirms the booking with his credit card.

In more detail, hotel booking has the following steps:

1. A user triggers a hotel booking request, which is sent to the service provider.
2. The service provider evaluates the query, that is, collects available hotels along the specified route or nearby a certain location, using the specified preferences. The list of hotels is returned to the vehicle.
3. The vehicle receives the list and lets the user decide the further proceeding. Usually, the user would select a hotel and book it.
4. The service provider handles the booking, using the given payment information. The confirmation or failure of the booking is reported to the user.

Privacy implications

The communication required for hotel booking reveals a number of personal information. Not only that an eavesdropper may learn about the position of the user, but he also gets further personal details, such as preferences for the room. When the booking is finished, also payment information is transferred, such as a credit card number, and the attacker knows where the user likely stays over the night. All of this data must be protected so that only the service provider has access to it. But also the service provider itself may endanger privacy, if requests and confirmed hotels are stored for a longer time. On the one hand, storing some information is necessary to handle the booking, but on the other hand, the information should not be abused.

Privacy policies

Similarly to use case 3, policies for the application on the vehicle should mandate that only a trusted service provider should be used, and communication must be kept confidential. On the service provider’s side, policies have to ensure that the sent data can only be retrieved from the database for accounting purposes, and hotel preferences should not be stored at all. They may be used only for filtering of suitable hotels. Likewise, the travelled route for which hotels are queried, should not be stored.
Information flow

Vehicle: Creation of hotel request. The creation of the hotel request is straightforward. When invoked by the driver, the vehicle gathers necessary information such as current position, planned route, user preferences, etc. This data is available either via sensor interface, queried from an HMI interface, or made available by other on-board systems like a navigation system. The PCM attaches the corresponding policies and sends the request via confidential communication to the service provider.

Server: Assembly of hotel list. The hotel request is received by the service provider over the confidential communication API component, which decrypts the data stream and passes it to the Importer. The decryption operations and subsequent integrity checks are conducted by the Trust Manager. The Importer passes the data to the controlled application environment, which invokes the appropriate application and caches the attached policies. The controlled application for hotel booking evaluates the received route and issues a query to the database (or an external application) to get a list of hotels nearby. Though the query contains user preferences for the hotel, the user identity is not revealed. This is enforced by the PCM, which evaluates the policies before the application query can continue. When the hotel list is complete, the controlled hotel booking application sends the result back to the requesting vehicle. This is done by a send query to the PCM, which grants the operation if the destination is correct. Together with policies, the Exporter passes the data to the confidential communication API, which returns the data encrypted to the vehicle.

Vehicle: Hotel selection and booking. The vehicle receives the hotel list over the confidential communication API, and likewise decrypts the data using the Trust Manager. The Importer then passes the data to the controlled application, which passes it to an uncontrolled HMI application for selection by the user. When the user has selected a hotel, the application places a query to the PCM, requesting the credit card number from the database. The PCM grants access only if a policy allows the application to retrieve the credit card number. After that, the application issues a send query to the PCM, which contains the data to confirm the hotel booking, including payment information. Again, policies are checked, and the information is handed over to the Exporter and the Confidential communication API.

Server: Booking and confirmation. The principal process is similar to the ones before. In this case, however, the application must be allowed to store the booking in the database. Special preferences may also be stored by the application. Moreover, if the policy mandates, the PCM schedules the deletion of the data after the maximum allowed storage period.

More details on various involved processes are modeled as UML sequence diagrams and can be found in Appendix A. For example, A.4 shows the process of sensor access by a controlled application.
3. Policies and Queries

The execution of queries in Privacy Aware Applications (PAA) needs additional considerations when compared with those in other (traditional) applications. Beside the processing of data and the evaluation of queries on the data we must respect specific privacy requirements. Therefore, we take into account (and evaluate) information on the application data regarding privacy and information about the context of executing a query. Such information is always represented as metadata and coupled both with the application data and with the request (query).

Usually, (query) execution environments transform a query on data into an internal (logical) representation. Executing a query mostly results in the execution of a sequence of operations on internal representation of the data (see Figure 3.1). At the end of a query execution the result is sent back to the requester.

PeRA data (information) is always coupled with information that describes the requirements of the stakeholders regarding privacy. Those requirements may restrict the execution of operations on data within PeRA. Thus, while executing queries we do not only perform operations on data; at the same time we also evaluate the corresponding privacy requirements.
information and compute the effect of the request on the privacy requirements of the data – see Figure 3.2. The effect of the query execution regarding privacy has to be considered for coupling the result of the request with its (new/updated) privacy information. PeRA supports the extensible PRECIOUSA Privacy aware Query Language (PPQL) to express requests on the system. Requests formulated with PPQL contain statements that request to execute an operation on data. The set of operations is extensible. For all supported operations PeRA calculates the privacy effect. Thereby, an operation can be complex. For instance we support an operation that executes a (lightweight) SQL query. Several other query languages exist which are optimized to support different application domains.

Figure 3.2.: Extended query execution environments according privacy requirements.

For PeRA we use the PRECIOUSA Privacy Policy Language (P3L) for describing the privacy requirements of the stakeholders (For a detailed description of the P3L language we refer Deliverable D6). A Stakeholder is either (a) the data subject - the person the information is about, (b) the data controller who is responsible for the privacy protection (thus, liable in case of privacy violation), (c) the data processor who implements/executes the application and acts in behalf of the data controller, (d) privacy agencies who advocate the legal framework for protecting privacy and set up the privacy requirements from the legal perspective, and others.

Application providers (data controllers in the legal sense) have to implement an appropriate protection of privacy. Therefore, they have to support a set/range of privacy policies. In PeRA, the data processor must provide a description of the supported range of privacy for his provided application. This description has to be attached to the request. While executing the request the PeRA-System evaluates if the privacy requirements of the requested data are compatible with the privacy specification requested by the requesting data processor.
Policies define permissions for performing operations on data. These permissions are only valid for specific situations – the context of the permissions. To describe such situations the permissions are combined with restrictions (constraints). These permissions are valid if a given situation fulfills all of the constraints. Thus, constraints define the context when a policy allows the system to perform specific operations on data. While executing a request the system checks if the context of the request matches the context of the policy.

3.1. Query Language PPQL

PeRA supports the extensible PRECIOUSA Privacy aware Query Language (PPQL) to express requests on the system. PPQL queries describe requests to execute operations on data (see Figure 3.3). The set of PPQL operations is extensible. For all supported operations in PeRA the Privacy Control Monitor is able to calculate the privacy effect of their execution. Therefore, an operation taking both aspects into account might be quite complex. For instance the operation process-query takes as one argument a query and executes it on a data source. Therefore, the Privacy Control Monitor is able compute the privacy effect from executing a query of the language. Several existing query languages are optimized to support different application domains. In general, it is impossible to calculate the privacy effects for any query in a given languages. Therefore, it becomes necessary to restrict the expressiveness of some query language.

For the prototypical implementation of the PeRA architecture we support a lightweight SQL language to query data of a relational database.

![Figure 3.3.: Executing PPQL requests.](image)

3.1.1. Operation Requirements

We define some requirements that must be realized by operations (including the execution of query languages) which are supported by the PeRA system. New operations have to
realize these requirements in order to extend the set of PeRA operations. The following list describes the operation requirements.

- All data sources that contain requested data are identified by URIs. URIs are used as identifiers (e.g. for data items, policies, applications, communication entities) within the whole PeRA system;

- The execution of operations (including embedded queries) on data can be evaluated to calculate the (proposed) effect on data regarding its privacy information. Thus, for every operation the query execution system can decide whether the execution of the operation conforms to the declared privacy requirements (policies).
  
  - Condition 1: Stakeholders have declared their permission to execute the operation for the context of the request.
  
  - Condition 2: The requesting data processor provides a range of privacy requirements which fits with the privacy requirements of the result.

- For every operation the query execution system can calculate/determine the privacy effect (new privacy information) which results from executing the operation; e.g. the degree of anonymity as obfuscation value.

- For every performed operation the query execution system can decide if the privacy effect of the performed operation is conform to the privacy requirements of the stakeholders and the requesting data processor.

- For every operation the query execution system can calculate or determine the change of privacy criteria (e.g. new privacy policy) after performing the operation on data.

3.1.2. Supported operations

PPQL defines a basic set of operations which realize the previously defined requirements. These basic operations enable some core functionality for a privacy aware processing of data (especially of personal information). PeRA implements this set of operations. The elements of this basic set are the operations

1. **access**: retrieve requested data from different sources (e.g. a sensor value, a value from a controlled application, or a result of a query executed on a data store).

2. **store**: stores the data handed over in a data store.

3. **send**: takes/accesses data from a data source and sends it to the destination; the data is not being changed.

4. **change**: selects some data from a data source and alters the data according to the specified expression (e.g. replaces an old value with a new one or applies a data transformation function on it). That is, the data will be changed at the data source. Different from the process-data operation no temporary values are produced.
5. **delete**: permanently removes data from a data store which has been selected by a filter predicate.

6. **process-data**: accesses data from a data source and applies a (data transformation) function on it. Such functions are provided as data transformation functions by the PCM or as (un)certified functions by Controlled Applications.

7. **retrieve**: accesses some data and returns it back to the caller. The caller might be a *Controlled Application or Uncontrolled Application* on the same or on a remote system.

8. **process-query**: accesses data from a data store by executing a query (e.g. in SQL-Light) that is specified as a parameter of this operation.

A request may have to perform a sequence of operations. The input of a subsequent operation may require the output of a preceding operation. Thus, we need a mechanism to hold and access temporary results. We implement temporary results by the keyword AS. The schema is “OP on Data AS TemporaryResult”. In our example statement shown in Figure 3.3 we access data from different sources, combine the data, store the result in a data store, and send the result to a traffic control center. While performing these requests, the temporary variables are coupled with the appropriate privacy criteria. All temporary variables will be deleted after the request has been performed. Intermediate results and temporary data can be stored and accessed using a datastore. Thus, PPQL supports to realize a session based computation of data that contains several requests. Controlled Applications can be utilized if the access time to the data store is not acceptable or if the data processor does not have the permission to store the temporary data.

### 3.1.3. SQL-Light

In our PeRA system we support the query language SQL-Light for the operator process-query. SQL-Light is a SQL based language to query data of a relational database. In general you cannot calculate all possible privacy effects for statements of the SQL language. There exist several variants to query for some specific information. To protect all variants is too complex for many cases. For instance, to get a value of a sensitive attribute describing a person you can retrieve this information using indirect queries; thereby you exclude all values that are not true for the person you have in mind. At the end you excluded all possible values except one without querying directly for the information. To avoid this complexity we restrict the expressiveness of SQL. Our criteria to restrict the SQL query language is that the Privacy Control Monitor can calculate the privacy effect from executing any query of the restricted language. For SQL-Light as for SQL we transform a query into a sequence of operations. In the PeRA prototype we evaluate every single SQL-Light operation regarding their privacy effects independently. Thus, in order to calculate the privacy effects of one operation we take as input the data the operation is performed on and the privacy information of the data. If we have previously performed other operations on input data the privacy effect of this operations is part of the privacy information of the data.
One future research challenge is to investigate if the execution of a sequence of operations has side effects that cannot be calculated by just considering a single operation. We still take into account the privacy effects of previously executed operations, but we do not consider the operations itself nor future operations. We restrict the set of SQL operations and support the following operations: a) select, b) project, c) join, d) group by, e) aggregate functions.

| Example 1 | SELECT AVG(speed) FROM Traffic, User WHERE User.name = 'Alice' and User.uid = Traffic.driver-uid |
| Example 2 | SELECT SUM(driven-distance) FROM TollDeclarations WHERE user = 'Bob' and BETWEEN (date, '2010-03-01', current-date) and street-type = 'highway' |
| Example 3 | SELECT vehicle-type, location FROM Traffic |

Table 3.1.: SQL-Light examples

We do not support language elements such as negation, union, intersection or nested queries. The schema of the human readable notation of SQL-Light is “SELECT expression FROM table-reference WHERE search-condition GROUP BY attribute-list”. The expression of the select clause consists of a list of attributes (e.g. location, name, speed) or aggregation functions (AVG, SUM, COUNT). The table-reference of the from clause consists of a list of tables (e.g. User, Vehicle). The search-condition of the where clause consists of simple constraints (e.g. contain a relational operator). Thus, inner joins are possible.

3.1.4. PeRA components realizing supported operations

In PeRA different components perform the supported operations. As shown in Figure 3.4 the components which execute the operations are Data Transformation component, Controlled Application, Importer, Exporter, and SQL-Light interpreter. Table 3.2 shows which component realizes which operation.

<table>
<thead>
<tr>
<th>Component</th>
<th>Realized operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Transformation:</td>
<td>extends set of basic operations with a set of trusted data transformation operations supported by PeRA</td>
</tr>
<tr>
<td>Controlled Application:</td>
<td>can provide (un)certified operations on data that are implemented by third parties; e.g. application provider supports specific functionalities</td>
</tr>
<tr>
<td>Importer:</td>
<td>provides access to the periphery of the system; e.g. sensors</td>
</tr>
<tr>
<td>Exporter:</td>
<td>provides communication to remote systems within PeRAs perimeter and to uncontrolled applications</td>
</tr>
<tr>
<td>SQL-Light interpreter</td>
<td>provides a query execution engine for SQL-Light</td>
</tr>
</tbody>
</table>

Table 3.2.: Components and their implemented operations

For every supported PPQL operation the PeRA system determines the privacy effect resulting from the operation. This privacy effect can be computed or is provided as metadata by certificates of the operation (e.g. operations that are implemented by certified controlled
applications). The Query and Policy Analyzer takes the privacy effect as input to decide if the performance of a requested operation is conform to the specified privacy criteria. We distinguish two types of operations that can be performed. One type of operations take data as input and create new data as the result (e.g. data transformation). After performing such a type of operation the Query and Policy Analyzer must determine the (new) privacy criteria (e.g. described by a P3L policy) of the result (e.g. the result – a calculated invoice – has to be deleted from the data store after two month). The second type of operations leaves data unchanged (e.g. store, send). After the decision whether the request is allowed, but before the operation is being performed the privacy and policy manager must determine the new privacy criteria (e.g. P3L policy). In general, data is coupled with the new privacy information before the operation is performed. One example for the second type of operation is the scenario of sending the current location information. The policy of the location information states that it can be stored in a data store and it can be send to a traffic control center. Furthermore, the policy states that the data that might be sent to the traffic control center can not be stored. Thus, before performing the send operation the permission to store the location information has to be removed from the coupled privacy information. One simple solution to implement such privacy criteria is to use the post policy mechanism of the P3L which is discussed in the next section in more detail. Figure 3.5 highlights the components that determine the privacy effects of supported PPQL operations.
We provide two formats to serialize PPQL statements. One format is defined in human readable form as shown above with the schemata and the examples. The other format is defined in XML format and mainly used for to exchange privacy information and to compute privacy effects in the PeRA system. The XML format of PPQL is defined by a XML schemata.

### 3.2. Privacy Metadata

In order to follow the privacy requirements as defined by a stakeholder, the PeRA system evaluates information describing the requested data regarding privacy aspects and information about the context of the query execution. Such information is always coupled as metadata with the data we manage in PeRA and the request.

In PeRA we consider different types of metadata while processing a request (see Figure 3.2). First, we consider is the metadata of the requested data. Before a request on data can be performed the privacy control monitor checks if the stakeholders of the requested data permit the system to perform the request. Such privacy criteria that restricts the performance of requests on data is attached as metadata (e.g. P3L policies). The second type of metadata that is attached on the requested data is information which describes

![Figure 3.5.: PeRA components providing privacy effect of PPQL operations.](image)
data quality aspects. For some applications it is necessary to know the preciseness of
data or they have requirements on the preciseness of results.

For example, the quality of a service might depend on the precision of the processed data. Some high quality services need very precise data. In contrast, most Privacy Enhancing Technologies (PETs) obfuscate data to protect users against privacy violations. Therefore, it is important to find the necessary balance between the degree of privacy protection and the provided quality of service. As a result we have to consider the two opposed dimensions of a) privacy requirements and b) data quality requirements:

- **Dimension 1**: privacy criteria - privacy measure e.g. obfuscation to reduce the information content (rising the entropy)
- **Dimension 2**: data quality requirement - one aspect of data quality e.g. preciseness

The second type of metadata the PeRA system has to consider is information describing the context of the request. The privacy criteria defined by the stakeholders permit the performance of requests for a set of contexts that are defined by constraints. The PeRA system provides the metadata that describes the context. The third type of metadata is attached with the request. This metadata describes the supported privacy dimension (e.g. supported range of P3L policies) of the requesting data processor and the requirements of the requester regarding data quality. i.e. preciseness. Thus, it is possible to check if the supported privacy dimension of a data processor conforms to the specified privacy criteria of the requested data. Such analysis can be done at design time and at runtime.

Regarding the feasibility of our approach for processing the different types of metadata we need to use a standard language(s) to specify metadata. Stakeholders have to agree on common languages; for instance, to express their privacy criteria. In Deliverable D6 [2] we discuss different languages for expressing privacy policies (e.g. P3P, APPEL, XACML, PRIME) and application requirements (e.g. CARML). We define the requirements of languages for expressing the PeRA metadata, show the drawbacks of the existing languages, and develop our own languages to express the metadata. Thereby, we show how existing approaches/languages can be integrated into our language.

To establish such a generic construct of different dialects we need an identification mechanism that is very flexible and commonly used. We have to identify different kind of resources as privacy policy, data sources, data schemata, etc.. Therefore, PeRA uses the standard web technology Uniform Resource Identifiers (URI). With URIs we can ease the handling of the metadata. Instead of communicating all metadata every time we use references (e.g. to (globally) known metadata representing standard privacy policies). The resolution of resources can be done transparently.

There exist two formats for the metadata in PeRA. As for PPQL one format is defined in a human readable form and the other format is defined for exchange and the processing in the PeRA system using XML schemata.
3.2.1. Policy Language P3L

In PeRA data (information) is always coupled with information that describes the requirements of the stakeholders regarding privacy (see Figure 3.6). Those requirements are described by statements of policies which restrict the execution of operations on data.

For PeRA we use the PRECIOSA Privacy Policy Language (P3L) to describe the privacy requirements of the stakeholders. Stakeholders can be (a) the data subject - the person the information is about, (b) the data controller who is responsible for the privacy protection (thus, liable in case of privacy violation), (c) the data processor who implements/executes the application and acts in behalf of the data controller, (d) privacy agencies who advocate the legal framework for protecting privacy and set up the privacy requirements from legal perspective, and others.

![Diagram: Assign policies with data and schemata or ontologies.]

Figure 3.6.: Assign policies with data and schemata or ontologies.

Considering the principles and approaches introduced in Deliverable D7 we define the requirements of a language which can be used to express privacy requirements in Deliverable D6 adequately. We show the drawbacks of the existing languages and develop our own language P3L.

The primary principles of using P3L are 1) to link user related data with the corresponding privacy policies which are specified by the stakeholders, 2) the linked privacy policies define all permissions to perform operations on the (user related) data, 3) to determine the (new) privacy policy which describes the corresponding privacy requirements of the data after performing a permitted operation on the requested data. Again the (new) privacy policy which is responsible after successfully performing a requested operation, will be linked with the original data or the result.

One important feature which we want to implement by using policies is that the stakeholders can clearly define their privacy criteria. Thereby, it is not necessary to describe the privacy criteria for every single application. In P3L the scope of validity of a policy is...
defined by the context of the policy. Thus, P3L has the flexibility to specify the privacy criteria for single applications or even a group of applications which can be categorized by their application type.

With P3L we can express a range of privacy requirements – from no restrictions (without required privacy protection) up to maximum privacy protection, where we enforce the highest privacy protection possible (services may not be flexible anymore or even not executable). We provide an additional argument of using policies and other metadata: In PeRA we are able to express explicitly the privacy criteria of stakeholders at the same time supporting privacy dimension of applications. Thus, it is possible to check if the supported privacy dimension of an application is conform to the specified privacy criteria of the requested data. Such analysis can be done at design time and at runtime. This enables the evaluation, the comparison, and the verification of applications, services, and systems regarding privacy protection. In cases of privacy violation we also may determine the responsibilities (in a legal sense).

In Deliverable D6 we developed P3L based on models and ontologies which describe different domains (see a simplified example in Figure 3.7) as ICT, ITS, and legal aspects as well as privacy aspects of this domains. This basic set of conceptual descriptions can be extended by ontologies that describe application specific domains. In Deliverable D6 we show how to use privacy models, privacy ontology, and engineering methods 1) to define a conceptualization for ITS applications to support the provision of privacy, 2) to

Figure 3.7.: Exemplary ontology for a cITS scenario

In Deliverable D6 we developed P3L based on models and ontologies which describe different domains (see a simplified example in Figure 3.7) as ICT, ITS, and legal aspects as well as privacy aspects of this domains. This basic set of conceptual descriptions can be extended by ontologies that describe application specific domains. In Deliverable D6 we show how to use privacy models, privacy ontology, and engineering methods 1) to define a conceptualization for cITS applications to support the provision of privacy, 2) to
describe privacy protection requirements, 3) to describe privacy specific properties, and
4) how to analyze and to enhance applications regarding privacy protection aspects.

The following list summarizes the major benefits from using a conceptualization in form of
models, metamodels, and ontologies

- Defining terms and their meaning for privacy criteria, privacy mechanisms, privacy measurement and auditing regarding privacy;
- Translating user privacy to technical privacy;
- Bridging the gap between legal requirements and technical solutions;
- Describing the behavior of active computing elements (involved in the operations of an application);
- Detecting privacy leakages;
- Describing the requirements for privacy protection;
- Providing the bases for defining appropriate privacy policies for applications;
- Mapping of requested/required privacy protection and provided privacy protection;
- Providing the bases for the specification and ((semi) automatic) verification regarding the compliance to guidelines and principles of
  - components,
  - the composition of components (system privacy),
  - systems,
  - application;
- Describing possibilities to derive (personal) information and defining rules for preventing thus inferences;
- Addressing intelligent privacy attacks that utilize semantic technologies, background knowledge, or data mining.

In the following we illustrate the most important concepts and elements of P3L with an example scenario. In the scenario we have the two data processing applications Collision Detection and Traffic Analysis of the Traffic Control Center. The Tables 3.3 and 3.4 show a snapshot of the data of the example scenario. For the scenario we consider a (restricted) ontology that contains the definition of the concepts of the scenario and the statements that the concept GPS isA LocationDescription and that the concept Licence-Plate isA Identifier. For analyzing purposes we define the following conceptual mappings for the schemata of the applications: Attribute GPS-Data of TCC-Schema corresponds to the concept GPS, Attribute Vehicle-Type of TCC-Schema corresponds to the concept VehicleType, Attribute Vehicle-Class of CD-Schema corresponds to the concept Vehicle-Type.

P3L statements describe permissions to perform operations on data. To express such permissions we have to identify the data that might be processed. Therefore, P3L provides different variants to identify data items. Thus, the coupling of policies with data is expressed on different levels of granularity. We consider the following levels:
1) the level
Table 3.3.: TCC Data

<table>
<thead>
<tr>
<th>ID</th>
<th>Time-Stamp</th>
<th>Position</th>
<th>Vehicle-Type</th>
<th>LicencePlate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12:01:37</td>
<td>90.20.3.5</td>
<td>truck</td>
<td>D B-A-34</td>
</tr>
<tr>
<td>2</td>
<td>12:02:38</td>
<td>90.20.3.7</td>
<td>motorcycle</td>
<td>D P-G-27</td>
</tr>
<tr>
<td>3</td>
<td>12:02:41</td>
<td>90.20.3.6</td>
<td>motorcar</td>
<td>D B-S-12</td>
</tr>
</tbody>
</table>

Table 3.4.: CD Data

<table>
<thead>
<tr>
<th>ID</th>
<th>Time-Stamp</th>
<th>Distance-Vector</th>
<th>Vehicle-Speed</th>
<th>Vehicle-Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12:01:35.55</td>
<td>3,2;4,3</td>
<td>80</td>
<td>truck</td>
</tr>
<tr>
<td>2</td>
<td>12:01:35.65</td>
<td>-2,-2;-1,-1</td>
<td>105</td>
<td>motorcycle</td>
</tr>
<tr>
<td>3</td>
<td>12:01:35.89</td>
<td>0,0;1,1</td>
<td>100</td>
<td>motorcar</td>
</tr>
</tbody>
</table>

of instances, 2) the level of schema elements, and 3) the level of concepts. At the instance level we select single values (e.g. the Vehicle-Type value truck of the TCC data in our example scenario). Such values can also be structured data as tuples in a relational database. At schema level we use expressions on schema elements to select a set of values (e.g. all values of the attribute Vehicle-Type in the schema of the TCC data). At concept level we select a set of values that share the same conceptualization (i.e. the same semantics). Therefore, the application data has to be mapped with expressions of the ontologies we use for identifying the data (e.g. the concept GPS of the ontology in the scenario selects all data of the attribute Vehicle-Type in the schema of the TCC and all values of the attribute Vehicle-Class in the schema of the CD).

In our example scenario the vehicle of type motorcar is requesting other vehicles for information as their distance to the motorcar, their direction, and their speed. The purpose of requesting and processing such information is to detect impending collisions. To enable the execution of such requests from the motorcar the other vehicles must have define the permission for sending and processing the requested information. Therefore, we exemplary define a P3L policy (see Example 3.8).

According to Example 3.8 every P3L statement consists a definition part and a statement part. The definition part contains elements that describe the metadata of the policy itself. The most important elements of the definition part are the description of Namespaces, Schemata (e.g. relational schema with tables, attributes, references to corresponding concepts), the scope of coupling data with the policy (couple with all data of schema elements, couple with data enclosed with a message, couple with all data that corresponds to the concept of the selected data), and policy description (e.g. policy ID, creator of policy, text description).

The statement part of a P3L policy contains the permissions to execute operations. These permissions are bound on a context and conditions that have to be fulfilled before and after processing the requested operations. The most important are the elements
NS=http://www.preciosa-project.org/P3L/PeRA/predef.p3l;
NS:CD=http://example.org/applications/ITS/Safety/CollisionDetection/;
NS:RV=http://www.preciosa-project.org/P3L/PeRA/predef-RV.p3l;
CD:Schema=(ID, Time-Stamp, Distance-Vector, Vehicle-Speed, Vehicle-Class)
PolicyID=http://www.preciosa-project.org/P3L/PeRA/WebInterface/User/MeyerJacobCD01.p3l
PolicyCreator=Jacob Meyer
PolicyLabel=P3L User Policy of Jacob Meyer for the application Collision Detection
PolicyDescription= This policy permits to execute a request for information to the purpose of collision detection. The Enquirer has to be another vehicle within a range of 30 meters. The policy permits the receiver of the request to send the requested information (Time-Stamp, Distance-Vector, Vehicle-Speed, Vehicle-Class) which has to be attached with a standard policy of the application type collision detection.'
Context(RV:EnquirerPurpose = CollisionDetection, RV:EnquirerType = vehicle)
{
Permit Access GPS-value FROM sensor.GPS AS vehicle-location
Permit Access speed-value FROM sensor.Speed AS vehicle-speed
Permit Access current-time FROM system AS vehicle-time
Permit Access vehicle-type FROM system AS vehicle-type
Permit CD:CalculateDistance[distance-vector](vehicle-location) AS distance
Permit send (vehicle-time, distance, vehicle-speed, vehicle-type) With CD:AbsoluteValueOfVector(distance, 'meter') < 30
PostPolicy = CD:process-CD-Data.p3l
}

Figure 3.8.: P3L user policy for collision detection

- Context (scope of validity) - is defined by conditions using the system/runtime variables; they describe the RuntimePreConditions which are defined independently of the data to be processed;
- Permit statement - defines the permission to process operations on data;
- Action statement - defines an action/operation that has to be triggered (e.g. delete data after 3 month);
- PreCondition of a permit statement - depends on the requested data (e.g. maximum speed value which is allowed to be processed);
- PostCondition of a permit statement - depends on the result of the operation(s) (e.g. the result of the operation has a minimum measure of anonymity);
- Temporary Variables - to establish/enable a chain of operations
- Post Policy - Reference to new privacy criteria for the result of performed operation and for original data after performing the operation;

We emphasize that it is important requirement to keep the set of elements as operations of P3L extensible. For PeRA we predefined a set of elements which enables a basic functionality. Thus, we support the implementation of the Use Cases introduced in Chapter 2. In Table 3.5 we summarize the content of the Namespace NS:PeRA=http://www.preciosa-project.org/P3L/PeRA/predef.p3l#.
As described in Deliverable D6 we provide two formats to serialize P3L statements. The first format is a human readable syntax similar to a script language or SQL (see previous examples) and a XML representation with a defined XML-Schema to exchange P3L statements within PeRA.
Additional metadata defines the preciseness, the privacy measure, and the context (provided by system variables). The structure of that metadata is similar to the structure of P3L statements. For further details we refer to Deliverable 6 [2].
4. Mandatory Privacy Control

One of the principles in PeRA is to ensure the tight coupling of privacy metadata (e.g., P3L policies) with any data item that is processed by the system. The metadata describes the conditions, under which operations on personal data are permitted. After storing data at the backend, direct access to data is only granted to the core system that mediates between requesters (applications) and stored data. Thus, applications never directly accesses data; instead an application always has to interact with the core PeRA components. These components in turn ensure that applications always obey the privacy criteria (P3L policies) set by the stakeholders (as data subjects). All data accesses that violate privacy criteria are denied. Therefore, the compliance to the specified privacy criteria in form of P3L policies is always assured. We call this concept of mandatory policy enforcement during data access Mandatory Privacy Control (MPC).

The two major components of the Mandatory Privacy Control are the Privacy Control Monitor and the Secure Data/Metadata Repository – the latter is described in Chapter 6. The purpose of the latter is to provide encrypted storage for both data and policies. Storing both data and policies always together ensures that any access of and computation on data always follows the specified policies – the underlying principle of our approach. To guarantee such privacy protecting access we assume that all components of the core architecture are in a verified and secure state. Furthermore, the Data/Metadata Repository provides mechanisms to store policies in an efficient way, i.e., using references instead of redundant storage. Additionally, the repository provides a tamper-proof way to store audit data that cannot be modified once written.

Whenever applications want to access data stored in the repository, this access is mediated by the Privacy Control Monitor (PCM). The PCM is composed of several sub-components and ensures that all access to data is policy compliant. We emphasize that the PCM does not only mediate access to already gathered data in the repository; this component also serves as a gateway to any data generated by in-vehicle sensors by attaching policies and, thereby, ensuring policy compliance upon data collection. The functionality of the PCM is mainly divided in query analyzing and policy management.

Inside the PCM, queries from applications are handled by the Query and Policy Analyzer. First, the Privacy Policy Manager is consulted to collect all policies of data items that would be affected by the query operation. If the policies in question are incoherent, it is also the Privacy Policy Manager’s duty to merge conflicting policies. If the query to be executed is policy-compliant, it is executed and the result is returned to the requesting application. If necessary, data transformations are applied to the data before it is handed out. Possible transformations are data fusion, obfuscation, or other aggregate query operations. These
Transformations must be implemented by the PeRA core to guarantee that no personal data is transferred to other entities without the existence of a permitting policy.

In addition to the query authorization by user policies, the **Query Intrusion Detection System (Query IDS)** component detects – if possible – suspicious query activities over time and blocks specific queries despite their policy-compliance. Finally, the **Privacy Trail Manager** logs all relevant operations performed on data to allow for later audits and heuristic checks by the Query IDS.

The PCM must offer additional system functionality that goes beyond the functionality previously described. Two types of applications might interact with the PCM by sending requests or receiving results: Uncontrolled applications and controlled applications. **Uncontrolled applications** are deployed with only minor modifications to existing code; however they are executed **outside** of the Privacy Enforment Perimeter (PEP – please see Chapter 5). They access data only through the query API provided by the generic core. Because all data access of external applications is mediated by the PeRA components, these external applications do not have to be verified manually. **Uncontrolled applications** are deployed and upgraded in a PeRA independent manner. However, access to unaggregated personal information is usually limited for Uncontrolled Applications since it is impossible to guarantee that those applications comply with the privacy preferences attached to the data. Outside the PEP their compliance to privacy preferences (policies) is on a best effort level.

If an application requires fine-grain access to personal information, it must be implemented as a **Controlled Application**. A Controlled Application executes inside a **Controlled Application Environment** which oversees the flow of data to and from the application. While access to personal information may be granted to controlled applications on a broader level (given that corresponding policies allow such access), communication with other components is tightly controlled by the application environment. The choice to use Uncontrolled Applications, Controlled Applications, or a combination of both highly depends on the application requirements and may only be decided on a per-application basis.

### 4.1. Privacy Control Monitor

<table>
<thead>
<tr>
<th>Name:</th>
<th>Privacy Control Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Execution of PPQL requests</td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Controlled Application Environment, Privacy Policy Manager, Secure Data and Metadata Repository, Importer, Exporter, Public and Confidential Communication APIs</td>
</tr>
<tr>
<td>Nature:</td>
<td>Mandatory component</td>
</tr>
</tbody>
</table>
4.1.1. Purpose

The **Privacy Control Monitor** (PCM) is the main component to orchestrate the complex execution of requests expressed in the **PRECIOSA Privacy aware Query Language** (PPQL requests) as described in Section 3.1. The main purpose of the PCM is to guarantee that only those operations on data are executed that are compliant with the privacy policies attached to the data. After performing such an operation the result data is attached with (new) appropriate privacy policies derived from the existing preferences (policies) again described in P3L policy language.

4.1.2. Interfaces and Services

Since it is the heart of the PeRa the PCM interacts with the following components:

- The Importer to receive requests and data;
- The Exporter to return data to Controlled Applications or Uncontrolled Applications;
- The Controlled Application Environment (CAE) to receive requests from or to return results to a Controlled Application;
- The Privacy Policy Manager (PPM) to receive requested privacy policies or to store policies;
- The Secure Data and Metadata Repository to store or to retrieve application data or metadata;
- The Trust Manager for administrative tasks regarding the database and the current state of the PCM.

In the following we describe the different interactions in more detail.

**Interaction with Controlled Application Environment**

The PCM is able to invoke the application in two different modi: one modus allows to invoke an application without any return value, the other modus expects a return value/object from the invoked application. The different interactions between the PCM and the CAE are summarized in Figure 4.1. More details of the interface are specified in Subsection 4.3.
Interaction with the Importer and the Exporter

To interact with components outside PeRA the PCM uses the Importer and Exporter components. The Importer might receive a request from another application to start (invoke) an application within the Controlled Application Environment or to execute a request (query or an operator of the Data Transformation Component). The PCM might call the Importer requesting (a list of) sensor values as input on behalf of an application (either uncontrolled or controlled).

The PCM is able to call the Exporter as a gateway to deliver data either to other PeRA instances or to Uncontrolled Applications. Additional metadata helps the Exporter to determine where to send the data (destination), how to configure the communication subsystem (in terms of encryption or pseudonym use), and how to attach privacy metadata.

The different interactions between the PCM and the Importer/Exporter are summarized in Figure 4.2.

Interaction with Privacy Policy Manager (PPM)

The PCM calls the Privacy Policy Manager (PPM) in various ways:
1. **getPolicy**: Requesting a specific policy based on a data item URI, a filter on data items; and a specific context;

2. **registerPolicy**: with a newly created policy the PCM may register this policy (at a specific site); the PPM handles all the details and returns a URI as the (new) reference to this policy;

3. **updateDataPolicy**: replace an old policy with a new one;

4. **updateDataPolicy**: replace existing policy-URIs with a new URI for all data items that qualify using the the provided filter.

The different interactions between the PCM and the Privacy Policy Manager (PPM) are summarized in Figure 4.3.

---

**Interaction with Trust Manager (TM)**

The PCM calls the Trust Manager (TM) for several administrative tasks:

1. **isSystemInTrustedState**: checks if the PCM is still executing correctly or if it has been modified/tampered with;

2. **unlockDataBase**: unlocks the database for accesses by the PCM and the PPM.

The different interactions between the PCM and the Trust Manager (PM) are summarized in Figure 4.4. More details can be found in Chapter 5.

---

**4.1.3. Description**

The PCM is the central component which decides whether a request is allowed or rejected. As one of the components of the PCM the **Query and Policy Analyzer (QPA)** implements the query execution system for PPQL requests. The QPA enforces the tight coupling of data and its privacy preferences (in form of metadata) at all times as follows.
Before performing a request the QPA checks the attached privacy policies to guarantee that permission exists – declared by the stakeholders using the PPL – to execute the operation in the context of the request. After performing a request the QPA ensures that the resulting data is combined with the requested privacy preferences (policies) as stated by the incoming policies possibly in combination with other sources.

If a request is rejected the caller receives a result that provides the reason for not executing the request properly. Such a behavior is necessary and important for safety applications such as collision detection to guarantee their correct execution.

PeRA supports some basic PPQL operations which are implemented by the different components within the PCM. The Data Transformation component provides basic data transformation operations such as anonymization of data or the computation of its anonymity level. The SQL-Light interpreter implements a lightweight SQL query language (a restricted subset of SQL) to request data from a relational data store. For all of the operations and data queries the PCM can calculate/provide the privacy effect from performing the operations on the requested data. The effects on privacy preferences by the request are calculated and are used either to allow or to deny a PPQL request, and if allowed to create the (new) privacy preferences (in form of policies).

In the following we describe additional subcomponents of the PCM in more details.

Query and Policy Analyzer

The main task of the Query and Policy Analyzer (QPA) is to handle requests (including privacy aware execution of queries). Specifically, the QPA ensures that data and attached
privacy policies are always processed together synchronously thus ensuring that the appropriate privacy preferences (policies) are attached to the result of the request.

To reach these goals the QPA analyses operations and policies. For this reason it must retrieve and access policies via the Privacy Policy Manager (PPM) before performing an operation on the data. These operations are executed by either

- the Data Transformation Component
- the SQL-Light Execution Engine

We notice that data might be coupled with sets of privacy policies to reflect the privacy preferences of different stakeholders (data subject, data controller/processor, privacy agency, system provider, ...). Therefore processing privacy preferences results in performing set-oriented operations on privacy policies possibly intertwined with generating new policies for the resulting data.

Before discussing the major steps of executing a query we emphasize that Privacy by Design requires that all stakeholders in the data (data subject, data processor, data protection laws) must have defined their privacy preferences beforehand. Only then the QPA is able to perform the following steps at runtime when executing a query:

1. Analyze context information of the query to later evaluate the privacy policies;
2. Fetch those policies specified for the data (using the URIs) that are schema dependent and independent of individual data items;
3. Determine if the specified privacy policies allow the execution of the specified query or request (input and output – on the schema level) in the specified context;
4. If allowed, process the query/request. We notice that some privacy policies are attached to individual data items which must be checked during query processing. That is, after partially performing a query (select, join) the QPA might have to check if the further processing is still allowed on the (intermediate) result;
5. Check if the final result follows the specified privacy policies. For example, we check if the computed result complies with the level of anonymity specified by the attached set of policies. We distinguish the following outcomes:
   a) If it does, the QPA returns the result to the PCM which in turn returns the result to the requester;
   b) If not, we see three different alternatives to return a result:
      i. Reject the query/request and return an explanation (return code) to the requester. The information on the rejection should be detailed enough for the requester to allow him to possibly respecify the query and/or the privacy/quality measure to then resubmit the query.
      ii. Transform the resulting data to the most restrictive policy/policies and return the transformed result. This alternative might lead to reduced data quality (utility);
iii. Select those data items for the result that match the quality criteria as specified by the requester. With this alternative the requester receives a result of a specified quality level, but the result might be incomplete.

For further references we call the first alternative the retry alternative (option) and the last two alternatives the adaptable alternatives (options). The adaptable option still needs some more discussion and explanation with respect to how to handle the changes on the result with respect to the requester:

1. the changes to intermediate/final results are transparent for the requester. That is, the requester is informed about the transformations without knowing the details;
2. The changes to the intermediate/final result are nontransparent for the requester. That is the requester is not informed about the transformations.

The latter alternative becomes a two-edge sword. Not informing the requester might be an additional privacy measure since it avoids a try-an-error approach by the requester to find out more about the data. At the same time, the changes made by the QPA might impact the requester dramatically. For example, leaving out positions of vehicles in safety applications like collision detection might lead to wrong conclusions and are therefore not acceptable. We conclude that for such applications the requester must know about changes made by the QPA to the result of a request to operate correctly (safely).

SQL-Light Interpreter

The SQL-Light Interpreter is solely responsible for executing SQL queries at the same time determining the effects of operations on privacy policies.

For the purpose of the prototype we restrict the subset of SQL queries to SPJG (Select, Project, Join, Group, and Aggregate) queries. For this reason we refer to the current component as the SQL-Light Interpreter. This operator view allows SQL-Light Interpreter to transform a submitted SQL query into a sequence of operations. For each of the operations the SQL-Light Interpreter is able to determine the effect on the attached set of policies as shown in Figure 4.5. Therefore, PeRA is able to evaluate if an SQL-Light query conforms to the specified privacy policies. The processing is summarized in Figure 4.5.

The processing within the SQL-Light interpreter consists of three major execution steps:

1. Preprocessing;
2. Hand over query to DBMS (maybe in iteration to get necessary policy without executing complete query);
3. Postprocessing.
Data Transformation Component

The *Data Transformation* component provides a set of operations (supported by PeRA) that can be invoked by either the PCM itself or by (controlled or uncontrolled) applications. For each operation of the Data Transformation component the PCM also calculates the privacy effects based on the attached policies and the additional context information (metadata).

A system provider may extend the Data Transformation component with additional data transformation operations by installing a new version of the PeRA system. Such a change causes to check and to verify the changed Data Transformation component (and therefore the PCM) to guarantee the integrity of the extended component.

Data transformation operations may fall into one of the two following categories:

1. Category 1:
   - Input to Data Transformation: data
   - Output of Data Transformation: privacy measure – for example, the anonymity level of the input data

2. Category 2:
• Input to Data Transformation: data, privacy measure
• Output of Data Transformation: obfuscated data that has the requested value of privacy measure.

The prototype implementation will implement the following data transformation operations:
• Aggregation
• Anonymization
• Pseudonymization

4.1.4. Platform Considerations

The PCM needs to run both on the server side, as well as the vehicle side. If implemented as a lightweight OSGi bundle, it is possible to run the same implementation of the PCM both on the vehicle side and in the backend. If one uses a heavyweight application server in the backend, for instance, a Java-Enterprise-based application server, a different solution could be necessary for the vehicle side. For the server side implementation complex computing can be necessary for analyzing big sets of collected data. The analysis could be done for application purposes or for purposes of privacy protection.

4.2. Privacy Policy Manager

<table>
<thead>
<tr>
<th>Name</th>
<th>Privacy Policy Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Manages the coupling of data and privacy metadata</td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Privacy Control Monitor, Secure Data and Metadata Repository, Importer, Exporter</td>
</tr>
<tr>
<td>Nature:</td>
<td>Mandatory component</td>
</tr>
</tbody>
</table>

4.2.1. Purpose

The Privacy Policy Manager (PPM) supports the tasks of finding privacy preferences in form of policies (metadata) to data. It also returns references to privacy policies or materializes those policies whenever necessary. Furthermore, the PPM also instantiates policy templates or profiles for data without such metadata. For example, when sensor data enters PeRA we must find and attach the required policies to these data items for appropriate privacy aware processing of the data at later time.
4.2.2. Interfaces and Services

The Privacy Policy Manager (PPM) offers the following interfaces as used by the PCM:

1. **getPolicy**: Requesting a specific policy based on a data item URI, a filter on data items; and a specific context;

2. **registerPolicy**: with a newly created policy the PCM may register this policy (at a specific site); the PPM handles all the details and returns a URI as the (new) reference to this policy;

3. **updateDataPolicy**: replace an old policy with a new one;

4. **updateDataPolicy**: replace existing policy-URIs with a new URI for all data items that qualify using the the provided filter.

Figure 4.6 shows the interaction between the PPM and the PCM.

![Diagram](image)

Figure 4.6.: Interfaces of the Privacy Policy Manager

4.2.3. Description

The PPM supports

- The coupling of data and privacy policies (metadata) by returning references to policies (in form of URIs) or by materializing policies (providing specific URIs);

- The retrieval and instantiation of policy templates or profiles for data (that enter PeRA without policies) that enter the system without it. For example, when sensor data is retrieved via the Importer, the PCM must attach appropriate policies first before any operation on that data is performed;

- The management of versioning on policies (which might support auditing processes to verify that data has always been protected by the correct policies).
Assuming that the Secure Data and Metadata Repository implements the relational model we suggest to attach policies using the primary key of the (application) relations. References to policies and the policies themselves (if cached) are stored in separate tables. Since data tuples might be governed by more than one policy the relationship between data items of the application tables and the policy tables must be 1:N.

4.2.4. Discussion

Using P3L it is unnecessary to describe the privacy criteria for every application separately. Instead, policies might describe the privacy needs for a group of applications (application type). Therefore, we create descriptions of privacy criteria (at least minimal privacy requirement) that are application independent.

4.3. Controlled Application Environment

<table>
<thead>
<tr>
<th>Name:</th>
<th>Controlled Applications Container</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Acts as a container for custom applications that need to access possibly sensitive information. Controls information flow to and from those applications to ensure policy compliance.</td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Privacy Control Monitor, Query + Policy Analyzer, Secure Data and Metadata Repository, Importer, Exporter, Public and Confidential Communication APIs</td>
</tr>
<tr>
<td>Interfaces/Services:</td>
<td>Application Management, Application Descriptors, Invocation Interface</td>
</tr>
<tr>
<td>Nature:</td>
<td>Mandatory component</td>
</tr>
</tbody>
</table>

4.3.1. Purpose

The controlled application environment (CAE) offers an execution environment for any custom, application-specific code that needs to be run inside the policy enforcement perimeter. Such controlled applications can only interact with the PCM and no other components. Due to application meta-data that is attached to application bundles (e.g., application provider, purpose, communication partners), as well as information provided by the CAE (e.g., history of accessed information items), the PCM can then decide whether data access is policy compliant and allow or deny access accordingly.

In many cases, it will be sufficient to interact with the personal information stored inside the PEP via uncontrolled applications. Uncontrolled here means that applications will run outside the PEP; thus, only minimal modifications to existing application code are necessary. Mainly, existing applications to be used in this way only need to use the PeRA query API instead of the query API of a database system directly. However, policies governing data access of uncontrolled applications will most likely mandate that personal information can only be accessed in aggregated form.
Deliverable 10

For many use cases, this type of access is sufficient. Yet, other types of applications, or components of applications, will require a broader access to personal information inside the PEP. For these cases, PeRA provides applications running inside a controlled application environment that will mediate all data flow to and from these applications. This allows application providers to run custom code without the need for manual code review while still providing technical policy enforcement. The general idea for such controlled applications is that read access to data is easily possible while communication with other components and writing data back to the repository is strictly limited.

On a generic level, we can distinguish the following usage scenarios for controlled applications:

- **Message Assembly.** Used to assemble application specific communication messages using sensor values or data already stored in the repository, which are then sent out via the communication API. Typically deployed in vehicles. Message assembly applications take the role of the application that they send data to. Data is read from local sensors and possibly the storage repository. No data is stored in the repository. Data sent out needs to be encrypted so that it can only be accessed by other entities within the PEP. This corresponds to the use case presented in Section 2.2.2.

- **Message Storage.** Used to process application specific communication messages upon reception, possibly filter the contained values, and store the necessary information in the repository. Deployed in back end systems or intermediary nodes that cache data, message storage applications complement the message assembly applications in the vehicles. Message storage applications, just like their counterparts in the vehicles, are authenticated with the application provider's code signature and data access is granted or denied based on that identity. Data is read from the communication message, decomposed into the relevant values, and stored in the repository. No data is sent out to other components apart from the repository.

- **User Surrogate.** Used to perform operations in back end systems on behalf of a single user. Examples include functions with high bandwidth needs and modification of personal information about the requesting user. Contrary to the first two patterns, user surrogates act in the role of a specific user. This allows them access to personal information about that user. Thus, one worker instance of such applications is instantiated per user request. Communication between different application instances is disallowed. The only possible communication channel is back to the requesting user.

- **Custom Data Transformations.** Used when custom data transformations, that is, an extension of the query language by custom information fusion or alteration mechanisms, is needed by an application provider. For instance, such custom data transformations could be business critical and optimized algorithms that outperform aggregate functions provided by a SQL-like query language. As data flow to and from such applications is arbitrary, they need to be manually code reviewed and certified.
These patterns allow a flexible deployment of custom application code that needs to access personal information from inside the policy enforcement perimeter. The strict application environment configurations alleviate the need for a manual code review for most use cases.

### 4.3.2. Dependencies and Prerequisites

The only component that the controlled application environment directly interacts with is the privacy control monitor. The PCM’s query interface is used by the environment, on behalf of a controlled application, to access data stored in the secure repository and to access sensor data. Furthermore, all invocations of the exporter and, ultimately, the communication APIs have to pass the PCM. Thus, the controlled application environment relies on the implementation of the privacy control monitor, the secure data and metadata repository, the importer, exporter, and the public and confidential communication APIs.

In addition, an application environment is needed in which controlled applications can run. This can be an application container, for instance, provided by Enterprise Java Beans (EJB) [7]. This container needs to ensure that all information flow to and from applications is strictly controlled. Controlled applications may not access any persistent storage despite the secure repository via the query interface provided by the PCM. A lightweight alternative to EJB is to use the OSGi framework for application access control and lifecycle management. Some application management functionality, which would otherwise be provided by the application container, then needs to be implemented by the CAE itself.

### 4.3.3. Interfaces and Services

The controlled application environment offers an interface to manage, i.e., install, update, remove, (re-)start, and stop, controlled applications. To be installed in the CAE, application bundles for controlled applications need to be accompanied by descriptors that state the application’s role, purpose, and other meta information required by the PCM to evaluate access to information. To allow incoming connections to controlled applications, the CAE further exposes an invocation interface. Finally, the CAE exposes a pass-through interface to the PCM functionality that can be used by controlled applications. Controlled applications cannot access the PCM API directly, because additional meta-data is needed that can only be provided by the CAE.

### Application Descriptors

Each controlled application is accompanied by an application descriptor that describes the application's nature, role, purpose, and other configuration values. The application descriptor, as well as the application itself, can be signed by a trusted third party, but it does not need to be. The fact whether contained information is signed and verified by a trusted party will be forwarded to the PCM with each query. It then depends on the data...
policies, whether access in the given configuration is granted. The following configuration values can be set in an application descriptor:

**ApplicationID** A unique identifier (e.g., application name with namespace) for the application. This identifier will be used for invocations of the application.

**Version** The version number of the application.

**Role** The role in which the given application should run (e.g., FCD provider).

**Purpose** A description of the application’s purpose (e.g., periodic FCD reports).

**Lifecycle** The lifecycle of the given application. Possible values are stateless and stateful. Stateless applications are destroyed and reinitialized before each invocation so that they cannot accumulate any internal state. Stateful applications can run indefinitely and manage their lifecycle internally. Note that this will usually imply stricter application verification by trusted third parties. Without such verification, an application may not be allowed to run stateless.

**Communication Endpoints** Identifiers for each communication partner of the application. Communication partners inside the PEP are identified by a public key ID and a corresponding server name or unique server ID. Communication to such endpoints is then encrypted using the corresponding cryptographic key.

**Periodicity** Specifies, if needed, a cron-like, periodic invocation pattern for an application. This is useful, for instance, for beaconing applications that want to send out current status information (e.g., speed, time, or temperature) periodically.

**Custom Transformations** If an application implements custom data transformations, e.g., custom aggregation functions on data, the provider needs to specify the post conditions that these operations have. For example, if one implements an application-specific average function, a post condition could be a certain $k$-anonymity that is guaranteed after the operation completed. Post conditions need to be extracted during a manual review process of the application. This information can then be used by the PCM for policy merging purposes.

**Data Tokens** If an application does not need access to data content for application logic, but only wants to send certain data, it should specify these data items as data tokens. See Section 4.3.4 for a more detailed discussion of the data token concept.

**Application Management**

Controlled Applications need to be installed, updated, and possibly removed from the system. Therefore, the CAE offers an application management interface. This interface cannot be called remotely but only by local management tools. Also, active security components of the PCM can use the application management interface to disable or uninstall controlled applications that do not behave in the way they are described.
installControlledApp

inputs:
    String applicationFilename;
    String applicationId;
exceptions:
    FileNotFoundException;
    CertificateInvalid;
    DescriptorInvalid;

Installs the given Application in the application repository with the given URI. The application JAR file must contain the application code itself, an application descriptor in the above-specified format, and a signature by either a trusted third party or the application provider. If the descriptor is well-formed and semantically correct, and the certificate is successfully verified, the application is installed and added to the CAE application repository. If the application is already present in the repository, but the installed version is different from the one to be installed, the given JAR will be installed in parallel.

disableControlledApp

inputs:
    String applicationId;

Disables the application with the given ID. This method should be called by active security components of the PCM when malicious behavior form a controlled application has been detected.

enableControlledApp

inputs:
    String applicationId;

(Re-)enables the application with the given ID.

uninstallControlledApp

inputs:
    String applicationId;

Completely uninstalls the application with the given ID from the system. This includes the application descriptor and all internal references to the application, for instance, entries for periodic invocation.
Invocation Interface

The invocation interface is used for all invocations of controlled applications. The reason for invocation can either be periodic invocation as defined in the application's descriptor, or invocation due to incoming data or requests by uncontrolled applications.

invokeApp

inputs:
  String applicationId;
  Object params;
output:
  Object result;
  MetaInfo info;

Invokes the application with the given ID and hands over the given parameters. The format of the parameters is application specific and should be handled internally by the application's implementation of the invokeApp Method (see Section 4.3.4). Note that, depending on the descriptor of the application in question, subsequent calls to this method can result in calls to different incarnations of a controlled application that cannot communicate with each other. The controlled application may return a result to the original caller. In this case, the CAE also has to return some meta information about the application to the PCM.

Interface for Controlled Applications

Because outgoing queries by controlled applications need to be accompanied by metadata that the CAE must provide, a proxy-interface for the PCM query methods is provided. These methods are called only by controlled applications.

executeRequest

inputs:
  String queryString;
outputs:
  Object result;

Executes the given query, which is formulated as a query string. The result is composed out of a structured data object and a privacy policy.
export

inputs:
   Object data;
   String destination;

Export the given data to the specified destination.

getSensorValue

inputs:
   String sensorId;
outputs:
   Object value;

Requests the value of the specified sensor. Again, a policy is attached to the resulting value.

4.3.4. Description

Controlled applications are used whenever applications need broad access to personal information, but can be restricted in terms of communication to other components. The CAE provides an environment for such applications. To achieve a secure environment for controlled applications, there are four required functionalities:

1. The CAE needs to encapsulate controlled applications in a way that they cannot directly access any interfaces provided by PeRA components. Instead, controlled applications must only be able to call functions of the CAE which in turn forwards the calls to the PCM.

2. As a result from the first functionality, the CAE needs to provide a way for controlled applications to query information from the database or sensors and also a way to send out data. Although this functionality is provided by the PCM, the CAE needs to add certain meta-information that the applications cannot provide themselves. For example, the CAE enriches the query by meta-information stating whether a controlled application has been self-signed or signed by a trusted third party. The PCM then can base policy decisions on this fact. Similarly, other information stated in the application descriptor or collected by the CAE is added to the queries.

3. The CAE must provide lifecycle management for controlled applications. This is especially important to ensure that certain types of controlled applications – namely, the stateless ones – cannot exchange state information between invocations. Moreover, lifecycle management provides the necessary functionality to install, uninstall, start, and stop controlled applications.
4. The CAE must offer the means to invoke controlled applications from the outside. This encompasses the mapping of application ids given with calls to instantiations of the corresponding code.

The first two items of functionality can be provided in part by the chosen application container framework, e.g., OSGi. Also, the management interface has already been discussed in the previous section. Thus, in the following we will concentrate on exemplifying the information flow to and from controlled applications as the two major functionalities provided by the CAE.

**Information Flow to Controlled Applications**

Controlled applications are invoked using a generic invoke method. This method receives two parameters: a unique application id and an application specific data object. First, the CAE resolves the application ID in its internal database of installed applications. If not present, an error is returned. If the application ID is found, the following actions depends on whether the application is stateless or stateful. If the controlled application runs stateful, the data object is handed over to the application directly. If, however, the selected controlled application runs stateless, a method call request for the specified application is generated and added to the request queue. The queue operates in the following fashion. For each entry, the stated controlled application is freshly started, then the method call is executed, and finally the application is stopped again. This ensures, that no state can be exchanged between subsequent calls even of the same controlled application.

In addition to being invoked by incoming calls, controlled applications can be invoked periodically by a local cron-table-like subcomponent. This allows for applications that need to be run without external trigger events to be implemented in a stateless way.

**Information Flow to the PCM**

When a controlled application wants to access information, it uses the execute method provided by the CAE. The general pattern is that the CAE never decides whether access to requested information or requests to send data can be granted. The CAE only adds certain meta information to the outgoing query, which the PCM can use to allow or deny the access based on policies. The following information is added to outgoing queries as meta data:

- All information provided in the application descriptor.
- The information contained in the application certificate together with the information whether the application bundle has been self-signed or is signed by a trusted third party.
- A set of all data items previously requested by the same instance of the application.
This information, especially the set of previously requested data items, ensures that the PCM can safely decide whether to allow the request. Because the meta information is provided by the CAE and not the application itself, it can be guaranteed that the provided data cannot be modified from inside the application. However, the fact whether the application is self-signed or signed by a trusted third party needs to be taken into account. If self-signed, the application can provide arbitrary information inside its application descriptor. The CAE can then only ensure that this information cannot be modified during runtime.

If access to the requested information is granted by the PCM, it first hands the information back to the CAE. Also, all requested data items are added to the information flow log of the current application instance. This information flow log later serves as a means for the PCM to decide whether an application, given the information it has previously accessed, can request further data or is allowed to communicate to other entities. Also, the information flow log is used to determine the policy of outgoing data in case of send requests by controlled applications. The requested information itself is then returned as-is to the controlled application for further processing.

If an application needs to send a certain item of information but does not need it for its internal application logic, said item should be requested as a token only instead of the real information item. This decision needs to be declared in the application descriptor as specified above. Figure 4.7 shows the information flow using tokens. All information items requested as tokens are not returned to the application after the successful completion of queries. Instead, the CAE generates an identifying token and stores the data item internally. This token can then be added to a subsequent send request by the application and will be replaced with the real data by the CAE. This way, the PCM can be sure that the data item in the send request is indeed the value provided by the secure data repository or read from a local sensor. Without tokens, any data item of a send request could contain arbitrary data, which was combined using all information that has been accessed before.
This mechanism allows for applications that would otherwise need to be manually certified by a trusted third party to run as self-signed applications.

4.3.5. Platform Considerations

The controlled application environment needs to run both on the server side, as well as the vehicle side. If implemented as a lightweight extension to an existing OSGi framework, it is possible to run the same implementation of the CAE both on the vehicle side and in the backend. If one uses a heavyweight application server in the backend, for instance, a Java-Enterprise-based application server, a different solution must be chosen for the vehicle side. However, it needs to be ensured that both versions rely on the same format of application certification and application descriptors.

4.3.6. Discussion

The presented approach is a trade-off between complexity of implementation and complexity of application deployment. If easy implementation is the goal, one could mandate the manual code review of all controlled applications. After such a review, a trusted third party can assert all necessary properties of such applications in their descriptor, and the PCM can make policy decisions based on these descriptors. However, this process is lengthy and needs to be repeated even for small updates to existing applications.

Another alternative would be to provide a customized scripting language for controlled applications. As such a scripting language would be evaluated during runtime, one could implement automatic reasoning about the data flow this way. Similarly, arbitrary application code can be allowed to be executed with the restriction that all outgoing information is strictly analyzed for covert data. However, both these possibilities require a complex implementation and chances for hidden security holes and remaining covert channels are high.

Therefore, we chose to implement the CAE using a compromise between manual certification and automatic reasoning about processed data. For simple application patterns, such as message assembly, message storage, and user surrogates, we offer mechanisms that allow for these types of applications to run without a manual code review. If more complex applications are necessary for service providers, they can still chose to have their applications certified by a trusted third party. This fact will be stated in the meta data provided by the CAE upon each query issued to the PCM, which can in turn decide to allow or forbid the access in question. This method enables us to keep the controlled application environment lightweight, yet remaining a high level of flexibility.
5. MPC Integrity Protection

Chapter 4 describes the components for Mandatory Privacy Control (MPC). MPC can only be effective if it can be guaranteed that MPC components work as expected, cannot be tampered with, and cannot be circumvented. In this Chapter, we specify MPC Integrity Protection (MIP) components to achieve this. MIP has several major tasks:

1. Provide users with means to establish trust in components running on remote systems.
2. Ensure confidentiality and integrity of personal information in storage and transit.
3. Ensure that access to stored or transmitted personal information is only granted if MPC components are in a trusted state.
4. Ensure binding of data and metadata (in transit and storage)

When these requirements are fulfilled mandatory enforcement of, and adherence to, privacy policies can be guaranteed, because MPC, MIP, and privacy policies form a protection chain for personal information, as explained in Section 2.1.1. This protection chain creates the Policy Enforcement Perimeter (PEP), which provides system-wide privacy.

MIP mechanisms are provided by two component units. The Trust Manager and its sub-components (Section 5.1) are responsible for integrity measurement of local MPC components, that information can only be accessed if the system is in a trusted state, and management of key material for storage encryption and communication encryption. The specific implementation of the Trust Manager strongly depends on the underlying platform, e.g., whether a Hardware Security Module (HSM) is available that can act as an anchor of trust. For the rest of this text, we assume that such an HSM is available. However, implementations with weaker requirements (and likely weaker assurance) can be imagined.

The Communication Endpoints (Section 5.2) are responsible for general communication with remote entities and applications. Thus, confidential communication inside the PEP is a major task, as well as abstracting from underlying communication stacks and systems. Together, Trust Manager and confidential communication enable trust establishment between remote systems.

In the following, the Trust Manager and the communication endpoints Importer and Exporter are specified in detail.
5.1. Trust Manager

<table>
<thead>
<tr>
<th>Name:</th>
<th>TrustManager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose:</td>
<td>Local trust anchor of PeRA. Ensures that access to data is only granted if PeRA components are in a trusted state.</td>
</tr>
<tr>
<td>Dependencies:</td>
<td>Hardware Security Module</td>
</tr>
<tr>
<td>Interfaces/Services:</td>
<td>System State, Database encryption control, Crypto services</td>
</tr>
<tr>
<td>Nature:</td>
<td>Core component</td>
</tr>
</tbody>
</table>

5.1.1. Purpose

The Trust Manager ensures integrity of local MPC components, and monitors the platform's current state and trustworthiness. Further, the Trust Manager only allows other components access to stored or received data if the MPC components are in a trustworthy state. The Trust Manager also exposes crypto functions to other PeRA components.

5.1.2. Dependencies and Prerequisites

- Hardware Security Module, i.e., TPM chip compliant with TCG TPM 1.2 specification
- Trusted system boot
- Trust Manager requires to be the first component of the PeRA to be loaded, in order to measure integrity of other components on load

5.1.3. Interfaces and Services

The Trust Manager exposes an interface to the PCM that enables it to query if the system is currently in a trusted state. Further the PCM can request the unlocking of specific databases. Additionally, the Trust Manager exposes a service interface for its crypto module to all PeRA components so that they can make use of provided encryption, decryption, signature verification, certificate verification, and key management capabilities. This interface may provide certain services that abstract from the underlying crypto functions.

**isSystemInTrustedState**

```java
inputs:
  void;
outputs:
  boolean inTrustedState;
exceptions:
```
Returns whether or not the system is currently in a trusted platform configuration. Access restricted to PCM component.

**unlockDatabase**

- **inputs:**
  - `String dbName;`
- **outputs:**
  - `uint ReturnCode;`
- **exceptions:**

If system is in a trusted state, the specified database is unlocked. The caller is informed about the outcome of the unlocking attempt. Access restricted to PCM component.

**encryptMessage**

- **inputs:**
  - `byte[] message;`
  - `string publicKeyId;`
- **outputs:**
  - `byte[] encMessage;`
- **exceptions:**
  - `KeyNotFound;`
  - `InvalidKey;`
  - `InternalEncryptionError;`

Encrypts the provided message with the public key corresponding to the specified id.

**decryptMessage**

- **inputs:**
  - `byte[] encMessage;`
  - `string privateKeyId;`
- **outputs:**
  - `byte[] message;`
- **exceptions:**
  - `KeyNotFound;`
  - `InvalidKey;`
  - `InternalDecryptionError;`
  - `UntrustedSystemState;`

Decrypts the provided message with the private key corresponding to the specified id. If the private key is sealed to platform configuration, decryption is only successful if system is in that configuration.
createSignature

    inputs:
        byte[] message;
        string privateKeyId;
    outputs:
        byte[] signature;
    exceptions:
        KeyNotFound;
        InvalidKey;

Creates a signature of the message with the private key corresponding to the specified key id.

verifySignature

    inputs:
        byte[] message;
        byte[] signature;
        string publicKeyId;
    outputs:
        boolean isValid;
    exceptions:
        InvalidSignature;
        KeyNotFound;
        InvalidKey;

Verifies the given signature with the public key corresponding to the specified key id.

verifyCertificate

    inputs:
        Certificate cert;
    outputs:
        boolean isValid;
    exceptions:
        InvalidCertificate;

Verifies the given certificate and the certificate chain to a trusted root certificate.
importCertificate

inputs:
    Certificate cert;
outputs:
    string publicKeyId;
exceptions:
    InvalidCertificate;

Imports a certificate and the contained public key into the secure key storage. The unique certificate id is used internally to reference the public key.

5.1.4. Description

In order to ensure that system components running on a remote node have not been tampered with, we need to establish trust into the integrity of the remote system. In the case of the PeRA, trust in the remote system means relying on components of the remote system to work as expected so that privacy policies coupled to data are enforced. In other words, we trust a remote system if we can be sure of the integrity of its MPC components. It is the Trust Manager’s purpose to provide respective functionality. We employ trusted computing principles to achieve this.

The Trust Manager relies on a tamper-resistant hardware security module (HSM). In the following, we assume a Trusted Platform Module (TPM) chip as the HSM. While not necessarily the most capable or suitable HSM implementation, the Trusted Computing Group’s (TCG)\footnote{The TCG is a not-for-profit organization formed by industry stakeholders to develop, define, and promote open standards for hardware-enabled trusted computing and security technologies, with the self-proclaimed aim of helping users to protect their information assets (data, passwords, keys, etc.) from compromise due to attacks or physical theft. TCG website: \url{www.trustedcomputinggroup.org}} TPM specification and its TPM Software Stack (TSS) constitute a widely and readily available standardized trust architecture to design against and draw from its terminology. The TCG TPM specification [8, 9, 10] may pose restrictions to the MIP components in some points, whenever possible, these restrictions are named and factored in, but in some parts the PeRA specification may extend or deviate from the feature set of the existing TPM specification. An internal HSM Adapter serves as an abstraction layer between other subcomponents of the Trust Manager and the HSM. As a result, the PeRA remains independent of specific HSM hardware or software implementations and can also be adapted to other HSM architectures than TPM. Sections 5.1.5 and 5.1.7 elaborate further on HSM issues.

Based on the HSM, a chain of trust can be established at system boot that starts with the hardware chip and subsequently asserts transitive trust to other hardware and software components of the platform. Integrity measurement capabilities are required at each execution level to determine if certain system components are in a trusted state before passing execution to them. Each execution level requires its own integrity measurement
component, but in all stages measurement results are stored as integrity digests, i.e., hash values, in the HSM by extending specific Platform Configuration Registers (PCRs). Note that integrity measurement does not prevent the platform from reaching an untrusted state, but it guarantees that the platform state is accurately measured and stored. The boot process starts with the core Root of Trust for Measurement (cRTM) which measures the integrity of the BIOS before executing it, then the BIOS measures the boot loader before executing it, which measures the operating system kernel before passing execution to it. The Trust Manager is the first component of the PeRA to be loaded. It measures the integrity of other PeRA components when they are loaded and while running. Figure 5.1 visualizes the flows of measurement and execution. This way, a verifiable and trusted integrity chain has been established.

Thus, the Trust Manager requires integrity measurement as well as integrity verification capabilities, which are both handled by the internal Integrity Monitor subcomponent. Implementation of a trusted boot sequence is assumed to be present, e.g., as implemented by TrustedGRUB\(^2\) or results from the openTC project\(^3\).

If the platform is in a trusted state, the Trust Manager grants MPC components access to received and stored data. Basically, the Trust Manager should only decrypt a received message or release the decryption key of the database if integrity of the system can be guaranteed. Furthermore, key material must be securely stored inside the HSM with

---

\(^2\)TrustedGRUB project webpage: [http://www.sf.net/projects/trustedgrub](http://www.sf.net/projects/trustedgrub)

\(^3\)openTC project page: [http://www.opentc.net/](http://www.opentc.net/)
only the Trust Manager having access to the key store. The Key Manager subcomponent manages securely stored key material. The combination of transitive trust assignment and integrity measurement enables the definition of trusted platform configurations, i.e., the recorded trustworthy state of all MPC components. Keys can be bound to trusted platform configurations, so that they are only usable if the platform is in the corresponding trusted state. Remote entities can seal messages against such a trusted platform configuration, i.e., the message is encrypted with a public key of which the corresponding secret key is locked to a specific trusted platform configuration. The Trust Manager’s Crypto Module provides encryption/decryption and related services to the Trust Manager as well as other components, like the communication endpoints (Section 5.2).

The Trust Manager exposes an interface to the PCM that enables the PCM to query if the system is currently in a trusted state. Further the PCM can request the unlocking of specific databases, then the Trust Manager interacts directly with the DBMS to perform the unlocking (decryption) of the specified database, if and only if the system is in a trusted state. Additionally, the Trust Manager exposes a service interface for its crypto module so that other PeRA components can make use of provided encryption, decryption, signature verification, certificate verification, and key management capabilities.

The Trust Manager’s Setup Module is used to initially select the trusted platform configuration during deployment of the PeRA. A trusted third party verifies that all PeRA components are in a trustworthy state and that a freshly generated key pair is locked to that trusted platform configuration. The trusted third party then certifies the platform’s trustworthiness by issuing a public key certificate. This enables non-interactive remote platform attestation. The trusted third party certifies the trustworthiness of the platform and that it works correctly. Remote entities trust the trusted third party’s judgment and encrypt messages with the public key from the certificate. The receiver can only decrypt the message if it is in the certified trusted platform configuration because the secret key can only be used in that configuration. A detailed description of non-interactive remote attestation is provided in Section 5.3.

Figure 5.2 gives an overview of the internal structure of the Trust Manager. In the following, the functionality of the Integrity Monitor, Key Manager, Crypto Module, and Setup Module are explained in detail.

**Integrity monitor**

The integrity monitor subcomponent handles integrity measurement of the PeRA components and provides means to verify the integrity of the current platform state.

Integrity measurement serves the purpose of determining if a platform is in a trusted state. To be able to successfully enforce privacy, it is essential to guarantee that all PeRA main components are in a trusted state. To ensure this, the Integrity monitor measures the integrity of the PCM, the secure data and metadata repository, the controlled application environment, and the local communication endpoints, i.e., Importer and Exporter. Only if these components are in a verifiable trusted state, access to stored data, received data, and sensor output is granted to the PCM and other core components.
Integrity measurements are stored as hash values inside Platform Configuration Registers (PCRs) of the TPM chip. The TPM ensures secure storage and maintenance of accurate values of integrity digests in PCRs. Usually, PCRs can only be extended not overwritten. Each execution level extends specific PCRs. The TPM also provides integrity protected reporting of stored integrity digests. Thus, PCR values cannot be modified and a currently running component can verify the PCR values of other components loaded before them to determine their state.

The integrity monitor of the Trust Manager takes integrity measurements of components and stores them inside the TPM in dedicated PCR slots. The integrity monitor on its own does not decide if a component is in a trusted state. It only takes measurements of the current state and stores them, and then provides a function to the Trust Manager main component to verify if the current state corresponds to a known trusted state. The Trusted platform configurations used for state verification are also stored inside the TPM.

The point in time when the integrity monitor measures the integrity of a PeRA component is configurable per component. The following configurations are possible:

**on load.** The integrity of a component is measured each time the component is loaded or started. For example, a hash value of the component’s binary files is computed and the component is only loaded if the hash corresponds to the recorded trusted hash value. For on load integrity measurement the integrity monitor may also verify the signature of code components signed by trusted third parties, e.g., a signed OSGi bundle.

**periodic.** The integrity of a component is measured at runtime in periodic intervals. Integrity measurement at runtime is problematic, because dynamic program parts are meant to change to some extent during their execution. Thus, taking a global hash
value as integrity measurement is not feasible. However, the integrity of static parts of the component, e.g., static procedures or the text segment of a process, can be measured, if the memory address is known to the integrity monitor.

**on request.** Each time the PCM requests access to the database, the integrity of the component is measured.

Which integrity measurement configuration is chosen depends on the adversary model and the level of assurance that needs to be provided. Thus, if it can be guaranteed that loaded application code cannot be modified in memory on load integrity measurement is sufficient. If a more capable adversary has to be assumed periodic or on request integrity measurement needs to be implemented. Measuring integrity at runtime may also entail performance degradation, which needs to be taken into account.

Resulting integrity digests are stored inside the HSM through the HSM adapter and the underlying HSM software stack, e.g., TSS in case of a TPM chip. How many PCR slots are required to store the system state depends on the required flexibility in measuring certain components separately. A single PCR slot is sufficient if all components are always measured together. The slot would be subsequently extended with the integrity digests of all components. Each component (or set of components) that needs to be measured separately from the other components requires a dedicated PCR slot. Before a new measurement is taken the PCR values need to be reset, because the stored PCR values should always reflect the current state of PeRA components. It is crucial that only the integrity monitor and no other components are able to reset these PCR values. **Locality levels,** as introduced by the TPM 1.2 specification [8] support this. PCRs 16-23 can be reset by specific locality levels. The TPM 1.2 specification defines five locality levels and their anticipated usage (see Table 5.1). The integrity monitor should operate with locality level 2 (acting logically as part of a trusted OS), while other PeRA components, as well as the Trust Manager main component and its other subcomponents, must operate at a lower level, e.g., 0. The integrity monitor could then reset PCRs 20-23 [11], thus enabling dynamic integrity measurement. Further, it needs to be addressed how dynamic measurement results are certified beforehand and how remote entities should interpret such data. While we foresee different integrity measurement configurations, development of dynamic integrity measurement solutions it is not further detailed here.

Once the measured system state has been stored, the Trust Manager can verify the trustworthiness of the measured state. For that purpose, the measured state must be compared to a previously recorded state that has been determined to be trusted. The trusted

### Table 5.1.: TPM locality levels and anticipated usage.

<table>
<thead>
<tr>
<th>Locality level</th>
<th>Anticipated Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Special initialization hardware</td>
</tr>
<tr>
<td>3</td>
<td>Trusted initialization software</td>
</tr>
<tr>
<td>2</td>
<td>Trusted operating system</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Other Software</td>
</tr>
</tbody>
</table>

Deliverable 10 1.0
state information needs to be integrity protected to prevent tampering or should be stored inside the TPM. Key material can also be directly sealed to a specific platform configuration, i.e., the TPM internally checks the platform state and refuses usage of that key if the platform is in a different configuration.

Key manager

The key manager’s main purpose is secure key storage and management. Specifically, the Key manager component handles the following key types and credentials:

**Platform’s encryption key pair.** The platform’s main key pair. The secret key should not leave the TPM chip after creation and is sealed to PCR values representing the trusted platform state (see description of Setup Module). The public key can be exported and a privacy authority issues a public key certificate to certify that the private key is sealed to a trusted state. The secret key may only be used for decryption of messages encrypted by other entities, and not for signing.

**Platform’s signing key pair.** The purpose of the platform’s signing key pair is to securely and explicitly bind metadata (e.g., privacy policies) to data. The secret key is used for signing only and should not leave the TPM. The corresponding public key can be exported and a privacy authority should issue a certificate for it. As an alternative, pseudonymous keys can be used to protect the platform’s identity, i.e., multiple signing keys exist with corresponding certificates that do not contain identifying information. This option may apply to vehicles.

**Platform’s database encryption credentials.** How database encryption is realized depends on the employed database management system and the encryption support of the underlying database engine, Chapter 6 describes DBMS specific details. Regardless of how the database is encrypted (e.g., on a database level, table level, or data item level), it can be assumed that symmetric encryption will be utilized in some form due to performance reasons. The required symmetric keys or credentials are managed by the key manager and stored inside the TPM.

**Imported public key certificates and public keys.** Imported public key certificates and public keys also need to be stored securely to prevent modification or tampering. In TPM terminology such keys are called bind keys if they originate from another TPM and legacy keys otherwise.

A TPM chip has a limited number of key slots in volatile memory to hold active keys. The Key Cache Manager (KCM) is part of the TPM Software Stack (TSS) and handles encryption of inactive keys and the externalized non-volatile storage of the encrypted key blobs. This allows to use the TPM as a general purpose secure storage devices independently of the memory space available on the chip. The KCM ensures that externalized keys or data are never available in the clear (unencrypted) outside of the TPM. Keys can also be stored in a hierarchical order by defining parent key/child key relationships. In a hierarchical storage scenario, the parent key must be loaded first and be usable (i.e., if the key is sealed to a specific platform configuration) before the child key can be loaded into the
TPM. While in theory the trusted storage functionality provided by the KCM can be used to store keys or data blobs of any size, it is optimized for small payloads. Thus, larger data needs to be divided into appropriate storage chunks.

The Storage Root Key (SRK) is the root key of the trusted storage hierarchy. The SRK is embedded inside the TPM, it is created when the `TPM_TakeOwnership` primitive is executed. It never leaves the TPM and is protected by the SRK passphrase. The SRK should not be used to directly encrypt data, it should rather be defined as the parent key of storage encryption keys.

The KCM is only concerned with caching of keys but not their usage, the Trust manager’s key manager component makes use of the KCM’s capabilities to efficiently manage secure key storage for the PeRA. The key manager component keeps a mapping of all stored keys and assigned key identifiers, potentially in cooperation with the KCM of the TPM Software Stack.

The encryption credential of the local data repository is only shared with the DBMS when the PCM requests unlocking of a specific database and if the platform is in a trusted state. The DBMS, or a DBMS proxy must provide an interface (provideEncryptionKey) that allows the key manager, and the trust manager respectively, to pass on the credential required for database decryption. Furthermore, a second method is required, that enables the key manager to tell the DBMS to forget (i.e., delete) the credential again. This way the key manager can control when and for how long the DBMS can work with and access a database at a time. These parameters can be configured per system. Obviously, the integrity of the DBMS has to be monitored to ensure compliance to these functions.

Note that protection against techniques for key extraction from volatile memory [12] are not considered, other research projects, like EMSCB and openTC, already strive to enhance security of host platforms and to establish a fully trusted computing base on top of and complementary to the TCG specifications.

**Crypto module**

The crypto module is the general crypto facility of the PeRA. It is exposed to other PeRA components through interfaces of the Trust Manager. This way, cryptographic functionality is concentrated in one component. Other PeRA components that make use of these facilities are namely the confidential communication endpoints, the Importer and Exporter. The crypto module abstracts from the internally used crypto engine or crypto provider. For example, RSA encryption could be performed internally either by the TPM or in software, e.g., with the openSSL crypto library.

The TPM contains a RSA engine. RSA asymmetric key pairs can be generated inside the TPM with dedicated usage properties, e.g., some keys are only available for signing.

---

4The actual payload depends on the key slot size of the employed TPM chip.


6Open Trusted Computing (openTC) project website: [http://www.opentc.net/](http://www.opentc.net/)

7openSSL project page: [http://www.openssl.org/](http://www.openssl.org/)
and others can only be used for storage encryption. The RSA engine enables signing, encryption, and decryption on the chip. As a result, secret keys do not have to leave the TPM because all operations involving the key can be performed by the chip.

The crypto module exposes an interface for encryption and decryption of messages. Keys to be used are identified with key ids which are resolved internally by the key manager. The decryption method handles messages encrypted with keys sealed to a trusted state as well as decryption of messages encrypted with normal public keys.\(^8\)

Additionally, the crypto module also provides signature verification and certificate verification functionality to other components. Certificate verification may also entail validation against certificate revocation lists (CRLs) and validation of certificate chains.

As a last function, the crypto module provides an interface to import public key certificates. Internally, these are passed on to the key manager which either stores them locally or in its secure key storage.

Setup module

The setup module of the Trust Manager supports the initial deployment of the PeRA on a new system. During deployment, the installed PeRA instance needs to be certified by a privacy authority (see Section 5.1.7 for a discussion of the privacy authority’s role). The certification process entails platform validation. The integrity of all instantiated PeRA components needs to be verified and the system state needs to be validated as trusted state \(\tau\). The integrity measurement results of this trusted state are recorded.

Then, the setup module supports the generation of the platform’s main public key pair \((PK_S, SK_S)\) and the sealing of the secret key \(SK_S\) to PCR values representing the recorded trusted platform state \(\tau\). In a final step, the privacy authority certifies that the platform’s main secret key is effectively sealed to the trusted state by issuing a public key certificate \(Cert(PK_S)\) for the just created public key. It is the setup module’s task to aid the privacy authority in this process, e.g., by exporting \(PK_S\) from the TPM.

How the certification process and platform validation is performed in detail depends on the deployment environment and the privacy authority. Two likely general approaches are:

**System validation.** Once the PeRA has been instantiated, the system’s overall compliance with the PeRA specification is validated. This may include on-site validation of the system setup and code review.

**Component validation.** Components are independently verified by the privacy authority before deployment. Verified components are digitally signed by the privacy authority.

\(^8\)In practice, a message will be encrypted with a randomly generated symmetric key which in turn is encrypted with the public key of the receiver.
Deliverable 10

The issue of privacy certification will be further discussed in deliverable D11 in context of guidelines for privacy aware cooperative applications.

The setup module may also provide an administration interface to enable trusted updates of existing PeRA components or registration of new components. Updates may entail re-certification and key updates, depending on how the main key pair is sealed to the trusted platform state. Section 5.1.7 discusses the tradeoff between securely binding a secret key to a trusted state and update flexibility.

5.1.5. Platform Considerations

The Trust Manager and its functionality are interweaved with the Hardware Security Module (HSM). Requirements for a HSM strongly depend on the deployment platform. A HSM for an in-vehicle embedded platform needs to be energy efficient and shock resistant, while also providing crypto acceleration. A HSM in a server environment has to scale to a potentially large number of client requests and support parallel operation. And a HSM solution for a road-side unit requires weather resistant housing.

Current HSM modules that are compliant with the TPM main specification v1.2 [8] are often not capable to fulfill these requirements, because they are targeted at the personal computing market and often lack an integrated crypto accelerator.

Several efforts are under way to develop more focused hardware security modules that would also be suitable for use in cooperative ITS scenarios. The TCG’s Server Working Group already released a generic server specification, in 2005, that details issues and general requirements for TPMs in server environments [13]. However, adequate solutions are not provided and left to implementation specific server specifications instead.

The EVITA project\textsuperscript{9} currently develops an architecture for secure and tamper-resistant in-vehicle platforms. The TECOM project\textsuperscript{10} works on trusted hardware and system software for embedded platforms. The EMSCB project\textsuperscript{11} is a European effort to realize a more capable secure computing base for client platforms and servers. These efforts seem promising and may eventually replace solutions based on TPM specifications of the TCG, or advance them.

Therefore, the Trust Manager and the rest of the PeRA specification is independent of a specific HSM as long as the functionality outlined above can be provided. This allows portability to other more advanced HSM solutions. A more advanced HSM could, for example, include a cryptographic co-processor to accelerate cryptographic operations and provide better physical security and tamper-resistance than ordinary TPM chips. At the same time, it is anticipated that future generations of TPM hardware will provide better scalability.

\textsuperscript{9} E-safety Vehicle Intrusion Protected Applications (EVITA) project page: http://evita-project.org/
\textsuperscript{10} Trusted Embedded COMputing (TECOM) project page: http://www.tecom-project.eu/
\textsuperscript{11} European Multilaterally Secure Computing Base (EMSCB) project page: http://www.emscb.com/
5.1.6. Additional Material

The TCG TPM Main specification Level 2 Version 1.2 (Revision 103) consists of three parts and has been accepted as ISO/IEC standard 11889:

- Part 1: Design Principles [8]
- Part 2: TPM Structures [9]
- Part 3: Commands [10]

The TCG Software Stack (TSS) specification [14] provides a standardized API for TPM usage.


5.1.7. Discussion

In this Section, extra discussions are dedicated to TPM as a HSM, the role of the privacy authority, and relations to trusted domain approaches.

**TPM as hardware security module**

The decision to use a TPM chip as a hardware platform module stems rather from the comprehensive specification and the wide availability of TPMs and TPM-related knowledge than the capability of these chips. This has already been discussed in the previous Section (see Section 5.1.5). Henceforth, it is not surprising that we are not the only ones to consider the use of TPM for security and privacy in inter-vehicular communications. For example, Guette and Heen [15] analyse how the TPM native functions can be used to implement a pseudonym system for inter-vehicular communications. One of the main challenges of PRECIOSA is the development of a system-wide privacy solution for cITS, spanning vehicle, access, and backend domain. The TPM approach seems most viable to achieve a system wide solution, although current TPM chips do not yet fulfill all hardware requirements, especially in terms of crypto acceleration.

A hardware trust anchor, trusted integrity measurement, and secure key storage are the main advantages of employing a HSM, as described in the above Sections. However, a HSM also introduces some issues that become especially apparent in the server environment. Replication is an important aspect that needs to be addressed for backup and load balancing purposes. Keys and data should be protected against misuse and their accompanying privacy policies should be enforced, but keys and data should not be destroyed or rendered unusable by software or hardware failure. Furthermore, hardware and software updates are inevitable in the server environment to support scalability and system security.
At first glance, binding data and key access to a trusted state, on the one hand, and supporting replication, backups, and updates, on the other hand, seem contradictory. However, virtualization is starting to be widely adopted in server environments. Solutions exist for providing TPM access in virtualized machine instances. Virtual TPMs can be made available for each virtual machine [16]. Virtualization could be used to enable replication and backups. A virtualized system (including a virtualized TPM instance) could run the PeRA. The privacy agency validates and certifies the virtual system instead of the physical system. Additionally, the virtualization system could also be verified. The certified virtual PeRA system could then be cloned and run in parallel for load balancing, and snapshots can be taken as backups. It is also imaginable to run instances of the same server with different configurations in parallel. For example, after an update of certain components the old instance could remain active for a while to handle requests from legacy clients. Data migration between virtualized instances can go through the PCM and the PeRA components, treating the second instance as a remote entity running the PeRA inside the PEP.

Another issue to consider is who takes ownership of the TPM in a server environment, because the TPM owner can reconfigure the TPM and its access control. A natural choice in larger companies would be the data protection or privacy officer, who monitors that a company adheres to privacy and data protection regulations. As an alternative, the privacy authority that certifies the platform could provide the service of taking on this responsibility.

The privacy authority

The privacy authority is a third party trusted by cITS users to represent their privacy concerns towards service and infrastructure providers. National or international data protection authorities could fill this role, so could councils or consortia specialized on privacy and data protection. Platform certification needs to be standardized to unify certification requirements and ensure comparability of certification results. It is imaginable that the outcome of the platform validation is a privacy compliance rating on a predefined scale, which is then certified. The scale must be meaningful enough to support users in estimating if the guaranteed privacy level of a particular service provider matches their personal trust requirements and privacy concerns. Deliverable D11 will further detail the certification process.

PeRA and TVD

Trusted Virtual Domains (TVD) [17, 18] are an approach, which seems similar to the PRECiosa PeRA, at first glance. Local trusted domains are extended into remote entities by ensuring the integrity of those systems. In TVD the attestation occurs on the machine level and a virtual trusted domain is created between all trusted nodes. Multi-layered TVDs [18] have been proposed that employ virtualization to enable application isolation and compartmentalization. The open trusted computing architecture [16] also fits this category. TVDs have also been employed for privacy-preserving information sharing [19]. However,
TVDs do not regulate information flows inside a trusted virtual domain. The PeRA approach goes further by enforcing fine-grained privacy policies for information processing and sharing. For trusted virtual domains each virtual machine would have to implement its own privacy enforcement.

TVD and PeRA are complimentary concepts which could be integrated in the future to benefit from each other. As has been mentioned earlier, introducing a virtualization layer simplifies backup and replication strategies, because the virtualized TPM could be backed up or replicated with the rest of the system. Having several virtual machines each running PeRA in a TVD would also facilitate interoperability and coexistence of PeRA implementations or applications provided by different vendors or stakeholders. On the other hand, TVDs have been designed to enforce security policies, an integration with PeRA would also enable efficient fine-grained privacy policy enforcement.

5.2. Communication Endpoints

Apart from the Query API, communication with the outside world is a key interface of the PeRA. The communication components have to ensure certain properties, such as anonymity and confidentiality of any exposed data. The first step towards the development of such communication components is to identify what kind of communication is necessary to support our use cases. This analysis of the use cases revealed that several communication patterns with different characteristics are required for ITS applications [20].

As a first distinction, communication may be either unidirectional or bidirectional. For example, in a floating car data use case (see Section 2.2.2), a vehicle sends data to a centralized traffic management center, but no information is required to be sent back to the vehicle. In the hotel booking use case (see Section 2.2.4), a vehicle sends a request to a server, which then has to reply to this request, sending the requested information back to the source of the request.

Another distinction between types of communication is to differentiate public and confidential communication. When a finite group of entities is involved in the communication, transferred data can be kept confidential among these. In case that a vehicle communicates with a previously known server in the backend, it is possible to establish a private channel between the two. In this case, other entities cannot access information transferred in the private channel.

In contrast, in other use cases potential receivers of communicated data are not known in advance. Typically, if broadcast communication is involved, the sender of information is not aware of all stations that may receive a message. From the considered use cases, the intersection collision warning (see Section 2.2.1) requires periodic broadcasts of vehicle status information in order to inform vehicles in the vicinity about its current status. In this case, preventive measures are required to ensure that no personal information is – directly or indirectly – revealed in the public communication.

Therefore, we distinguish between two different communication APIs. The confidential communication API is intended to establish a private connection between two PeRA
Deliverable 10

capable units. Hence, data transferred over a confidential channel stays within the privacy enforcement perimeter. That is, personal information is not exposed during transport and under control of the PeRA at the peers. The second form is the **public communication API**. In this case, a communication privacy system ensures that no personal information is contained in publicly distributed messages (minimum disclosure). Moreover, permanent identifiers of nodes are replaced by pseudonymous identifiers in order to avoid linkability of data. The communication privacy system for public communication is not part of the PeRA itself but rather an external subsystem tailored to the specific deployment environment. A communication privacy framework for vehicles based on the SeVeCom architecture [21, 22] is outlined in Chapter 8.

The confidential communication API is implemented by the **Exporter** and **Importer** components on top of public communication. They act as local trusted communication endpoints for unidirectional and bidirectional confidential communication. The Exporter encrypts outgoing messages and the Importer decrypts incoming messages. Furthermore, Importer and Exporter are the only PeRA components directly exposed to the outside world. The Importer handles all incoming data, messages, and query requests; the Exporter handles all outgoing messages and query results. The Exporter has dispatching capabilities and ensures that configuration of (public) communication APIs adheres to privacy policies of the messages to be sent. Specifically, the Exporter configures the public communication API in terms of anonymity requirements. The Importer also provides an interface for PeRA components to local sensors.

Importer, Exporter, and the Trust Manager together form the local integrity protection part of the Policy Enforcement Perimeter (PEP) spanning all distributed PeRA instances. Inside the PEP, data does not roam freely. Each data item or message is explicitly bound to an accompanying privacy policy (by the Exporter), and strong encryption ensures that only specified recipients are able to access and utilize the data (see Figure 5.3). The MPC integrity protection components provide the means to establish trust between two entities so that the sender of a message can be sure that it is only accessible to the desired recipient and that the recipient adheres to any policies coupled to the data.

For handling local query requests and replies some type of session management is required so that the Exporter can reliably dispatch query results to the entity that posed the respective query request. Importer and Exporter need shared access to the session management. Whether session management is realized as an additional session manager component or if the information is encapsulated in a query context which is passed along the processing chain of PeRA components is left to specific implementations.

In the following, Importer and Exporter are described in detail.
Figure 5.3.: Importer, Exporter and Trust Manager of multiple nodes form the Policy Enforcement Perimeter (dark blue).

5.2.1. Importer

Purpose

The Importer is the communication endpoint for incoming requests from local uncontrolled applications and incoming messages via public or confidential communication. Further, the Importer provides an interface to PeRA components to retrieve local sensor values.

Dependencies and Prerequisites

- Trust Manager, for crypto support and key management.

Interfaces and Services

The Importer does not provide many interfaces to other PeRA components because it mainly reacts to events from the outside world. Towards the outside, a query interface for local applications and interfaces to communication systems are provided.
getSensorValue

inputs:
   String sensorId;
outputs:
   Object sensorOutput;
exceptions:
   UnknownSensor;

Returns the current sensor value for the specified sensor. The format of the return value depends on the sensor type. This service is only available internally to PeRa components.

putSensorValue

inputs:
   String sensorId;
   Object sensorInput;
outputs:
   uint returnCode;
exceptions:
   UnknownSensor;
   UnknownDataFormat;

External interface. Allows sensors to push data into the PeRA, which will be pooled by the Importer. Controlled applications are able to request this sensor information by the getSensorValue method.

queryApi

inputs:
   byte[] encryptionHeader; (confidential communication)
   byte[] message;
outputs:
   uint returnCode;
exceptions:
   UnknownDataFormat;

External interface. (Encrypted) requests received by the public or confidential communication system can be passed to the Importer via this interface. See Section 3.1 for details on how the query interface is designed.
Description

As the endpoint for incoming communication, the Importer handles requests from local uncontrolled applications and remote entities. Messages from remote entities may either be received via public communication (i.e., unencrypted) or via confidential communication. In the latter case, message content is protected by the Policy Enforcement Perimeter. Furthermore, the Importer also provides an interface to other PeRA components to retrieve local sensor values. Each of the listed tasks and how it is realized by the Importer is outlined below.

Requests from uncontrolled applications When an uncontrolled application locally poses a query request to the PeRA, the Importer receives the request and passes it on to the PCM via the PCM’s `executeRequest` interface. Before passing it on, the Importer enriches the query with additional metadata and context information. The caller should already provide `role` and `purpose` information, and the Importer may add an internal `request id` or a `caller id and address`. Either way, it must be guaranteed that the Exporter can later associate a query reply with the corresponding request, so that replies get delivered to the correct requester or requester instance.

The request interface for the outside application is basically the Query API (see Section 3.1). How the Query API is exposed depends on the chosen caller model of a specific implementation. It can either be tightly coupled (method invocation, event-driven, ...) or loosely coupled (MPI, Web services, ...). For tightly coupled systems, a synchronization issue arises, because the PeRA is internally organized as a one-way processing chain. The Importer receives a request, the PCM processes it, and the Exporter dispatches the result. The synchronization issue can be addressed with a session management component that provides the actual interface to applications and covers the asynchronicity.

Receiving messages via public communication When a message is received via public communication, the Importer is triggered by communication components of the underlying system. The Importer receives a message (probably as a blob). First, the message is unmarshalled or deserialized, if necessary. The resulting message should have three parts:

- Data (personal information, application specific)
- Metadata (privacy policies, Preciosa Policy Language)
- Binding data (digital signature explicitly binding data and metadata, public key certificate (optional))

The data is the message’s main content and may constitute application specific data or a query request phrased in the Preciosa query language (see Section 3.1). The accompanying metadata (see Section 3.2) consists of privacy policy information (see Section 3.2.1) governing the message’s data, and potentially further context information, e.g., identifiers for public key certificates if no certificate is attached. The binding data is a digital signature.
performed by the message sender or data subject $S$ on a hash value computed from the data and metadata:

$$\text{Sig}_S = SK_S(H(\text{data} \parallel \text{metadata}))$$

$\text{Sig}_S$ explicitly couples data and metadata. In order to verify $\text{Sig}_S$, the Importer requires the public key $PK_S$ corresponding to $SK_S$. To achieve a certain level of authentication, $PK_S$ should be tied to an identity by a public key certificate issued by some CA. Note, that this identity does not need to uniquely identify the sender, a pseudonym is sufficient.

In the next step, the Importer verifies the data-metadata binding. Because the message has been received via public communication, i.e., outside the PEP, the Importer can only verify that the message has not been altered since the signer sends it, but not if the signer is also the origin of the data.

If the check succeeds, the data and metadata is passed on to the PCM, either as a query request or as an application invocation request, if the message was addressed to a controlled application (specified in the metadata).

**Receiving messages via confidential communication** The steps involved in receiving a message via confidential communication, i.e., inside the PEP, are very similar to the public communication case. The Import receives a message (most likely as a blob) from the underlying system’s communication unit. The message is unmarshalled and consists of two parts:

- Encryption header (encryption information)
- Encrypted message (data, metadata, and binding data encrypted with the receiver’s public key)

The Importer evaluates the encryption header, and cooperates with the Trust Manager\textsuperscript{12} for decryption of the encrypted message. Encryption succeeds, if and only if, the platform is currently in the trusted state the secret key is sealed to. If the platform is in an untrusted state, the received message remains inaccessible until the correct trusted state is restored.

Assuming the message can be decrypted, the subsequent handling is the same as for public messages. The decrypted message is unmarshalled or deserialized, the data-metadata binding is verified, and data and metadata are passed on to the PCM. Verification of the data-metadata binding allows to identify the PeRA instance that created the message and the policies. This information can be stored locally, e.g., by the Privacy Trail Manager. This way, a distributed processing trail is created which can be followed up if disputes arise about how data has been used or if false policies have been attached.

\textsuperscript{12}The Trust Manager’s crypto module is involved, which in turn requires the TM’s Key Manager to load the appropriate secret key.
Local sensors

In addition to handling incoming messages and queries, the Importer provides an interface for other PeRA components (namely the PCM) to access local sensors (location sensors, speed, heading, ...).

Other PeRA components can request a sensor value with the `getSensorValue` interface. The corresponding sensor is accessed and the current sensor value is returned. The Importer keeps a registry of locally available sensors, their assigned identifiers, and how to access them. Unless sensor access can be standardized, a handler for each sensor type is required and needs to be registered with the Importer on deployment. Note, that a default policy also needs to be specified for each sensor so that sensor output can be properly utilized by the PeRA.

Protection of sensors is out of scope of PRECIOSA, it is being addressed by other projects, e.g., EVITA. We assume for the PeRA that sensor output is trustworthy and correct. Here, correct means the sensor output has not been tampered with or subsequently modified. Sensor accuracy is not addressed.

5.2.2. Exporter

Purpose

The Exporter is the communication endpoint for outgoing messages. Every query reply or message passes the Exporter before it leaves the local PeRA instance. Depending on provided configuration parameters, the Exporter dispatches a message or query to the correct communication outlet and configures it as mandated. For example, a query reply is passed to the local uncontrolled application that posed the request. Encryption of messages for confidential communication inside the PEP is also handled by the Exporter. Both, public and confidential communication is then dispatched to the communication privacy system, (see Chapter 8), if required by the configuration parameters.

Dependencies and Prerequisites

- Trust Manager, for crypto support and key management.
- Communication privacy system, e.g., SeVeCom architecture.

Interfaces and Services

The Exporter provides an export interface, to be called by the PCM only.
export

inputs:
  Object data;
  String destinationId;
  Object exportConfig;
outputs:
  uint returnCode;
exceptions:
  DestinationUnresolvable;
  ExportConfigurationError;

Internal interface. The data (also containing a privacy policy) is serialized or marshalled (if required) and dispatched to the specified destination if export configuration can be realized.

Description

The Exporter acts as a dispatcher. Any message or query reply leaving the local PeRA instance arrives at the Exporter and is dispatched in accordance with provided export configuration parameters. The Exporter does not make dispatching decisions on its own. Instead, the PCM provides required configuration parameters and the Exporter ensures that the required configuration is provided. We can distinguish three major dispatching scenarios. The Exporter has to pass a query reply to a local uncontrolled application, a message needs to be send via public communication, a message needs to be encrypted and send via confidential communication.

Public communication is mainly used for broadcast based scenarios where receivers are potentially unknown. Whenever possible, confidential communication should be used to provide enhanced privacy protection for messages containing personal information. Regardless if communication is public or confidential, a communication privacy system should be in place that employs pseudonymous network identifiers and supports common communication patterns. A communication privacy system is described in Chapter 8.

The dispatching destination is set by the export configuration provided by the PCM.

Replies to uncontrolled applications  When the Exporter receives a query reply, which needs to be passed on to a local uncontrolled application, it has to determine the appropriate receiver first. As mentioned in the description of the Importer, a session management component can be introduced to provide this functionality, otherwise information about the correct recipient needs to be encoded in the context information or metadata of the request and subsequently the reply. The Exporter has to evaluate this information, locate the associated query handle and dispatch the reply. Note, that query replies and messages leaving the PeRA should be bound to a policy, although enforcement can not be
guaranteed and adherence to policies in only best-effort, in the case of uncontrolled applications. Binding of data and metadata is discussed as part of the Sections on sending messages via public or confidential communication.

**Sending messages via public communication** Some applications and use cases require data to leave the privacy perimeter. In this case, however, we want to ensure that no personal information leaks and that exposed information cannot be linked to an individual easily. At the same time, it should be guaranteed that any sent data is authenticated as being created by a valid (certified) sender and that the data cannot be altered by others after being sent.

The first measure to fulfill this goal is to use pseudonymous source identifiers, which are changed from time to time. In order to ensure authenticity and integrity, messages are additionally signed by the sender, and a certificate is attached. The certificate is required to establish trust in the sender, and maybe left out if it is presumably known to the receiver.

These measures are taken regardless of other characteristics of the communication. For example, the same should be applied both for broadcast or unicast, if the communication is exposed to (potential) public reception.

The cryptographic elements such as keys and pseudonyms as well as cryptographic operations of creation and verification of signatures are all provided by a communication privacy system. In our case, we use the SeVeCom framework. For details on how these components work, see Chapter 8.

Before messages are handed to the communication privacy system, the following steps are performed by the Exporter:

First, the Exporter has to perform the binding of data and metadata. The binding is created by signing a hash value of, both, data and metadata, with the platform’s signing key:

\[
\text{Sig}_S = SK_S(H(\text{data} \parallel \text{metadata}))
\]

The signature generation is performed in cooperation with the Trust Manager’s crypto module and key manager. This way, cryptographic functionality does not need to be replicated. Depending on how public keys are made available, the Exporter attaches the corresponding public key certificate to the message.

In the next step, data, metadata, binding data, and public key certificate (optional) are serialized or marshalled. The result is a blob which is ready to be passed on to a communication system.

Then, the Exporter determines the destination and resolves it to some address, if required. Before, dispatching the message to the public communication system, the communication system is configured according to the requirements specified by the PCM in the export configuration. Potential configuration parameters are the use of pseudonyms, pseudonym change frequency, or the selection of specific communication patterns. Once all necessary configurations have been performed successfully, the serialized message, including
a data-metadata binding, is dispatched to the appropriately configured communication system for sending.

**Sending messages via confidential communication** The sending of messages via confidential communication is quite similar to using public communication, except for a few additional steps.

Again, the data-metadata binding is created first by signing a hash value of both with the platform's signing key. Then the message is marshalled or serialized.

Now, the message needs to be encrypted with the recipient's public key. The Trust Manager's key manager stores imported certificates of other entities or can fetch them from a central repository, if not locally available. The certificate should be issued by a privacy authority and certify that the remote platform's corresponding secret key is sealed to a certified and trusted state of the remote platform's PeRA instance. This means once encrypted with that public key, the recipient can only decrypt it if all its PeRA components are in a trusted state.

Once the message has been encrypted, the Exporter prepends an encryption header that gives details about the used key, algorithm and so forth. Both, encryption header and encrypted message are again serialized and are ready for sending. Then, the Exporter configures the communication system according to the export configuration specified by the PCM. Only if the required configuration can be guaranteed the encrypted message is finally dispatched for sending.

**Platform Considerations**

The Exporter requires a communication privacy system to dispatch messages to. For vehicles, pseudonymization of messages is mandatory to prevent trackability. Servers and road side units do not require pseudonym usage and should use unique identifiers in most cases, so that they can be clearly identified by recipients. The SeVeCom architecture is a viable solution for communication privacy for vehicular communication. Backend components, however, may require different solutions.

**Additional Material**

Chapter 8 outlines a communication privacy framework for vehicles based on the SeVeCom architecture. [21] and [22] give a comprehensive overview of the SeVeCom results, focusing on architecture and design, and implementation, performance, and research challenges.

Schoch et al. [20] define communication patterns for inter-vehicular communication.
5.3. Non-interactive Remote Attestation

The concept of non-interactive remote attestation defers from typical trusted computing scenarios, where remote platform attestation or direct anonymous attestation (DAA) [23] would be employed to convince a remote entity (e.g., a vehicle) of the integrity of the local entity’s system (e.g., a server) before the remote entity sends any sensitive information. Simplified, the vehicle would request the server’s current state, and a state report signed by the TPM would be returned. The vehicle would decide if it trusts the server in the current state, and then send its message or not, optionally sealed to the reported state. However, in a cITS environment such bidirectional transactions may be impossible, expensive, or may even be privacy infringing themselves. For example, a reply message from a server could reveal a vehicle’s location in case of position-based routing.

Therefore, we outline a new approach in the following, which we call non-interactive remote attestation. The policy enforcement perimeter established by the distributed PeRA instances can achieve a similar effect by only requiring one-way communication. During system setup, a privacy agency verifies the integrity of all PeRA components and checks that they have not been modified. The HSM measures the platform state and generates an asymmetric key pair \( (PK_S, SK_S) \) that is sealed to the measured PCR values of state \( \tau \). The secret key \( SK_S \) is securely stored and managed by the HSM and should never leave it. The trusted privacy agency issues a certificate \( Cert_S \) for the public key \( PK_S \), that asserts the trustworthiness of the platform and especially that the corresponding private key is only usable if the platform is in this state. Such certificates can then be deployed to vehicles, e.g., together with the server’s message assembly application running on the vehicle. Vehicles could also periodically update service provider certificates from a public repository.

Now, a vehicle could encrypt messages with \( PK_S \) and be sure that the server can only decrypt it if it is in state \( \tau \), because \( SK_S \) is not usable otherwise. Of course, this requires that server and vehicle both trust the privacy agency to work correctly. The difference to a general PKI scheme is the sealing of the secret key to a platform state and configuration deemed trustworthy. As is typical with industry certifications, users or clients entrust a third party to correctly perform and attest the certification process.

**Example.** A short example may serve as an illustration on how non-interactive remote attestation is employed by the PeRA. Figure 5.4 depicts the situation. Whenever a vehicle sends a data set to a service provider, the Exporter encrypts the data set \( D \) together with the policy \( Pol \) with the service provider’s public key \( PK_{SP} \) taken from \( Cert_{SP} \). The result is \( C = Enc_{PK_{SP}}(D|Pol) \) sealed to the certified state of the service providers’s server. Sealing expresses the fact that data can only be decrypted by the receiving system’s HSM if the integrity of the destination system is ensured. Then, \( C \) is sent to the service provider.

At the receiver side, the Importer passes \( C \) to the Trust Manager, which controls the HSM containing \( SK_{SP} \). The Trust Manager checks whether all system components are in a state corresponding to the integrity measurement values of a trusted state. Only if this
verification is successful, the encrypted message is decrypted inside the HSM and then data and policy are placed in the data repository. Note, that sealing to the trusted platform state is enforced by the hardware chip not by software components.

This process ensures that personal data is only accessible within the Policy Enforcement Perimeter established by the Trust Managers of participating entities. Inside the PEP, it is guaranteed that any data is always coupled with its according policies and that those policies are adhered to, because entities with modified PeRA instances would neither be able to access data received from other nodes nor locally stored personal information of other nodes. Thus, when data can be accessed it is guaranteed that a PCM is in place and intact which checks data access against attached policies.
6. Secure Data and Metadata Repository

6.1. Data storage capabilities

The primary responsibility of the data repository is secure storage of data and metadata. There are two main principles for data and metadata storage according to the Privacy Enforceable Runtime Architecture introduced in deliverable D7 [1].

**Restricted information flow** Application data can be accessed only through PCM and all queries passed to the repository must be authorized by PCM. The query results are sent through PCM to requester.

**Data access dependent on platform integrity measurement** Before accessing any application data in repository, the integrity of the system must be guaranteed. Only when system is in an integral state data and metadata can be accessed.

Data repository has no its own privacy policy compliance and platform integrity measurements, it must rely on PCM and Trust Manager. For administration and deployment purposes there are also possibly other interfaces offering specific functionality for administrators, security officers and other authorised users.

6.2. Data model

6.2.1. Policies

From a storage point of view, privacy protection policies, introduced in Chapter 3, are XML documents with unique a identifier represented in the form of an URI. In case of issuing a new policy or a new version of a policy it has always a new identifier, modification of existing policies is forbidden, therefore there are only two operations which the metadata repository must support for policies: store and retrieve policy. The specification for URI length does not dictate a minimum or maximum, to store URIs in database using native URI type is suggested. If DBMS does not support native URI type one of the generic types like CHAR, VCHAR, CLOB can be used, provided that the database engine supports unique indexing on this type and that the maximum length of URIs will not exceed the column size. There are two ways to represent privacy policies in the datastore:
• Using a generic datatype like BLOB (Binary Large OBject), CLOB (Character Large OBject) or equivalent datatypes for storing whole documents. In this case, the datastore can store a privacy policy as a whole document without the possibility to perform any operation on a separated part of the policy and without any validation. According to the information flow in PeRA, the datastore does not perform any operations on policies except store and retrieve, thus this method is adequate for storing policies in PeRA.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI_ID</td>
<td>DMBS native URI type or equivalent</td>
<td>Primary key, unique URI</td>
</tr>
<tr>
<td>CLOB_Policy</td>
<td>CLOB (Character Large Object) or equivalent</td>
<td>Policy document</td>
</tr>
</tbody>
</table>

Table 6.1.: data model for policies

• Using native XML datatype. Several available database management systems (DBMS) offer a native XML type for representing XML documents in database. The specific data types offers additional features for processing XML files like: XML validation, XML modification, XSL transformations, XPath searching etc. This representation of XML documents in database has some limitations: XML documents stored in XML type must be syntactically correct, operations performed on XML type documents are not efficient because of costly operations on XML text. Some of the modern databases offer a XML-schema based XML type, which provides better performance but requires a repository of XML schemas, and stored documents must be valid with respect to a XML schema.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI_ID</td>
<td>DMBS native URI type or equivalent</td>
<td>Primary key, unique URI</td>
</tr>
<tr>
<td>XML_Policy</td>
<td>DBMS native XML format</td>
<td>Policy document</td>
</tr>
</tbody>
</table>

Table 6.2.: data model for policies

Which of the above representations of policy in database is suitable for a certain implementation depends on system requirements, available capabilities and resources. The suggested method is to store policy as generic datatype CLOB or equivalent until there is a need to access or update a piece of policy.

6.2.2. Ontology

An Ontology is a set of concepts and relationships between those concepts. The most convenient way for storing an ontology in databases is to break each ontology statement down into a <subject, predicate, object> triple. This triple is effectively modelled as a directed graph: The subject and object of the triple are modelled as nodes, and the predicate (or property) as a directed link that describes the relationship between the nodes. The direction of the link always points towards the object.
This method for storing ontologies in databases does not make assumptions about any particular application domain, nor does it define the semantics of any domain.

Enterprise class DBMS offer native data types for storing and processing ontologies in databases using triplets in RDF\(^1\) format. Subject, predicate and object in RDF triplets usually have the form of URIs.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.exp.org/PNet/S1">http://www.exp.org/PNet/S1</a></td>
<td><a href="http://www.exp.org/PNet/P1">http://www.exp.org/PNet/P1</a></td>
<td><a href="http://www.exp.org/PNet/O1">http://www.exp.org/PNet/O1</a></td>
</tr>
</tbody>
</table>

Table 6.3.: RDF ontology representation

DBMS with native RDF support, besides triplets storing, may offer wide set of tools for managing ontologies, like an inference engine, indexing etc. Native support for RDF format in data repository is not required for PeRA, however if data repository offers such capability, it is suggested to use the RDF format for storing and processing ontologies.

To improve efficiency this model can be normalized.

\(^1\)http://www.w3.org/RDF/
Applications can store application specific data, including privacy sensitive data, in the secure datastore. According to PeRA principles, an application can store its data only in secure datastore sending queries through PCM. An Application may store in the repository two kinds of data: privacy sensitive data and not sensitive data. To preserve privacy protection, all privacy sensitive data must be coupled with a privacy policy in the datastore. It is not possible to create one universal data model for all possible applications hosted inside PeRA, instead it is assumed that all data are coupled with policy, which for non privacy sensitive data can be an empty policy. The consequence of this assumption is that all operations on the data need an attached policy. Applications can create and use their own data model, the only requirement is that a correct policy must be attached to the data. Moreover, the PCM can analyze operations independent of the data model. The data store must offer for applications logically separated storage areas with ability to create data models. Exclusive storage areas for each application protects against unwanted interferences with another application.

The data repository must assure that all application data is stored and retrieved together with its protection policy or policy identifier.

6.2.4. Audit data

For audit operations, compliance checking, and the investigation of privacy incidents, systems collect audit data in the data repository. Audit data must be a trusted source of information for auditors, security officers, and law enforcement agencies, therefore, this
data must be protected against unauthorised modification and deletion. From an information flow point of view, the data store must work as a “write only device” for audit data. Each audit data record must be stamped with a timestamp by the data repository.

During system exploitation, the amount of audit data is continuously growing, therefore the scope of logged information must be limited to the information required. Moreover, unnecessary data must be deleted by an authorised user or automatic maintenance procedures.

DBMS systems acting as the data repository for audit data must offer separation of privileges allowing “write only” access to the repository. Moreover time stamping of the data inside repository should be possible.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Number(unique)</td>
<td>Unique identificator</td>
</tr>
<tr>
<td>Timestamp</td>
<td>DBMS native timestamp format</td>
<td>Time stamp added to data by data repository</td>
</tr>
<tr>
<td>AuditData</td>
<td>Implementation dependent</td>
<td>Audit data</td>
</tr>
</tbody>
</table>

Table 6.5.: data model for audit data

### 6.2.5. Capabilities for data storage in PeRA

To sum up above considerations, the data store must provide the following functionality:

1. Capability to guarantee uniqueness of URI for policy
2. Logically separated storage areas for applications
3. Capability to restrict write only access for audit data
4. Support for storing big XML policy documents
5. Guarantee coupling of data with policy

Most available database systems offer most of those capabilities inside DBMS engine, the others must be implemented for the specific system.

### 6.3. Platform considerations

On the market, there is no data repository available which is fully compliant with PeRA. For this reason, the data repository has to be build from one of available DBMS wrapped by an additional middleware layer supporting extra functionality. From the system point of view, the data repository offers interface for data manipulation and internal representation of the data is invisible. It broadens the choice of available data storage methods and platforms. This flexibility is very important especially for small embedded devices, where available
resources are very limited. For small embedded solutions, data can be stored in files or device memory, for big enterprise class systems the best solution is to use enterprise class DBMS with advanced administrative capabilities.

6.4. Discussion

There are still some issues which should be discussed and resolved in deliverable D11 – "Guidelines for privacy aware applications".

Encryption support Encryption protects data at rest against unauthorised access or modification, but encryption decreases data processing efficiency and requires basic administration task on data storage, like key management, update of CRL (Certificate Revocation List). For small, zero administration, devices encryption support may not be suitable. For big enterprise solutions, where data are stored in well protected server rooms with limited access and trained staff, encryption adds unjustified overhead for data processing. Decision about encryption of the data in data store should depend on risk analysis for certain systems.

Relational vs. key-value data model From a system point of view, internal representation of the data in DBMS is invisible for applications. Both, relational model and key-value model can be used. Which one is suitable for a certain system depends on system requirements and available resources.

Query language Query API offers restricted access to the data in repository. The query language supported by Query API on the one hand must be flexible to support application data models, on the other hand must be easy to control by PCM.
7. Additional Components

Privacy Enforceable Runtime Architecture is a referential composition of functional blocks and interfaces between blocks. PeRA is an abstract concept of a referential architecture applicable for all possible systems preserving privacy according to privacy principles. Most of the specified components are essential for preserving privacy protection and all implementations compliant with PeRA must contain such components. Some of the components are optional, and decision about implementation of such components is dependent on system requirements. In this chapter optional components for the Privacy Enforceable Architecture are described.

7.1. Query IDS

**Purpose** The Query Intrusion Detection System (Query IDS) is responsible for on-line monitoring all queries passing through the PCM. The primary task of Query IDS is to detect privacy intrusion inside PeRA.

**Prerequisites and Dependencies.** The PCM must send copies of all queries passing through to Query IDS. Moreover in addition to query, PCM must send with query, context data like timestamp, client identification, associated policy. Query IDS need also access to Data/Metadata repository for storing query history and all implementation specific data.

**Interfaces and Services.** The Query IDS receives all queries and requests passing through the PCM.

**Description.** The Query IDS is analogous to Intrusion Detection Systems widely used in protection of IT networks. Query intrusion detection systems can be

- **Passive** In a passive system, the intrusion detection system detects a potential privacy intrusion, logs appropriate information and signals an alert to system owner or administrator. Further processing is dependent on human reaction.

- **Reactive** In a reactive system, also known as Intrusion Prevention System (IPS), in case of detection potentially privacy threat, the IPS automatically takes action to prevent privacy intrusion.

Query intrusion detection systems, taking into account historical records, must decide if a query or sequence of queries is a privacy intrusion or not. Privacy intrusion can be detected using different techniques and criteria:
Rule-based Query IDS  The simplest implementation of Query IDS detects privacy invasion on the basics of fixed set of rules. Such rules may limit frequency of requests, volume of received data per time period, specific sequences of requests.

Anomaly-based Query IDS  Query IDS monitors application traffic and stores parameters of typical queries and query sequences. Any queries or query sequences which are to different form typical application activity are detected as a privacy invasion.

Signature-based Query IDS  Query IDS has a repository of preconfigured and predetermined attack patterns known as signatures. IDS compares query or sequence of queries with signatures and decides if query is a privacy invasion or not.

Knowledge based Query IDS  From a privacy perspective, each query is an information about the requester's intention, what he wants to know, each response is a information he received. By analysing the history of communication between requester and database, the IDS can infer what knowledge the requester acquired and if he had knowledge sufficient to infer privacy sensitive data for which he is not authorised. On the basis of acquired knowledge, the IDS can decide if a query is a privacy invasion or not.

Privacy metrics based Query IDS  Query IDS can measure privacy protection level of queries and requests using privacy metrics like k-Anonymity. Based on a permissible value of the privacy metric, the IDS classifies a query as a privacy invasion or not.

The actions taken in case of detection of a privacy intrusion can vary depending on implementation.

- log into logfile: in the simplest case, the Query IDS logs privacy intrusion alert into logfile. Regular analysis of Query IDS files is in responsibility of system administrator
- send alert: In case of privacy intrusion detection Query IDS sends alert to system administrator.
- Block requests from certain application. Application can be unlocked only by system administrator.
- Block all request from certain application for defined period of time.

Platform Considerations.  Query IDS can be implemented from simple rule-based detector to sophisticated, heuristics engine. For mobile devices, processing locally stored data, simple rule-based Query IDS is adequate and sufficient. For bigger enterprise class solutions, processing many kinds of information from different service providers, IDS should be a more advanced solution configurable and manageable by system administrator.
Discussion. Detecting privacy invasions is a major problem for IDS/IPS system. There is no simple answer how to detect privacy intrusions, when a query or set of queries is a normal and privileged system activity and when it becomes to be privacy intrusion. For real privacy intrusion detection, the system should build a model of knowledge accumulated by requester or adversary in the context of privacy, then measure this knowledge and decide if privacy intrusion happened. This kind of privacy invasion detection is a challenge for future projects. Intrusion detection systems must be configured with care because of high risk of negative influence on the system. If the IDS is too sensitive it may disrupt or totally jam system functioning, on the other hand, a non sensitive IDS gives an adversary the chance of going unnoticed.

7.2. Privacy Maintenance

Purpose The primary responsibility of the Privacy Maintenance component is to ensure that the principles of limited retention, limited collection and limited use are followed. This component consists of two functional blocks: Data Collection Analyser and Data Retention Manager, both are performing certain off-line tasks on data stored in repository, including application data.

Prerequisites and Dependencies. Privacy Maintenance component is highly dependent on data stored in Secure Data/Metadata Repository. For proper functioning, sufficient data set collected from PCM component must be available. This is also supporting Query IDS, performing analysis of spatiotemporal data.

Interfaces and Services. Access to data repository is required.

Description. Privacy Maintenance component has two functional blocks

Data Collection Analyser This component performs certain operations on the stored data: controls whether information is being collected and used in accordance with principle of limited collection, analyses whether data is stored no longer than is specified by data subject or is necessary, collects historical queries for detecting privacy intrusions, collects data for audit and compliance check. The responsibility of this component is to detect any noncompliant with privacy principles data storage operations.

Data Retention Manager Data Retention Manager ensures that any data in repository, which have limited retention time, will be not available for applications after permitted retention time. In the most cases privacy sensitive data should be deleted after retention time, but depersonalisation or aggregation of the data is also acceptable. In case when data must be stored after retention time for law enforcement agencies, instead of deletion data can be marked as deleted or moved to a separate storage area.

Platform Considerations. The Privacy maintenance component can be implemented using any job scheduler available on the execution platform or operating system. For enterprise class solutions, job schedulers performing privacy maintenance tasks can
be available on the database level. Most of the operating systems offer job sched-
ulers (e.g. cron in unix/linux systems), which can perform privacy maintenance tasks in batch mode.

7.3. Privacy Trail Manager

**Purpose**  The Privacy Trail Manager (PTM) component is a central place for logging infor-
mation required for later audit operations, compliance checking and the investigation of privacy incidents.

**Prerequisites and Dependencies.** PTM receives queries and requests sent by the PCM for audit and compliance checking.

**Description.** The primary responsibility of Privacy Trail Manager (PTM) is to ensure safe audit records storage and non-repudiation. For this purpose PTM may use cryptographic functions for signing and encryption of the data. All records stored by PTM must be stamped by reliable time source.

**Platform Considerations.** In most cases, PTM will be implemented using database privileges system and encryption support. For more advanced solutions specialised tamper resistant write-once hardware modules will be more suitable. For processing huge amounts of audit data one of the Hardware Security Modules (HSM) can be required: Cryptographic accelerator to improve cryptographic operations efficiency, Trusted Platform Module (TPM) as a keystore and protection against data stealing.
8. Communication Privacy

As stated in Section 5.2, the public communication API is a communication privacy system that directly appends to the Exporter and Importer to ensure that no personal information is transmitted in publicly distributed messages. Privacy-preserving communications are achieved internally by the components in the public communication API. This chapter will specify these components in details.

8.1. Requirements on Communication Privacy

Before we specify the components for communication privacy in details, let us first look at the requirements on privacy-preserving communications. In other words, what kind of goals the communication privacy framework should achieve.

8.1.1. Review of use cases

From the communication perspective, we identify a number of communication requirements from the selected use cases in the context of cITS. The list of the consolidated uses cases as described in Section 2.2 are:

- Collision Warning
- Floating Car Data (FCD)
- Online Navigation
- Hotel Booking

These use cases are representative because they represent not only PRECIOSA, but also the interests of stakeholders such as Article 29 Working Group and other ITS projects like CVIS. Therefore, they provide us with very good insights into vehicle-related communications.
8.1.2. Communication patterns

The use cases considered in PRECiosa require communication with partially differing characteristics. For example, intersection collision warning is a vehicle-to-vehicle application which is supposed to make use of IEEE 802.11p communication technology. Moreover, the use case is commonly seen to send data via single-hop broadcast on a periodic schedule. This way, other vehicles in the vicinity are made aware of the presence and the current status of a vehicle. By using the received status data from all other vehicles around, collision warning applications can predict potentially hazardous situations.

In summary, we can compile a communication pattern based on characteristics such as message **trigger**, **addressing**, **direction**, **transported data**, **quality of service requirements** and so on [20]. For the collision warning, we denote the pattern as beaconing and define it as a permanently ongoing, periodic service, which sends vehicle status data unidirectionally via single-hop broadcast. The receivers cannot be determined in this case, because any vehicle which is reached by a wireless broadcast transmission is implicitly addressed. From a privacy perspective, data sent via broadcast can be received by anyone, which means that no personal information should be contained or derived by the contained data.

The same characterization can be done for the other considered use cases, leading to other communication patterns. Unlike collision warning, FCD applications send data to a defined receiver, that is a traffic management center (TMC). Hence, addressing is unicast instead of broadcast, and the receiver of the data is previously known to the sender. But still, the application will send periodic status reports, although potentially less frequently than collision warning. Thus, the communication trigger is periodic as well, like the transported data, which also likely contains status data such as position and speed. Similarly, data is delivered only in one direction, as no answer is required to deliver FCD status data to the TMC. Quality of service considerations are much less relevant than for the time critical collision warning.

When it comes to online navigation and hotel booking, several characteristics change. While the connection is also unicast, the sender expects responses from the peer, e.g., the requested route or the confirmation of the hotel booking. Hence, the communication is bidirectional. Additionally, these use cases do not act periodically, but based on events such as the vehicle passenger trying to find a hotel or, or when an route update is necessary.

Table 8.1 gives a short overview on the communication patterns required in the PRECiosa use cases, and how they fit into PeRA.

As we can see in Table 8.1, several considered use cases require unicast communication, where the communication peer is known. If the peer is PERA-capable, the communication can be encrypted so that it can only be deciphered by the appropriate peer with correct key material. For most inter-vehicle communication use cases such as collision warning, broadcast communication is prevalent. In this case, any data leaves the policy enforcement perimeter, so that such data ought to be sufficiently anonymized.
Table 8.1.: Communication pattern analysis

<table>
<thead>
<tr>
<th>Use case</th>
<th>Communication pattern</th>
<th>Comm. interface to PeRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision Warning</td>
<td>Periodic Beaconing</td>
<td>Public Comm. API</td>
</tr>
<tr>
<td>Floating Car Data</td>
<td>Unidir. Unicast</td>
<td>Confidential Comm. API</td>
</tr>
<tr>
<td>Online Navigation</td>
<td>Bidir. Unicast</td>
<td>Confidential Comm. API</td>
</tr>
<tr>
<td>Hotel Booking</td>
<td>Bidir. Unicast</td>
<td>Confidential Comm. API</td>
</tr>
</tbody>
</table>

8.1.3. Privacy requirements

Based on the analysis, we can derive the privacy requirements as:

- **ID privacy** specifies how much the identity of the sender should be kept secret, depending on the applications.

- **Location privacy** has different levels, which range from distributing location information freely throughout the network to totally keeping it private.

- **Jurisdictional access** public authorities should have the access to the identity or location information of cars in case of criminal or liability investigations.

Due to the mission-critical nature of some of the safety applications, privacy requirements are constrained by a set of security requirements. Although these are not privacy requirements, these requirements have direct influence on the solution space of privacy communication framework, the relevant security requirements are:

- **Authentication** Trust is crucial in safety-related applications, in which vehicles react according to legitimate messages they received. Authentication ensures that the sender of a message is correctly identified. With **ID authentication**, the receiver is able to verify an unique ID of the sender. The ID could be the license plate or chassis number of the vehicle. Yet, in many cases, the actual identity of nodes does not play an important role – receivers are satisfied if they are able to verify that the sender has a certain property. Hence, **property authentication** is a security requirement that allows verifying properties of the sender, e.g. that the sender is a car, a traffic sign etc. For applications using location informations, **location authentication** allows to verify that the sender is actually at the claimed position, or that the message location claim is valid.

- **Integrity** Applications requiring **integrity** specify that the transported information must not be altered between sender and receiver.

Notice that the requirements on security and privacy play an important role in shaping the solution from the SEVECOM project, which in turn is integrated into the PRECIOUSA public communication API.
8.2. Components in Public Communication API

Many eSafety applications are built upon public communications to function properly, e.g., broadcasting beaconing messages in Collision Warning. Therefore, orthogonal privacy mechanisms are required to ensure the user-specified privacy level for such applications. As already discussed in Section 6.9 of D7, the SEVECOM baseline architecture [24] is adopted to enforce user privacy in public communications.

In the section we summarize the components and modules in the SEVECOM communication framework. A detailed description on the design and implementation of these components and modules can be found in [21, 22].

8.2.1. Components overview

In the context of PeRA, public communications are conceptually performed within the public communication API. In general, components that implement the mechanisms for privacy in public communications are self-contained, i.e., the components function without the dependencies on the rest of the components in PeRA. Figure 8.1 gives an overview of the components within the Public COMM-API. Notice that the direction of the data flow are bidirectional, which means Figure 8.1 presents both the inward and outward Public COMM-APIs in PeRa. Data in communication is accompanied by a privacy policy. The PeRA Exporter configures the security manager in accordance with the privacy policies of the data being sent. Subsequently, the security manager in Figure 8.1 enforces the privacy policy on the data when the data traverse the network stack. In case of inward data flow, the security manager performs necessary processes on the data, e.g., verifies a digital signature in a beacon message, and forwards the data and policy to the PeRA Importer.

In this deliverable, we summarize the components most relevant to privacy enforcement for PeRA, i.e., secure communication, identification & trust management, and privacy management, while leaving out the specifications on In Vehicle Security and Crypto Support modules, because these two modules are very hardware security-centric. However, in actual implementation, hardware security modules could be consolidated with the security module of the Trust Manager in PeRA to reuse the existing functionalities provided by the trust manager in PeRA.

Table 8.2 lists the components for pseudonymous public communications in Public COMM-API, which correspond to the three modules highlighted in Figure 8.1. The next section will specify these components in detail.

8.2.2. Security manager

The Security Manager is responsible for overall system organization, instantiation and configuration of components, hooking of the security subsystem into the communication stack, and dispatching of (some) calls between components.
8.2.3. Identification Manager

This component provides the means to uniquely identify communicating entities in vehicular networks, that is, vehicles or road side units at the wireless part of the network and trusted third parties at the wire-line part of the system. This component describes the details of public key creation and management.
We assume there is an asymmetric crypto-system. For our purpose, the key pairs must be suitable for creation and verification of signatures. Furthermore, we assume a suitable administrative process that will initialize identifiers for new vehicles or on-board units. During this process, names need to be assigned to entities and corresponding key pairs must be generated and installed in the vehicle.

```plaintext
PublicKey init_device(Identifier ID)
```

We denote the identifier of an entity \( X \) by \( ID_X \). The long-term identity is represented by \( ID_X \) and is associated with a name of \( X \), a cryptographic key pair \((SK_X, PK_X)\), and a set of attributes of the entity.

The exact format of this unique long-term identity \( ID_X \) is not specified here, as it will be the outcome of an agreement between car manufacturers and authorities, similar to the use of Vehicle Identification Numbers (VINs). Such identifiers of the same format will be assigned both to vehicles (own or of other's) and road-side units.

Each identifier (and thus entity) is bound to an asymmetric key pair \((SK_X, PK_X)\). A variety of asymmetric (public-key) cryptosystems is available; we recommend the use of Elliptic Curve cryptography (e.g. based on the IEEE P1363 standard). The exact form of implementation will also be influenced by an agreement of which parts of the standard (if not all) will be supported, and the characteristics of the Hardware Security Module (HSM).

The private key \( SK_X \) is generated by and stored in the HSM. Since the HSM is assumed to have limited storage capacity. Therefore, the public key \( PK_X \) is not stored in the HSM, but in the OBU.

### 8.2.4. Trust Manager

This component describes the trusted third party (TTP) needed to provide the certificates that are necessary for the identification component described in Identification Manager. We denote here the TTP, referred to interchangeably as a Certification Authority or CA, by \( T \). Notice that it should not be confused with the Trust Manager of PeRA.

This component relies on the data structures defined in the Identification component. It further assumes the provision of a revocation component. Finally, for each certificate validity period \( E \) the availability of reliable communication channel between system entities and the TTP \( T \) is assumed for the time needed to perform the certificate update protocol.

The TTP can be operated by local authorities, for example, at a region, state, or country level; or it can be operated by an international organization. \( T \) has a policy that determines that any newly manufactured car equipped with an onboard unit or roadside unit, or any onboard that is manufactured for retrofitting, will receive one certificate.

The trusted third party, \( T \), creates a certificate of the form:
Deliverable 10 1.0

\[ \text{cert}_{PK_X} = (ID_X, PK_X, A[], E, T, \text{sig}) \]

where \( ID_X \) and \( PK_X \) are the identity and public key of the entity \( X \), \( T \) is the identity of the TTP, \( A[] \) is an attribute list by which \( T \) declares its trust in certain attributes of \( ID_X \) (and thus \( X \)), and \( E \) is the validity period of the certificate, determined by two time-stamps, a start and an end time. The generation and binding of all this material by the TTP is achieved by the \( \text{sig} \) field, a signature calculated by the TTP \( T \) over the rest of the certificate data, as follows:

\[ \text{sig} = \text{sign}_{SK_T}(ID_X, PK_X, A[], E, T) \]

Each device \( ID_X \) is initialized with a certificate using a trustworthy communication link to \( T \) (which is assumed to be available during manufacturing). To obtain credentials (certificate) and cryptographic material at a later point in time, a two-party protocol with \( T \) is used.

At a point in time \( \tau \) time units before the expiration of its certificate, \( X \)’s HSM generates a new key pair \((SK'_X, PK'_X)\). It then generates a Certificate Request (CR) for \( T \), providing the identity of the TTP, the entity’s identity \( ID_X \), its new public key \( PK'_X \), and \( T \)-specific information in a string \( \text{info} \); we denote this as \( CR_{data} = (T, ID_X, PK'_X, \text{info}) \).

Then, \( ID_X \) signs the \( CR_{data} \), where the request is uniquely identified by the \( \text{info} \) string, which includes a nonce, i.e., a not-previously-used identifier with respect to \( T \) and \( ID_X \), and the new validity period \( E' \). The request sent

\[ CR = (CR_{data}, \text{sign}_{SK_X}(CR_{data})) \]

If the request is authenticated and the \( ID_X \) has not obtained yet a certificate for the period \( E' \), \( T \) generates a new certificate:

\[ \text{cert}_{PK'_X} = (ID_X, PK'_X, A[], E', T, \text{sig}) \]

with its fields defined above. \( T \) sends this certificate to \( ID_X \) as a Certificate Response (CRS). The protocol concludes with the transmission by \( ID_X \) of a Certificate Acknowledgement (CAck):

\[ CAck = (T, ID_X, \text{info}, \text{sign}_{SK'_X}(T, ID_X, \text{info})) \]

Upon reception of the CAck, \( T \) considers the installation of the new certificate successful. Note that the duration of the \( E' \) is a system parameter, depending on the \( T \), and unless the value provided in CR is compliant with the system operation, a new certificate will not be provided. \(^1\)

\(^1\)Note that in the case of retrofitting, that is, installation of an OBU in used cars, an off-line identification process is necessary to ascertain the correctness of used attributes (e.g., physical or other attributes).
The CA provides a remotely accessible interface for certificate renewal, performed as defined above. Using this interface, vehicles that reach the end of their certificate lifetime can generate a new asymmetric key pair, send the new public key to the CA, authenticated by the currently available and valid cryptographic key, and receive a new certificate.

The presence of the HSM ensures that the vehicle will not utilize the newly acquired certificate and the corresponding private key during the validity period $E$ of its current key and certificate. This is so, as $E$ and $E'$ are partially and to a small extent overlapping, exactly to enable the renewal.

The CA certificate is always valid, in the sense that the CA itself ensures the distribution of a new certificate when necessary. The CA certificate format follows that of the certificate formats for the entities:

$$\text{cert}_{PK_T} = (T, PK_T, A_{CA}[], E_{CA}, I, \text{sig})$$

where $T$ is the unique identity of the CA or TTP we denote as $T^2$, $PK_T$ is its public key, $E_{CA}$ is the validity period of this certificate, $A_{CA}[]$ is a list of attributes for $T$ (such as the geographic area it covers), and $I$ the identifier of the certificate issuer. $E_{CA}$ is significantly larger than any $E$ for the validity of entity certificates. The issuer of such certificates for a CA is either a hierarchically superior authority $R$, or in the case of several independent authorities operating in the absence of an $R$, $I = T$. In that latter case, to ensure interoperability and enable secure communication between vehicles registered with distinct authorities (trusted third parties), each trusted third party $S$ can generate a certificate for each trusted third party $T \neq S$ if CA policies are compatible. This is essentially the concept of cross-certification. In that case, for example, $I = S$ above, and

$$\text{sig} = \text{sign}_{SK_S}(T, PK_T, A_{CA}[], E_{CA}, S)$$

### 8.2.5. Pseudonym Manager

The Pseudonym Management is responsible for administration of the on-board pseudonym pool. The functionalities of this component include the initiation of pseudonym generation, adding and deleting pseudonyms in the pool, monitoring the pseudonym usage status, and the configuration of the pseudonym refill policy.

Besides, this component also defines the structure of pseudonyms, which are short-term certified public keys that do not provide additional identifying information. They can, however, include a list of vehicle attributes that need to be attested by a TTP. These attributes should be sufficiently generic so that an identification of an individual vehicle is not possible based on the attribute list.

---

2 For simplicity, we avoid here the distinction between $T$ and $ID_T$. 

30.03.2010 IST-224201
This component requires each vehicle to have obtained a unique long-term identifier $ID_X$ which bounds to a corresponding certificate $cert_{PK_X}$ provided by the Identification & Trust Management Module.

It is prerequisite that there is a PP in the Public Key Infrastructure (PKI), who can issue certified public keys to individual vehicles (i.e., issue pseudonyms to legitimate vehicles), and such pseudonyms are trusted by all nodes in the network.

Additionally, it is assumed that there exists a Hardware Security Module (HSM) where secret keys from pseudonyms can be stored securely and that will do all secret key operations.

In the following, we describe the function of the Pseudonym Manager.

**Pseudonym**  Pseudonyms are a set of distinct certified public keys that do not provide additional identifying information.

Instead of using a long-term identifier for signing messages, each vehicle is equipped with a set of short-term identifiers (e.g., $\{psnym_{X1}, \ldots, psnym_{Xk}\}$) that consist of a key pair and corresponding certificates. Those pseudonyms are similar to the long term identifiers with the exception that they do not include any information which identifies an individual.

The pseudonym key pair of vehicle $X$ ($PSNYM - SK_{XI}, PSNYM - PK_{XI}$) is a key pair. A pseudonym certificate for this key pair has the following format:

$$psnym - cert_{XI} = (PSNYM - PK_{XI}, A[], E, PP, sig)$$

where $A[]$ is the attribute list by which $PP$ declares its trust in certain attributes of $X$, $E$ is the validity period of the certificate, determined by two time stamps, a start and an end time. $PP$ denotes the pseudonym provider which created this pseudonym certificate and $sig$ is a signature over the certificate data calculated using $PP$s private key:

$$sig = sign(SK_{PP}, PSNYM - PK_{XI}, A[], E, PP)$$

The pseudonym provider $PP$ is an authority issuing pseudonym certificates similar to the CA.

When using pseudonyms, a safety-related message from a vehicle $X$ will roughly have the following format:

$$MSG = data | sig | PSNYM$$

Details of message formats are given in the respective components described in the components of the Secure Communication Module.
Pseudonym generation The generation of new pseudonyms basically involves two steps:

1. **Generation of key pairs:** the HSM module generates a set of new key pairs 
   \[ (PSNYM − SK_{Xi}, PSNYM − PK_{Xi}) \forall i = 1 \ldots n. \]

2. **Retrieval of certificates:** the OBU contacts the pseudonym provider, sends all the pseudonym public keys plus authentication information, and in turn receives one certificate per key pair.

The communication with the PP has to be done via an authenticated and confidential communication link. The exact details of the pseudonym provider protocol where a PP communicates with vehicle X are as follows:

\[
\begin{align*}
X \to PP & : \text{PSNYM} – PK_{X1}, \text{PSNYM} – PK_{X2}, \ldots, \text{PSNYM} – PK_{Xn} \\
PP \to X & : N \\
X \to PP & : \text{sign}(\text{PSNYM} – SK_{X1}, N), \text{sign}(\text{PSNYM} – SK_{X2}, N), \ldots \\
PP \to X & : \text{psnym} – \text{cert}_{X1}, \text{psnym} – \text{cert}_{X2}, \ldots
\end{align*}
\]

Pseudonym certificates are only issued, if vehicle X was previously able to authenticate to PP using ID_X and if X can prove knowledge of the corresponding secret keys PSNYM – SK_{Xi}, N by signing a nonce N.

Pseudonym Storage The pseudonym certificates are stored in the OBU, the secret keys are stored in the hardware security module. Certificates are transmitted together with packets sent and contain only public information. Therefore, they do not need to be protected. Knowledge of secret keys of other vehicles’ pseudonyms, however, would allow impersonation of those other vehicles and need to be stored in a protected way.

The storage space allocated for pseudonyms in the OBU depends on the available space in the OBU and the frequency of pseudonym usage (i.e., how often a vehicle needs to sign the messages with the same private key and the corresponding pseudonym).

Pseudonym Refill Frequent change of pseudonyms used in communication ensures the privacy of vehicles. The Pseudonym Management should constantly monitor the number of valid pseudonyms in the storage, how often/long they have been used, and when they will expire. When the number of available pseudonyms falls below a certain threshold, the Pseudonym Management will restart the pseudonym generation procedure and obtain a new set of pseudonyms. This is called ‘Pseudonym refill’.

Pseudonym Resolution When issuing pseudonym certificates, the pseudonym provider PP stores the mapping between ID_X and all PSNYM – PK_{Xi} at a pseudonym resolution authority PRA. The mapping is stored for the lifetime of the pseudonym plus some extra time. Based on certain legal conditions, public bodies, for example, vested with the power of law enforcement, can contact the PRA and request a resolution for any

30.03.2010 IST-224201
The Pseudonym Application provides pseudonyms which are used in secure communications. Beside providing valid pseudonyms, this component decides how long a pseudonym is allowed to be used in the communications and when to change to another pseudonym, according to the privacy policy defined either by the user or the VC system. It specifies a framework, on which a vehicle’s decision of pseudonym changes are based. The component is also responsible for coordination of changes of identifiers in other layers in the communication stack.

This component assumes that a vehicle $X$ has a set of valid pseudonyms available. Details about format and creation of pseudonyms is given in the Pseudonym Management component. It is also assumed that the lower layer protocols, e.g., at the data link layer, can respond to the changing address command and update their addresses accordingly if possible.

The Pseudonym Application component provides a method which returns the key id of the pseudonym that is currently to be used. This key id is then to be given as an argument in calls to the HSM requesting e.g. signing of data. Based on this information, the HSM can then select the proper key material. The signature of this method is:

$$PSYNM, ID_{key} = \text{PseudonymApplication.current();}$$

For privacy reasons, each pseudonym will only be used in the communication for a short period of time and then discarded. The time-lapse since a pseudonym is used is denoted as $\tau$. We suggest that $\tau$ is in the range of seconds to a few minutes.

After providing the pseudonym, Pseudonym Application sends a `change_address((<layer>,<address>),...)` command to the communication stack instructing it to change the addresses on the respective layers. The communication stack will reset the current used identifiers as soon as all messages currently queued in the stack have been sent.

The decision to switch to a new pseudonym is based on how long the current pseudonym has already been used ($t_{used}$). This time is calculated as the current time $t_{current}$ minus the time of the last pseudonym change $t_{last}$, i.e. $t_{used} = t_{current} - t_{last}$. The Pseudonym Application component compares $t_{used}$ with the pseudonym changing interval that consists of a fixed time interval $\tau$ randomized by a random variable $\delta$, and switches to a new pseudonym, when the following condition is satisfied:

$$t_{used} \geq \tau + \delta$$

\[ i.e., \text{having already left the Secure Communication Module but having yet to be send by the communication stack} \]
where $\delta$ is in a configurable interval $[-\epsilon, +\epsilon]$. This randomization is necessary as otherwise pseudonym changes could easily be tracked based on the pseudonym change timing observed by an attacker.

### 8.2.7. Secure Communication Module

The Secure Communication Module in the SeVeCom framework aims to secure vehicle-oriented communications (i.e., vehicle-to-vehicle and vehicle-to-infrastructure communications). This module consists of the Secure Beaconing component, the Secure Flooding component, and the Secure Routing component, which provide security to different communication patterns for vehicular communications.

Although each component of the Secure Communication module has specific operations on the messages traversing the communication stacks, with respect to privacy-preserving public communications, we can abstract away the detailed security operations such that a message has a generalized format as

\[
\text{Header} \mid \text{Payload} \mid \text{Sig}(SK_i, \text{Header} \mid \text{Payload}) \mid \text{Cert}(PK_i)
\]

The secure communication module relies on the existence of an established Identity and Trust management, which provide credentials (i.e., certified public key $\text{Cert}(PK)$) to fulfill the security goals. However, if pseudonyms are in use, the way of signing and verifying messages does not change – the Privacy Management Modules have to take care which certificate and their corresponding keys to use.

### Hooking

The components in the Secure Communication Module provide an interface which allows them to be hooked into the process of communication. Therefore, a component is notified when packets are to be sent, received, or forwarded.

In most cases, a total control over the processing of such packets is required. For example, a component in the Secure Communication Module may decide whether it is secure to accept a packet or if it has been manipulated and should be discarded. This can be done either by this component itself, or by the network layer implementation. The interface design here assumes that the network layer is able to process certain return codes and has to drop a packet accordingly, if the secure components indicate so.

Because we distinguish between packets being received, forwarded and sent, we assume that the hooks can be inserted at defined points of the network stack. This could be similar to the Linux netfilter architecture, as depicted in Figure 8.1.
8.2.8. Hardware Security Module and Crypto Support

The general purpose of the Hardware Security Module (HSM) and Crypto Support is to provide the implementation of the cryptographic operations needed by other modules and components in the SEVECOM communication framework. These cryptographic operations can be called through the APIs provided by the components of the Crypto Support Module. The basic functionalities provided by the HSM include public key cryptographic operations, such as digital signature generation and verification. The private keys are stored in an independent tamper-resistant device.

From an implementation point of view, HSM and Crypto Support can be implemented in the public communication API as an embedded component. Alternatively, we could also utilize the existing implementation of the Trust Manager in PeRa in the actual implementation, if such a tight coupling of PeRA and the SEVECOM communication framework is desired.

8.3. Enhancing Conditional Pseudonymity

Section 8.2.5 describes the pseudonym manager and related functionality, as defined in the SeVeCom baseline architecture [24]. The pseudonym manager interacts with a pseudonym provider to refill pseudonyms. The pseudonym provider in turn stores identity-pseudonym mappings to provide accountability in case of misuse. While this kind of conditional pseudonymity, i.e., the usage of resolvable pseudonyms, is a common approach, the resulting pseudonym-identity mappings are privacy sensitive. Thus, strong protection is required to prevent abuse or leakage. In [25] we propose a new approach for conditional pseudonymity that does not rely on pseudonym-identity mappings to be stored by any party. Instead, resolution information is embedded in pseudonyms and can only be accessed when multiple authorities cooperate. The privacy-preserving pseudonym issuance protocol ensures that pseudonyms contain valid resolution information but prevents issuing authorities from creating pseudonym-identity mappings.

In the following, we outline the $\mathcal{V}$-token approach as a privacy enhancement for the pseudonym management lifecycle described in Section 8.2.5. As mentioned before, when pseudonyms are used, accountability may be desired to evict misbehaving nodes or assign liability after fatal accidents. So some authorities must be able to resolve pseudonyms to vehicle identities in certain situations. Conditional pseudonymity provides privacy for vehicles and also accountability. However, vehicles have to trust pseudonym issuing authorities to store and manage resolution information securely and responsibly. If resolution information leaks or becomes openly available, the privacy protection provided by pseudonyms is undermined. In this context, we advocate [25] reconsidering the common assumption that authorities can be fully trusted with managing information that could render privacy mechanisms ineffective. Instead, authorities should practice minimum disclosure and separation of concerns, characteristics also reflected in the principles for hippocratic cITS defined in Deliverable D7 [1]. Only entities responsible for identity resolution...
Figure 8.2.: Enhanced pseudonym issuance split into authentication and acquisition phase (1+2), resulting in a pseudonym for inter-vehicular communication (3).

should be able to access resolution information while other entities, like pseudonym issuance authorities, should neither store nor have access to it.

The $\mathcal{V}$-token approach achieves accountability without requiring authorities to store resolution information and prevents them from keeping it. As a result, drivers have to place less trust in authorities. The scheme also benefits authorities and pseudonym providers by helping them comply with privacy regulations and reducing the amount of sensitive information to be managed. Further, we enforce the cooperation of several authorities for pseudonym-identity resolution to ensure multiple parties agreeing on necessity for resolution. The resolution protocol also provides perfect forward privacy [26], i.e., only linking information for the current pseudonym is made available while other pseudonyms and messages of that user remain unlinkable.

The $\mathcal{V}$-token approach [25] is based on the idea of embedding resolution information directly in pseudonym certificates rather than having authorities store them. The vehicle identifier $id_V$, the identifier of the vehicle’s registration authority $id_{CA_1}$, and a unique randomization factor $r$ are encrypted with $PK_{RA}$, the commonly known public key of the resolution authorities. Resulting ciphertexts, we call them $\mathcal{V}$-tokens, are unlinkable.

Pseudonyms with embedded $\mathcal{V}$-tokens are issued in a two phase protocol, which ensures that $\mathcal{V}$-token content is valid but prevents issuing authorities from linking pseudonyms to vehicles (see Section 8.3.1). Vehicle $V$ uses the resulting pseudonym $P_i$ for normal message authentication by signing messages with $SK_{P_i}$ and attaching $P_i$ to the message. Figure 8.2 depicts this process. Receivers verify $P_i$ and the signature. Thus, embedding the $\mathcal{V}$-token in $P_i$ does not affect how $P_i$ is used in communications.

If required, pseudonym-identity resolution is performed collaboratively by a minimum number of authorities. They need to jointly decrypt the $\mathcal{V}$-token embedded in a $P_i$ to retrieve the linking information. Section 8.3.2 details the resolution protocol. Fischer et al [27] also propose a scheme that enforces collaborative identity resolution. However, resolution authorities need to participate in pseudonym issuance which is not desirable. Ideally, we
Deliverable 10

would prefer a strict separation of concerns so that each entity in our system model has a clear task and can be implemented independently of other functionality.

In the following, vehicle \( V \) is identifiable by a unique long-term identifier \( \text{id}_V \), e.g., an identity certificate and the corresponding key pair. \( V \) is registered with an authority \( CA_h \), its home CA identified by \( \text{id}_{CA_h} \). \( CA_h \) manages \( V \)’s virtual identity and issued \( \text{id}_V \). In practice, a regional vehicle registration authority could take on this role, thus consolidating authority over \( V \)’s virtual identity and physical license plates.

\( V \) can obtain pseudonyms \( P_i \) from pseudonym providers \( PP_k \). Pseudonym providers are independent from \( CA_h \) so that \( V \) can engage with arbitrary \( PP_k \). Before new \( P_i \) are issued, \( V \) is authenticated and it is verified that \( V \) has not been revoked. A pseudonym \( P_i \) is a public key certificate for a key pair \((PK_{P_i}, SK_{P_i})\), containing no information linking \( P_i \) to \( V \) or any \( P_j \) \((j \neq i)\). When communicating, \( V \) signs messages with secret key \( SK_{P_i} \) of the current pseudonym \( P_i \). The signature and \( P_i \) are attached to the message for verification by receivers.

The resolution authorities \( RA_i \) take part in pseudonym-identity resolution. A subset of them has to cooperate in the process. \( RA_i \) should be independent from authorities involved in the issuance of a pseudonym.

### 8.3.1. Privacy-preserving Pseudonym Issuance

The privacy-preserving issuance protocol employs a blind signature scheme to prevent issuing authorities from learning linking information. In the **authentication phase**, vehicle \( V \) first obtains blindly signed \( \mathcal{V} \)-tokens from \( CA_h \). Subsequently, \( \mathcal{V} \)-tokens are used in the **acquisition phase** to obtain pseudonyms from a pseudonym provider \( PP \).

#### Authentication phase

\( V \) is first authenticated by \( CA_h \) and \( CA_h \) checks that \( V \) has not been revoked. \( CA_h \) then sends \( V \) the identifier string \( \text{id} \) to be included in the \( \mathcal{V} \)-token, the public key \( PK_{RA} \) of the resolution authorities, an expiration date \( \text{exp} \), and a request for \( n \) commitments. \( V \) then generates \( n \) \( \mathcal{V} \)-tokens \( \mathcal{V}_i \) by choosing a unique random \( r_i \) that is appended to \( \text{id} \), before encrypting it with \( PK_{RA} \), \( \text{exp} \) and \( \text{id}_{CA_h} \) are appended to each \( \mathcal{V}_i \). The expiration date limits the lifetime of a \( \mathcal{V} \)-token. \( \text{id}_{CA_h} \) is required for verification purposes in the acquisition phase.

\( V \) then chooses \( n \) random distinct blinding factors \( b_i \), with inverse \( b_i^{-1} \). Each \( \mathcal{V} \)-token is blinded, resulting in \( n \) commitments \( C_i \), which are sent to \( CA_h \). Now, \( V \) is committed to the content encoded in all \( C_i \) in the sense that it cannot manipulate or change the content anymore. Subsequently, \( V \) has to prove probabilistically to \( CA_h \) that the encoded content contains \( \text{id} \) as provided by \( CA_h \) earlier, by revealing \( b_i \) and \( r_i \) of \( h \) commitments \((h \geq n/2)\) randomly chosen by \( CA_h \). If all opened commitments can be verified successfully, \( CA_h \) trust the remaining commitments to be correct as well. The probability that \( V \) managed to
cheat is then negligible, due to the fact that \( V \) does not know beforehand which \( C_i \) will be opened. See [28] for a formal analysis of the security of commitment schemes.

\( CA_h \) signs the remaining \( n - h \) commitments with its secret key \( SK_{CA_h} \), yielding \( n - h \) blind signatures \( \sigma_{CA_h}(C_i) \) which are sent to \( V \). \( V \) can now remove the blinding factor by applying the corresponding unblinding factor due to the homomorphic nature of the encryption scheme. This way, \( V \) obtains \( n - h \) \( \mathcal{V} \)-tokens \( \mathcal{V}_i \) with valid signature from \( CA_h \), without \( CA_h \) learning which \( \mathcal{V}_i \) it signed.

**Acquisition phase**

Once in possession of valid \( \mathcal{V} \)-tokens, \( V \) can use them as anonymous credentials to obtain pseudonyms from a pseudonym provider \( PP \). In the acquisition phase, it is essential that \( V \) only communicates anonymously with \( PP \) to ensure unlinkability between \( V \) and the resulting pseudonyms. Either \( V \) uses a previously issued pseudonym to communicate anonymously or an anonymization mechanism like onion routing can be used. A signed \( \mathcal{V} \)-token is a credential which certifies that its holder has been successfully authenticated by \( CA_h \). Each \( \mathcal{V} \)-token authorizes the holder to obtain one pseudonym certificate. To obtain a pseudonym, \( V \) generates a public key pair \( (PK_P, SK_P) \), and presents the public key together with a signed \( \mathcal{V} \)-token to \( PP \) in a request encrypted with \( PK_{PP} \).

\( PP \) decrypts the request and verifies the signature on the \( \mathcal{V} \)-token. If valid, \( PP \) proceeds by checking that \( \mathcal{V}_i \) has not expired and has not been used before to prevent double spending. For this purpose, pseudonym providers can cocollectively operate a distributed \( \mathcal{V} \)-token clearing house \( CH \) in which hash values of used \( \mathcal{V} \)-tokens are stored. If all checks succeed, \( PP \) includes the plain \( \mathcal{V} \)-token \( \mathcal{V}_i \) (without \( \sigma_{CA_h}, \text{exp}, \text{id}_{CA_h} \)) in a pseudonym certificate \( P_i \) for \( PK_P \). \( P_i \) also contains an expiration date \( \text{exp}_{PP} \) and \( \text{id}_{PP} \). Then, \( PP \) sends the resulting pseudonym \( P_i \) back to \( V \). \( V \) stores \( P_i \) together with \( SK_P \), and can delete the used \( \mathcal{V} \)-token \( \mathcal{V}_i \). Now, \( V \) can use \( P_i \) for message authentication in inter-vehicular communication.

The protocol effectively preserves \( V \)'s privacy against \( CA_h \) and \( PP \). The issued pseudonym \( P_i \) cannot be linked to \( V \) by any of these authorities, not even if \( CA_h \) and \( PP \) would collude. See [25] for a formal analysis of the presented protocol.

**8.3.2. Collaborative Identity Resolution**

While identity resolution is part of conditional pseudonymity to prevent misuse and abuse of a system, it also exposes users to potential privacy infringement. Therefore, the information required for identity resolution needs to be protected, so that it is only available to a restrictive number of authorities in very specific situations. Separation of duties is a common principle to prevent intentional or unintentional misuse of information or processes. We apply separation of duties to the protection of identity resolution information. For this purpose, we distribute the ability to perform identity resolution between a number...
of authorities and enforce their collaboration to perform identity resolution with a threshold encryption scheme.

The secret key of the resolution authorities $SK_{RA}$ is split among $n$ resolution authorities, so that each holds only a share of $SK_{RA}$. Cooperation of a subset of $k$ of $n$ RAs is required to decrypt a $\mathcal{V}$-token, which has been encrypted with $PK_{RA}$. It is important to ensure that neither $SK_{RA}$ nor individual shares of it are revealed in the process of collaborative $\mathcal{V}$-token decryption. Secret sharing homomorphisms proposed by Benaloh [29] enable entities to jointly recover a secret without revealing their individual shares in the process. The concept can be combined with homomorphic encryption schemes (e.g., ElGamal [30]) to enable homomorphic threshold decryption: A ciphertext $E_{PK}(m)$ is passed between a number of entities, with each entity applying its share of the secret key to it. Once the threshold is reached, i.e., the $k$-th entity applied its secret share, $E_{PK}(m)$ is fully decrypted and the plaintext $m$ is revealed. With this approach, no party learns the global secret or secret shares of any other entity. Additionally, the order in which entities apply their secret share is irrelevant as long as the $k$-th entity is the one that should learn $m$.

In our case, the initial input for the collaborative identity resolution protocol is a pseudonym certificate $P_i$ containing a $\mathcal{V}$-token $\mathcal{V}_i$. For protocol description, we assume a threshold $k = 3$ and three resolution authorities: a law enforcement agency $L$, a judge or juridical institution $J$, and a data protection agency $DP$. $L$ wants to identify the sender of a message with pseudonym $P_i$, $J$ decides if evidence provided by $L$ is sufficient to justify identity resolution, and $DP$ surveys privacy breaches.

$L$ should be convinced that resolution of $P_i$ is justified and has to provide corresponding evidence to the other authorities involved. $L$ presents the $\mathcal{V}$-token $\mathcal{V}_i$ and the evidence in turns to $J$ and $DP$. Both assess the evidence and decrypt $\mathcal{V}_i$ with their share of $SK_{RA}$ if they approve identity resolution. Note that as long as $\mathcal{V}_i$ has been decrypted by less than $k - 1$ RAs, no information about the plaintext is revealed.

Eventually, $L$ can apply its own share of $SK_{RA}$. The threshold $k = 3$ is reached and the $\mathcal{V}$-token is successfully decrypted, yielding the identifier $id$. Note, that only $L$ learns the linking information $id$ because it applies its secret share last.

With the obtained information, $L$ can contact $CA_h$ and request further information about vehicle $V$. $CA_h$ looks up $id_V$ in its database and returns information about $V$ to $L$. If required, $CA_h$ can revoke $V$’s long-term identity to prevent $V$ from obtaining new $\mathcal{V}$-tokens and pseudonyms.

$L$ has successfully linked pseudonym $P_i$ to vehicle $V$ and has sufficient information to hold $V$ accountable. The protocol provides a straightforward approach for identity resolution with enforced distribution of resolution authority. It is also extensible and flexible. In [25], different ways of representing more complex hierarchies of authorities are discussed, which can be instantiated to reflect the external and internal organizational structure of RAs and how secret shares are distributed and divided further internally.
9. Conclusion

This document provides a comprehensive specification of the PRECIOISA Privacy enforcing Runtime Architecture (PeRA). The PeRA is build around the concept of privacy policy enforcement, which is realized in a 3-tier approach, consisting of privacy policies, mandatory privacy control, and MPC integrity protection. Data subjects specify privacy policies for their personal information and data. Components of the mandatory privacy control layer, namely the PCM, ensure policy compliant data processing. Below that, the MPC integrity protection components ensure that MPC components can only access data when they are in a correct state, thus preventing modified MPC components from bypassing policy compliance. The Importer and Exporter as confidential communication endpoints provide secure coupling of data and metadata. The communication privacy system provides pseudonymization and privacy protection on the communication level, both, for confidential and public communication.

The PRECIOSA approach towards privacy in cITS rests on three pillars, which have been described conceptually in deliverable D7 [1]:

1. The principles for hippocratic cITS (HcITS) set the requirements and provide ideas for privacy enhancement.
2. The privacy-aware design process for cITS applications advocates a privacy by design approach for application design.
3. The privacy-enforcable runtime architecture (PeRA) is a framework that implements the HcITS principles, which can be considered as a deployment platform in the privacy aware application design process.

In the following, we analyze how and to what extend the principles for hippocratic cITS are realized by the PeRA.

9.1. Compliance to the principles for hippocratic cITS

We defined ten HcITS principles in deliverable D7 [1], and will discuss for each principle how it is supported by the PeRA.
1. Purpose specification For all personal information that is communicated between or stored by participating components of an ITS system the purposes for which the information has been collected shall be associated with that information.

In the PeRA, several entities specify purposes. The data subject can specify purposes for which created data can be used. The purpose is part of the metadata that is securely coupled to any data item. Controlled and uncontrolled applications that wish to access data have to state the purpose for which they want to access the data and their role together with their query request.

2. Consent. The donor of the information must provide consent for the usage of information for a specified purpose. This consent might be restricted to one specific purpose or to a set of purposes; the data subject should also have the right and ability to revoke his consent for the future.

A data subject gives explicit consent by specifying a privacy policy that governs the use of her personal information. Data may only be used in compliance with these policies; usage purposes not specified by the policy should be denied by default. While not part of the PeRA specification, a controlled application on the data controller’s platform can be envisioned, that enables users to adjust and revoke their consent for specific purposes by updating a given policy. See the description of user surrogate application pattern in Section 4.3 on how this could be supported.

3. Limited Collection. Information related to the data subject shall be limited for communication, storage, and collection to the minimum necessary for accomplishing the specified purposes.

Practicing limited collection needs to be part of privacy-aware application design and cannot be provided by the PeRA per se. Nevertheless, the PeRA supports limited collection by ensuring that personal information is communicated confidentially and stored securely under supervision of trusted PeRA components.

4. Limited Use. The cITS shall execute only those operations on the personal information that are consistent with the purposes for which the information was collected, stored, and communicated.

The purpose of mandatory privacy control (see Chapter 4) is to enforce limited use by assuring policy compliance.

5. Limited Disclosure. The personal information related to the data subject and operated on by the cITS shall not be communicated outside the cITS for purposes other than those for which the data subject gave his/her consent.

Limited disclosure is closely related to limited use. Disclosure occurs when data leaves the policy enforcement perimeter. For example, when an uncontrolled application requests
information the result is disclosed and from then on not under control of any PeRA components anymore. The PCM ensures that data only leaves the PeRA if the associated policies allow that and if all post processing requirements specified in policies are fulfilled (e.g., a certain degree of anonymity). Thus, disclosure is always only performed in accordance with privacy policies.

6. Limited Retention. *Personal Information related to the data subject shall be retained by the cITS only as long as necessary.*

Retention periods are specified in privacy policies. Once data is imported into a local PeRA instance, corresponding privacy maintenance events are generated which are queued by the Retention Manager. The Retention Manager (see Section 7.2) ensures that data is removed once the retention period is over.

7. Accuracy and Context Preservation. *Personal Information related to the data subject and stored by the cITS shall always be accurate, up-to-date, and never be decoupled from its context and purpose.*

The PeRA preserves the accuracy and context of personal information by always storing coupled metadata and privacy policies together with data. In transit, data and metadata are securely coupled.

8. Security. *Personal information related to the data subject shall be protected by appropriate security measures against unauthorized use or use in conflict with the consent of the data subject.*

The MPC Integrity Protection components, namely the Trust Manager, ensure that stored or received data can only be accessed if all relevant components of the local PeRA instance are in a trusted and validated state. Data is stored in encrypted databases and communicated confidentially. Required key material to gain access to data is stored inside a Hardware Security Module which can only be accessed and controlled by the Trust Manager.

9. Openness. *A data subject shall be able to access all information that is stored in the cITS and is related to the data subject.*

As mentioned in the discussion of the second principle, a user surrogate application could provide a data subject with access to all data items stored about that data subject by another PeRA instance.
10. Compliance. A data subject shall — directly or indirectly — be able to verify the compliance of the cITS with the above principles.

In practice, a data subject will not be able to verify compliance of all entities it communicates with by herself. Therefore, PeRA instances are certified by a privacy authority, which is trusted by the data subjects as well as the data controllers. The privacy authority verifies that all components of a specific PeRA instance are in proper and trusted states, and that key material for data decryption can only be accessed in that state. See Section 5.1 on a discussion of the role of the privacy authority.

9.2. Outlook

The PeRA is a potent system realizing the HcITS principles. Application designers can rely on the functionality provided by the PeRA to enforce compliance to privacy policies set by data subjects and these principles in a wider sense. Furthermore, the PeRA provides a comprehensive solution for privacy in cooperative ITS, which provides system privacy rather than focusing on a single domain. The PeRA has been designed to be deployable in the vehicle domain, access domain, and backend domain, alike.

The next step in the PRECISOA project is the integration of the Privacy-enforcing Runtime Architecture with specific ITS applications to demonstrate the feasibility of the PeRA approach in actual use cases. Deliverable D14 will report on the integration process and results. In addition, D13 will devise principles, mechanisms, and support tools to verify the privacy level.
A. Sequence Diagrams

Figure A.1.: A sensor provides data to the PERA

Figure A.2.: A controlled application requests data from the DBMS
Figure A.3.: A controlled application exports data to an uncontrolled application

Figure A.4.: A controlled application requests data from a sensor
Figure A.5.: An uncontrolled application requests data from the DBMS

Figure A.6.: An uncontrolled application invokes a method of a controlled application
Bibliography


