PaDS: An adaptive and privacy-enabling Data Pipeline for Smart Cars

1st Yunxuan Li
University of Stuttgart
Stuttgart, Germany
yunxuan.li@ipvs.uni-stuttgart.de

2nd Christoph Stach
University of Stuttgart
Stuttgart, Germany
christoph.stach@ipvs.uni-stuttgart.de

3rd Bernhard Mitschang
University of Stuttgart
Stuttgart, Germany
bernhard.mitschang@ipvs.uni-stuttgart.de

***Abstract*—The extensive use of onboard sensors in smart cars enables the collection, processing, and dissemination of large amounts of mobile data containing information about the vehicle, its driver, and even bystanders. Despite the undoubted benefits of such smart cars, this leads to significant privacy concerns. Due to their inherent mobility, the situation of smart cars changes frequently, and with it, the appropriate measures to counteract the exposure of private data. However, data management in such vehicles lacks sufficient support for this *privacy dynamism*. We therefore introduce PaDS, a framework for Privacy adaptive Data Stream. The focus of this paper is to enable adaptive data processing within the vehicle data stream. With PaDS, Privacy-Enhancing Technologies can be deployed dynamically in the data pipeline of a smart car according to the current situation without user intervention. With a comparison of state-of-the-art approaches, we demonstrate that our solution is very efficient as it does not require a complete restart of the data pipeline. Moreover, compared to a static approach, PaDS causes only minimal overhead despite its dynamic adaptation of the data pipeline to react to changing privacy requirements. This renders PaDS an effective privacy solution for smart cars.**

***Index Terms*—Smart Car, Privacy-Enabling Data Pipeline, Data Stream Runtime Adaptation, Mobile Data Privacy Management**

1. Introduction

With the help of various onboard sensing equipment, smart cars (SCs) can collect, process, and share information regarding themselves and the environment [1]. These mobile data offer detailed insights into various aspects, including the vehicle's location, activities, and information regarding the driver. Collecting and sharing of these data can be beneficial for achieving autonomous driving or creating a safer road environment. Moreover, vehicular context can be derived through the processing and fusion of these data, facilitating the development of situation-aware applications [2].

Given the rich information encapsulated in vehicular data, the collection and sharing of such data also raise significant privacy concerns. For instance, if vehicle data are disseminated or published without adequate protection, the behavior of the driver can be subject to unwanted analysis [3]. According to one of Mozilla's reports [4], modern automobiles are capable of gathering large amounts of details about the driver. Among 25 inspected car manufacturers, almost all of them indicate that further inferences of the interests of the user are made from the collected data (88%), and the majority of the car manufacturers stated in their policy that they can further share (84%) or even sell (76%) the collected personal data. Thus, privacy protection has become an important topic in the automotive domain.

To protect users' sensitive information from unwanted processing, Privacy-Enhancing Technologies (PETs) such as data-hiding [5] or restructuring-based technology, including randomization and perturbation [6], can be utilized. As outlined by Blarkom *et al.* [7], PETs are measures designed to protect informational privacy by minimizing personal data, thereby preventing unnecessary processing without compromising data usability.

As SCs encounter different situations while traveling, their inherent mobility becomes a special factor in the privacy protection of SCs. In addition, research has shown that the sensitivity of personal data is related to when and where as well as for what purpose the data are collected [8]. Thus, the privacy demands of a driver can show variability across different applications and vary in response to evolving situations. For instance, drivers may decide to hide their driving behavior by sending perturbed data at all times. Nonetheless, in critical situations like accidents, they would prefer immediate and accurate data sharing with emergency services. It is therefore necessary to develop technical measures that enable drivers to apply different PETs tailored to varying scenarios directly in their car before data are made available [9]. However, the challenge arises due to the typical static structure of current SC data processing pipelines, which often lack the flexibility to accommodate such *privacy dynamism*. This limitation emphasizes the necessity of developing mechanisms that enable the seamless adaptation of various PETs within a vehicle's data processing pipeline.

To address this challenge, we introduce a Privacy adaptive Data Stream (*PaDS*) framework that dynamically adapts PETs within a vehicle data stream. The focus of this paper is on the autonomous adaptation mechanism of data processing

Supported by SofDCar (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action, GSaME.

© 2024 IEEE. This is the author's version of the work. It is posted at https://opencms.uni-stuttgart.de/fak5/ipvs/departments/as/publications/stachch/mdm_24_pads.pdf by permission of IEEE for your personal use. Not for redistribution. The definitive version was published in *In: Renso, C. et al. (Eds.) Proceedings of the 2024 25th IEEE International Conference on Mobile Data Management. IEEE Computer Society, Brussels, Belgium*, pp. 41–50, 2024, doi: 10.1109/MDM61037.2024.00026.

operators, which allows for the dynamic adaptation of PETs. Our framework incorporates continuous situation recognition to assess the current vehicular context. Upon detecting a change in the situation, PaDS autonomously initiates the adaptation of PETs without requiring user intervention and minimizing the impact on normal vehicle functionalities. The key contributions of our research are: 1) We introduce a novel adaptive and privacy-enabling data pipeline, *PaDS*, to accommodate the evolving privacy requirements in the SC domain; 2) We propose a novel switching strategy in *PaDS* architecture, facilitating seamless PET adaptation and accurate data processing during switching phases; 3) We demonstrate the feasibility of *PaDS* with a prototype and evaluate *PaDS* in comparison to state-of-the-art and related work.

The remainder of the paper is structured as follows: In Section 2, we outline the requirements for the PET switching within a vehicle data pipeline. In Section 3, we introduce the *PaDS* concept and discuss state-of-the-art stream adaptation strategies in Section 4. In Section 5, we provide detailed insights into the adaptive PET switching strategy proposed in *PaDS*, followed by its evaluation in Section 6. Finally, we compare our work with the current research state in Section 7 and conclude the paper with Section 8.

2. Requirements

In this section, we analyze the requirements for seamless PET adaptation which we derived from the literature.

R_1 Automatic PET Switching: In similar domains such as mobile phones, addressing potential changes in privacy demands is commonly facilitated through pop-ups. For example, when users launch an application in a different country, a pop-up may inform them about the processing and storage of their data on a different server, prompting a potential adjustment of privacy preferences. However, a direct application of this approach in the automotive domain is not possible. Based on research by Klauer *et al.* [10] and Strayer *et al.* [11], it is evident that both visual and cognitive distractions during driving pose significant threats to safe driving performance. Consequently, drivers have to be enabled to specify their privacy preferences in advance by means of a privacy policy. As the vehicle traverses the road and encounters new situations, the PETs within the current data pipeline should automatically switch to the desired PETs, aligning with the driver's privacy preferences for the new situation. The switching mechanism is designed to occur seamlessly, eliminating the necessity for any driver actions or even requiring their notice, thereby minimizing potential distractions during the driving task.

R_2 Accurate PET Enforcement: As underlined in Section 1, the strength of privacy protection offered by user-specific PETs can differ significantly between two consecutive situations, particularly when switching from not using PETs to utilizing PETs or the reverse. In streaming systems, delays often occur between the time an event happens and the time it undergoes processing within the system [12]. Consequently, a potential scenario arises where the situation has already changed, but the actual adaptation of PETs has not yet begun. This may result in data that should be protected by the new PET being processed by the previous PETs, thereby posing a risk of privacy breaches. Hence, the switching mechanism must address this challenge by ensuring that vehicle data are processed by the correct PETs during the adaptation process, preventing unintentional privacy breaches.

R_3 App-Specific Privacy Protection: While prioritizing privacy considerations, it is imperative that the switching mechanism keeps the impact on normal vehicle functions minimal, particularly for safety-critical functions such as Advanced Driver Assistance Systems (ADAS) [13]. Moreover, it is noteworthy that user privacy preferences may vary depending on the application. This implies that PET switching may only occur for specific applications when the situation changes. Consequently, if a situation change necessitates a PET adaptation for one application, the switching mechanism should guarantee that any potential side effects are restricted to the data processing of that specific application, leaving other applications unaffected.

R_4 Timely PET Switching: Numerous applications in the vehicular domain have timeliness requirements for data. While information services, such as the exchange of information about available parking spaces, can typically tolerate certain delays, safety applications are more sensitive to latency and often require a bounded delay of messages [14]. It is evident that switching from one PET to another introduces additional overhead into the standard data processing pipeline, potentially leading to increased data latency. Nonetheless, the switching mechanism must aim to minimize this additional latency, ensuring that the impact on the functionality of vehicle applications is kept to a minimum.

3. Privacy adaptive Data Stream

To address these requirements, we developed *PaDS* to continuously provide querying applications with privacy-protected mobile data while adapting to situational changes. In this section, we give an overview of the *PaDS* framework in Section 3.1 and introduce its architecture in Section 3.2.

3.1. Overview

PaDS is specially designed to address the challenges of situation-aware privacy-preserving processing of stream data originating from vehicle sources. To give drivers full control over their data, *PaDS* has to be deployed directly in their SC. In this way, no third party has access to or influence on *PaDS*. This ensures that no data can bypass *PaDS* and secretly be forwarded to data consumers inside or outside the car. This deployment decision has an immediate impact on the supported types of PETs. According to the categorization proposed by Kaaniche *et al.* [15], PETs can be classified into three categories based on their deployment layer. The first category contains user-side PETs, incorporating techniques directly applicable on the end-user side, such as randomization and perturbation. In parallel, there are server-side PETs, i.e., techniques applicable on the server side. These PETs are

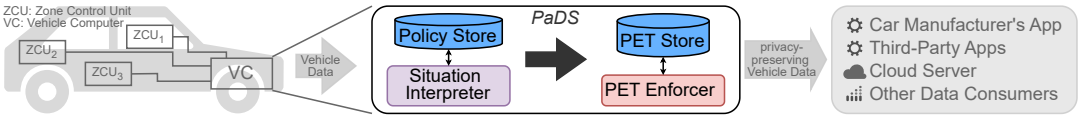


Figure 1: Deployment Strategy for *PaDS* and Overview of its Main Components

commonly used to anonymize databases for data exchanges (e.g., differential privacy [16]) or to perform computations over encrypted data (e.g., homomorphic encryption [17]). The third category includes channel-side PETs, which secure communication channels, for instance, by end-to-end encryption. As discussed before, *PaDS* is designed for a local deployment onboard a SC. Thus, only user-side PETs are considered in *PaDS*.

Figure 1 provides an overview of *PaDS*. As indicated by the grey car, we assume that the underlying SC has a zone-oriented architecture [18], aligning with the current trends in vehicular electrical and electronic architecture development. In this architecture, the zone control unit collects and forwards data from their respective zone to a centralized and more powerful vehicle computer responsible for executing complex functions such as ADAS or third-party applications. This also affirms the feasibility of executing PETs in a SC. As illustrated in Figure 1, *PaDS* is designed to intervene in the data flow. Instead of sharing vehicle data directly with applications or external data consumers, *PaDS* is deployed as an intermediary to ensure the protection of sensitive information within the collected vehicle data prior to its dissemination.

PaDS employs the privacy policy proposed in our previous work [19]. This policy allows users to specify desired PETs tailored for individual applications and diverse situations at a granular level, extending to specific vehicle data types. The user-defined privacy policies are stored in the `Policy Store` and are made available to *PaDS*. The input of *PaDS* contains the vehicle data collected by the central vehicle computer, including GPS data, camera data, and other relevant information about the vehicle's current state. The component `Situation Interpreter` (the purple box in Figure 1) allows *PaDS* to react to situational changes by constantly evaluating the current situation in the context of the incoming vehicle data. If a change in the situation requires action, the component `PET Enforcer` (the red box in Figure 1) is informed. This component is responsible for managing the switching between various PETs and processing the vehicle data with the desired PET algorithm. To facilitate dynamic PET switchings, *PaDS* mandates that all PETs are locally accessible prior to their intended switching. Within *PaDS*, a dedicated `PET Store` has been conceptualized to manage the storage and provision of PETs. The data sink of *PaDS* (the grey box on the right in Figure 1) is a conceptual component where privacy-preserving vehicle data processed by `PET Enforcer` is made available for sharing with intended applications or any other data consumers.

3.2. Architecture and Data Flow

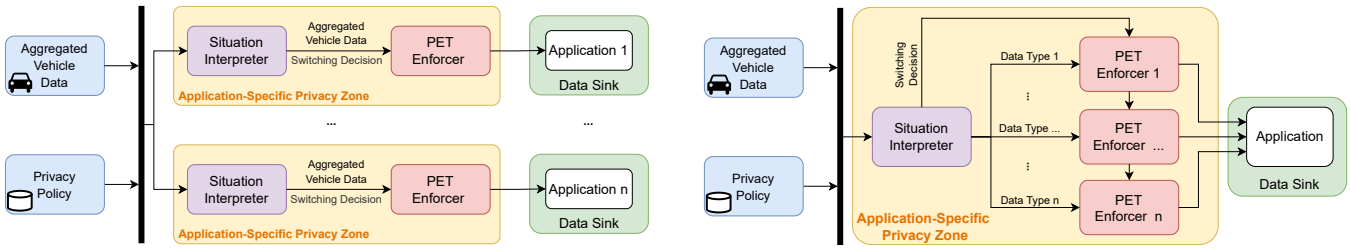
In this section, we provide a comprehensive explanation of the *PaDS* architecture and its data flow.

Input of *PaDS*: In *PaDS*, we assume the input data are aggregated vehicle data rather than individual data streams from a vehicle's sensor network. With the underlying zone-oriented architecture, the central vehicle computer can undertake the aggregation of data from different zones, validating our assumption. Furthermore, the potential out-of-order sensor readings resulting from fluctuation in transmission within the sensor network are also neglected in *PaDS*, as this can also be handled by the vehicle computer. Thus, we assume that the input data are maintained in chronological order as determined by its generation timestamp.

The privacy policy is also considered as one of the input streams, enabling *PaDS* to adapt to privacy policy changes, for instance, administrative policy received over-the-air, while the vehicle is in motion. It is important to highlight that *PaDS* only executes the input privacy policies without examining the validity of the policy. In other words, the system does not assess whether the PETs defined in the privacy policy meet users' privacy preferences or if potential conflicts exist among multiple defined policies. Consequently, the verification of the privacy policy's validity becomes essential during the policy creation phase. To address this, we introduced a privacy context model in our earlier work to assist users in determining suitable PETs [20]. The assessment of validity and potential conflicts can be entrusted by privacy experts.

***PaDS* Architecture:** Recall the requirement R_3 in Section 2, where the PET switching for one application should not impede the data processing of another application. To fulfill this requirement, *PaDS* introduces a distinct privacy-enabling data pipeline for each application. As depicted in Figure 2a, each application-specific pipeline consists of a `Situation Interpreter` and a `PET Enforcer`, which forms the application-specific *privacy zone* dedicated to processing the vehicle data before they are shared with the application. The component `Situation Interpreter` is responsible for recognizing the current situation in accordance with situations specified in the user-defined privacy policies and determining policies applicable to the application under the current situation. The evaluation of applicable policies for an application is described in our prior work [19] and thus is not further discussed here.

Note that instead of deploying a central `Situation Interpreter` for all applications, we utilize the parallel processing capability of the stream processing system to concurrently evaluate distinct situations tailored for each application. To enable thorough situation recognition, each



(a) *PaDS* Architecture

(b) *PaDS* Application-Specific Privacy Zone

Figure 2: Architecture of the *PaDS*

pipeline receives an identical copy of the aggregated vehicle data as input, ensuring the evaluation of all potential situations. Should the user decide to add or delete an application in SCs, the corresponding pipeline can also be initiated or removed.

App-Specific Privacy Zone: The architecture of the privacy zone for a specific application is further detailed in Figure 2b. As mentioned before, the privacy policy used in *PaDS* empowers the user to restrict an application's data access to the level of a certain vehicle data type. This intention can result in a fine-grained privacy policy wherein PETs may vary according to distinct data types. Furthermore, PETs show strong distinctions depending on the data type to which they are applied. For instance, a PET tailored to obfuscate location data cannot be directly applied to camera data for concealing location-specific details such as landmarks. Instead, a specialized PET designed explicitly for camera data should be utilized. Thus, to accommodate the aforementioned privacy policy and the diverse spectrum of PETs, *PaDS* incorporates an individual `PET Enforcer` for each unique vehicle data type requested by an application. This design choice also leverages the parallel processing capabilities inherent in the stream processing system, facilitating the concurrent execution of individual PETs.

Data Flow: Upon activation of the SC, *PaDS* initiates a dedicated data pipeline for each application and loads all privacy policies from the `Policy Store` into each application-specific `Situation Interpreter`. In parallel, the vehicle computer commences data collection across different zones and forwards it to *PaDS*. Based on the initial vehicle data and the stored privacy policies, the initial PET to be deployed in each `PET Enforcer` is determined based on an evaluation of the vehicle's current situation, namely, its stationary state at the parking location. As the SC transitions into motion, it continuously streams data to *PaDS*.

For the incoming aggregated vehicle data, *PaDS* forwards it to each application-specific pipeline. Within the designated privacy zone of each application, the `Situation Interpreter` evaluates the current situation using the input vehicle data. When `Situation Interpreter` detects a situational change that requires PET switching, it generates a *switching decision*, which encapsulates subsequent PETs derived from the applicable policies. The resulting *switching decision* is then broadcast to all `PET Enforcers`. The aggregated vehicle data is also decomposed into distinct data types, and each type is transmitted to the corresponding `PET Enforcer` tailored for that data type.

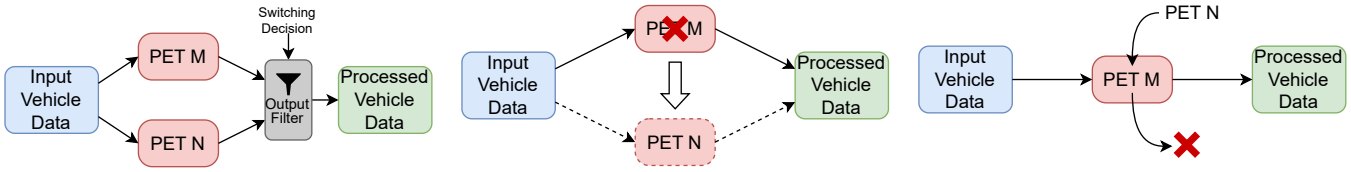
Upon receiving a *switching decision*, the `PET Enforcer` initiates the switching to the subsequent PET. During the switching period, vehicle data processing is temporarily paused, with incoming data being stored in a buffer. Once the switching is completed and the new PET is operational, the data processing is resumed, starting with the buffered data until it has been entirely processed. The updated PET then takes over the processing of ongoing vehicle data until the next switch is initiated. Finally, the processed data is forwarded to the respective application.

Given that situation recognition is a well-researched domain, *PaDS* leverages the SitOPT framework [21] for situation recognition. Without restricting the generality, however, a different framework could also be applied. While *PaDS* has no restrictions on selecting situation recognition frameworks, certain adaptations may be necessary, for instance, extending the output with a *switching decision*. In the following, we focus on the switching mechanism that facilitates the switching between PETs.

4. Switching Implementation Strategy

In this section, we delve into various implementation strategies of the dynamic PETs switching mechanism within a data type specific `PET Enforcer`. In the following, we first introduce important criteria for the switching implementation strategy, followed by a comprehensive analysis of state-of-the-art switching strategies and our proposed strategy.

With over-the-air (OTA) software updates becoming critical to the advancement of SCs, it is essential to ensure that privacy protection for SCs is also prepared for OTA updates of PETs. Recall the automatic switching requirement (R_1), the `PET Enforcer` must facilitate the automatic and seamless switching of PETs upon receiving a *switching decision*. This includes not only switching between PETs within the current `PET Store` but also incorporating PET updates delivered over-the-air. For instance, a PET provider might release a hotfix or a more efficient version of a PET. If users indicate that they prefer using the latest PETs, these updates will come with an administrative policy mandating an immediate switch to the latest PETs. Therefore, the PET switching implementation needs to be flexible enough



(a) S1: Output Filtering (b) S2: Dynamic Pipeline (c) S3: Static Pipeline and Dynamic Loading

Figure 3: Implementation Strategy of Dynamic PET Switching Mechanism

to accommodate switching between all PETs in the `PET Store`, including those introduced via OTA updates.

Moreover, the switching implementation should maintain lower complexity and minimize computational resource costs. Considering the timely switching requirement (R_4), while the switching process may introduce a brief pipeline downtime for processing vehicle data for certain applications, a lower complexity would positively impact the duration of this downtime. Although we mentioned in Section 3.1 that modern SCs are typically equipped with powerful vehicle computers, it is crucial to note that computational resources, including storage, RAM, and processing power, remain finite. Excessive utilization of these resources could impede the responsiveness and energy efficiency of the onboard computing system. This limitation not only constrains the potential for extending functionalities but also intensifies the demand for additional computational hardware.

Following the outlined requirements, we reviewed several state-of-the-art switching strategies from the literature (details in Section 7). These strategies can be roughly summarized into Output Filtering (e.g., DASH-PC [22]) and Dynamic Pipeline (e.g., Enorm [23]) as depicted in Figure 3. The pipeline structure in Figure 3 is an excerpt from Figure 2b focusing on the PET switching within a `PET Enforcer`. Thus, components such as `Situation Interpreter` are omitted in Figure 3, and the blue node as well as the green node represent the input vehicle data and the corresponding processed data, respectively. In the following, we provide a detailed explanation of these switching strategies.

Strategy 1: Output Filtering. As depicted in Figure 3a, the first approach utilizes a *static* pipeline in which every PET from the `PET Store` is concurrently deployed in a dedicated operator. During the runtime of *PaDS*, the input vehicle data are forwarded to each deployed PET algorithm. The processed data are then filtered by an Output Filter according to the incoming *switching decision*, ensuring that only data processed by the desired PET is forwarded.

Given its static pipeline structure, this approach faces challenges when confronted with OTA updates of PETs. Specifically, it is unable to handle the switching of PETs that are introduced by OTA updates, as such switching would necessitate a complete reconstruction of the data pipeline. While the switching complexity for PETs already deployed in the data pipeline remains relatively low, involving only adjustments of filter criteria in the Output Filter, the complexity for PETs introduced by OTA updates becomes exceedingly high, necessitating a complete halt of the entire data pipeline for reconstruction. Furthermore, this approach places substantial demands on computational resources due to the engagement of all candidate PETs in the computation, irrespective of their current relevance or applicability.

Strategy 2: Dynamic Pipeline. The second approach is based on the principle of a *dynamic* pipeline. Similar to the last approach, each candidate PET is implemented within a dedicated operator. Upon receiving a *switching decision*, the pipeline undergoes a dynamic structural transformation. As illustrated in Figure 3b, this process involves stopping the currently active operator of PET M, instantiating another operator with the next PET N, and establishing a connection to the new operator with PET N.

It is evident that this approach effectively manages the switching of PETs already stored in the `PET Store`. Moreover, in the event of switching between PETs introduced by OTA updates, this approach demonstrates the capability to address such scenarios by initializing a new operator dedicated to the new PET and dynamically establishing connections accordingly. Notably, this approach involves solely the execution of applicable PETs during runtime. Hence, the computational resource cost for this approach remains at a low to moderate level.

Nevertheless, the inherent switching mechanism in this approach indicates a considerable level of complexity, as it demands a modification in the topology of the pipeline. As of the current development of stream processing technology, this approach requires a temporary termination of the stream processing operator to reconstruct the pipeline for a new operator equipped with the next PET. Subsequently, the new operator is started. While the interruption has no effect on the processing of data types that do not require a PET switch, it inevitably leads to a temporary downtime for data types necessitating a PET switch.

The characteristics of Strategies 1 and 2 are outlined in the first two rows of Table 1. Strategy 1 incurs minimal latency, as all PETs are processed in parallel, and only the output stream of the correct PET has to be forwarded to the respective application. However, this strategy demands high computational resources and lacks the capability to switch between PETs introduced by OTA updates. Conversely, Strategy 2 offers extensive support for PET switching but necessitates a temporary suspension of the underlying data pipeline for each switch. Both strategies fall short of meeting *PaDS*'s criteria for optimal PET switching. Therefore, we

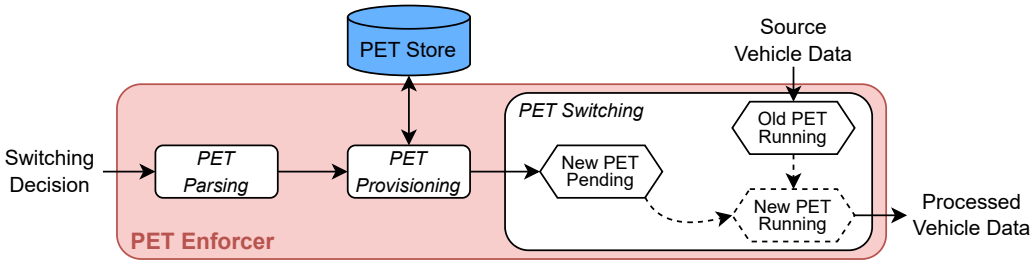


Figure 4: PET Switching Stages in a `PET Enforcer`

introduce in *PaDS* a novel third strategy that combines the flexibility of the Dynamic Pipeline with the static pipeline structure of Output Filtering, thereby mitigating the limitations inherent in the first two strategies.

Strategy 3: Static Pipeline and Dynamic Loading. Our approach is demonstrated in Figure 3c. Similar to Strategy 1, the pipeline topology remains constant, where a dedicated operator is assigned to each data type for executing the PET algorithm. However, the PET algorithms running in the operator are switched in accordance with the *switching decision*. As depicted in Figure 3c, the operator controls the lifetime of the PET inside. Upon receiving the *switching decision*, the operator dynamically loads the new PET N and unloads the previous PET M.

The static pipeline structure of this approach mandates the implementation of a standardized interface across the PETs engaged in the switching process. It is imperative that all PETs provisioned by the `PET Store` are developed in a modularized manner. Such a modular design not only simplifies the switching process between different PETs but also increases the compatibility and extensibility of *PaDS*, particularly in integrating future PETs.

Similar to Strategy 2, the last approach can effectively handle the dynamic switch between PETs introduced by OTA updates. The switching complexity lies in the dynamic loading mechanism. Upon the arrival of the *switching decision*, the operator only needs a brief pause of the data processing to initiate the subsequent PET and set it to functioning, after which it resumes data processing seamlessly with the updated PET. Thus, the potential downtime induced by this approach is significantly shorter compared to the second strategy. Furthermore, this approach demands low to moderate computational resources, as it exclusively executes applicable PETs during runtime and introduces only minimal overhead in preparing new PETs. The characteristic of this strategy is also summarized in Table 1.

	over-the-air PET Update	Switching Complexity	Computation Resource
S1: Output Filtering	-	low	high
S2: Dynamic Pipeline	+	high	moderate
S3: Static Pipeline & Dynamic Loading	+	moderate	moderate

Table 1: Comparison of three Switching Strategies

5. Adaptive PET Switching

In this section, we introduce the details of the proposed Static Pipeline and Dynamic Loading strategy. The procedure of PET switching within a single `PET Enforcer` can be split into three main stages: PET Parsing, PET Provisioning, and PET Switching, as illustrated in Figure 4. Upon receiving the *switching decision*, the `PET Enforcer` initiates the switching procedure by parsing the PET annotation within the incoming message, thereby identifying the subsequent PET. It then retrieves the desired PET from the `PET Store`. Subsequently, the new PET is initialized and set to pending. Once the preceding PET completes processing the current data, the new PET is seamlessly switched in and set to an active state. In the following, we explain each stage in detail.

In the first stage, PET Parsing, the subsequent PET to be executed is determined based on the PET annotation within the *switching decision*. This annotation should be written in a machine-readable form and include PET's metadata, such as its name, version, and the input parameters required for its initialization. Subsequently, this annotation is transformed to a format that matches the PET metadata in the `PET Store`, ensuring accurate identification and retrieval of the desired PET. Note that the development of the PET metadata is beyond the scope of this work, and thus, the details of the parsing process are not further discussed in this context.

In the PET Provisioning stage, the `PET Enforcer` queries the `PET Store` for the corresponding algorithm and initializes it with the specified parameters. The newly instantiated PET is temporarily set to be pending. Once feasible, the newly instantiated PET algorithm will be set to active, substituting the current running PET.

The final stage of PET Switching does not necessarily take place immediately after the completion of PET Provisioning. Recall the requirement R_2 of accurate PET enforcement outlined in Section 2, the activation of the new PET is contingent upon the completion of data processing by the preceding PET for all data points generated prior to the change in situation. To facilitate this, we introduce the concept of *triggering data*, defined as a data point responsible for provoking the situation change. This concept shares similarities with the watermark notion used in streaming processing systems [12], where the observation of a watermark indicates that all input data occurring before the time of the watermark has been observed by the system.

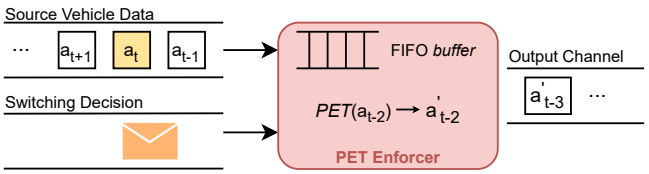


Figure 5: Input and Output Channel of a `PET Enforcer`

Note that *PaDS* is specifically designed for a local execution within an SC, operating under the assumption of a timely ordering of input vehicle data within the system. Consequently, the *triggering data*, under this assumption, serves as a perfect watermark [12], signifying that all data belonging to the last situation have already been observed by the system. In *PaDS*, the `Situation Interpreter` is responsible for marking a data point as the *triggering data*, for instance, by adding a flag to the data point.

It is noteworthy that the `PET Enforcer` receives input vehicle data and *switching decisions* through distinct input channels. As illustrated in Figure 5, the upper input channel facilitates the flow of input vehicle data into the `PET Enforcer`, while the lower channel is dedicated to receiving the *switching decision* from the `Situation Interpreter`. As of the current development of stream processing technology, the `PET Enforcer` is limited to consuming and processing one message from an input channel at a time. Specifically, if a vehicle data point is consumed, the `PET Enforcer` processes it using the currently deployed PET. Conversely, upon the consumption of a *switching decision* (depicted as the envelope icon in Figure 5), the `PET Enforcer` initiates the PET switching procedure.

Moreover, the consumption of messages from these two channels occurs in a non-deterministic manner. Consequently, there is no guarantee that the *triggering data* (yellow marked data point a_t in Figure 5) will be directly consumed after the processing of the *switching decision*. While it is possible to intentionally define an order of message consumption within the `PET Enforcer`, such a method could adversely impact system throughput as extra delay would be introduced.

To address the challenge of non-deterministic message consumption order, we introduce a First-In-First-Out (FIFO) *buffer* within the `PET Enforcer`. This *buffer* serves the purpose of temporarily storing incoming vehicle data that arrives successive to the *triggering data* in case the *switching decision* has not yet been processed. Note that the *triggering data* itself is also buffered. Subsequently, upon the consumption of the *switching decision*, the initialized PET can be directly switched to an active state within the PET Switching stage, thereby initiating the processing of data points from the *buffer*. Once the *buffer* is fully released, the `PET Enforcer` resumes the consumption of data from the input channel. Conversely, if the *switching decision* is consumed prior to the *triggering data*, the initialized PET is set to a pending state at the end of the PET Provisioning stage. Meanwhile, the existing PET continues processing any incoming data until the *triggering data* is processed. Upon the consumption of the *triggering data*, the new PET is then activated and commences the data processing of the *triggering data*. This adaptive switching procedure ensures seamless switching between PETs while maintaining the second requirement of accurate PET enforcement.

6. Evaluation

In this section, we evaluate the *PaDS* framework based on the requirements outlined in Section 2. For the evaluation, the requirements are divided into qualitative (R_1, R_2, R_3) and quantitative requirements (R_4). In the following, we first evaluate the qualitative requirements based on the design of *PaDS* and then evaluate the quantitative requirements based on a prototype of *PaDS*.

Requirement R_1, Automatic PET Switching, is facilitated by two primary components within *PaDS*. Primarily, the framework leverages existing situation recognition frameworks to respond to changes in the environment autonomously. Upon detection of such changes, a *switching decision* is promptly generated and transmitted to the `PET Enforcer`. Within the `PET Enforcer`, the subsequent PETs are seamlessly switched into the data pipeline as described in Section 5. Thereby, the entire process of PET adaptation is initiated and executed automatically in response to situational changes without user intervention.

As described in Section 5, the second requirement (R_2 Accurate PET Enforcement) is ensured through the incorporation of *triggering data* and buffering mechanisms. While the *switching decision* and source vehicle data are managed as distinct data streams, the introduction of *triggering data* enables the `PET Enforcer` to determine the relationship between messages within these streams. Consequently, if the *triggering data* is delayed, the initiated PET temporarily enters a pending state to wait for its arrival. Similarly, should the *switching decision* encounter delays, the *triggering data* initiates a temporary buffering of incoming data until the PET is switched. Therefore, source vehicle data are always processed by the intended PETs, even during the PET switching phases.

The last qualitative requirement (R_3 App-Specific Privacy Protection) is ensured by the architectural design of *PaDS*, which allocates a specific privacy zone to each installed application. This isolation guarantees that the switching action for one application remains independent of the data processing of another. Moreover, the selected switching strategy does not necessitate the need to halt the data processing of an application during the PET switching phase.

To assess the final requirement (R_4 Timely PET Switching), we implemented a proof-of-concept prototype of *PaDS* to quantitatively assess the latency implications introduced by the additional steps involved in PET switching. For a comprehensive analysis, we implemented Strategy 1 as a comparative baseline. While Strategy 1 lacks the capability to switch between PETs updated over-the-air, it offers minimal latency in its switching mechanism, making it a suitable

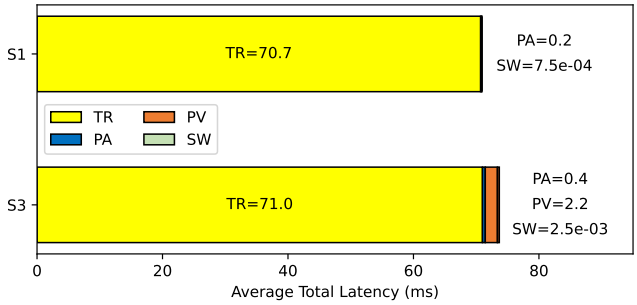


Figure 6: Avarage Total Latency of S1 and S3

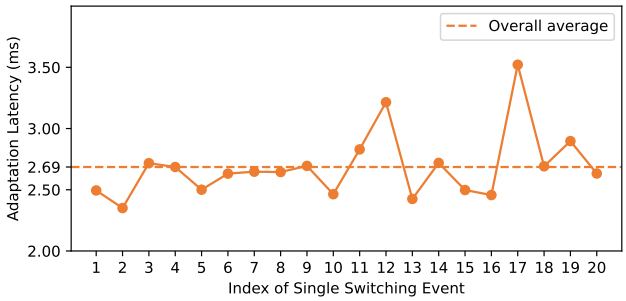


Figure 7: Adaptation Latency of Strategy 3

point of reference. Conversely, Strategy 2 was excluded from this comparative analysis due to its inherently higher latency from its more complex switching mechanism. Based on our experience, reconstructing the data pipeline at runtime utilizing popular streaming platforms generally introduces latency in the order of a tenth of a second per switch, rendering further evaluation impractical due to excessive overhead for Timely PET Switching (R_4).

The total latency we measured during the evaluation extends from the generation of a *switching decision* to the activation of a new PET within the data pipeline. As the runtime of the `Situation Interpreter` depends on the specific situation recognition framework employed, therefore, this latency is not included in our evaluation. Additionally, the runtime of PETs is subject to its algorithm and implementation, thus, it is also excluded from the measurement. The measured total latency can be further divided into transition latency (TR), parsing latency (PA), provision latency (PV), and switching latency (SW). The transition latency denotes the duration of transmitting a *switching decision* from the `Situation Interpreter` to the `PET Enforcer` (to component Output Filter in Strategy 1). The parsing latency, provision latency, and switching latency quantify the time required for PET Parsing, Provisioning, and Switching stages within the `PET Enforcer`.

For the prototype implementation, Apache Flink was chosen as the underlying streaming processing platform due to its high flexibility, extensibility, and inherent exactly-once semantics. Particularly, the exact-once semantics allow us to avoid undesired loss or duplication of data without additional development efforts. The application-specific pipeline was realized as a Flink job. In our prototype, we did not incorporate a comprehensive `Situation Interpreter`. Instead, we simulated situational changes by injecting predefined *switching decisions*. The switching procedure described in Section 5 was implemented within the `PET Enforcer` operator. For PET Parsing, we developed a simple PET annotation scheme to identify the PETs utilized during evaluation. The dedicated `PET Store` was integrated as a built-in library of *PaDS*.

In order to highlight the experimental effect, image data were selected as the target data type due to their high refresh frequency and larger data volume. Specifically, image data from the KITTI dataset [24] was employed, with each image having a resolution of 1226 × 370 and a sampling rate of 10Hz. For the experiment, we adapted a camera PET from the data hiding category [5]. The PET utilizes image segmentation techniques to identify sensitive information within an image and render it through pixelation or blurring to make it unrecognizable. Each variation of the PET was implemented as a modularized PET within the built-in library. In the experiment, 1000 image data was fed into the prototype at a rate of 10Hz. In order to stimulate PET adaptation at a higher frequency, *switching decisions* were generated every 50 frames, leading to 20 switching events throughout the experiment. The generated *switching decision* was configured to switch between pixelation PET and blurring PET.

The experiment was conducted utilizing the Apple M1 Pro processor with eight cores, as from our perspective, it approximates the minimum computational power expected within an SC. The average total latency is depicted in Figure 6, where two stacked bars represent the total latency of Strategies 1 and 3, respectively. As illustrated in Figure 6, the average transition latency for Strategies 1 and 3 are similar, averaging around 70 ms. The parsing latency for both strategies is comparable and minimal, at under 0.5 ms. In Strategy 1, the PETs are implemented as fixed operators, thus, there is no provision latency. Moreover, the switching mechanism in this strategy is to modify the output filter criteria, resulting in a negligible switching latency of around 0 ms. Conversely, Strategy 3, the dynamic switching mechanism employed in *PaDS*, holds a provision latency of approximately 2.2 ms. This latency is attributed to the retrieving and instantiating processes necessary for preparing a PET. The switching latency in Strategy 3 is also negligible, as it involves merely the activation of the prepared PET.

During our evaluation, we noted that when the *triggering data* experiences delays, resulting in the new PET pending in the `PET Enforcer`, the average wait time for the *triggering data* is approximately 12 ms. This pending latency is primarily due to data transfer latency between different components and is therefore not included in the total adaptation latency (sum of PA, PV, and SW). As depicted in Figure 7, once the prototype achieves operational stability, the adaptation latency of a single switch fluctuates between a small range from 2.5 ms to 3 ms, which indicates that the switching latency is independent of the specific PET.

Lessons Learned. Our comparative analysis reveals that Strategy 3 incurs significantly lower latency for each switch compared to Strategy 2, which typically results in a latency close to one-tenth of a second. Although there is a perceptible latency with Strategy 3 compared to Strategy 1, it remains relatively low, at approximately 2.7 ms. It is important to note that this latency also depends on the computational power of the underlying platform. Given that our experimental setup reflects the minimum expected computational power in an SC, future improvements in vehicle computing power are likely to decrease this latency further. Furthermore, Strategy 3 is designed to execute only applicable PETs at runtime, thereby optimizing computational resource consumption. Conversely, Strategy 1 deploys all PETs regardless of their current applicability, leading to increased computational overhead. Moreover, Strategy 3 supports the switching between PETs introduced to the `PET Store` through over-the-air updates. Once a PET is made available within the `PET Store`, it can be switched in a `PET Enforcer` following the switching procedure introduced in Section 5.

Although we focus on PET switching in this paper, our proposed switching strategy is not limited to switching between PETs. For instance, Strategy 3 can be used to switch between a variety of general data processing algorithms, such as data transformation or data cleansing algorithms. This flexibility highlights the adaptability of our switching strategy to a broad spectrum of computational tasks beyond the scope of privacy protection.

7. Related Work

In recent years, a variety of studies have been conducted to address privacy protection issues in the smart car domain from multiple perspectives. Despite this diversity, the majority of these studies focus on privacy protection within specific use cases, such as intelligent transportation management. For example, Tan and Yang [25] proposed a mechanism to protect vehicle data in traffic control by leveraging secure multi-party computation and differential privacy. In a similar vein, Herges *et al.* [26] introduced the Ginger framework, utilizing access control to protect the privacy of vehicle data against potential privacy violations posed by telematics applications. Moreover, a considerable amount of studies in this area focus on the privacy protection of specific vehicle data types, especially trajectory data. For instance, techniques such as spatial location cloaking [27], temporal cloaking [28], and path confusion [29] are popular approaches to trajectory data protection. Due to their focus on specific applications or data types, these frameworks often lack adaptability to use cases they were not designed for. In contrast, *PaDS* is a general-purpose privacy-preserving framework applicable to a broad range of potential applications and data types within the SC context. Moreover, these studies lack consideration for situation awareness in privacy protection for SCs, providing fixed privacy protection levels in varying situations.

To the best of our knowledge, only a few studies have addressed situation-aware privacy protection of vehicular data, primarily focusing on adaptive vehicle identifier changes. Emara *et al.* [30] introduced CAPS, a location privacy scheme that changes a vehicle's pseudonym in response to its surroundings. Saini *et al.* [31] developed a pseudonym changing scheme that leverages the vehicle context and traffic patterns to determine a suitable situation for changing pseudonyms. Lu *et al.* [32] proposed an ID-based authentication framework utilizing adaptive pseudonyms for communication among roadside units and vehicles. Nevertheless, these studies often focus on specific data types, such as vehicle pseudonyms. Conversely, *PaDS* offers situation-aware privacy protection for arbitrary vehicular data types.

While research on situation-aware privacy protection in the SC domain is limited, this concept is gaining more attention in similar fields. For instance, Elkhodr *et al.* [33] proposed a context-aware approach for protecting user location data utilizing a Dynamic Location Disclosure Agent. In the field of Ubiquitous computing, Schaub *et al.* [34] introduced a context-adaptive framework that supports the privacy decision-making of users by leveraging the context of the user and respective context changes. As for data streams in the IoT domain, Stach *et al.* [35] introduced PATRON to control access to information patterns by concealing privacy patterns. However, several studies also focus on specific data types, such as IoT location data. The data type independent approaches either help users in identifying their privacy preferences but do not adaptively ensure these preferences [34], or employ pre-configured privacy mechanisms for various privacy patterns [35]. On the contrary, *PaDS* is an adaptive and privacy-enabling Data Pipeline that ensures adaptive privacy protections by utilizing different PETs according to situation changes.

As for the runtime adaptation of Data Stream Processing Systems, most researched adaptations are designed to adapt to variations in their working environment (e.g., processing migration or scaling), only a few studies focus on the runtime adaptation of data processing operations. For instance, DASH-PC [22] adapts between diverse quality versions of point cloud frames based on constrained resources such as bandwidth. Qin and Eichelberger [36] proposed an approach to re-route a data stream between multiple processing algorithms. Madsen *et al.* [23] introduced Enorm to dynamically split and combine operators, which change the stream topology at runtime. To reduce the reconstruction time for combining, they also support running the old operators in parallel during adaptation. These switching strategies, as discussed in Section 4, either deal with a re-routing of data within a static stream pipeline or create a disruption of the data processing to rebuild the pipeline. Conversely, the switching strategy we proposed in *PaDS* mitigates these limitations so that the switching between PETs introduced by OTA updates is also supported and the possible downtime of data processing is kept minimum.

8. Conclusion

While privacy protection in SCs is a prevalent research area, studies addressing situation-aware privacy protection remain limited. In this paper, we introduce *PaDS*, a novel

adaptive and privacy-enabling data pipeline to accommodate privacy requirements that arise when SCs encounter various situations while moving. *PaDS* introduces a novel switching strategy for data processing operators such as PETs. By facilitating automatic PET switching, *PaDS* provides effective situation-aware privacy protection for SCs while ensuring accurate PET processing of mobile data. With the proposed switching strategy, *PaDS* only executes applicable PETs during runtime, consuming only low to moderate computational resources compared to the strategy where all PETs run in parallel. With a comparison of state-of-the-art switching strategies, we demonstrate that *PaDS* effectively handles switching between PETs in the `PET Store` and those updated over-the-air with minimal adaptation latency.

For future work, we plan to extend *PaDS* to support adaptation between general data processing operators instead of only PETs. We also plan to investigate the feasibility of implementing *PaDS* in real vehicles and evaluate its performance against real-world use cases.

References

[1] N. Lu *et al.*, "Connected Vehicles: Solutions and Challenges," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 289–299, Aug. 2014.

[2] F. Terroso-Sáenz *et al.*, "A complex event processing approach to perceive the vehicular context," *Information Fusion*, vol. 21, pp. 187–209, Jan. 2015.

[3] J. Joy and M. Gerla, "Internet of Vehicles and Autonomous Connected Car - Privacy and Security Issues," in *ICCCN '17*, 2017.

[4] C. Jen *et al.*, "It's Official: Cars Are the Worst Product Category We Have Ever Reviewed for Privacy," Mozilla, Sep. 6, 2023. [Online]. Available: https://foundation.mozilla.org/en/privacynotincluded/articles/its-official-cars-are-the-worst-product-category-we-have-ever-reviewed-for-privacy/.

[5] J. R. Padilla-López *et al.*, "Visual privacy protection methods: A survey," *Expert Systems with Applications*, vol. 42, no. 9, pp. 4177–4195, Jun. 2015.

[6] S. Turgay and İ. İlter, "Perturbation Methods for Protecting Data Privacy: A Review of Techniques and Applications," *Automation and Machine Learning*, vol. 4, no. 2, pp. 31–41, 2023.

[7] G. W. van Blarkom *et al.*, Eds., *Handbook of Privacy and Privacy-Enhancing Technologies: The case of Intelligent Software Agents*. PISA Consortium, 2003, ISBN: 90-74087-33-7.

[8] M. Gharib *et al.*, "An Ontology for Privacy Requirements via a Systematic Literature Review," *Journal on Data Semantics*, vol. 9, pp. 123–149, Dec. 2020.

[9] A. Fieschi *et al.*, "Privacy in Connected Vehicles: Perspectives of Drivers and Car Manufacturers," in *SummerSOC '23*, 2023.

[10] S. G. Klauer *et al.*, "The Impact of Driver Inattention on Near-Crash/Crash Risk: An Analysis Using the 100-Car Naturalistic Driving Study Data," National Highway Traffic Safety Administration, Technical Report DOT HS 810 594, Apr. 2006. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.gov/files/810594.pdf.

[11] D. L. Strayer *et al.*, "Cell phone-induced failures of visual attention during simulated driving," *Journal of Experimental Psychology: Applied*, vol. 9, no. 1, pp. 23–32, 2003.

[12] T. Akidau *et al.*, Eds., *Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing*. O'Reilly Media, 2018, ISBN: 978-1491983874.

[13] M. Lu *et al.*, "Technical Feasibility of Advanced Driver Assistance Systems (ADAS) for Road Traffic Safety," *Transportation Planning and Technology*, vol. 28, no. 3, pp. 167–187, 2005.

[14] J. E. Siegel *et al.*, "A Survey of the Connected Vehicle Landscape—Architectures, Enabling Technologies, Applications, and Development Areas," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2391–2406, Aug. 2018.

[15] N. Kaaniche *et al.*, "Privacy enhancing technologies for solving the privacy-personalization paradox: Taxonomy and survey," *Journal of Network and Computer Applications*, vol. 171, no. 102807, Dec. 2020.

[16] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, Aug. 2014.

[17] M. A. Will and R. K. L. Ko, "A guide to homomorphic encryption," in *The Cloud Security Ecosystem: Technical, Legal, Business and Management Issues*, R. Ko and K.-K. R. Choo, Eds., Syngress, 2015, ch. 5, pp. 101–127, ISBN: 978-0-12-801595-7.

[18] V. M. Navale *et al.*, "(R)evolution of E/E Architectures," *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, vol. 8, no. 2, pp. 282–288, 2015.

[19] Y. Li *et al.*, "Ensuring Situation-Aware Privacy for Connected Vehicles," in *IoT '22*, 2023.

[20] Y. Li *et al.*, "CV-Priv: Towards a Context Model for Privacy Policy Creation for Connected Vehicles," in *CoMoRea '23*, 2023.

[21] P. Hirmer *et al.*, "Situation recognition and handling based on executing situation templates and situation-aware workflows," *Computing*, vol. 99, pp. 163–181, Feb. 2017.

[22] M. Hosseini and C. Timmerer, "Dynamic Adaptive Point Cloud Streaming," in *PV '18*, 2018.

[23] K. G. S. Madsen *et al.*, "Enorm: Efficient window-based computation in large-scale distributed stream processing systems," in *DEBS '16*, 2016.

[24] A. Geiger *et al.*, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, Sep. 2013.

[25] C. Tan and K. Yang, "Privacy-preserving adaptive traffic signal control in a connected vehicle environment," *Transportation Research Part C: Emerging Technologies*, vol. 158, no. 104453, Jan. 2024.

[26] D. Herges *et al.*, "Ginger: An Access Control Framework for Telematics Applications," in *TrustCom '12*, 2012.

[27] L. Šikšnys *et al.*, "Private and Flexible Proximity Detection in Mobile Social Networks," in *MDM '10*, 2010.

[28] M. Gruteser and D. Grunwald, "Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking," in *MobiSys '03*, 2003.

[29] B. Hoh *et al.*, "Achieving Guaranteed Anonymity in GPS Traces via Uncertainty-Aware Path Cloaking," *IEEE Transactions on Mobile Computing*, vol. 9, no. 8, pp. 1089–1107, Aug. 2010.

[30] K. Emara *et al.*, "CAPS: Context-aware privacy scheme for VANET safety applications," in *WiSec '15*, 2015.

[31] I. Saini *et al.*, "A Context Aware and Traffic Adaptive Privacy Scheme in VANETs," in *CAVS '20*, 2020.

[32] H. Lu *et al.*, "A novel ID-based authentication framework with adaptive privacy preservation for VANETs," in *ComComAp '12*, 2012.

[33] M. Elkhodr *et al.*, "A contextual-adaptive Location Disclosure Agent for general devices in the Internet of Things," in *LCN Workshops '13*, 2013.

[34] F. Schaub *et al.*, "Context-Adaptive Privacy: Leveraging Context Awareness to Support Privacy Decision Making," *IEEE Pervasive Computing*, vol. 14, no. 1, pp. 34–43, Feb. 2015.

[35] C. Stach *et al.*, "How a Pattern-based Privacy System Contributes to Improve Context Recognition," in *CoMoRea '18*, 2018.

[36] C. Qin and H. Eichelberger, "Impact-minimizing Runtime Switching of Distributed Stream Processing Algorithms," in *BDPR '16*, 2016.