Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Preserving Privacy in Software Defined Car Environments

Yunxuan Li

Course of Study:	Informatik
Examiner:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Supervisor:	Dr. rer. nat. Pascal Hirmer
Commenced:	October 1, 2021
Completed:	April 1, 2022

Abstract

Modern vehicles are becoming more and more intelligent. With new sensors and software that are available for Connected Vehicles (CVs), they are capable of collecting, processing, and sharing data with various participants in connected car environments. Although they bring us lots of convenience and connectivity, they also introduce new threats, such as security, reliability, and privacy. In this thesis, we focus on the privacy aspect and analyze the privacy requirements of connected car environments. To ensure users' privacy, we propose the Privacy for Connected Vehicle Framework. In general, our framework behaves as an access control system between source data generated in CVs and various end-point services. To protect privacy, our framework only shares data that are perturbed based on users' privacy requirements. To ensure maximum service quality, our framework does not interfere any business logic of end-point services. In addition, our framework can provide protection for both situational privacy patterns and individual privacy demands. Moreover, our framework always puts users' safety before privacy and can be deployed in both edge environments and fog environments.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

Acronyms

ABAC Attribute-Based Access Control. 27

ADAS Advanced Driver-Assistance Systems. 17

app application. 19

CV Connected Vehicle. 3

DRL Drools Rule Language. 47

DSRC Dedicated Short Range Communication. 20

IoT Internet of Things. 17

IoV Internet of Vehicles. 17

ITS Intelligent Transportation System. 17

JSON JavaScript Object Notation. 31

Privacy4CV Framework Privacy for Connected Vehicle Framework. 3, 17

PT Privacy Technique. 18

RSU Roadside Unit. 17

SLA Service Level Agreement. 27

SofDCar Software-Defined Car. 19

STP Situational Policy. 33

TLP Trust-Level Policy. 33

V2C Vehicle-to-Cloud Communication. 19

V2I Vehicle-to-Infrastructure Communication. 19

V2N Vehicle-to-Network Communication. 19

V2P Vehicle-to-Pedestrian Communication. 19

V2S Vehicle-to-Sensors Communication. 19

V2V Vehicle-to-Vehicle Communication. 19

V2X Vehicle-to-Everything Communication. 20

1 Introduction

In recent years, the booming development of the Internet of Things (IoT) has also put the Internet of Vehicles (IoV) in the spotlight. As one of the key members of IoT, IoV is enabled by the rapid growth of Connected Vehicles (CVs). According to Placek [Pla21], there will be over 400 million connected cars on the road by the time of 2025. A recent market report from MordorIntelligence [Mor21] also predicts that the share of CVs among newly produced vehicles will rise from 35% in 2015 to almost 100% by 2025.

Connected Vehicles have brought lots of new connectivity to traditional road environments. Nowadays, CVs can be connected to each other, to infrastructures, such as Roadside Units (RSUs), to systems, such as Intelligent Transportation Systems (ITSs), and to the internet and cloud. Furthermore, they are capable of collecting, processing, and sharing data with various participants in connected car environments. These data can be extremely helpful in achieving autonomous driving or creating safer road environments. For example, Advanced Driver-Assistance Systems (ADAS) can take advantages of these data to warn drivers of potentially dangerous situations. Moreover, other participants in connected car environments can also use these data to provide drivers with more personalized and individual services. For example, insurance companies can reward their users for a good driving style based on the data collected and shared from their CVs.

Despite all of its advantages, CVs also bring new issues and challenges. As more electronic components and software systems have been deployed in modern vehicles, security and reliability become major concerns for CVs. When it comes to the connectivity and data sharing that are available in connected car environments, privacy becomes a vital aspect that cannot be ignored. According to McKinsey [McK14], 51% people in Germany and 45% people in US are reluctant to use connected services in vehicles because they want to protect their privacy.

In this thesis, we only focus on the privacy aspect of connected car environments. Both misusage and unwillingly publishing of sensitive data or privacy patterns are privacy concerns in connected car environments. A privacy pattern in this relation is defined as a group of data that is usually measured or gathered from different sensors or sources and requires special protection. For example, drivers may not consider the speed data as sensitive when it is shared alone. However, if the speed data is above the speed limit in a certain location, drivers might not be willing to share the speed data in combination with location data to avoid speeding penalties. Hence, the speed or location data itself may not be considered as sensitive data, but the combination of them under certain circumstances forms a privacy pattern that requires special protection.

Thus, we propose the Privacy for Connected Vehicle Framework (Privacy4CV Framework) in this thesis that aims to provide situation-aware and individual privacy protections for users in connected car environments. Our framework behaves as an access control system between various end-point services and data generated in CVs. Unlike traditional access control system where an access request can be denied, our framework does not reject any services' requests. Instead, it perturbs source data based on users' specific privacy demands before data is shared.

Furthermore, our framework enables users to create individual and situational privacy policies so that both sensitive data and privacy patterns can be protected on demand. In addition, our framework does not interfere with any business logic of end-point services so that maximum service quality can be provided. Moreover, our framework always puts users' safety before privacy. We achieve this by only allowing domain experts to define law and safety related privacy policies. For data perturbations, we use different Privacy Techniques (PTs) in a modular manner. This enables an easy switch of new technologies in the future as well. Our framework is also very generic so that it can be deployed in both edge environments (e.g., on the CV) and fog environments (e.g., on the RSU).

The rest of the thesis is organized as follows: Chapter 2 gives an overview of Connected Vehicle and its environment. In Chapter 3, we analyze the privacy concerns and propose several privacy requirements for connected car environments. Then, we present our Privacy for Connected Vehicle Framework in Chapter 4 and provide a proof-of-concept prototype in Chapter 5. After that, we evaluate our framework and discuss its limitations in Chapter 6. Finally, we review the related works in Chapter 7 and conclude this thesis with a summary and outlook for further works in Chapter 8.

2 Background

More than 30 years ago, the first software component has been deployed in cars. Since then, more and more innovative software components have been developed dedicated for cars. Nowadays, modern vehicles usually have millions of lines of software code running within them [Ala16]. As the concept of Connected Vehicle (CV) or Software-Defined Car (SofDCar) is getting more and more popular these years, the definition of the term "connected-car" has also evolved. According to Coppola and Morisio [CM16], this term was used to referring wireless in-vehicle connectivity of a large amount electronic components. However, in present days, a Connected Vehicle or a Software-Defined Car, according to Alam [Ala16] or Coppola and Morisio [CM16], refers to a vehicle that is capable of accessing the internet at any time, has the ability to interact with other vehicles or smart devices, and is equipped with modern software applications (apps) so that individual services or assistance can be provided at driver's need.

As introduced by Lu et al. in [LCZ+14], there are four types of connectivity in connected car environments:

- Vehicle-to-Sensors Communication (V2S): also known as intra-vehicle connectivity. This kind of communication refers to the wired or wireless data transmission between sensors and components within CVs.
- Vehicle-to-Vehicle Communication (V2V): or inter-vehicle connectivity, refers to the wireless exchange of information between CVs in real-time. The communication range of V2V is usually less than 300 meters. Generally, the information shared through V2V contains speed, location, heading and other relevant data. This data can be used by appropriate software to determine potentially dangerous traffic situations.
- Vehicle-to-Infrastructure Communication (V2I): which is called "vehicle-to-road infrastructure" communication in [LCZ+14], refers to the data exchange between CVs and RSUs. A RSU is usually an intelligent road infrastructure that is capable of collecting and processing data, providing connectivity and information support to passing vehicles.
- Vehicle-to-Network Communication (V2N): named "vehicle-to-internet" communication in [LCZ+14], refers to the internet access of CVs. This kind of communication becomes a fundamental and must-have feature for modern vehicles.

Apart from the four categories of communications described above, several new types of communications within connected car environments are becoming popular these years. For example, Vehicle-to-Pedestrian Communication (V2P), which indicates the data exchange between CVs and smart devices of pedestrians, helps to protect the safety and increase the convenience of vulnerable road users. Vehicle-to-Cloud Communication (V2C) enables CVs to connect to cloud systems so that cloud connected services, such as smart home systems, can be integrated with CVs.

To conclude, all communication means explained above can be classified as Vehicle-to-Everything Communication (V2X), which consists all kinds of communication types between CVs and all possible entities in connected car environments.

Currently, Dedicated Short Range Communication (DSRC) is the most promising standard used in V2V, V2I, and V2P communications. According to Morgan [Mor10], DSRC uses a two-way radio in the 5.850 to 5.925 GHz band range. Its transmission range is limited to a maximum of 1 km. The above transmission spectrum is further divided into several channels. For example, channel 172 is designated for exchanging safety messages between CVs or infrastructures, according to Kenney [Ken11].

As Bai et al. introduced in [BSK10], one important usage of connected car connectivity using DSRC is to build cooperative vehicle safety communications systems where CVs periodically broadcast their current status information together with their sensor information to each other. The receiving CV then runs some appropriate algorithms based on the received data along with its own data to calculate the possibility of the occurrence of dangerous situations.

In addition, the connectivity provided by CVs also benefits the ADAS, where the inputs are no longer limited to data sources within a CV, but also data sources from other CVs, infrastructures or even pedestrians with the help of V2V, V2I and V2P communications. Moreover, the ITS, proposed by Dimitrakopoulos and Demestichas in [DD10], which aims to improve the efficiency and safety of transportation and traffic management systems, relies on V2V and V2I communications as well.

Despite the benefits CVs or SofDCar can bring, Coppola and Morisio also point out the potential issues and challenges in connected car environments. The major concern of CVs is security. As CVs introduce much more connectivity to systems, infrastructures, and internet, they also become more vulnerable against attackers from outside. Then, the reliability is another big challenge. It is inevitable that sensors or hardware may provide incorrect data, systems may malfunction, and communications may fail. It is crucial that CVs can still provide reliable services under such circumstances, as a simple failure of the system in other domains may only result in financial loss, it might cause fatal consequences in connected car environments. Finally, privacy is an important issue as well. As the modern vehicles become more and more connected with internet, users have less control of the usage of their sensitive data. Thus, in this thesis, we propose a privacy-preserving framework for connected car environments to help users take back controls over their data.

3 Privacy Analysis and Requirements

In this chapter, we analyze the privacy concerns in connected car environments and summarize them into several privacy requirements. We assume that every end-point service in connected car environments is greedy. That is, end-point services always want to retrieve every possible source data from a CV even though they do not need them for their service functionalities. Furthermore, they always try to derive hidden information from the data they retrieved to gain maximum knowledge of users.

On the other hand, the general demand of users is to utilize as many functionalities provided by end-point services as possible while having their sensitive data protected. This means, users usually do not want to share their source data with untrusted participants in connected car environments but only perturbed data. Based on these two assumptions, we analyze the privacy demands based on a motivation example in Section 3.1 and then introduce the privacy requirements in Section 3.2.

3.1 Privacy Analysis

This section focuses on the privacy analysis based on different CV scenarios. These scenarios are extended from the privacy pattern described in Chapter 1. In the following, we first illustrate this privacy pattern in the base scenario with more details based on Figure 3.1. Then, we conduct the privacy analysis for the base scenario and different extended scenarios.

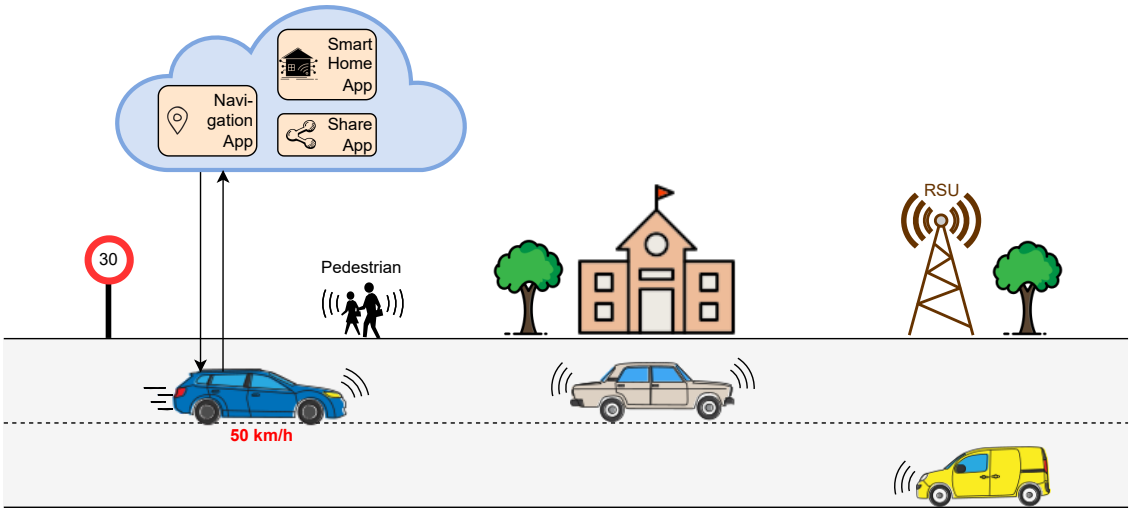


Figure 3.1: Detailed Motivation Scenario

3.1.1 Base Scenario: Speeding

As shown in Figure 3.1, our user Alex is driving a blue CV at 50 km/h. He just entered a campus area, but he did not notice the speed limit sign. In his car, he had several third party apps installed that have to be connected to the cloud. On the road, there are other CVs. One beige car that drives in the same direction as Alex and a yellow car that drives in the opposite direction as Alex. In addition, on the roadside, there are two pedestrians walking along the road and one RSU located near the school. However, this RSU cannot measure the speed of the car and there is no speed camera located in this area.

In this base scenario, the privacy pattern that Alex wants to hide from any other party (e.g., his car insurance company) is the speeding behavior in a campus area. On the other hand, he could not hide this behavior from parties (pedestrian, drivers of beige and yellow CV) that are currently present in the scenario.

Thus, one main goal of our framework is to provide users with the ability to hide their sensitive data, including specific privacy patterns, from any party that cannot sense the information itself. In the base scenario, this means, if Alex does not want his insurance company to know that he is speeding in a campus area, our framework will guarantee that. However, if there is a speed camera in this campus area that captured the speeding behavior of Alex, then, even if Alex wants to hide this behavior from the law department, our framework could not help with that.

3.1.2 Extended Scenario 1: Individuality

For the first extension of the base scenario, we focus on the three apps installed in Alex's CV. As shown in Figure 3.1, Alex has three different cloud-based apps: *Navigation* app, *Smart-Home* app, and *Share* app.

In this scenario, Alex is still speeding in the campus area, but he has different privacy demand on these three apps. For the *Navigation* app, he does not mind this app to know he is currently speeding, because an accurate routing service is more important to Alex. Then, for the *Smart-Home* app, he only wants to let the app know his approximate location. Finally, for the *Share* app, he is willing to share his location and speed with the app, but he does not want this app to know his speeding behavior.

It is nature that privacy demands for specific data section, particular end-point service, or certain situation may vary. The other goal of our framework is to provide individual and situation-aware privacy protections for different users. This means, for a certain scenario, our framework supports users to define individual privacy demands for different end-point services. In the meantime, our framework also supports users to define different privacy patterns for the same end-point service in various scenarios.

3.1.3 Extended Scenario 2: Safety

In this extension of the base scenario, we change our view to the other CVs on the road and the pedestrians on the roadside. Imagine that in the beige CV, there is an ADAS that gathers information from other CVs near it and alerts its driver of possible accidents. Moreover, imagine that one pedestrian is wearing a smart device that also gathers information from other CVs near it and indicates its wearer whether it is safe to cross the road now.

Recall that in the base scenario, we discussed that the pedestrians and the drivers of other CVs in the scenario know Alex is speeding. This is because they can perceive this reality through their own eyes. However, we cannot assume the same to the ADAS or to the wearable smart device, unless they are equipped with a speed measurement sensor. Hence, to realize their service functionalities, they need data shared from Alex's CV.

Usually, users do not trust other CVs or (wearable devices of) pedestrians on the roadside but prefer to trust RSUs, as they are usually managed by the government. Thus, Alex does not want to share his speed data to other CVs directly but through a RSU so that his identity still stays hidden. However, in certain critical situations (e.g., speeding) every potential latency in the data communication might increase the chance of a car accident. Therefore, even though Alex does not want to share his data to other CVs directly, it is inevitable in this situation.

Another important aspect in this scenario is that Alex needs to share his real speed data, as the functionalities provided by these two services are safety-related. Assume that Alex shares a false speed data (e.g., 30 km/h) because he wants to hide his speeding behavior. Then, the two services mentioned above would make predictions based on the fact that Alex is driving around 30 km/h even though he is actually driving at 50 km/h. This would also increase the chance of a car accident.

Thus, one important principle in our framework is that users' privacy decision should never affect the safety of any user, including themselves. For our particular scenario here, this means that Alex needs to broadcast his real speed data to other CVs or pedestrians near him. Yet, our framework could still protect Alex's privacy on a certain degree. For example, a pseudonym could be used to present Alex's CV. In addition, for non-safety related services, the framework should still fulfill Alex's privacy demands as described in Section 3.1.2.

3.1.4 Extended Scenario 3: Accident

In the third extension of the base scenario, we consider a tragic situation where Alex has a car accident in the end, because of his speeding behavior. In this case, the law enforcement will arrive after the car accident and gather evidence at the scene.

Clearly, our framework must not help Alex hide his speeding behavior from the law enforcement. Instead, our framework should help the law enforcement to get hands on the very source data of Alex's CV before the car accident occurs. This means, the source data of a CV needs to be saved in our framework for a certain time. In addition, users should not have the access to modify these source data.

Recall that Alex wants to hide his speeding behavior from his car insurance company as introduced in Section 3.1.1. Our framework could still fulfill his privacy demands even if a car accident has occurred. This is because an insurance company is usually a commercial organization and does

not stand for law. On the other hand, our framework cannot prevent Alex's insurance company from knowing his speeding behavior or the car accident if his insurance company could obtain this information from law enforcement.

3.1.5 Extended Scenario 4: Broadcast

For the last extension of the base scenario, we consider a specific communication technology that are widely used in connected car environments – broadcasting. As introduced in Chapter 2, lots of communications between CVs and other parties, such as V2V and V2I, use this kind of technology. One specific characteristic of broadcasting is that it does not have fixed receivers, which means, whoever is in the broadcasting range and has a supported receiver can receive the message.

This requires our framework to have all the data processed or perturbed within the framework and then send or broadcast the data to other participants in connected car environments. In other words, our framework should not depend on any other third parties to protect user's privacy.

3.2 Privacy Requirements

After we have discussed different privacy demands in various scenarios in Section 3.1, we now conclude them into following privacy requirements:

- **Transparency (before law)**: the source data of a CV should be transparent before law enforcement, especially when a law-breaking behavior occurs
- **Safety First**: the safety of any user always comes first, i.e., the privacy demands of a user can only be fulfilled if it does not harm anyone's safety
- **Equality**: whatever information that cannot be learned in a non-connected car environment, should not be learned in a connected car environment, unless the user agrees to publish
- **Individuality**: each user's individual privacy demands, and situational privacy pattern should be fulfilled and protected
- **Local Processing**: the source data should be processed within the framework to a state that, even if the processed data are being published later, it does not break users' privacy demand

Apart from these essential requirements, we consider the following optional privacy demands to be valuable for connected car environments as well:

- **Transparency (before user)**: the data flow should be transparent before user, which means users are aware of the exact data various services have required
- **Accountability**: if a privacy breach occurs, the responsible parties could be identified
- **Secure Deletion**: users have the ability to request data collectors to delete their data securely

4 Privacy4CV Framework

Based on the privacy requirements described in Chapter 3, we propose a situation-aware Privacy4CV Framework to ensure privacy in connected car environments. The goal of our framework is to provide a concept that can protect users' privacy and ensure services' quality at the same time. In addition, our framework is generic enough so that it can be applied to both edge environment (e.g., on the CV) and fog environment (e.g., on the RSU). Thus, the CV model used to develop our framework is also generalized in this thesis.

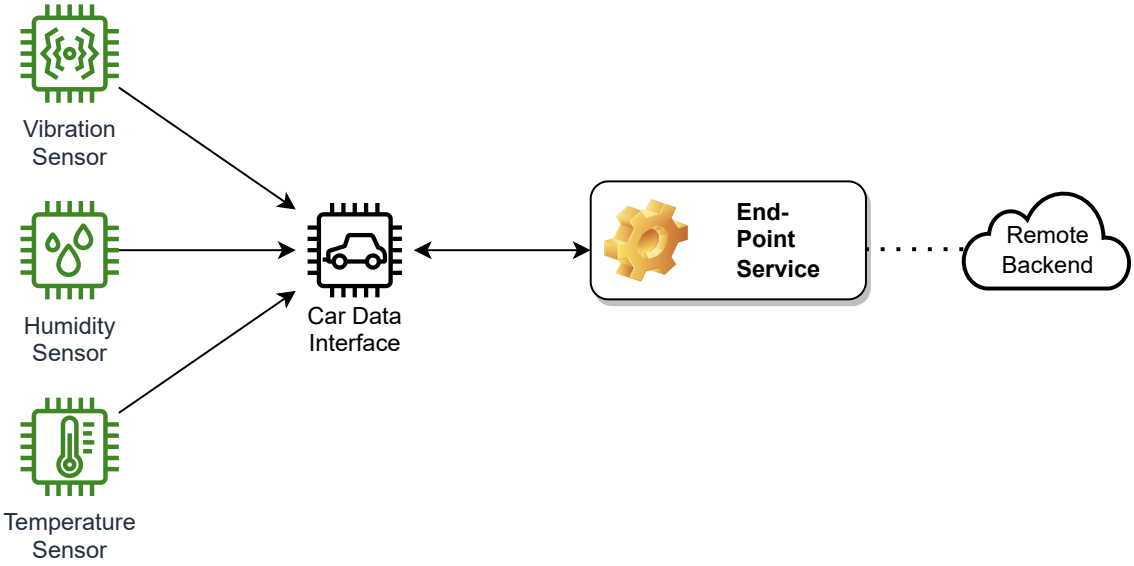


Figure 4.1: Connected Vehicle Model

As depicted in Figure 4.1, we assume each CV has an `Interface` that gathers various metrics from different sensors equipped in cars and exposes these source data to the end-point services installed in cars as data streams. The end-point services can be completely local, or they can be connected to a remote back-end. Notice that this assumption is very general and does not bring any restrictions to the framework. If a CV does not provide such an interface, we could easily implement this by gathering data from whatever interfaces provided by that CV and expose them to the end-point services. In addition, we assume that the underlying security is guaranteed. This means, for any security methods we consider in the framework, we do not consider privacy leaks if an attacker breaks this security method.

Then, on a very high level, our framework works like an "access control system" between the `Interface` and various end-point services. With our framework being deployed in between, the end-point service cannot access the source data directly. Instead, it gets the perturbed data that has

been processed by the framework based on users' specific privacy demands. As end-point services can only get data through our framework now, they cannot retrieve arbitrary data they want and, in most cases, they will only get modified data that are necessary for their service functionality.

Before we dive into the details of our framework, we first introduce some prerequisites in Section 4.1. Then, we give a thorough explanation of the architecture of our framework in Section 4.2. After that, we describe the privacy policy and PT in Section 4.3 and Section 4.4, respectively. Finally, we illustrate the user interaction with our framework in Section 4.5 and conclude this chapter with a discussion in Section 4.6.

4.1 Prerequisites

To ensure that our framework can protect users' privacy and maintain maximum service functionality at the same time, two essential prerequisites need to be fulfilled. In the following, we explain why we need these two prerequisites and the details of them in Section 4.1.1 and Section 4.1.2 separately.

4.1.1 Authority

Recall that in Section 3.1.3 and Section 3.1.4 we argued that in some cases we must reveal the source data of our CV to other parties to ensure safety of all users or because we need to obey the law. As we also assume that all end-point services are greedy, it is easy for them to claim that all of their service functionalities are safety related so that they can get hands on the unmodified source data. Moreover, they could even camouflage themselves as law departments to achieve this goal as well. On the other hand, it is very hard for our framework or our users to distinguish these disguises. Thus, a government authority or an industry association is needed to review and audit every functionality provided by end-point services in connected car environments.

The major responsibility of the government authority or industry association is to issue signed certificates to service providers. Then, it can be ensured that only end-point services which have a valid certificate could require data from our framework. In addition, important metadata of end-point services should be presented in the certificate as well. To determine the metadata, three sub-tasks need to be accomplished by the authority: 1) assign function tag, 2) determine locality, and 3) identify essential and optional data sections for service functionalities.

For the first task, the authority needs to review the service purpose and issue a `functionTag` to the service functionality. Currently, the supported `functionTag` in our framework are `LAW`, `SAFETY` and `COMMERCIAL`. The `LAW` or `SAFETY` `functionTag` indicates that the corresponding functionality is law or safety related whereas the `COMMERCIAL` `functionTag` is used for any other kinds of services. With all services being divided into three categories, our framework can then force that only domain experts are qualified to define privacy requirements for `LAW` or `SAFETY` functionalities and general users are only allowed to specify their individual or situational privacy demands for `COMMERCIAL` services.

When defining privacy policies for `LAW` or `SAFETY` functionalities, we trust our domain experts to always put service qualities (in this case, protect users' safety or obey the law) before users' privacy so that no life will be endangered. On the other hand, we provide maximum flexibility to

users regarding their individual privacy demands of COMMERCIAL services. Notice that supported functionTags in our framework are not fixed. If there are new types of services showing up in the future, it can be easily adapted by our framework through adding new functionTags accordingly.

The second task of the authority is to distinguish the locality of a service functionality. Similar to catalogs of external services introduced by Plappert et al. in [PZK+17], we define local service as services whose computations are all done locally, i.e., within the car, and remote service as services that need to send car data to components that are external to the car, e.g., a remote back-end, for its computations. The separation is needed as our framework provides local services with full access to source data since the data never leave the car. However, as we do not trust any service functionalities to declare their locality truthfully, we need an authority to classify them for our framework.

For the last task, the essential and optional data sections required to accomplish the service functionality need to be identified. Recall that service providers are always greedy. Thus, it is highly possible that they require unnecessary data in order to collect more information from users. Therefore, an authority is needed to identify the essential data for a service's core functionality, optional data for service's additional functionality, and unnecessary data which is not needed at all.

Together, for each eligible service functionality, the authority issues a signed certificate that also includes its metadata as mentioned above. Then, whenever a new service is installed in a CV, this certificate should be downloaded, verified and saved in the CV as well, so that our framework is aware of the service's metadata when it requires data later.

4.1.2 Service Level Agreement

To maintain maximum service quality, our framework does not interfere with any business logic of service functionalities or their communications with their remote back-end. This means, once the data is handed over to the service provider, especially for remote service, our framework has no control over the data anymore. Thus, in the Service Level Agreement (SLA) between a service provider and its user, the service provider must explain clearly, how it will further process, store or share the data they received from our framework.

Furthermore, if a service provider delivers multiple functionalities with different functionTags, then this service provider needs to clarify how it organizes various data retrieved from different functionalities in the SLA as well. For example, whether the service provider integrates the data they retrieved from its SAFETY and COMMERCIAL services of the same user. On the other hand, it is our framework's responsibility to guarantee that the data we forward to the end-point service does not have any sensitive information or can be used to derive hidden knowledge of the user anymore.

4.2 Architecture

Recall that our framework works like an Attribute-Based Access Control (ABAC) system between the data stream of CV's source data and different end-point services. However, classical ABAC model proposed by Hu et al. in [HFK+13] is used to determine authorization decisions of certain

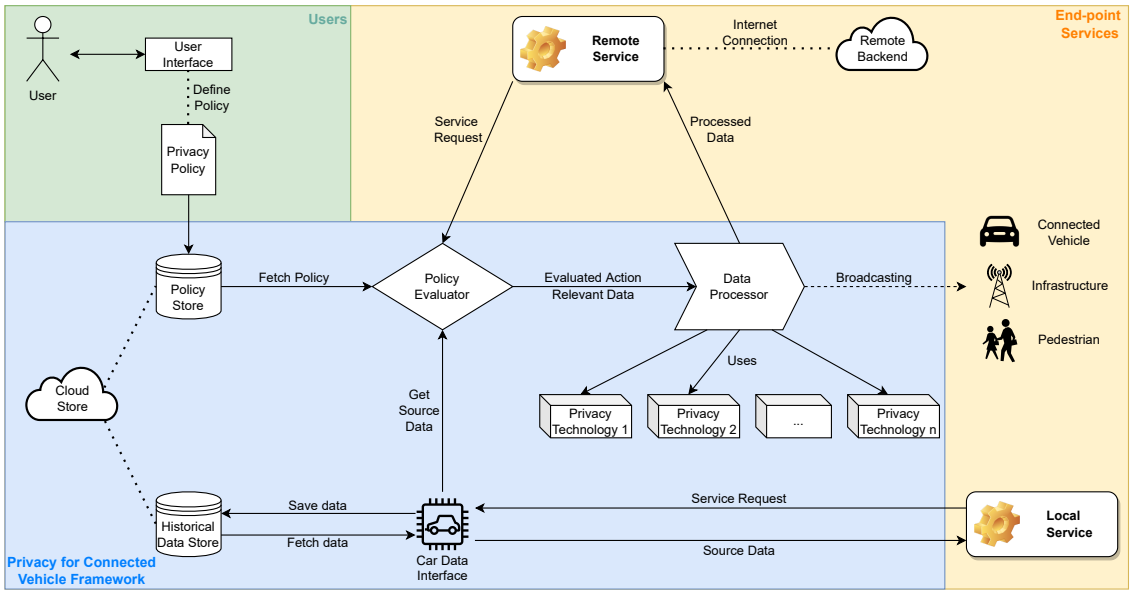


Figure 4.2: Privacy for Connected Vehicle Framework

resources by evaluating attributes against policies. In our framework, the resource is the source data stream of a CV, the policy represents user's specific privacy demands, and the authorization decision is no longer a simple permit or deny, but different Privacy Techniques (PTs) that need to be applied to the source data stream.

As depicted in Figure 4.2, there are two parties that interact with our framework: users and end-point services. Users of our framework can be further divided into three user groups: system admins, domain experts and general users where system admins are responsible for configuring the framework, domain experts are capable of defining domain related policies and general users are drivers or passengers of CVs. As mentioned in Section 3.1 and Section 4.1, end-point services of our framework can be classified into three categories as well: `local` service, `remote` service, and receivers in broadcasting range.

As for the Privacy4CV Framework itself, it consists of a `User Interface` where users can interact with our framework, a car data `Interface` where the source data stream comes from, a `policy store` where users' privacy policies are persisted, a `historical data store` where unmodified source data are stored for a certain period, a `policy evaluator` where the source data and the metadata of end-point services are evaluated against deployed privacy policies, and a `data processor` where various PTs are applied to the source data according to the evaluation result.

Figure 4.2 also provides an overview of our framework. In the Privacy4CV Framework, users are enabled to define their individual or situational privacy demands through the `User Interface`. These privacy demands are then used by the `policy evaluator` together with the metadata of end-point services to determine the desired PTs. After that, the `data processor` applies corresponding PTs to the source data stream. Finally, the processed data is forwarded to the end-point services accordingly. With this overview in mind, we introduce the detailed data flow of our framework in the following.

To start with, users need to specify their individual or situational privacy demands through the User Interface. This User Interface helps users to formalize their privacy requirements into privacy policies that are supported by our framework. The created privacy policies are then saved in a policy store. This store is located in CVs, however, it can also be configured to connect to a remote cloud store, based on users' demands.

The cloud store needs to provide user-based access control so that users can only access their own privacy policies. In addition, the synchronization between the cloud store and the local policy store should be conducted at a time when the system's performance is not affected and through a secure channel. With the privacy policies saved in cloud, users can always use the same privacy policies without re-defining them in different CVs. This helps especially in car sharing situations. Furthermore, the cloud store could also enable users to share their privacy policies as templates to help other users create their own policies more easily.

Naturally, any user-defined privacy policies can be later viewed, modified, or deleted. However, as mentioned in Section 4.1.1, all expert-defined LAW or SAFETY related privacy policies can only be read by users and only the qualified experts are able to modify or delete them.

Every time a privacy policy has been created or modified, a re-deployment of all privacy policies is conducted in policy evaluator so that it is always up-to-date with all policies saved in policy store. It is recommended that users should only interact with the User Interface when CVs are parked. Thus, any privacy requirements users want to use for their next trip must be defined before the trip starts.

When the policy creation is finished and the CV starts moving, a constant stream of data is forwarded to our framework through the car data Interface. For every source data our framework receives, a copy of it is saved in the historical data store. This data store is needed in case law enforcement requires access to unmodified data as described in Section 3.1.4, or an end-point service requires both current data and historical data. The historical data store should also guarantee that the data saved in it is not modifiable. Moreover, this data store should be implemented in a way that only eligible parties (e.g., law enforcement) and our framework have access to it directly. For any other parties such as users or various end-point services, they can only access the historical data through our framework.

According to McKinsey [McK14], a modern CV generates around 25 Gigabytes of data per hour. Thus, it is not likely to save all source data of a CV for long period purely in the car. In our framework, users can choose a certain period (e.g., a week) that they prefer to save their original source data in the store. After the defined period is over, the newly arriving data will overwrite the old data. However, users cannot choose arbitrary period for the historical data store. The selected period must exceed the minimum period defined by the authority that is introduced in Section 4.1.1. On the other hand, the maximum period is bound by the storage capacity of the underlying CV.

Therefore, if users want to save more source data than the CV's storage can hold, they could choose to connect the historical data store to a remote cloud store. The requirements for this cloud store are similar to the requirements of the policy's cloud store. In addition, the data saved in the cloud store should be unchangeable, same as the requirements for the historical data store in CV. Naturally, users can still view or download the data from the cloud store through our framework, and they can also modify the downloaded data as desired. Furthermore, as the very

source data of users are saved in the cloud store that are external to the CV, the cloud store provider is responsible that the saved data is not leaked or misused against users' will. For more information and details about the historical data store please refer to Section 4.6.1.

Recall that an end-point service in our framework can be classified into three categories: local service, remote service, and receivers in broadcasting range. In the following, we explain the data preparation for these three types of end-point services separately.

We start with the simplest case, data preparation for local services. As no data leaves the CV during the computation of local services, our framework provides them with direct access to the source data. Since there are no privacy leak risks, the access to the source data guarantees the highest service quality of local services. With the same reason, local services can also get direct read access to the historical data store if they need any historical data.

On the other hand, remote services usually transfer users' data to their remote back-end for computation, which increases the risks of a privacy leak. Therefore, only the data that are processed based on users' privacy demands is forwarded to remote services. In our framework, this is achieved by routing the source data to the policy evaluator, instead to the remote services directly. The source data can either be current data from the source data stream or historical data from the historical data store. Then, when the source data arrives in the policy evaluator, it evaluates this data together with the metadata of the remote service against all privacy policies deployed on itself. The result of the policy evaluation contains a set of desired PTs that needs to be applied to the source data and a list of data sections that users are willing to share with the remote service. This result is then forwarded to the data processor together with the source data. After that, the data processor fetches required PT modules and applies them to the source data. It also filters the source data (after all PTs have been applied) so that the processed data only contains data sections that users are willing to share. Finally, the processed data is forwarded to the remote service.

For the third type of end-point services, broadcasting can be seen as a special case of remote services where the end-point service is unknown. Recall that when evaluating privacy policies, the metadata of end-point services is also required. As the receiver of broadcast messages varies, in this case, the metadata is the information about the underlying broadcast channel that is used to send messages. Analogous to the remote services' situation, the source data that needs to be broadcast is first evaluated against users' privacy policies together with the metadata of the broadcasting channel. Then the corresponding PTs are applied to the source data and finally, the processed data is broadcast through the desired channel.

The above description of our framework mainly explains how it can be used in edge environment, i.e., on a CV. In the following, we interpret that our framework can also be applied in fog environment, e.g., on an RSU. The architecture of our framework does not need to be changed when used in fog environment. However, the group of users, the source data that comes into our framework and the set of PTs do need to adapt.

First of all, the general users of our framework in edge environment are drivers of CVs whereas the user group in fog environment is usually managers or administrators of the RSU. Then, in edge environment, we only consider data from a single CV as the source data for our framework, but in fog environment, the source data are data measured or gathered by the RSU which usually include data from different CVs or pedestrians near the RSU. Finally, the PTs used in both environments are

usually different as well, as the source data in these two environments differ. For example, a PT that is defined for data from a single CV is usually not applicable on a group of data from multiple CVs or on data from pedestrians.

In addition, the RSU that uses our framework can be seen as a trusted and centralized anonymizer for CVs that also have our framework deployed. In this case, PTs that realize the group-based anonymization models, such as k-anonymity (proposed by Sweeney [Swe02]) or ℓ-diversity (proposed by Machanavajjhala et al. [MKGV07]), can be provided to individual CV as well. To use a PT from the RSU, the source data of a CV and the name or indicator of user's desired PT are sent to the RSU. The RSU then applies the desired PT on the received data together with other data that are currently available in the RSU. Finally, the processed data is sent back to the requesting CV.

Notice that in this special case, we let the source data of a CV leaves the car unmodified. However, as the data is only traveling from one instance of our framework to another, we do not consider potential privacy breaches as long as the underlying communication channel is secure.

4.3 Privacy Policy

In this section, our focus is the privacy policy used in Privacy4CV Framework. In general, users have individual privacy preferences for specific data section, particular end-point service or certain circumstance. They can define these preferences in our framework through the `User Interface` depicted in Figure 4.2 with high flexibility. However, all of these privacy preferences need to be transformed into a privacy policy with a given format, so that they can be evaluated by the framework. In the following, we first introduce the privacy policy format in Section 4.3.1, and then describe two kinds of policies that are used in our Privacy4CV Framework in Section 4.3.2. After that, we explain the potential conflicts that could exist in a set of privacy policies and how to resolve them in Section 4.3.3. Finally, we illustrate the evaluation using a set of privacy policies in Section 4.3.4.

4.3.1 Policy Format

A privacy policy is defined in JavaScript Object Notation (JSON). Recall the base scenario we introduced in Section 3.1.1 where Alex wants to hide the privacy pattern of his speeding behavior in a campus area from his insurance company. A sample privacy policy that fulfills Alex's privacy demand is presented in Listing 4.1.

As shown in Listing 4.1, each policy has a unique `policyID` that can be used to identify the policy and a `meta` field that records the general information about the policy. Following the `meta` field, there is a `priority` value that indicates the precedence of the policy. The policy `conditions` field defines the constraints of a policy, and the policy `actions` field describes the desired PTs for different data sections or end-point services that should be executed successively. In the following, we introduce these four fields in detail.

Listing 4.1 Sample Situational Policy

```
{
  "policyID": "e7d626c3-e3fe-48c9-9cf0-a93fa6c5566c",
  "meta": {
    "name": "speeding privacy policy",
    "lifetime": "ALWAYS",
    "isEnabled": true,
    "creator": "user-pjeXDbRozh",
    "createAt": "2022-04-13T09:01:47",
    "ownership": [
      "user-pjeXDbRozh"
    ]
  },
  "priority": 2,
  "conditions": {
    "date": {
      "weekday": [1, 2, 3, 4, 5],
      "frequency": "EVERY_WEEK"
    },
    "time": {
      "begin": "06:00",
      "end": "20:00",
      "frequency": "EVERY_DAY"
    },
    "location": {
      "neighbourhood": "CAMPUS"
    },
    "speed": {
      "useSpeedLimit": true,
      "speedLimitTolerance": 0.1
    },
    "functionTag": "COMMERCIAL"
  },
  "actions": [
    {
      "targetServices": [
        "Ins4Car"
      ],
      "targetDataSections": [
        "SPEED"
      ],
      "privacyTechnique": "randomization",
      "privacyParameters": [
        "SPEED_LIMIT",
        "0.1"
      ]
    }
  ]
}
```

Policy Meta

The policy `meta` describes the metadata of a privacy policy. It contains a user-defined policy `name` that can help users to easily search and then modify the policy using the `User Interface`. Apart from the policy's `name`, the metadata also contains information about the policy's effective period. The `lifetime` attribute can be set to `ALWAYS`, `THIS_MONTH`, `THIS_WEEK` or `TODAY` to describe a policy's lifetime and the `isEnabled` attribute is used to indicate whether a policy is currently activated. Notice that the supported value of `lifetime` for a privacy policy is configured by a system admin and can be changed through system update.

The last part of the metadata is the creation and ownership information of the privacy policy. The `creator` attribute contains an ID of the user who created the policy, the `createAt` attribute describes the date and time when the policy has been created, and the `ownership` attribute indicates which group of users can modify the policy after it has been created.

Policy Priority

Each policy contains an integer value that indicates its priority. In our framework, the lower the priority value, the higher the policy priority. Notice that the highest (priority value 0) and the second highest (priority value 1) policy priority are preserved for `LAW` and `SAFETY` tagged service functionalities, respectively. In addition, policies of these two priorities can only be created by authorized domain experts. Apart from these two preserved priority values, there is one more restriction for the policy priority: the lowest priority (in Privacy4CV Framework, we choose priority value 10 for the lowest priority as default value) is preserved for all Trust-Level Policies (TLPs).

All other priority values (by default from 2 to 9) can then be used for user-defined Situational Policies (STPs) to provide maximum flexibility. Notice that there is no restriction to the maximum priority level. This means, if users want additional priority levels, they could change the maximum priority level to an arbitrary integer n that is bigger than 3. Thus, they have a priority range from 2 to n-1 for their STPs, as the TLPs always use the lowest priority (in this case priority value n).

For more information about TLP, STP and the difference between them please refer to Section 4.3.2.

Policy Conditions

The policy `conditions` field specifies a set of conditions when the policy should be evaluated to `true`. The condition constraints are very flexible as there are no restrictions from the framework. However, it does suffer from limitations of the underlying hardware. Therefore, the actual supported constraint types are configured by a system admin based on the available sensors, the accuracy of their measurement, and so on.

The policy `conditions` field is the key to situational privacy protections. It enables users to specify privacy preferences for different situations or various privacy patterns. In the following, we take a closer look at the `conditions` field defined in the sample policy. As shown in Listing 4.1, this `conditions` field consists of five constraints. Each constraint and the meaning of it is listed below:

- `date` constraint: every week from `Monday` to `Friday`

- time constraint: every day from 6am to 8pm, combined with the date constraint it means: every workday from 6am to 8pm
- location constraint: in CAMPUS area
- speed constraint: exceed the speed limit by more than 10 percent
- functionTag constraint: the targeting end-point service functionality is marked as COMMERCIAL

Altogether, the conditions field of this policy describes: if the user is speeding in a CAMPUS area on a workday between 6am to 8pm and the data is providing to a COMMERCIAL end-point service functionality, then the defined policy actions should be executed.

Notice that there is more than one constraint defined in the sample policy. When evaluating the policy, the relation between all these constraints is always AND. However, the user can define OR or even XOR connected constraints through the User Interface, but it should then be transferred into multiple privacy policies with the given format.

Policy Actions

The policy actions field contains a set of actions that will be executed when the policy is evaluated to true. As presented in Listing 4.1, each action is represented by four attributes. The targetServices attribute contains a list of its target service's name or indicator. The targetDataSections attribute indicates a list of the targeted data section(s), namely the data section(s) that needs to be perturbed by the given PT. The last two attributes, privacyTechnique and privacyParameters, specify the concrete PT and its parameter(s), respectively.

Notice that targetServices and targetDataSections attributes act as sub-constraints within each policy action. These two attributes also enable users to define individual privacy demands for particular end-point services and specific data sections. Furthermore, same as the constraint types available in policy conditions, the supported data sections and PTs are configured by a system admin and can be changed through system updates. Moreover, all actions that are defined in one policy share the same policy conditions. This means, a user can define different privacy actions for different end-point services that share the same trigger in one privacy policy.

It is important that there are no conflicts between actions listed in policy actions. A conflict of policy action occurs when:

- the two policy actions have at least one same entry in targetServices, and
- the two policy actions have at least one same entry in targetDataSections and,
- the two policy actions use different PT, or
- the two policy actions use the same PT but with different privacyParameters

If an action conflict is detected, users are informed to change one of the actions to resolve the conflict. As long as an action conflict exists in a privacy policy, this policy is marked as inactive (i.e., isEnabled attribute in policy meta is set to false) and will not be deployed in the policy evaluator until the conflict has been resolved.

4.3.2 Trust-Level Policy vs. Situational Policy

Based on the privacy policy format that has been explained in Section 4.3.1, two kinds of policies are defined in the Privacy4CV Framework, namely the Trust-Level Policy (TLP) and Situational Policy (STP). In the following, we first introduce these two types of privacy policies and then explain the difference between them.

Trust-Level Policy

A Trust-Level Policy (TLP) is a privacy policy whose `conditions` field only contains one constraint regarding the trust-level of end-point services. When a new third party app is installed in the car, a default trust-level is assigned to this app. Users can change the default value and the trust-level of a specific app through the `User Interface` anytime.

As the TLP is applied based on the trust-level of end-point services, there is no need to specify any target services in `targetServices` attribute of each policy action. A sample TLP is shown in Listing 4.2. We can see that there is only one constraint `trustLevel` defined in the policy `conditions` field and as explained before, the `targetServices` attribute in each policy action is left empty.

Situational Policy

On the other hand, a Situational Policy (STP) is a privacy policy whose `conditions` field is usually more flexible, complex, and situational. As shown in Listing 4.1, this STP is targeting a very specific privacy pattern that might occur in the user's daily routine. The STP is aimed to provide users with high flexibility to define fine-grained privacy policies. Users can use this kind of privacy policy to define both individual privacy requirements for particular end-point services or specific data sections and situational privacy demands for different situations or various privacy patterns.

Notice that STPs always have higher priority than TLPs. Thus, it is not recommended using the `trustLevel` constraint in STPs.

Comparison

Comparing these two types of policies, we can see that TLP is aimed at general purpose privacy protection whereas STP is more focusing on individual and situational privacy requirements. Therefore, TLP always has the lowest priority, as users' specific privacy requirements should always be fulfilled first. However, this does not mean that TLP is redundant. TLPs are the default policies that come with the system, thus, even if users forget to define any STP before their first drive, their privacy is still protected by TLPs. Furthermore, it is because of the TLPs that users do not have to define STPs for every possible scenario, privacy pattern that might occur or every end-point service that is installed in car. All undefined situations will be covered by TLPs.

As mentioned above, the default TLPs are activated when the Privacy4CV Framework is deployed in a CV. Naturally, users can change these default TLPs based on their desire, but users cannot delete them. In addition, the supported trust-levels are configured by a system admin and can be changed through system updates.

Listing 4.2 Sample Trust-Level Policy

```
{
  "policyID": "949ae0eb-81d4-49cc-b837-17906b5034a5",
  "meta": {
    "name": "semi-trusted level privacy policy",
    "lifetime": "ALWAYS",
    "isEnabled": true,
    "creator": "user-pjeXDbRozh",
    "createAt": "2022-04-10T18:32:28",
    "ownership": [
      "ALL"
    ]
  },
  "priority": 10,
  "conditions": {
    "trustLevel": "SEMI_TRUSTED"
  },
  "actions": [
    {
      "targetServices": [],
      "targetDataSections": [
        "META"
      ],
      "privacyTechnique": "anonymization",
      "privacyParameters": [
        "pseudonymization"
      ]
    },
    {
      "targetServices": [],
      "targetDataSections": [
        "STATUS",
        "SPEED",
        "LOCATION"
      ],
      "privacyTechnique": "NONE",
      "privacyParameters": [
        "NONE"
      ]
    }
  ]
}
```

4.3.3 Policy Conflict

Usually, users have more than one privacy policy defined in their CV and when they are driving, there is usually more than one privacy policy active. Thus, it is vital to make sure that there are no conflicts between active policies. Otherwise, the evaluation order of policies is not deterministic.

Due to the consideration of user's safety, the Privacy4CV Framework does not recommend creating privacy policies or resolving policy conflicts on the fly, namely, when CVs are moving. Thus, our framework checks the policy conflict every time a new policy has been created.

For both TLPs and STPs policy conflicts could occur. The policy conflict in TLPs is easy to detect as for each trust-level, there should only be one TLP. On the other hand, two STPs have a policy conflict when the following statements are evaluated to true at the same time:

- if two STPs should be activated at the same time,
- if two STPs have the same `priority`,
- if policy `conditions` of two STPs can be evaluated to `true` at the same time,
- if action conflict described in Section 4.3.1 exist between any pair of policy actions from these two STPs

Notice that there could never be a policy conflict between TLPs and STPs as TLPs always have lower `priority` than STPs.

If any conflict is detected, several suggestions will be displayed on the `User Interface` to help users resolve the policy conflict. Usually, users are asked to change the `priority` of one policy, modify the policy `conditions` field, or adjust policy `actions` field based on where the conflict is detected. Then, when users have resolved the conflict, the system will re-check the policy conflicts.

This procedure should be repeated until there are no policy conflicts in the system anymore. However, if users ignore the conflict alarm and do not resolve conflicts before driving, then, all conflicting STPs will be marked as inactive until all policy conflicts have been resolved. Similarly, for conflicting TLPs of a certain trust-level, the default TLP will be used.

4.3.4 Policy Evaluation

When a CV is moving, there are usually multiple privacy policies active. Thus, evaluating these policies in the right order is very important. In the Privacy4CV Framework, one rule is applied: for a target data section of an end-point service functionality, **only** the `actions` field from the policy that has the highest priority will be executed. In the following, we illustrate how this rule works with an example shown in Figure 4.3.

In the example, we have four different privacy policies defined. The `policy-1` is an expert-defined `LAW` related policy that has the highest priority (priority value 0). This policy targets data section `C` of both service functionality 1 and 2. The `policy-2` is also an expert-defined `SAFETY` related policy that has the second highest priority (priority value 1). This policy targets only data section `B` of service functionality 1. Then, we have a user-defined STP `policy-3` that has a priority value m with

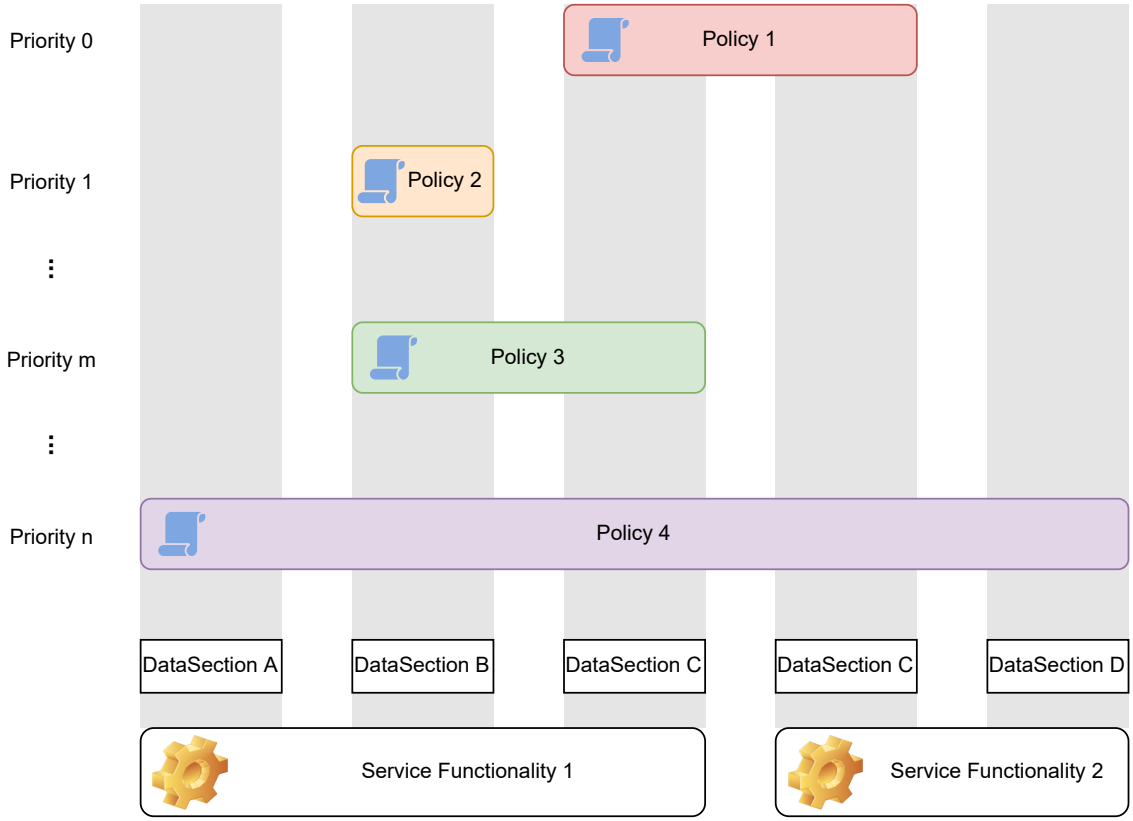


Figure 4.3: Policy Evaluation

$2 \leq m < n$. This policy targets both data section `B` and `C` in service functionality 1. Finally, we have a default TLP `policy-4` with the lowest priority (priority value n). This policy covers all data sections in all service functionalities.

Now, assume that the current source data forwarded by the car data `Interface` satisfying every constraint in all four policies' `conditions` field, namely all four policies are evaluated to `true`. Then, for the three data sections that are required by the service functionality 1 we have:

- for data section `A`, the privacy `actions` field defined in `policy-4` will be executed, as there are no other policies defined for this data section of this service functionality.
- for data section `B`, **only** the privacy `actions` field defined in `policy-2` will be executed, as `policy-2` is the policy that has the highest priority defined for this data section.
- same as above, for data section `C`, **only** the privacy `actions` field defined in `policy-1` will be executed.

Analogously, for service functionality 2, the privacy `actions` field defined in `policy-1` will be executed for data section `C` and the privacy `actions` field defined in `policy-4` will be executed for data section `D`.

Notice that the privacy `actions` field from user-defined STP `policy-3` is never executed in this example. The reason is that, in the example situation, the `LAW` and `SAFETY` related policies are activated at the same time. Since these two policies have higher priority, no privacy `actions` from

policy-3 can be executed. In the Privacy4CV Framework, following the law and protect user's safety always come before preserving user's privacy. Thus, in our framework, a user-defined privacy policy could never have a higher priority than an expert-defined LAW or SAFETY related privacy policy.

In addition, this example also shows the necessity of TLPs. As the user only defines one STP for data section B and C of service functionality 1, the other data sections of service functionalities 1 and 2 are left unprotected from the user's point of view. Now, imagine we do not have policy-4, then the very source data of data section A and D will be sent to service functionality 1 and 2, respectively, which might lead to a privacy breach. Hence, in the Privacy4CV Framework, TLPs always have the lowest priority since they act as a last defense of user's privacy.

4.4 Privacy Technique

In our framework, Privacy Technique (PT) is the actual technique or algorithm that is used to perturb source data in a privacy-preserving way. In general, there are no restrictions from our framework regarding the selection of PTs. However, certain adoptions might be necessary.

Foremost, the desired technique should be implemented as modules that are exchangeable. This means, all PTs used in our framework need to implement two interfaces that parse the source data and the corresponding parameters from data processor as input data and convert the perturbed output data into a given format that is supported by the data processor again. By treating different PTs as modules, our framework can always provide users with the latest technique by switching out outdated techniques and including the latest one. Recall that the sets of PTs for edge environment and fog environment are usually different. With the modularization, it is also easier for system admins to manage these PTs' set.

Another point that is worth to mention here is the locality of PTs. Generally, our framework prefers to use PTs that have all of its computation processed locally, i.e., within the CV or RSU. In spite of that, our framework also accepts PTs that require a remote back-end. However, as the source data now leaves our framework during the PT's computation, an SLA needs to be provided by the technique provider. Similar to what is introduced in Section 4.1.2, this SLA needs to inform its users regarding the usage of their data and it is presented to users when they decide to use such a PT during the policy creation.

Recall that the goal of this thesis is to propose a privacy-preserving framework for connected car environments but not to create new PTs. Thus, it is PT's responsibility to ensure that the perturbed data is still useful for different end-point services. If the output data from a PT has negative effects on certain service qualities, users should also be informed during the policy creation. Furthermore, our framework has no restrictions on the number of data sections a PT can modify at once. This allows PTs that manipulate multiple data sections (e.g., speed and location data) to be used in our framework as well. In this case, the consistency of the perturbed data needs to be taken care of by themselves.

Altogether, the information mentioned above needs to be included in PT's metadata. This metadata will then help system admins to decide whether to include a certain PT in our framework and let users know the capability and limitation (w.r.t. service quality and data consistency) of a certain PT during the policy creation phase.

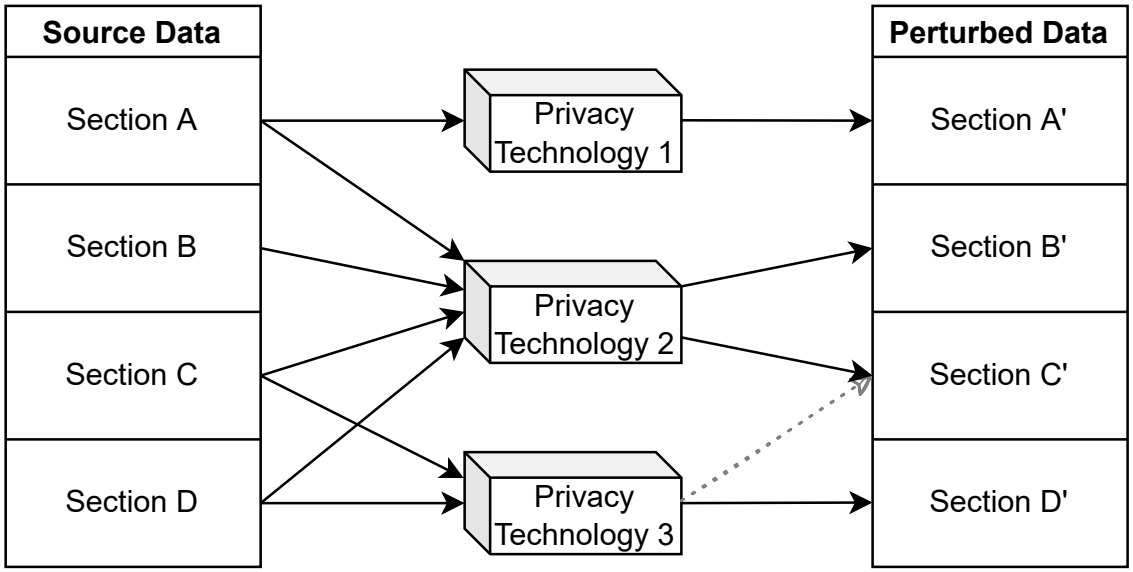


Figure 4.4: Privacy Technique Execution in Data Processor

Finally, we demonstrate the execution of different PTs in `data processor` with Figure 4.4. Recall that each PT defined in a policy action is targeting a specific data section or a set of data sections. As shown in Figure 4.4, the source data that arrives at the `data processor` has four data sections and the desired PTs derived by the `policy evaluator` are `PT1` (targeting data section `A`), `PT2` (targeting data section `B` and `C`), and `PT3` (targeting data section `D`).

The execution of PTs does not have to follow a specific order, as each PT process independently. In addition, each PT takes a copy of source data with all data sections as input (this is not illustrated in Figure 4.4 to avoid visual cluster). Thus, the execution of PTs can be parallel. Back to the example shown in Figure 4.4, `PT1` only modifies data section `A`, `PT2` modifies both data section `B` and `C`, `PT3` modifies two data sections `C` and `D` as well. The output data of each PT are then combined into the perturbed data by the `data processor` based on the targeting data section(s) defined in the policy `actions` field.

Notice that the data section `C` is being modified by both `PT2` and `PT3`. However, only the data section `C` from `PT2`'s output data is used in perturbed data, as the targeting data section for `PT3` is only `D` (that's why the arrow from `PT3` to data section `C′` of the perturbed data in Figure 4.4 is dotted). This means, users can use PTs that modify multiple data sections for the perturbation of only one or a subset of modified data sections. Certainly, users should be warned of the potential inconsistency in the processed data when they decide to use a PT in such way.

4.5 User Interaction

In this section, we briefly explain the interaction between users and our framework. Recall that users of our framework can be divided into three user groups: system admins, general users, and domain experts. The system admins are a group of users that determine the configurations of the framework. The general users are the actual user group that use our framework to protect their

privacy in connected car environments. Finally, the domain experts can be seen as a combination of both system admins and general users as they are able to modify configurations based on their domain knowledge same as admins, and they are required to create LAW and SAFETY related privacy policies similar to general users. In the following, we introduce the interactions between our framework and the system admin, general users and domain experts in Section 4.5.1, Section 4.5.2 and Section 4.5.3, respectively.

4.5.1 System Admin

There are three main tasks that a system admin needs to accomplish to configure our framework. As the overall purpose of system admins is different from general users, a specific `Admin Interface` should be provided to help admins to fulfill their tasks.

Recall that the available constraint types of policy `conditions` field and supported data sections of policy `actions` field are configured by system admins. Thus, the first task of system admins is to select supported constraint types and data sections for various car models of different car manufacturers. The available constraint types and data sections are derived by domain experts based on accessible sensors. To enable system admins to easily select the desired entries, the `Admin Interface` could provide admins with a list of all available constraint types and data sections with a checkbox beside each entry.

The second task of system admins is to specify PTs that can be used in our framework. In this case, the `Admin Interface` could provide a *Privacy Technique Market* where various PTs are exhibited. All PTs that are available in the market should be modularized and compatible with our framework, as described in Section 4.4. Besides, this market should also provide admins with detailed information of PTs, such as an introductory description and the metadata of PTs. Then, for each data section selected, system admins should choose at least one PT. If there are no available PTs for certain data sections, admins should either remove them from supported data sections' set or mark them in a way so that users can be informed when they define their privacy policies.

The last task of system admins is to configure the supported trust-levels of our framework and specify a default trust-level for newly installed apps. Furthermore, for each supported trust-level, system admins need to determine a default TLP, if experts have multiple TLPs defined for the same trust-level.

Finally, the system configuration is saved in a `config` file and is delivered to every CVs that have our framework deployed. Notice that this `config` file should be periodically updated so that our framework is always up-to-date with the latest techniques w.r.t. new sensors and PTs. Depending on the system management, this periodically updated could be either time-based (e.g., every month) or event-driven (e.g., whenever new techniques are available).

4.5.2 General User

The main interaction between general users and our framework is defining their individual and situational privacy policies. To hide the complicated policy format from users, a user-friendly `User Interface` should be implemented to ensure good user experience. As mentioned before, our framework does not recommend users to interact with the `User Interface` while driving due to the consideration of user's safety.

When users first interact with our framework, a short introduction of the Privacy4CV Framework should be displayed on the `User Interface`. Naturally, this interface should also allow users to explore more about our framework if they want to. Then, when users start to create new privacy policies, the `User Interface` should first let users choose whether they wish to create a TLP or STP. After that, a corresponding policy creation *interface* should be presented to users to help them construct privacy policy.

During the creation, the `config` file managed by system admins should be used to determine the supported trust-levels, constraint types, data sections and available PTs. Moreover, metadata of end-point services, broadcast channels and PTs should be provided in the *interface* as well. Naturally, this *interface* should also allow users to modify their self-defined policies when they are not driving. The policies are then re-deployed after each policy modification. For more discussion regarding the policy update time, please refer to Section 4.6.2.

There are no specific requirements for the appearance of the `User Interface`. Thus, the implementation of the `User Interface` should contain a *translator* that converts user's input of a privacy policy from the *interface* into the policy format described in Section 4.3.1. Furthermore, the *translator* can be used with an optional *optimizer*. The *optimizer* is responsible for policy optimizations such as combining policies that share the same policy `conditions` field or marking policies that could never be triggered as deactivated.

Apart from the *interface* and *translator* components, the implementation of the `User Interface` should also realize the following functionalities:

- **Conflict Warning**: when action conflicts or policy conflicts occurs, the `User Interface` should be able to detect them and inform users. It should also help users to resolve them by identifying possible conflict positions.
- **Overwrite Warning**: the `User Interface` should warn users when their STPs will be overwritten by expert-defined policies, namely when there are policy conflicts between STPs and expert-defined policies regardless of policy `priority`.
- **Suppression Warning**: users are free to choose whether they want to suppress certain data section(s) for specific end-point service or not, even if the data section(s) are marked as `essential` or `optional` in service's metadata. In this case, the `User Interface` should warn users of the potential negative effect of service qualities.
- **Privacy Leak Warning**: users should be warned if any of their privacy policies share their source data to untrusted parties directly.

- **Inconsistency Warning**: the `User Interface` should warn users when user-defined privacy policies may share inconsistent data to end-point services. The inconsistency detection should be conducted with help of expert-defined related data section groups (more explanation in Section 4.5.3), e.g., if individual PTs are used for different data sections in one group.

Notice that users are not forced to take any actions if any warnings listed above occur. However, users need to confirm that they are ignoring the warnings on purpose and are aware of the potential consequences.

4.5.3 Domain Expert

Domain experts are both responsible for system configurations and creating `LAW`, `SAFETY` related privacy policies. Thus, experts are the only user group that require interactions with both `Admin Interface` and `User Interface`.

As mentioned in Section 4.5.1, domain experts are in charge of determining available constraint types and data sections. The `Admin Interface` should provide experts with a list of accessible sensors based on different car manufacturers or even car models. The corresponding metrics that a sensor can measure, and its accuracy should be presented to experts as well. The domain experts should then derive and create available constraint types and data sections for various car models of different car manufacturers based on the information listed in the `Admin Interface`.

Furthermore, as mentioned in Section 4.5.2, domain experts are responsible for determining the so-called "related data section groups". This group contains data sections that are usually correlated or dependent from each other. For example, *time*, *speed*, and *location* data are in one related data section group as the knowledge of any two data sections in this group leads to the exact information of the third data section. In this case, the `Admin Interface` should provide experts with tools that could identify related data section groups easily.

Despite the system configuration aspect, domain experts are required to create `LAW`, `SAFETY` related privacy policies and default TLPs for each admin-defined trust-level. When defining STPs for `LAW` or `SAFETY` related functionalities, we trust our domain experts to always put users' safety before privacy. To create these policies, the same `User Interface` introduced in Section 4.5.2 is used. Notice that expert-defined privacy policies are global policies which are downloaded to every CVs that have our framework deployed. Thus, it is important that experts do not create conflicting policies. This means, every policy conflicts warned by the `User Interface` must be resolved. Other interactions between domain experts and the `User Interface` are similar to general users and therefore are omitted here.

There are no restrictions or requirements from our framework regarding the implementation of `Admin Interface` or `User Interface` mentioned above. In general, the `Admin Interface` should be available through a web service while the `User Interface` should be embedded in the infotainment system in CVs. On the other hand, domain experts also need to create privacy policies through the `User Interface`. Thus, a web service based `User Interface` could be provided as well. In addition, this version could also be provided to general users so that they could manage their privacy policies anytime. Then, users only need to download their predefined privacy policies every time they start their CVs.

4.6 Discussion

In this section, we discuss the remaining two open topics for our framework. In the following, we first discuss the consistency issue of historical data in Section 4.6.1 and then explore the possible policy update time in Section 4.6.2.

4.6.1 Historical Data

As mentioned in Section 4.2, a `historical data store` is necessary so that unmodified source data of CVs are available when critical situations occur, such as a car accident (as described in Section 3.1.4). To achieve this requirement, any read-only data store is sufficient for our framework.

On the other hand, this data store also provides historical data as source data for different end-point services. For this purpose, the core issue is the potential data inconsistency for data that is shared to an end-point service multiple times. For example, a certain data was first shared to an end-point service as current data from the car data stream. After some time, this end-point service requests to access this data from the `historical data store` again. If the data shared in these two cases are inconsistent, our framework could be exploited in an oracle attack, as explained by Coombs [Coo19], to crack certain PT.

To resolve the potential data inconsistency issue, the `historical data store` or a separate data store is required to save the historical policies as well. In addition, the mapping of the corresponding randomness used by a PT for a certain historical data should also be saved. Clearly, this increases the storage space needed for the `historical data store`. However, with the extra information recorded, the same PTs (including the randomness used) can be re-applied to a certain historical data. When an end-point service requires to re-access it, the same data that the end-point service received before can be provided.

Another issue that is worth to mention here is policy changes between multiple data accesses of the same data. Assume that data A' was first shared to an end-point service based on policy P from current data stream. In the meantime, the original data A, the policy P and the corresponding randomness used by PTs defined in P are saved in `historical data store`. After some time, the user decides to modify P and change it into policy P'. From that time, the data received by that end-point service is processed based on the new policy P'. Now, when the end-point service requests to access data A again, two options are possible:

1. the same data A' should be forwarded
2. data A'' which is generated by applying P' on data A should be forwarded

Clearly, the data consistency is guaranteed for the first option as the end-point service always gets the same data A'. However, the latest privacy demands from the user for that end-point service is left unconsidered. On the contrary, the second option does take the user's current privacy requirements into consideration, but the end-point service might notice the inconsistency of data A. As there are both benefits and drawbacks from these two options, there is no absolute correct solution for this particular situation. Thus, it is the user's choice which option should be used for this specific scenario. Therefore, the `User Interface` should provide users with the ability to specify whether a modified or newly created policy should be applied to historical data or not.

4.6.2 Policy Update Time

In general, our framework recommends users to create or modify their privacy policies only when CVs are parked. Thus, we do not expect any policy changes while CVs are moving, namely while data are generated to the data stream continuously. However, we cannot ignore the case where passengers update policies when CVs are on the move. In this case, the policy update time could be an interesting point.

As mentioned before, one possible point of time is whenever CVs are parked, namely the generation of data stream stops. In this case, the policies that are changed during driving time are first saved and then re-deployed once CVs are parked. This means, the live change of policies has no effect on already deployed policies. Therefore, for end-point services, the data they received are always consistent (w.r.t. previous data) as there are no policies changes after the data stream of CVs starts. On the other hand, one drawback of this point of time is that users' latest privacy demands cannot be fulfilled immediately. If users really want to have their modified policies to be deployed as soon as possible, they need to find a place to stop their CV and restart it.

The other possible solution for policy update is live update. This means, no matter if the CV is moving or not, policy changes are taken into effect the moment users finished the modification. If the techniques used to implement the `policy evaluator` supports live policy update, then this solution can be realized with minimal cost. The only drawback of this solution is that the end-point services may notice the policy change, since a data inconsistency might occur in the processed data stream they received.

However, if the underlying techniques do not support policy re-deployment on-the-fly, then the implementation of this solution may introduce severe side effects. For example, one possible implementation could be that passengers are provided with the ability to trigger the policy update by themselves, even if the CV is stilling moving. Then, any data shared during the re-deployment time is unprotected. Furthermore, it is also easier for end-point services to notice the policy change with this kind of implementation.

Similar to Section 4.6.1, both policy update times have their advantages and disadvantages, so it is users' choice what update method should be used in our framework.

5 Proof-of-Concept

To demonstrate our framework, we develop a proof-of-concept demo in Java. The goal of our implementation is to demonstrate and verify our framework. Therefore, only the core concept of our framework is realized. The underlying car data and various end-point services are only mocked and simulated. Furthermore, we develop a light-weight user interface for demonstration purposes. In the following, we first introduce the technologies used in our demo in Section 5.1. Then, we describe the implementation of our demo in Section 5.2. Finally, we present our demo with different use cases in Section 5.3.

5.1 Technologies

Before we explain the implementation of our proof-of-concept demo, we first introduce the technologies used in our demo.

The main technology of our demo is **Apache Kafka** [Apa] which is an open-source event streaming platform. The key concept of `kafka` is the `topic`. In `kafka`, a `topic` is a log of events where each event is modeled as a key-value pair. Then, applications can interact with event streams by publishing (write) or subscribing (read) to the desired `topic`. In our demo, we use the `kafka` platform provided by Confluent [Con] to simulate the underlying data stream from the car data `Interface` to different end-point services. In our implementation, we use `Producer API` to publish mock car data into data streams and use `Kafka Stream API` to realize `policy evaluator` and `data processor` from Privacy4CV Framework as stream processing applications. The end-point services are then simulated as consumers using `Consumer API`.

The other major technology in our demo is **Drools** developed by Red Hat and its community [Hat]. `Drools` is a business rules management system solution, however, only its business rules engine is used in our demo. This rules engine stores, processes, and evaluates data against predefined rules that are written in Drools Rule Language (DRL). Each rule defined by this language has a *conditions* section and an *actions* section. The *conditions* are used to evaluate data points, and *actions* are executed on matching data points. In our demo, we use `Drools`' business rules engine in our `policy evaluator` for policy evaluations.

Apart from these two technologies, we deploy the `kafka` platform provided by Confluent in **Docker**. We also use **Jackson** library as the JSON parser in Java, **Java Faker** library for generating mock car data, **opencsv** library for writing and reading data from `csv` files, and **JavaFX** client toolkit to build the demonstration interface.

5.2 Implementation

For the proof-of-concept demo, not every component of our framework is implemented. As mentioned before, only the `policy evaluator` and `data processor` are realized in the demo. The two data stores: `policy store` and `historical data store` are omitted in our demo as many traditional data stores with desired access control could fulfill the purposes of them and the data interaction of these two stores are less interesting compared to the rest of the data flow. We also implemented a light-weight user interface for demonstration purposes in which only the conflict detection functionality, mentioned in Section 4.5, is implemented.

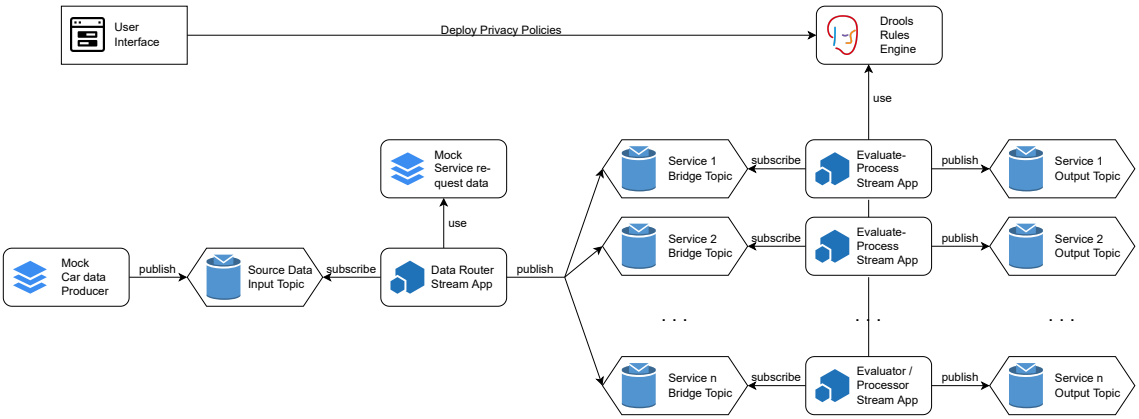


Figure 5.1: Architecture for Proof-of-Concept Demo in Edge Environment

Our demo is fully developed on `kafka` stream platform. All stream processing components are communicated through publishing to or subscribing from various `topics`. As depicted in Figure 5.1, the data stream is produced by the `Mock Car data Producer`. This producer generates mock car data on demand and publishes them to the `Source Data Input Topic`. Then, the first stream app `Data Router` subscribes data from this input `topic` and routes it to different bridge `topics` based on the mock service request data (i.e., mock service metadata that should be contained in the certificate issued by the authority). Recall that we do not implement actual end-point services in our demo. Thus, the interaction between an end-point service and our framework is simulated by the `Data Router`.

After that, for each end-point service, an `Evaluate-Process` stream app is deployed. Notice that both `policy evaluator` and `data processor` of Privacy4CV Framework are implemented within this single stream app. For the `policy evaluator`, we use the rules engine from `Drools` to evaluate privacy policies defined by users. This requires that all privacy policies are converted to rules that are written in DRL. The transformation of privacy policies from the policy format described in Section 4.3 to rules that `Drools` supports is provided by our user interface. Furthermore, thanks to the *actions* section in each `Drools`' rule, we do not need to explicitly implement the `data processor` but to put the corresponding PTs in a rule's *actions* section, so that they will be automatically executed when the rule is evaluated to `true`.

Then, for each data point read from the bridge `topic` for the specific end-point service, `Evaluate-Process` stream app first evaluates the data against all rules deployed in `Drools`' rules engine and then applies PTs on that data according to evaluation result. Finally, the stream app

publishes the processed data to the dedicated output `topic` for that end-point service and the consumption of data from the end-point service is simulated using a `kafka` consumer (which is omitted in Figure 5.1).

Each mock car data generated in our demo has four data sections: META, STATUS, SPEED, LOCATION. As shown in Figure 5.4a, META section contains an `UUID` that represents the car identifier and a `String` that indicates the car brand. STATUS section contains a `boolean` that specifies whether the CV is currently in an accident and two `double` values that represent the fuel consumption (FC in Figure 5.4a) and emission, respectively. SPEED section only has a `double` value for the current speed and LOCATION section contains two `double` values that specify the longitude and latitude (in meters) of the CV, respectively. In addition, each mock data is associated with a timestamp indicating when the data is generated and some environmental information, such as surrounding area or speed limit (not all listed in Figure 5.4a).

There are several PTs provided in our demo. As mentioned in Section 4.4, we implement them in a modular manner, so that when users change the desired PT in policies, no code needs to be modified. In the following, we shortly describe the PTs used in our demo for different data sections.

First, for the META data section, we provide `pseudonymization` where user's car identifier is replaced with a pseudonym, and `de-identification` where the identifier is being removed in the output data. Then, for the SPEED data section, three PTs are available: `aggregation` allows users to use the moving average of its SPEED data as their current speed, `randomization` allows users to use a random number in a given range as their current speed and `bounding` enables users to bind their current speed to a given boundary. Finally, for the LOCATION data section, we implement `rounding` which allows users to round their longitude and latitude either to the nearest integer or the nearest landmark, and `randomization` based on randomize, N-Rand, N-Mix techniques, as described by Wightman et al. in [WCJ+11].

Furthermore, for our demo system, we configured it to support four data sections of mock car data and three trust-levels: `TRUSTED`, `SEMI-TRUSTED`, and `UNTRUSTED` where `UNTRUSTED` is used as default trust-level for newly installed apps. For each trust-level, a default TLP is provided. For `TRUSTED` end-point services, TLP `policy-0` is defined where the very original data are shared. The TLP `policy-1` for `SEMI-TRUSTED` services is listed in Listing 4.2 where only `pseudonymization` is applied to the car identifier in META section. Similarly, the `UNTRUSTED` TLP `policy-2` also defines `pseudonymization` on the car identifier, and it specifies `aggregation` on SPEED data and `randomization` for LOCATION section as well.

In addition to these default TLPs, we prepared two expert-defined STPs for `LAW` and `SAFETY` tagged services, respectively. The `LAW` related STP `policy-3` is only activated when an accident occurs. In this case, all original data of car META, STATUS, and LOCATION is shared to the end-point service. As for the `SAFETY` related STP `policy-4`, it defines that no PTs should be used on SPEED and LOCATION data sections but `pseudonymization` is still applicable to the car identifier.

5.3 Demo

In this section, we demonstrate our framework based on different scenarios described in Section 3.1. For our demo, we combine the motivation scenario and all its extended scenarios into one story. As shown in Figure 5.2, user Alex is driving a CV from his workplace to home. Shortly after he

leaves the workplace at $(0.5, 0.5)$, he feels hungry. Thus, he stops at $(3, 3)$ to search for a restaurant nearby. Then, he drives to $(5, 1.5)$ to have his dinner and continues to drive home. However, Alex is very tired, he overlooks the speed limit sign in campus area which results in a car accident at $(7, 5)$ later. Hence, he has to take a detour to repair his CV at $(4, 7)$ before he finally goes home.

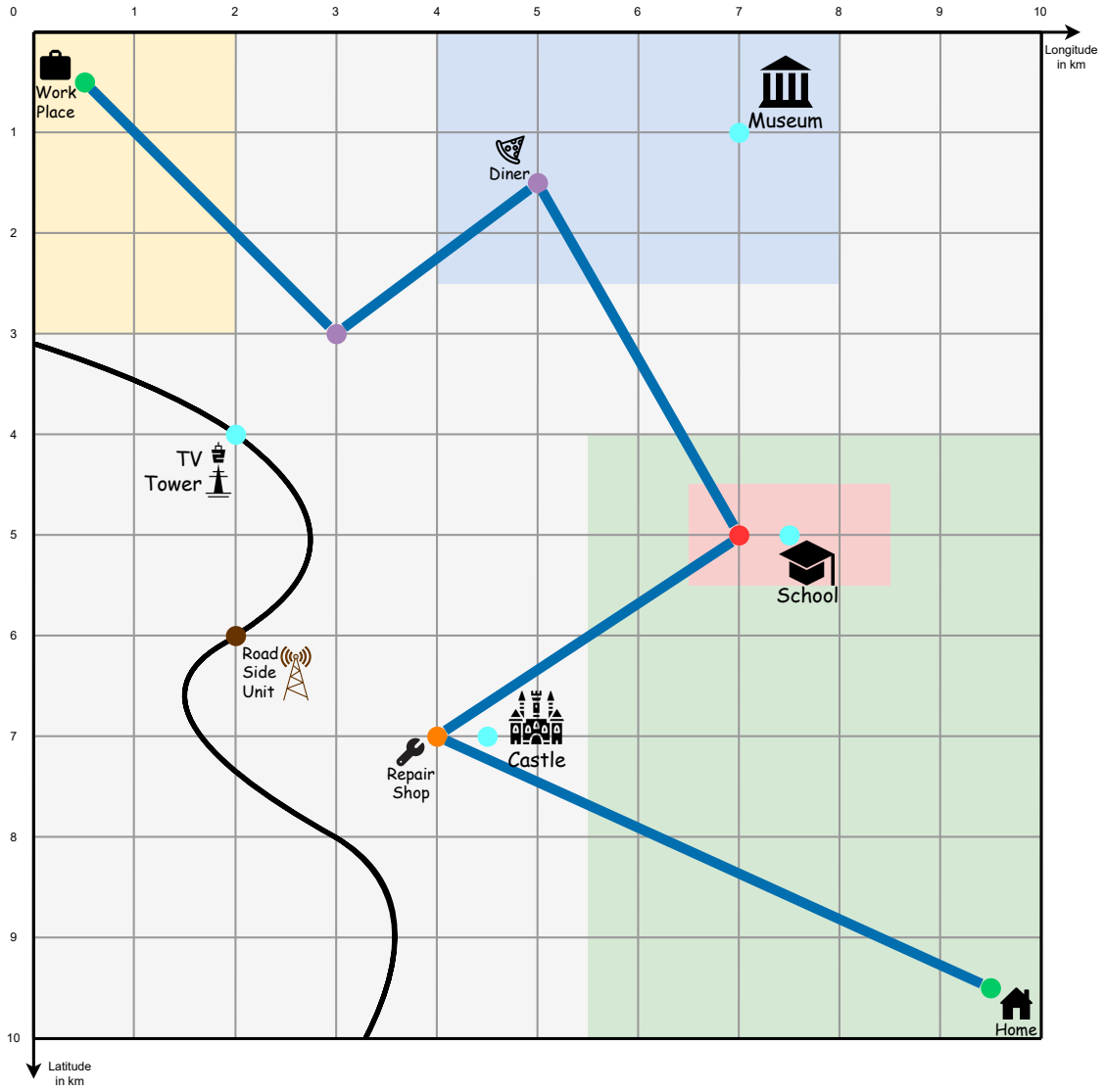
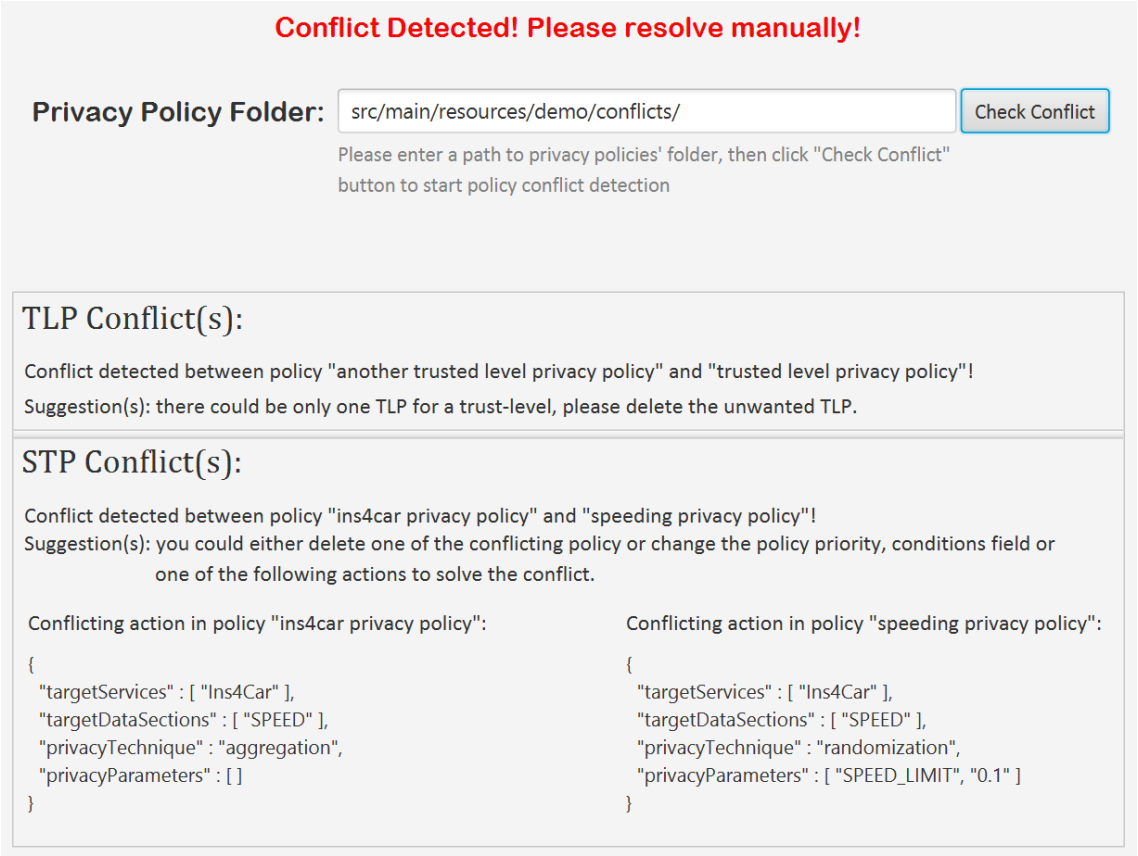


Figure 5.2: Demo Story Map

The dark blue line in Figure 5.2 illustrates the trajectory of Alex in the story. All special points, such as the stop point (purple dot), accident point (red dot), and landmarks (light blue dot) are highlighted on the map. In addition, industrial area (yellow block), business area (blue block), residential area (green block), campus area (red block), and suburban area (light gray block) are marked in Figure 5.2 as well. Recall that our framework can be deployed in both edge and fog environment. Thus, we included another road in the map (black line) with an RSU being deployed on this roadside at $(2, 6)$. In the following, we first present our demo in edge environment (i.e., within Alex's CV) based on the demo story in Section 5.3.1 and then exemplify two use cases of our framework in fog environment (i.e., within RSU) in Section 5.3.2.

5.3.1 Edge Environment

Before Alex drives his CV for the first time, he created several privacy policies for potential situations he might encounter. Before deploying these policies to `Drools`' rules engine, potential conflicts between policies need to be checked and resolved. For our demo story, we prepared a set of privacy policies that contain a policy conflict in TLPs and a conflict in STPs.



(a) Policy Conflict Detection

(b) Convert Policy to Rules

Figure 5.3: Demo User Interface

As shown in Figure 5.3a, when Alex presses the *Check Conflict* button in our demo interface, policy conflicts are checked. Then, the result of the conflict detection is highlighted at the top of the interface with red text to indicate that at least one conflict is detected and green text to inform users that no conflicts are found. When there are conflicts between policies, all conflicting policies are listed at the bottom of the interface and suggestions are provided to help users resolve the conflicts.

When all conflicts are resolved, Alex should see the demo interface as depicted in Figure 5.3b. Here, he could click on *Convert All* button to transform all privacy policies to rules written in DRL. When the transformation is finished, he is informed through the orange text displayed at the top of the interface.

For Alex's trajectory shown in Figure 5.2, we generated a list of 330 mock car data points. For a better presentation of source data and processed data, we route outputs of `Mock Car data Producer` and `kafka` consumers of different end-point services to tables in our demo interface. In the following, we demonstrate the capability of our framework based on different events in demo story.

Normal Driving

In this scenario, Alex has two STPs defined for two different `COMMERCIAL` end-point services: his car manufacturer's app *manu*4*car* and his car insurance's app *ins*4*car*. For *manu*4*car*, the required data sections are META and STATUS. For this app, Alex created a STP `policy-5` that allows our framework to share his original car STATUS data to *manu*4*car*. However, for META data, it can only be shared after PT `de-identification` has been applied. Then, for *ins*4*car* which requires META, SPEED, and LOCATION sections, Alex has defined another STP `policy-6` to share his car META data with `pseudonymization` used on car identifier, his aggregated car SPEED data, and his car LOCATION data which is randomized within a 50 meters' range.

Figure 5.4a shows the unmodified source car data when Alex starts his CV. Figure 5.4b displays the processed data for *manu*4*car* where car META and STATUS are the only data sections that are available, and the car identifier `ID` is removed based on Alex's privacy demand. Analogously, Figure 5.4c lists the processed data for *ins*4*car*. In Figure 5.4c, we can see that the car identifier is being substituted with a pseudonym (marked with prefix "pd_" for demonstration purpose) and the moving average is being used on car SPEED data so that it looks much smoother. Finally, comparing the LOCATION data shown in Figure 5.4a and Figure 5.4c (displayed in meters), we can see clearly that the shared LOCATION data is different from the original data and the difference is within 50 meters.

Finding Restaurant

After driving for a while, Alex feels hungry and decides to eat at a restaurant nearby. As he does not have any services installed in his CV that provide this functionality, he downloads a new app *find-restaurant* while he is parked. When the download is finished, Alex uses it immediately without defining any STPs for it. Thus, the default trust-level `UNTRUSTED` is assigned to *find-restaurant*, and TLP `policy-2` is used to protect Alex's privacy when he is using the app.

Timestamp	Meta		Status			Speed	Location		Area
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude	
2022-04-15T16:30:10	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	0.00	0.00	0.00	500.00000	500.00000	INDUSTRIAL
2022-04-15T16:30:15	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.28	92.57	39.72	539.00717	539.00717	INDUSTRIAL
2022-04-15T16:30:20	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.45	96.42	37.08	575.42095	575.42095	INDUSTRIAL
2022-04-15T16:30:25	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.29	96.24	39.90	614.60853	614.60853	INDUSTRIAL
2022-04-15T16:30:30	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.23	92.31	40.17	654.05784	654.05784	INDUSTRIAL
2022-04-15T16:30:35	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.72	97.72	43.98	697.24944	697.24944	INDUSTRIAL

(a) Original Car Data

Timestamp	Meta		Car Status			Speed	Location	
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude
2022-04-15T16:30:10		S-Car	false	0.00	0.00			
2022-04-15T16:30:15		S-Car	false	4.28	92.57			
2022-04-15T16:30:20		S-Car	false	4.45	96.42			
2022-04-15T16:30:25		S-Car	false	4.29	96.24			
2022-04-15T16:30:30		S-Car	false	4.23	92.31			
2022-04-15T16:30:35		S-Car	false	4.72	97.72			

(b) Processed Car Data for *manu4car*

Timestamp	Meta		Car Status			Speed	Location	
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude
2022-04-15T16:30:10	pd_8ada0c00-6c12-4d5b-9b53-1f72d48e4c47	S-Car				0.00	481.96637	517.11588
2022-04-15T16:30:15	pd_fae55950-89c6-4435-93fc-53d3a7fcf70c	S-Car				9.93	557.14023	515.67869
2022-04-15T16:30:20	pd_fb69af1c-d5f4-4e39-b8de-de25995105d2	S-Car				15.36	600.79197	586.51263
2022-04-15T16:30:25	pd_614c4c86-6f48-40ba-a975-0aba002cbdc5	S-Car				19.45	643.30316	634.55492
2022-04-15T16:30:30	pd_79ec32f0-c06b-4005-9054-a3979d0414f3	S-Car				22.41	677.19518	678.03229
2022-04-15T16:30:35	pd_fdacc23b-9e1a-4762-8255-da3628c2a827	S-Car				28.69	669.84700	686.80385

(c) Processed Car Data for *ins4car*

Figure 5.4: Data during normal driving

Timestamp	Meta		Status			Speed	Location		Area
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude	
2022-04-15T16:37:00	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	0.00	0.00	0.00	3000.00000	3000.00000	SUBURBAN
2022-04-15T16:37:05	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	0.00	0.00	0.00	3000.00000	3000.00000	SUBURBAN
2022-04-15T16:37:10	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	0.00	0.00	0.00	3000.00000	3000.00000	SUBURBAN
2022-04-15T16:37:15	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	0.00	0.00	0.00	3000.00000	3000.00000	SUBURBAN
2022-04-15T16:37:20	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	0.00	0.00	0.00	3000.00000	3000.00000	SUBURBAN
2022-04-15T16:37:25	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	0.00	0.00	0.00	3000.00000	3000.00000	SUBURBAN

(a) Original Car Data

Timestamp	Meta		Car Status			Speed	Location	
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude
2022-04-15T16:37	pd_49183a9e-b7aa-4961-a60f-36574e85d284	S-Car					3011.38765	2960.69884
2022-04-15T16:37:05	pd_d842f4a2-8f56-4a2d-9247-3bcb880ab4ad	S-Car					3023.64943	2972.25831
2022-04-15T16:37:10	pd_d9d90ee2-b95d-4eec-9f8d-e88ab1d2d065	S-Car					3021.07182	2954.95924
2022-04-15T16:37:15	pd_f7d6b407-385e-4568-9f98-43b610faebb4	S-Car					2964.87501	3015.82843
2022-04-15T16:37:20	pd_4acfc4c6-7dd8-48c0-84ce-8034307711be	S-Car					3036.05913	3019.07043
2022-04-15T16:37:25	pd_5014652d-1d22-4c8e-889f-0c8d6cdc0927	S-Car					3030.46130	3005.69870

(b) Processed Car Data for *find-restaurant*

Figure 5.5: Data during finding restaurant

According to the certificate issued by the authority (mocked in demo), *find-restaurant* only requires car META and LOCATION. Therefore, although the default TLP for UNTRUSTED services has three privacy actions defined for META, SPEED, and LOCATION, separately, only the processed car META and LOCATION are shared with *find-restaurant*. As shown in Figure 5.5b, it is clear that a pseudonym is used for Alex's car identifier and the LOCATION is also randomized compared to the source data listed in Figure 5.5a.

Speeding in Campus Area

When entering campus area, Alex missed the new speed limit sign (30 km/h) and continues driving at a speed around 50 km/h which is significantly higher than the new speed limit. As Alex knows he sometimes overlooks the speed limit sign, he had a specific STP `policy-7` defined for *ins*4*car* in this situation so that his car insurance company does not detect his speeding behavior. The policy listed in Listing 4.1 is exactly the same STP we used in our demo for this scenario. In this STP, Alex demands that when he is speeding in a campus area during a certain time, a random SPEED data around the current speed limit should be shared to *ins*4*car*. Recall that Alex has another STP `policy-6` defined for *ins*4*car* in the "Normal Driving" scenario where the aggregated SPEED data should be shared. This will be overwritten by `policy-7` as Alex sets a higher priority of `policy-7` than `policy-6`.

Timestamp	Meta		Status			Speed	Location		Area
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude	
2022-04-15T17:26:20	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.30	96.98	46.50	6655.57826	4397.26196	RESIDENTIAL
2022-04-15T17:26:25	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.58	99.32	46.94	6687.92254	4453.86445	RESIDENTIAL
2022-04-15T17:26:30	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.35	96.83	51.16	6723.17941	4515.56397	CAMPUS
2022-04-15T17:26:35	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.60	90.17	50.04	6757.65803	4575.90155	CAMPUS
2022-04-15T17:26:40	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.69	95.92	53.56	6794.56574	4640.49004	CAMPUS
2022-04-15T17:26:45	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.42	96.57	53.41	6831.37094	4704.89914	CAMPUS

(a) Original Car Data

Timestamp	Meta		Car Status			Speed	Location	
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude
2022-04-15T17:26:20	pd_57c829f2-d50a-4edc-ada3-8e3bc245cab0	S-Car				51.22	6674.30650	4369.46837
2022-04-15T17:26:25	pd_a4e01234-7982-401d-8a69-85ef5156c5d0	S-Car				48.79	6672.73181	4439.63383
2022-04-15T17:26:30	pd_db4d8082-2859-487e-9922-545fa396d209	S-Car				32.20	6737.80149	4535.35436
2022-04-15T17:26:35	pd_f9eeed54-121a-4b99-be98-49dbeb8eb3e9	S-Car				28.13	6737.47759	4546.59549
2022-04-15T17:26:40	pd_31774bf6-5bf5-426e-807a-87efb1f9fc0a	S-Car				32.56	6817.21994	4655.37077
2022-04-15T17:26:45	pd_ed3305aa-1679-46bb-91a2-c32d0f9293f2	S-Car				32.50	6853.72488	4732.81285

(b) Processed Car Data for *ins*4*car*

Timestamp	Meta		Car Status			Speed	Location	
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude
2022-04-15T17:26:20	pd_9151da59-0bee-4bce-9c82-a272a64604f0	S-Car				46.50	6655.57826	4397.26196
2022-04-15T17:26:25	pd_63c741ad-7917-4e31-a47a-79c415def438	S-Car				46.94	6687.92254	4453.86445
2022-04-15T17:26:30	pd_ec9a798b-0f65-4c16-8bd2-f1fe2d994607	S-Car				51.16	6723.17941	4515.56397
2022-04-15T17:26:35	pd_a985647e-0617-4034-b0fb-c12a182e5b55	S-Car				50.04	6757.65803	4575.90155
2022-04-15T17:26:40	pd_faebeb29-1bb2-4a29-891a-a5ef30d3ed50	S-Car				53.56	6794.56574	4640.49004
2022-04-15T17:26:45	pd_1bdd4721-e6ff-4d5a-96c3-9a850a1b7950	S-Car				53.41	6831.37094	4704.89914

(c) Processed Car Data for SAFETY related Broadcasting

Figure 5.6: Data during speeding in campus area

As shown in Figure 5.6b, when Alex has not entered campus area (first two rows), the SPEED data shared to *ins*4*car* still follows the privacy action defined in the policy `actions` field of `policy-6`. Thus, PT `aggregation` is used to process these data. When Alex enters campus area (start from third row), the privacy action for SPEED data defined in the policy `actions` field of `policy-7` is executed. Hence, we see that the data shared with *ins*4*car* from that point of time is around 30 km/h instead of the original or aggregated SPEED data.

Recall that in Section 3.1.3, we discussed that sharing false SPEED and LOCATION data to SAFETY related services usually increases the possibility of a car accident. Thus, for other CVs on the road that are near Alex, the broadcast data is processed based on expert-defined SAFETY STP `policy-4`. As shown in Figure 5.6c, the unmodified SPEED and LOCATION data are broadcast to other CVs with the car identifier being replaced by a pseudonym as defined in `policy-4`.

Car Accident

Unfortunately, a car accident occurs because of Alex's speeding behavior. The moment Alex's CV found itself in an accident state, the *first-aid* app is activated. As its name indicates, the functionality of this app is to provide drivers with immediate assistance when an accident arises. As this app is being classified as a LAW related service by the authority, the expert-defined STP `policy-3` is used to process data for *first-aid*. As shown in Figure 5.7, when the attribute `inAccident` in car STATUS becomes `true`, the unmodified source data of required data sections: META, STATUS, and LOCATION are shared to *first-aid* as defined in `policy-3`.

Timestamp	Meta		Status			Speed	Location		Area
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude	
2022-04-15T17:27:00	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.77	91.14	54.30	6931.89146	4880.81006	CAMPUS
2022-04-15T17:27:05	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.63	98.83	52.78	6968.25808	4944.45165	CAMPUS
2022-04-15T17:27:10	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00	0.00	7000.00000	5000.00000	CAMPUS
2022-04-15T17:27:15	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00	0.00	7000.00000	5000.00000	CAMPUS
2022-04-15T17:27:20	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00	0.00	7000.00000	5000.00000	CAMPUS
2022-04-15T17:27:25	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00	0.00	7000.00000	5000.00000	CAMPUS

(a) Original Car Data

Timestamp	Meta		Car Status			Speed	Location	
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude
2022-04-15T17:27:10	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00		7000.00000	5000.00000
2022-04-15T17:27:15	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00		7000.00000	5000.00000
2022-04-15T17:27:20	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00		7000.00000	5000.00000
2022-04-15T17:27:25	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00		7000.00000	5000.00000
2022-04-15T17:27:30	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00		7000.00000	5000.00000
2022-04-15T17:27:35	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	true	0.00	0.00		7000.00000	5000.00000

(b) Processed Car Data for LAW related Services

Figure 5.7: Data during car accident

Timestamp	Meta		Status			Speed	Location		Area
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude	
2022-04-15T18:02:25	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.79	94.62	64.20	5096.47337	6269.01775	SUBURBAN
2022-04-15T18:02:30	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.51	96.26	74.13	5010.81115	6326.12590	SUBURBAN
2022-04-15T18:02:35	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.57	92.20	63.88	4936.98670	6375.34220	SUBURBAN
2022-04-15T18:02:40	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.57	93.05	72.88	4852.75929	6431.49381	SUBURBAN
2022-04-15T18:02:45	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.63	90.03	67.87	4774.32868	6483.78088	SUBURBAN
2022-04-15T18:02:50	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car	false	4.66	95.55	72.95	4690.02080	6539.98613	SUBURBAN

(a) Original Car Data

Timestamp	Meta		Car Status			Speed	Location	
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude
2022-04-15T18:02:25	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car				64.20	5096.47337	6269.01775
2022-04-15T18:02:30	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car				74.13	5010.81115	6326.12590
2022-04-15T18:02:35	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car				63.88	4936.98670	6375.34220
2022-04-15T18:02:40	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car				72.88	4852.75929	6431.49381
2022-04-15T18:02:45	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car				67.87	4774.32868	6483.78088
2022-04-15T18:02:50	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car				72.95	4690.02080	6539.98613

(b) Processed Car Data for *navigation*

Timestamp	Meta		Car Status			Speed	Location	
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude
2022-04-15T18:02:25	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car					5096.47337	6269.01775
2022-04-15T18:02:30	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car					5010.81115	6326.12590
2022-04-15T18:02:35	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car					4500.00000	7000.00000
2022-04-15T18:02:40	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car					4500.00000	7000.00000
2022-04-15T18:02:45	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car					4500.00000	7000.00000
2022-04-15T18:02:50	59aa5a36-954c-479e-ba10-07493e4f5437	S-Car					4500.00000	7000.00000

(c) Processed Car Data for *location-share*

Figure 5.8: Data for TRUSTED services with different policies

Location Sharing

After the car accident status has been lifted by law enforcement, Alex decides to repair his CV before going home, so that his family will not know about this car accident. Before he drives to the repair shop, he remembers that the *location-share* app is always active in his CV so that family members always know the location of each other. As Alex assigned the trust-level TRUSTED to this app, the default TLP `policy-0` always shares Alex's unmodified LOCATION data with the app. To hide the fact that Alex is going to a repair shop from his family, he created another STP `policy-8` for *location-share* only on that day where PT `rounding` to nearest landmark should be applied to his LOCATION data when he is within a 1 km range of the repair shop.

As depicted in Figure 5.8c, when Alex is near the repair shop located in $(4, 7)$, the LOCATION data shared to *location-share* is modified to $(4.5, 7)$ (LOCATION data is presented in meters in Figure 5.8c) which is the nearest landmark *Castle* to the repair shop. In Figure 5.8 we also presented the processed data for *navigation* app, which is another TRUSTED app installed in Alex's CV that requires the LOCATION data. If Alex had not created STP `policy-8`, these two apps should get the same LOCATION data, as TLP `policy-0` should be applied to both of them. However, as STPs always has higher priorities than TLPs, the privacy action defined in `policy-0` targeting LOCATION data is overwritten by privacy action defined in `policy-8` when its policy `conditions` field is evaluated

to `true`. Furthermore, as `policy-8` only declares one privacy action for LOCATION data, the META data for *location-share* is still processed based on `policy-0`, which also corresponds to the policy evaluation rule described in Section 4.3.4.

5.3.2 Fog Environment

As the main purpose of our proof-of-concept demo is to present our framework in edge environment, we only prepared two use cases for our demo in fog environment. Thus, the implementation for fog environment is much simpler. As shown in Figure 5.9, the fog environment demo is still developed on `kafka` stream platform but has only two pipelines. The architecture of both stream pipelines is identical where stream apps read mock car data from individual input `topics`, process data within the apps, and then publish the data to dedicated output `topics`.

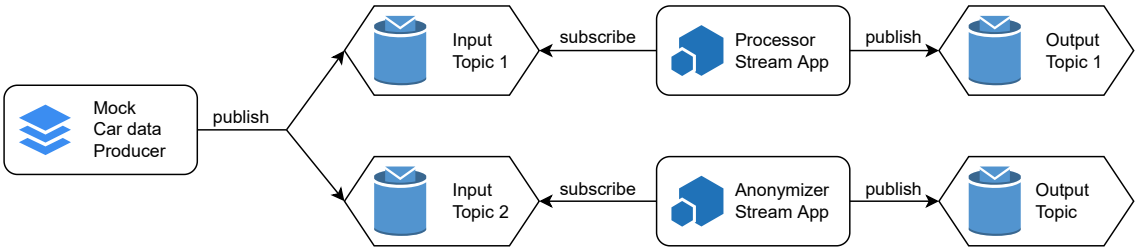


Figure 5.9: Architecture for Proof-of-Concept Demo in Fog Environment

The first pipeline is dedicated to demonstrating the use case, where our framework can be used in an RSU to protect users' privacy same as in a CV. However, unlike in an edge environment where source data are usually generated from the same CV and are being processed one by one, source data for fog environments (gathered or measured by RSUs) always contain various CVs' information and should be processed as a group. This also enables the usage of PTs that are designed for data groups. In our fog environment demo, we use `windowing operation` provided by `kafka` to implement the PT for groups of data in `Processor` stream app and we decided not to include a rules engine here to keep the example simple. Thus, there are no policies used in fog environment demo.

As mentioned in Section 4.2, an RSU with our framework deployed can be treated as a trusted, centralized anonymizer for CVs that have our framework equipped as well. The second pipeline depicted in Figure 5.9 aims to provide another example for this use case. For the fog environment, we assume that our framework is deployed in an RSU located at $(2, 6)$ (marked as brown dot in Figure 5.2) that has a communication range of 1 km, and we generated 100 mock car data points near the RSU as source data. In the following, we demonstrate the capability of our framework for these two use cases in detail.

Process data gathered by RSU

In this example, we use generated car data to simulate data gathered by the RSU. As listed in Figure 5.10a, the structure of mock car data in fog environment are the same as in edge environment. For this use case, we assume that the RSU publishes the average speed of CVs within its communication range every 15 seconds based on car types (indicated by attribute `Brand` in META

Timestamp	Meta		Status			Speed	Location	
	ID	Brand	inAccident	FC	Emission	Speed	Longitude	Latitude
2022-04-15T14:41:00	d236045c-466c-4592-8b96-78c89b5834ca	LKW-X	false	7.30	124.61	60.70	1.64846	5.52082
2022-04-15T14:41:02	0707bd39-0707-415b-a224-12bc7a183a02	LKW-Z	false	7.98	122.58	63.10	2.49872	6.41586
2022-04-15T14:41:03	36227ebd-f87a-4592-8a90-c23d92b9ad9a	LKW-Z	false	6.93	119.62	66.44	1.93676	5.61279
2022-04-15T14:41:05	cfdb6cc6-0bb3-48c0-b2c1-965edce81510	LKW-Y	false	6.39	123.91	65.69	1.56684	5.94041
2022-04-15T14:41:06	f9983dfd-71c6-493f-a442-b04ad3e17035	PKW-B	false	4.56	96.28	78.87	2.27891	6.10368
2022-04-15T14:41:08	a481acf5-fe9e-4015-b3c0-75cf01ad8733	PKW-C	false	5.05	94.45	85.74	2.44365	5.68296

(a) Generated Car Data

Timestamp	Car Type	Avg. Speed	Location		Environment Information		
			Longitude	Latitude	Neighbourhood	Speed Limit	Road Surface
2022-04-15T14:41:12	LKW	63.37	2.00000	6.00000	SUBURBAN	70.00	BITUMINOUS
2022-04-15T14:41:14	PKW	84.38	2.00000	6.00000	SUBURBAN	70.00	BITUMINOUS
2022-04-15T14:41:29	LKW	66.78	2.00000	6.00000	SUBURBAN	70.00	BITUMINOUS
2022-04-15T14:41:26	PKW	75.69	2.00000	6.00000	SUBURBAN	70.00	BITUMINOUS
2022-04-15T14:41:44	LKW	67.10	2.00000	6.00000	SUBURBAN	70.00	BITUMINOUS
2022-04-15T14:41:38	PKW	79.16	2.00000	6.00000	SUBURBAN	70.00	BITUMINOUS

(b) Average Speed Data based on Car Type

Timestamp	Meta		Cloaking Area			
			Top Left		Bottom Right	
	ID	Brand	Longitude	Latitude	Longitude	Latitude
2022-04-15T14:41:00	d236045c-466c-4592-8b96-78c89b5834ca	LKW-X	1.50	5.50	2.00	6.00
2022-04-15T14:41:02	0707bd39-0707-415b-a224-12bc7a183a02	LKW-Z	1.00	5.00	3.00	7.00
2022-04-15T14:41:03	36227ebd-f87a-4592-8a90-c23d92b9ad9a	LKW-Z	1.50	5.50	2.00	6.00
2022-04-15T14:41:05	cfdb6cc6-0bb3-48c0-b2c1-965edce81510	LKW-Y	1.50	5.50	2.00	6.00
2022-04-15T14:41:06	f9983dfd-71c6-493f-a442-b04ad3e17035	PKW-B	1.00	5.00	3.00	7.00
2022-04-15T14:41:08	a481acf5-fe9e-4015-b3c0-75cf01ad8733	PKW-C	2.00	5.50	2.50	6.00

(c) Spatial Cloaked Location Data

Figure 5.10: Data for fog environment

section where “PKW” represents passenger cars and “LKW” represents trucks). The receiver of this information could be an ITS, so that a corresponding traffic light timer can be adapted based on real-time traffic condition.

The PT used here is the group based average speed computation. Although it is not very complicated, it satisfies our goal in this use case where a specific user’s SPEED data is hidden, and the average SPEED data still expresses overall traffic condition in a degree. As shown in Figure 5.10b, the average SPEED data for passenger cars and trucks are computed separately. The average speed of passenger cars is around 80 km/h while the average speed of trucks is near 65 km/h. In addition, the location of the RSU and the environmental information sensed by the RSU are also published in the result data. Notice that the `Timestamp` in Figure 5.10b only represents the point of time when the last passenger car or truck has been seen in last 15 seconds, so this column may seem disordered.

RSU as trusted centralized anonymizer

In this use case, we consider the RSU as a trusted, centralized anonymizer. The generated data listed in Figure 5.10a are used to simulate data sent by CVs that require a spatial cloaking based on other CVs within the RSU's communication range. The PT used in this example is the Adaptive-Interval Cloaking Algorithms proposed by Gruteser and Grunwald [GG03]. This algorithm assumes that every user has the same degree of privacy k. Then for a given location of a user U and a set of other users' location, this algorithm cloaks the location of U into a region which contains at least k users.

Figure 5.10c shows the data processed by the RSU. The cloaked region produced by Adaptive-Interval Cloaking Algorithms is presented by the longitude and latitude of its top left and bottom right point. In our demo, the algorithm is applied to each requesting LOCATION data and 20 previous LOCATION data with k equals to 5. When the RSU first started, it does not process any incoming LOCATION data until k data points have been seen. Then, the RSU processes the first k data points at once and starts its normal processing for every incoming LOCATION data. This guarantees that no users' privacy demand will be violated if there are not enough data points. Notice that the car META data shown in Figure 5.10a and Figure 5.10c is only used to match data in this demo and can be omitted in other usages.

6 Evaluation

In this chapter, we first evaluate the Privacy4CV Framework in Section 6.1 based on the privacy requirements proposed in Section 3.2 and then discuss the limitations of the framework and this thesis in Section 6.2.

6.1 Evaluation

Recall that we introduced five essential and three optional privacy requirements for connected car environments. In the following, we only evaluate our framework regarding the five essential privacy demands: Transparency (before law), Safety First, Equality, Individuality, and Local Processing.

First of all, Transparency (before law) requires that law enforcement can always get source data of CVs especially when law-breaking behavior occurs. This is guaranteed in our framework by introducing the `historical data store`. As mentioned in Section 4.2, the `historical data store` is designed to be read only and saves a copy of every incoming source data for a certain period. In addition, the minimum preserving period is defined by the government authority so that users cannot turn off the `historical data store` by setting the period to 0. Furthermore, this data store is designed to be protected by access controls. This guarantees that only eligible parties (e.g., law enforcement) and our framework have access to the `historical data store` directly, and the very source data is not shared with any other parties (i.e., various end-point services) without being processed by our framework first.

Secondly, the Privacy4CV Framework always puts Safety before privacy. We achieve this by only allowing authorized experts to define `SAFETY` and `LAW` related STPs, and granting them higher priorities than any other user-defined STPs. As mentioned in Section 4.5.3, we always trust domain experts to put users' safety before privacy. Furthermore, by giving expert-defined STPs higher priority, the policy evaluation introduced in Section 4.3.4 guarantees that only expert-defined policy `actions` field will be executed even though users may have other `actions` fields defined for the same situation. This point is being demonstrated by our proof-of-concept demo in "Speeding in Campus Area" situation in Section 5.3.1 as well. In that scenario, although Alex defined a STP so that his car insurance company does not notice his speeding behavior, he has no right to define any STP for `SAFETY` related services or broadcast messages. Instead, the expert-defined STP is used to process the `SAFETY` related data that need to be broadcast to other CVs. Thus, as depicted in Figure 5.6, we see that the SPEED and LOCATION data broadcast to other CVs are unmodified, according to the expert-defined STP.

Furthermore, Equality demands that the information a third party could retrieve from a CV in both connected and non-connected car environments should be the same, unless users are willing to share. Our framework achieves this requirement by giving users the ability to define arbitrary privacy policies, that specify both individual privacy requirements for particular end-point services

or specific data sections and situational privacy demands for different situations or various privacy patterns. These privacy policies are then translated into multiple STPs that will be used in our framework to modify data before sharing them to `remote` end-point services. In addition, even if users forget to define any STPs before their first drive, their privacy is still protected by our expert-defined default TLPs.

Equality is being demonstrated in multiple situations by our proof-of-concept demo in Section 5.3.1 as well. First of all, in the "Normal Driving" scenario, the data Alex shares to *manu*4*car* and *ins*4*car* are modified based on his privacy demands so that no critical data are being shared unwillingly. Then, in the "Finding Restaurant" scenario, even if Alex forgets to define STPs for *find-restaurant*, his exact LOCATION data is not revealed because of the expert-defined default TLPs. After that, in the "Speeding in Campus Area" scenario, Alex's speeding behavior is completely hidden from his car insurance company, according to his privacy demand.

Moreover, our framework guarantees Individuality by providing high flexibility in defining privacy policies, especially for STPs. As explained in Section 4.3, there are no restrictions from our framework's side regarding defining policy `conditions` fields and `actions` fields. Thus, users are enabled to create different STPs for the same end-point service in distinct situations and for various privacy patterns or define different STPs for individual end-point services and data sections in the same scenario. The only limitation would be the underlying sensors equipped in CVs and available PTs included in the framework. However, this information is being periodically updated by the system admin so that it is always up-to-date. Individuality is demonstrated throughout the proof-of-concept demo. In our demo story, we see that Alex has multiple STPs defined for various end-point services in different situations. For example, in the "Location Sharing" scenario, although *location-share* and *navigation* are both `TRUSTED` apps, they get individually processed data as Alex has different policies defined for them.

Finally, Local Processing is guaranteed by our framework's design where only processed data are shared to `remote` services. As our framework is designed to work in both edge and fog environment, this enables users to use RSUs with our framework deployed as trusted, centralized anonymizers, so that more complex PTs can be used as well. Our proof-of-concept demo exemplifies this use case in Section 5.3.2, where a PT that requires a group of data is provided to each CV. Notice that our framework's architecture only guarantees that no source data will be shared without users' consent or without being modified first. It is PTs' responsibility that no critical data can be derived from the perturbed data.

6.2 Limitations

In Section 6.1, we evaluate the capability of Privacy4CV Framework regarding the five essential privacy requirements introduced in Section 3.2. However, the three optional privacy demands are not satisfied by our framework. In this section, we point out several limitations of our framework and this thesis.

Due to the time limitation of this thesis, we analyze the privacy requirements for connected car environments based on several assumptions and the motivation example. If time permits, a thoughtful user study of the privacy demands from driver's point of view should be conducted. Furthermore, privacy requirements from legal perspective should be taken into consideration as

well. In addition, when designing the Privacy4CV Framework, we used an abstracted model for CVs. The concrete architecture and the potential data flows between different components within CVs are not considered.

Then, for the three optional privacy requirements, both Transparency (before user) and Accountability require a monitoring or logging system that keeps track of data flows for different end-point services in the framework. Furthermore, for Accountability, it may require this monitoring system to follow data flows that even leave our framework to determine the responsible party when privacy leaks occur. This is far beyond the capability of our framework in the current version. Finally, Secure Deletion is aimed to give users ability to revoke their privacy decisions. This means, all participating end-point services need to support secure deletion methods based on a standard, which also indicates that potential modification or adaption of end-point services are required. However, this is against our framework's principle that no end-point services' business logic should be interfered with. Notice that this does not mean that these three optional privacy demands cannot be achieved in general, they are only restricted in our current framework's design.

Another limitation of our framework is regarding complex PTs that are only applicable on groups of data. Currently, these kinds of PTs can only be provided through our framework being deployed in fog environments. It is very inconvenient in places, such as suburban areas, where no RSUs are deployed. Finally, in this thesis, we assume that security in connected car environments is guaranteed. However, this is a very broad and rough assumption, and security itself is a very interesting research field in connected car environments.

7 Related Work

In this chapter, we review the related works that have been presented in recent years and examine the difference between them and our framework.

The Data Protection framework proposed by Duri et al. in [DGL+02] aims to achieve both data privacy and data security protection while enabling the data sharing. This framework has a layered architecture where each layer of hardware and software provides its own security functions. As for privacy protection, this framework also allows users to specify complex privacy policies and grants authorization for release of data by matching the individual's privacy policies with the data requests. In addition, this framework minimizes the amount of private data that leaves the trusted computing system by deploying data aggregation applications inside the trusted environment and only allowing aggregated results to be sent to service providers. However, this framework does not support privacy protections for particular situations or privacy patterns and there is no special attention given to PTs other than aggregation.

Plappert et al. [PZK+17] proposed a privacy-aware data access system for automotive applications. This system monitors and informs users of various data flows by enforcing every external service to implement the system. It also enables users to control third-party access to their personal data through privacy policies and privacy enhancing technologies. However, the privacy policy used in this system is usually applied globally on all services installed on the vehicle. Although it provides features, such as white-list and customized policy where users can set allowances for single data types, filter apps for specific functions or select specific services, there is no support from this system for situational or privacy pattern protections. Furthermore, the privacy enhancing technologies used in this system are fixed and broadcasting messages are not considered by this system as well. Moreover, as any access systems, this system may deny service requests based on the processing result of the policy.

Ghane et al. [GJK+21] proposed the Differentially Private Data Stream (DPDS) system that is designed for a distributed edge computing system where the server is untrusted. When transferring the data to the untrusted controller, vehicles subscribed to that controller form a group, and the data is first sent to the group leader. The group leader then compresses, perturbs, and filters the data before forwarding it to the controller. The DPDS system ensures that no source data will be sent to untrusted servers unmodified. However, source data are still being transferred from vehicles to vehicles unprotected. In addition, the DPDS system does not support users to define individual privacy demands nor provide situational privacy protections.

As the concept of Blockchain is getting popular these years, there are several frameworks proposed for connected car environments based on this technique. The Block4Forensic framework introduced by Cebe et al. [CEA+18] aims to provide a trustless, traceable, and privacy-aware framework for post-accident analysis. During the normal usage of vehicles, the hash of car data is periodically written to the Blockchain. These data should be disclosed to investigation parties after a car accident occurred. The nature of the Blockchain ensures the integrity of car data. The privacy of drivers is

protected by using the hash of data and pseudonym identities when they interact with the Blockchain. However, due to its focus on post-accident analysis, the privacy protection for other scenarios in connected car environments is not being considered in this framework.

Cebe et al. [CKO+19] proposed a Blockchain based data sharing framework for driver privacy and connected vehicle transportation. In this framework, the privacy is enabled by the channel feature of Hyperledger. It behaves like an access control system where organizations can only view data published in the same channel. However, there are no more privacy protections provided for data shared within a channel and there is no support for situational or privacy pattern protections. Li et al. [LGNS20] proposed another Blockchain based privacy-preserving framework for traffic management system. Similarly, this framework also uses the channel feature and access control to ensure intra-blockchain (i.e., within same channel) privacy. As it uses the same technology for privacy protection, it suffers from the same drawbacks as mentioned above.

In the following, we introduce another set of related works which, from our perspective, focus more on the privacy protection of a specific data type or a combination of data types. Thus, we do not consider them as general privacy-preserving frameworks for connected car environments, but we do believe that with a proper adaption, they can be used as PTs in our framework.

Joy and Gerla [JG17] proposed the notion of Haystack Privacy that aims to strengthen privacy yet maintain accuracy when answering counting queries. The mechanism of Haystack Privacy enables users to privatize their data independently based on a series of Bernoulli trials. When calculating the aggregating result, the expected value of the privacy noise due to the Bernoulli trials is calculated and removed.

Kalnis et al. [KGMP06] proposed a framework for preserving the anonymity of users issuing spatial queries to location based services. K-anonymity based transformations are used in the framework to compute exact answers for range and nearest neighbor queries. The framework also prevents the revelation of sensitive information about users.

Ma et al. [MZL+19] proposed a privacy-preserving mechanism RPTR that protects the privacy of a vehicle's real-time trajectory data release. The core idea of RPTR is to use a modified data, which is either a predicated data or a perturbed data based on the differential privacy's requirement and the used privacy budget, on trajectory data release.

Huang et al. [HLN+21] proposed an accountable and data sharing scheme (ADS) which provides data owners with the ability to identify the responsible data receiver(s) when data breaches occur. To achieve the accountability, ADS uses oblivious transfer protocol together with the zero-knowledge proof to embed the private key of data receiver into the shared data. When a data breach occurs, the private key can be automatically revealed so that the responsible data receiver can be identified. This technique may be very useful in helping our framework achieving accountability.

8 Summary and Future Work

Currently, privacy has been an active research field in connected car environments. In recent years, many new privacy-preserving frameworks have been proposed in other research. However, as introduced in Chapter 7, none of them provides protection for situational privacy demands or privacy patterns. In this thesis, we proposed a new situation-aware privacy-preserving framework for connected car environments. With the flexibility provided by our STPs, users are not only enabled to define their privacy demands for individual data section or end-point service, but also have the ability to specify privacy requirements for particular situations, i.e., privacy patterns.

Apart from STPs, our framework provides another set of privacy policies – TLPs, which help users to define global privacy demands based on trust-level and act as the last defense of users' privacy in circumstances where users forget to define any STPs. In addition, our framework does not interfere with any business logic of end-point services. The data perturbation in our framework only relies on different PTs that are implemented in a modular manner. This also enables easy switch of new technologies in the future. Then, for each data point generated in a CV, our framework ensures that only the data modified based on users' privacy demands will be shared.

Furthermore, our framework always puts law and users' safety before privacy. This is guaranteed by only allowing domain experts to define law and safety related privacy policies. Moreover, our framework provides a `historical data store`, where the unmodified source data of CVs are saved for a certain period, so that when an accident occurs, the source data is available for corresponding law enforcement or investigation parties. Finally, our framework is designed to be generic in two ways: 1) it can provide privacy protection for data sharing between CVs and remote end-point services or broadcast receivers, and 2) it can be deployed in both edge environments (e.g., on the CV) and fog environments (e.g., on the RSU).

As future work, we will focus on extending our framework to achieve optional privacy requirements, such as Transparency (before user) or Accountability, by integrating with new techniques, such as the ADS proposed by Huang et al. We also plan to expand our framework with other frameworks or techniques, such as Blockchain, for specific use cases. In addition, it is interesting to investigate the technical details of CVs so that our framework can handle peer-to-peer data communications and PTs, that require a group of data, can be used in edge environments as well. Furthermore, it is worthwhile to conduct user studies to understand the privacy demands from drivers' and manufacturers' perspective.

Bibliography

[Ala16] M. Alam. "The Software Defined Car: Convergence of Automotive and Internet of Things". In: *Wireless World in 2050 and Beyond: A Window into the Future!* Ed. by R. Prasad, S. Dixit. Cham: Springer International Publishing, 2016, pp. 83–92. ISBN: 978-3-319-42141-4. DOI: 10.1007/978-3-319-42141-4_8. URL: https://doi.org/10.1007/978-3-319-42141-4_8 (cit. on p. 19).

[Apa] Apache. *APACHE KAFKA*. URL: https://kafka.apache.org/ (cit. on p. 47).

[BSK10] F. Bai, D. D. Stancil, H. Krishnan. "Toward Understanding Characteristics of Dedicated Short Range Communications (DSRC) from a Perspective of Vehicular Network Engineers". In: *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*. MobiCom '10. Chicago, Illinois, USA: Association for Computing Machinery, 2010, pp. 329–340. ISBN: 9781450301817. DOI: 10.1145/1859995.1860033. URL: https://doi.org/10.1145/1859995.1860033 (cit. on p. 20).

[CEA+18] M. Cebe, E. Erdin, K. Akkaya, H. Aksu, S. Uluagac. "Block4Forensic: An Integrated Lightweight Blockchain Framework for Forensics Applications of Connected Vehicles". In: *IEEE Communications Magazine* 56.10 (2018), pp. 50–57. DOI: 10.1109/MCOM.2018.1800137 (cit. on pp. 65, 66).

[CKO+19] Y. Cao, A. Kurkcu, K. Ozbay, et al. "Blockchain: A Safe, Efficient Solution for Driver Privacy and Connected Vehicle Transportation Data Sharing". In: (2019) (cit. on p. 66).

[CM16] R. Coppola, M. Morisio. "Connected Car: Technologies, Issues, Future Trends". In: *ACM Comput. Surv.* 49.3 (Oct. 2016). ISSN: 0360-0300. DOI: 10.1145/2971482. URL: https://doi.org/10.1145/2971482 (cit. on pp. 19, 20).

[Con] Confluent. *Get started with Apache Kafka®*. URL: https://www.confluent.io/lp/apache-kafka/ (cit. on p. 47).

[Coo19] J. Coombs. *Understanding Oracle Attacks on Information Services*. 2019 (cit. on p. 44).

[DD10] G. Dimitrakopoulos, P. Demestichas. "Intelligent Transportation Systems". In: *IEEE Vehicular Technology Magazine* 5.1 (2010), pp. 77–84. DOI: 10.1109/MVT.2009.935537 (cit. on p. 20).

[DGL+02] S. Duri, M. Gruteser, X. Liu, P. Moskowitz, R. Perez, M. Singh, J.-M. Tang. "Framework for Security and Privacy in Automotive Telematics". In: *Proceedings of the 2nd International Workshop on Mobile Commerce*. WMC '02. Atlanta, Georgia, USA: Association for Computing Machinery, 2002, pp. 25–32. ISBN: 1581136005. DOI: 10.1145/570705.570711. URL: https://doi.org/10.1145/570705.570711 (cit. on p. 65).

[GG03] M. Gruteser, D. Grunwald. “Anonymous usage of location-based services through spatial and temporal cloaking”. In: *Proceedings of the 1st international conference on Mobile systems, applications and services*. 2003, pp. 31–42 (cit. on p. 59).

[GJK+21] S. Ghane, A. Jolfaei, L. Kulik, K. Ramamohanarao, D. Puthal. “Preserving Privacy in the Internet of Connected Vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 22.8 (2021), pp. 5018–5027. DOI: 10.1109/TITS.2020.2964410 (cit. on p. 65).

[Hat] R. Hat. *Drools*. URL: https://www.drools.org/ (cit. on p. 47).

[HFK+13] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, et al. “Guide to attribute based access control (abac) definition and considerations (draft)”. In: *NIST special publication* 800.162 (2013), pp. 1–54 (cit. on p. 27).

[HLN+21] C. Huang, D. Liu, J. Ni, R. Lu, X. Shen. “Achieving Accountable and Efficient Data Sharing in Industrial Internet of Things”. In: *IEEE Transactions on Industrial Informatics* 17.2 (2021), pp. 1416–1427. DOI: 10.1109/TII.2020.2982942 (cit. on pp. 66, 67).

[JG17] J. Joy, M. Gerla. “Internet of vehicles and autonomous connected car-privacy and security issues”. In: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–9 (cit. on p. 66).

[Ken11] J. B. Kenney. “Dedicated Short-Range Communications (DSRC) Standards in the United States”. In: *Proceedings of the IEEE* 99.7 (July 2011), pp. 1162–1182. ISSN: 1558-2256. DOI: 10.1109/JPROC.2011.2132790 (cit. on p. 20).

[KGMP06] P. Kalnis, G. Ghinita, K. Mouratidis, D. Papadias. *Preserving anonymity in location based services*. Tech. rep. 2006 (cit. on p. 66).

[LCZ+14] N. Lu, N. Cheng, N. Zhang, X. Shen, J. W. Mark. “Connected Vehicles: Solutions and Challenges”. In: *IEEE Internet of Things Journal* 1.4 (2014), pp. 289–299. DOI: 10.1109/JIOT.2014.2327587 (cit. on p. 19).

[LGNS20] W. Li, H. Guo, M. Nejad, C.-C. Shen. “Privacy-Preserving Traffic Management: A Blockchain and Zero-Knowledge Proof Inspired Approach”. In: *IEEE Access* 8 (2020), pp. 181733–181743. DOI: 10.1109/ACCESS.2020.3028189 (cit. on p. 66).

[McK14] McKinsey. *What’s driving the connected car*. Sept. 1, 2014. URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/whats-driving-the-connected-car (cit. on pp. 17, 29).

[MKGV07] A. Machanavajjhala, D. Kifer, J. Gehrke, M. Venkitasubramaniam. “l-diversity: Privacy beyond k-anonymity”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1 (2007), 3–es (cit. on p. 31).

[Mor10] Y. L. Morgan. “Notes on DSRC & WAVE standards suite: Its architecture, design, and characteristics”. In: *IEEE Communications Surveys & Tutorials* 12.4 (2010), pp. 504–518 (cit. on p. 20).

[Mor21] MordorIntelligence. *TELEMATICS MARKET - GROWTH, TRENDS, COVID-19 IMPACT, AND FORECASTS (2022 - 2027)*. 2021. URL: https://www.mordorintelligence.com/industry-reports/telematics-market (cit. on p. 17).

[MZL+19] Z. Ma, T. Zhang, X. Liu, X. Li, K. Ren. “Real-time privacy-preserving data release over vehicle trajectory”. In: *IEEE transactions on vehicular technology* 68.8 (2019), pp. 8091–8102 (cit. on p. 66).

[Pla21] M. Placek. *Connected cars worldwide - statistics & facts*. Dec. 8, 2021. URL: https://www.statista.com/topics/1918/connected-cars/ (cit. on p. 17).

[PZK+17] C. Plappert, D. Zelle, C. Krauß, B. Lange, S. Mauthöfer, J. Walter, B. Abendroth, R. Robrahn, T. von Pape, H. Decke. “A privacy-aware data access system for automotive applications”. In: *15th ESCAR Embedded Security in Cars Conference*. 2017 (cit. on pp. 27, 65).

[Swe02] L. Sweeney. “k-anonymity: A model for protecting privacy”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 557–570 (cit. on p. 31).

[WCJ+11] P. Wightman, W. Coronell, D. Jabba, M. Jimeno, M. Labrador. “Evaluation of location obfuscation techniques for privacy in location based information systems”. In: *2011 IEEE Third Latin-American Conference on Communications*. IEEE. 2011, pp. 1–6 (cit. on p. 49).

All links were last followed on March 20, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature