PECSo: A Privacy Enhancing Framework for Applications in the Automotive Domain

Giampaolo Bella[1], Sergio Esposito[1], Mirko Giuseppe Mangano[1], Salvatore Riccobene[1], Daniele Francesco Santamaria[1] and Gabriele Veneziano Broccia[1]

[1]*University of Catania, Italy*

Abstract

With the advancement of technology in the landscape of automotive, individuals' control over their personal data is increasingly at risk. Modern cars, equipped with a myriad of sensors and network capabilities, continuously collect vast amounts of data — from driver behaviour and media consumption to precise geolocation tracking. While these innovations enhance the driving experience and vehicle functionality, they also introduce significant privacy risks. The Privacy Enrooted Car Systems (PECS) project was developed to address these concerns by embedding robust privacy safeguards directly into automotive systems. By adopting a proactive approach to data protection, privacy, and cybersecurity, the PECS project aims to deliver a secure and privacy-focused driving environment. In this contribution we delve into the PECSo module of PECS, a framework to provide individuals with the capability to obfuscate their personal data collected by cars before sharing it with third parties, thus ensuring data protection right from the outset. To allow this, PECSo defines three tiers of compliance that impose progressively stricter adherence to PECS specifications. Prescriptions include implementing Privacy Enhancing Technologies (PETs) to safeguard personal data, receiving user-defined privacy policy from another PECS component, and segregation of duties between the application interface and the PET application. We demonstrate the practicality of this framework through two application prototypes tested on a real vehicle, featuring Secure Multi-Party Computation and Federated Learning as PETs to protect data privacy.

Keywords

Automotive Security, Privacy, Data Obfuscation, Federated Learning, Secure Multi-Party Computation

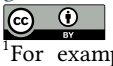
1. Introduction

To enhance user experience and provide them with a wide range of services while staying seated and driving, cars manage a plethora of user personal data, thus significantly raising privacy risks. A concerning aspect of this technological advancement is the inconsistency in compliance with the EU General Data Protection Regulation (GDPR) across car manufacturers. GDPR sets stringent standards for personal data protection, demanding transparency, user consent, and user control over their data. Despite these regulations, recent evaluations and historical evidence [1] reveal that many car manufacturers have not fully aligned their data collection and processing practices with GDPR mandates. This misalignment ranges from inadequate data handling disclosures to insufficient mechanisms for user consent and control, leading to potential privacy breaches. Instances of non-compliance are not merely isolated cases but represent a broader systemic issue within the industry.[1] The privacy implications of new automotive technologies frequently remain an afterthought, resulting in practices that can jeopardise user trust and safety. Such non-compliance underscores the critical need for initiatives that aim to embed robust privacy controls directly into vehicle systems, thereby addressing these challenges head-on. The PECS [2] project represents a proactive approach to addressing cybersecurity and privacy challenges within

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT

[†] These authors contributed equally.

giampaolo.bella@unict.it (G. Bella); sergio.esposito@unict.it (S. Esposito); mirkogiuseppe.mangano@studium.unict.it (M. G. Mangano); riccobene@unict.it (S. Riccobene); daniele.santamaria@unict.it (D. F. Santamaria); gabrieleveneziano.b@gmail.com (G. Veneziano Broccia)



[1] For example, see https://foundation.mozilla.org/en/privacynotincluded/articles/its-official-cars-are-the-worst-product-category-we-have-ever-reviewed-for-privacy/

the automotive industry. Through its multidimensional approach and rigorous methodologies, the project aims to foster a safer and more privacy-centric environment for car users.

PECS, funded by the NGI Trustchain initiative,[2] enhances the management of consents and privacy preferences, and builds a secure framework for data exchange and privacy-aware data processing. PECS enroots the technological systems of modern cars into the very bedrock of privacy by implementing three modules, namely:

1. The **PECS Interface (PECSi)** module, which serves as the user interface, allowing users to make subjective privacy preferences about the personal data processed by service providers. Users can set several pre-set preferences that apply globally to all services or choose to set specific preferences for each service. The user interface also alerts users whenever a service violates their privacy preferences;
2. The **PECS Obfuscation (PECSo)** module, which functions as the component for obfuscating personal data when processed by services. In this work, we show two Privacy Enhancement Technologies (PETs), that is, secure multi-party computation and federated learning, which can be implemented by service applications or can be included in PECSo;
3. The **Blockchain** module, which stores the privacy preferences set by users, ensuring further security properties and public verifiability of the preferences themselves. The preferences are linked to the user via a pseudonym to avoid exposing sensitive information, and can be later verified via digital signature.

In this contribution, we present the PECSo module of PECS by outlining its technical specifications, software implementation and use cases. The paper is organized as follows: Section 2 presents the related work; Section 3 briefly introduces PECS and its components; Section 4 then presents PECSo, showing how it works and how it interacts with applications; Section 5 shows two test applications we developed to demonstrate the potentialities of PECSo; finally, Section 6 concludes the paper with some future directions.

2. Related Work

Research on privacy in the automotive domain has gained traction as modern vehicles increasingly collect and process vast amounts of personal data. PRICON [3] addresses these issues by introducing a user-centered privacy-aware control system that integrates judicial, technical, and user perspectives. Unlike many solutions driven solely by technological innovations, PRICON empowers users to define and enforce self-determined privacy policies directly within the vehicular system. Its evaluation demonstrates a balanced approach to privacy, ensuring compliance with legal standards, technical feasibility, and user accessibility.

Built on similar principles, the PRICAR framework [4] offers a structured methodology for vehicular data sharing with third parties, underscoring the importance of compliance with data protection regulations such as the GDPR. It proposes a robust mechanism for ensuring that consent and data usage policies are transparent and actionable by users, thereby fostering trust in connected car ecosystems.

More broadly, Elmimouni et al. [5] explore the characteristics of Privacy-Enhancing Technologies (PETs) across various contexts, including vehicular applications. Their work examines laypersons' and experts' perceptions of PETs, providing valuable insights into their usability and effectiveness. This study underscores the necessity of designing PETs that are intuitive yet robust, ensuring privacy without compromising functionality.

Our work contributes to this growing body of research by introducing PECS, a privacy-focused system for connected vehicles. Specifically, the PECSo module leverages state-of-the-art techniques such as federated learning [6] and secure multi-party computation [7] to enhance privacy while maintaining service quality and its functionalities. By embedding PETs directly into automotive systems, our approach aligns with and extends the principles established in prior research.

[2] https://trustchain.ngi.eu/

3. PECS General Architecture

With the aim of improving consent and privacy preference management in the automotive environment, the Privacy Enrooted Car Systems (PECS) deeply embed the technology systems of contemporary cars into the foundation of privacy, building upon the aforementioned fundamental technologies, which we hereby reintroduce briefly: (a) PECSi, the interface with which the user can interact to define their data processing policies and to view alerts, (b) PECSo, the component that is responsible for data obfuscation, and (c) the Blockchain, which guarantees the integrity of the user-defined policies.

This results in multiple services running on the infotainment system: after their implementation, we were able to verify that all services and applications work on the Android operating system, which can be found on aftermarket head units.[3] PECS inherits the open-source nature of Android, and public access not only will be granted to the PECS technology, but we will also discuss the advantages this brings when it comes to verifying the fairness of the data processing of applications complying with PECSo. This will guarantee that people are in control of their data, ultimately helping with GDPR compliance in this field.

Figure 1 shows the architecure of PECS and how its different components interact with each other. While all component exchange messages to perform their different operations, the user only interacts with PECS through PECSi. The full features of PECS, offered by PECSi, PECSo and the Blockchain, are supported by some Android services that have diiferent functions, such as reading data from the car (e.g., from the OBD-II interface or from the CAN Bus), storing the user-defined policies where applications (and PECS itself) can subsequently retrieve them, sending messages on the CAN Bus to activate haptic feedback when PECSi detects a violation, and more. However, following the least privilege principle, only the services that actually need root privileges are run with elevate privileges (i.e., the ones that manage engine data readings and haptic feedback signaling on the CAN bus), while all the other PECS services are run with user privileges.

While PECSi provides drivers with an immersive experience that helps them comprehend the different kinds of personal information that the various services use, PECSo acts as a shield for the personal data, obfuscating them while preserving functionality. PECSo guarantees that personal information remains private and secure even in the face of advanced data analytics by combining and customising a variety of techniques to the specific target domain, such as federated learning and secure multi-party computation. Moreover, public verifiability and integrity of users privacy policies is guaranteed through their upload on the permissioned blockchain Hyperledger Fabric, after signing the policy with a private key associated to the user. Because the blockchain guarantees verifiability and integrity, in the event of a dispute between the user and the service provider on the data processing, the user can publicly prove that they selected a particular policy with public verifiability; because only the pseudonym, and not the real identity of the user, is stored on the blockchain, the policy remains not linkable to any physical person until, in case of dispute, the user discloses to be the policy owner by proving it with its digital signature. It is worth remarking that safeguarding the user identity is paramount to avoid the risk of profiling — in fact, if all publicly agreed upon policies from all users were available online and linkable to their identities, it would be easy for anyone to profile all users and even infer Personally Identifiable Information (PII). Additionally, policy integrity also safeguards the service provider, as the Blockchain inherently guarantees non-repudiation for all uploaded policies. Hence, the user cannot deny they had set a specific policy, in case of dispute.

As we will see in the next sections, PECSi communicates with the PECSo modules using AIDL transactions, while the mechanisms that manage the communications between PECSi (including its inner components) and the Blockchain involve FTP and are not discussed within this paper.

[3]For example, see https://www.amazon.com/PEMP-Screen-Android-Qualcomm-2005-2010/dp/B08V8T22KV/

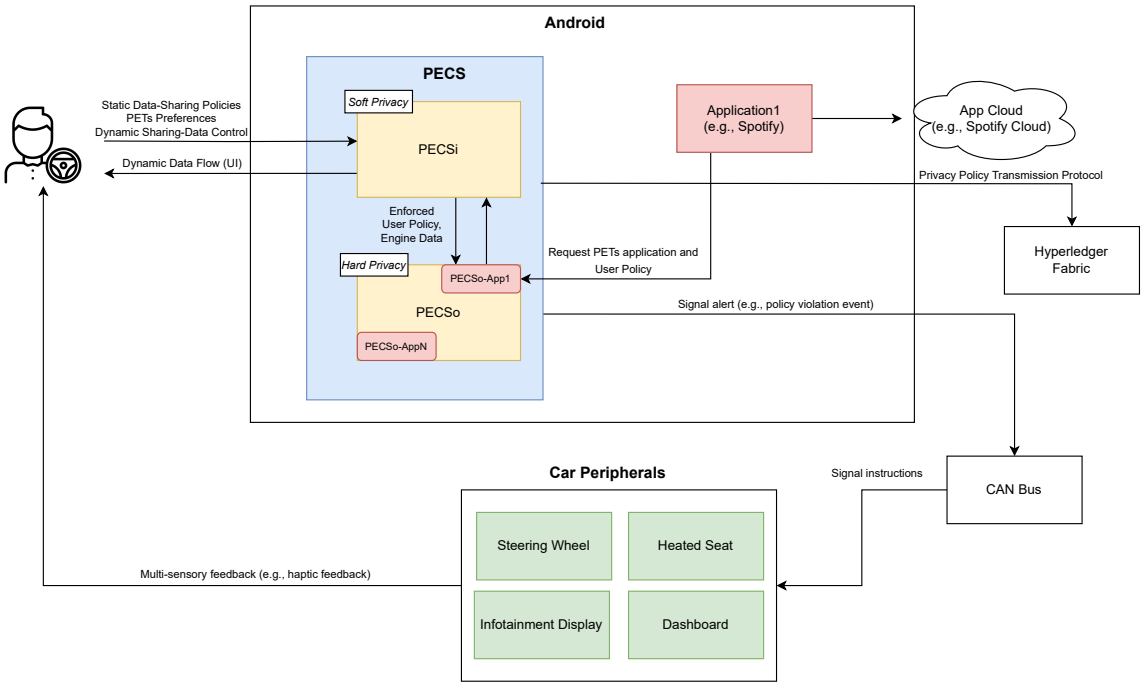


Figure 1: The PECS Architecture.

4. The PECSo Module

In this section, we focus on the design of PECSo, showing its peculiarities and how it can be used to protect user data in automotive environments.

4.1. PECSo Compliance for Applications

As mentioned earlier, PECSo serves as a shield of protection for personal data, by obfuscating them with one or more PETs while preserving functionality. Instead of being a single piece of software, PECSo aims to be a standard for developers to build PECS-compliant applications that implement one or more PETs while maintaining software functionalities and business logic for the service providers. In fact, building a single component that applies multiple PETs on data coming from different applications, in different formats, with multiple goals, and preserving business logic would be a nearly-impossible task with the technologies currently available. Instead, we build PECSo as a container for applications compliant to the PECSo standard[4]: as compliance to all the rules we set to ensure that data is protected adequately can be a daunting process for applications that are already in the maintenance phase of the Software Development Life-Cycle, we designed three compliance levels for the PECSo environment to increase accessibility, so that service providers can choose the level of compliance they want to achieve, related with the resources they are willing to spend for the process.

An application to be used within the automotive context is considered to be PECS-compliant if it respects one of the following tiers of compliance:

Tier 1: PET Implementation. The service provider implements, within its application, any Privacy Enhancing Technology to protect data used in one or more of its functionalities. The user must be able to choose from the application interface whether they want to use the available PET or not. All services offered by the application must be guaranteed in either case. The service provider is allowed to inform the user on how the quality of the service can change if the user enables the PET (e.g., the

[4]In this paper, we also identify these applications as “PECS-compliant applications”, using these two terms interchangeably.

application will be less precise in suggesting the next song to listen to, if it does not have access to the GPS location). The application must inform the user on the differences in the processing of their data (e.g. if the user enables Federated Learning, no data will be sent to the service provider's servers, but only insights learned by the FL model will be shared).

For example, a Tier-1 application that counts the average hours of music listening of all users in a certain group, could give the group members the ability to calculate this data using Secure Multi-Party Computation instead of sending the data to the servers. In this way, the average hours of music listening will be calculated anyway, but the server did not take part in the computation, ignoring the total hours of music listening of each participant. This is of course assuming that the server did not have this information already, for example because the served application is not a music streaming service.

In brief, in order to be compliant with Tier 1 it is required that the application:

1. Implements a PET to protect data processing in one or more application features;
2. Checks user's agreement to adopt the PET;
3. Shows an information paragraph that explains how the quality of service and data processing changes if the user decides to use the PET.

Tier 2: User Policy Check. The application must listen to Android IPC communications by implementing the AIDL (Android Interface Definition Language) interface that we called PECSo-Binder. Said transactions will be sent by PECSi and will contain updates on the policies that the user decides via the PECSi interface. The policies sent by PECSi are encoded in JSON format. Each time the application needs to use a permission, and each time the application needs to request a new permission, it must check if the last policy sent by PECSi allows the use of said permission. If not, the application must not use the permission, even if it was granted by the user at a previous time, and even if it limits the application features. The application is allowed to show a notification that explains that the permission was not granted by the chosen PECSi policy.

For example, if there is a Tier-2 application a that reads engine data in order to analyse the user's driving style and give some feedback regarding how to improve it, this application checks first if the user did manually grant these privileges to a on PECSi. If reading engine data was allowed, then the application can read the data and perform the analysis; otherwise, it will not do so.

To summarise, in order to be compliant with Tier 2 it is required that the application:

1. Is fully compliant to Tier 1;
2. Implements the AIDL interface for listening to the PECSi communications containing the latest user policy;
3. Stores the received policy locally, and extracts permissions given by the user;
4. Verifies if the policy allows the use of a certain permission, before using or even requesting it.

Tier 3: Segregation of Duties. The application is split in two modules, that we call (1) the PECSo-APP module and (2) the Interface module. The PECSo-APP module is an *open-source* Android application that runs as a background service and that can be developed by the service provider (i.e., the same entity that develops the application), or by any third party. This module receives the user policy directly from the PECSi backend service, which runs with root privileges, hence, it has writing rights on the PECSo-APP's internal storage. Additionally, the PECSo-APP module is also responsible of executing the PETs necessary for Tier 1 compliance. Hence, it must keep listening for AIDL transactions coming from the Interface module, that is, the module that is responsible for all other functionalities that are expected from the application. The Interface module allows the user to choose if they want to use a PET, and explains the data processing changes if the PET is allowed, in the same way that was necessary for Tier 1 compliance. Each time the Interface module needs a permission, it asks the PECSo-APP module the latest policy, and acts accordingly, as explained for Tier 2. Essentially, the Tier 3 compliance consists in the split of the PET implementation from the rest of the other application functionalities, to achieve a logical separation between the hard privacy technique and the rest of the application. The PECSo-APP

module can be loaded in the public repository of PECS, so that everyone can verify the application of the PET before sending data to the Interface module. Hence, if the user chose to activate the PET, the Interface module should never see any raw data, but only their representation after the PET processing.

In fact, if we look back again to Figure 1, we see that Application1, that is, the Interface module of the application, interacts with its PECSo counterpart (i.e., PECSo-APP) to retrieve user policies and request PET application. In other words, Application1 does not have access to data for features that are protected by the PET. PECSi communicates engine data and user policies to all PECSo-APPs, but not to Application1, guaranteeing data segregation. While Application1 could be downloaded from the normal application stores, PECSo-APPs should be downloaded from the open-source PECS repository only, to guarantee their authenticity and to be always peer-reviewable, that is, everybody can verify the correct implementation of the PETs. The repository can be maintained and supervised by the PECS service provider, although third parties and application developers might help in developing a PECSo-APP module for some applications. Finally, although it is known that AIDL transactions might be vulnerable to MITM attacks from malicious applications installed on the device, the likelihood of applications that exploit this vulnerability being deployed in real application stores is decreasing in time, also thanks to solutions like Play Protect that analyse applications before making them available for download.

Summarising, in order to be compliant with Tier 3 it is required that the application is split in the two aforementioned components, which in turn must meet their related requirements, taken from Tiers 1 and 2.

Hence, it is required that the PECSo-APP component:

1. Implements a PET to protect data processing in one or more application features;
2. Implements the AIDL interface for listening to the PECSi communications;
3. Stores the received policy locally, and extracts permissions given by the user.

Additionally, it is required that the Interface component:

1. Checks user's agreement to adopt the PET;
2. Shows an information paragraph that explains how the quality of service and data processing changes if the user decides to use the PET;
3. Verifies if the policy allows the use of a certain permission, before using or even requesting it.

5. Use-Case Scenarios

During the final phases of the development of PECS, we created a prototype within a real car, mounting an Android head unit on which we had installed the whole PECS system, including two test PECS-compliant applications we developed ourselves. While analysing all parts of the prototype is outside the scope of this work, in this section we focus on the two PECS-compliant applications that we designed and tested. While one application uses Federated Learning and the other one uses Secure Multi-Party Computation, it is worth mentioning that is the service provider (or the developer) who choses which PET to introduce in their application, so that they can choose the PET that better aligns with their business model.

Before describing each PECS-compliant application, in the next sections we first summarise how the selected PETs work and then we give some details on our implementation.

5.1. A Tier-1 Compliant App: "Average Speed"

Average Speed is a test application that calculates the average vehicle speed of all its users at any given moment. The application implements Secure Multi-Party Computation as a PET, because it easily allows to compute averages in a secure way.

5.1.1. Secure Multi-Party Computation Basics

Secure Multi-Party Computation (SMPC) allows different clients to perform a computation using inputs from other clients, but at the same time, this happens in such a way that no individual party gains access to the private inputs of the others, without the need of a trusted third party [8]. While traditional methods for data sharing could allow an adversary to retrieve the content of the transmitted data, hence potentially violating the user's privacy, in SMPC only part of the secret, called a "share", is sent to the other clients for the computation. Therefore, even if the adversary were listening to the transmission, they are unable to reconstruct the secret knowing only one share. Another way to perform secure multi-party computation is homomorphic encryption, which allows specific computations to be performed on ciphertexts, generating an encrypted result that yields the result of the operations performed when decrypted [9].

Being able to perform a computation without disclosing its inputs bears significant advantages related to the privacy of the users, as everybody can contribute to the computation without giving away sensitive information. This is particularly true when dealing with special categories of personal data, such as health status, political opinions and religious beliefs, but also with personal data that can violate the user's privacy, such as geo-localisation or the driving style.

Hence, in the Average Speed application, the computation of the average vehicle speed is done in a secure way if the user enables SMPC, without the need to actually disclose the current speed to the other parties involved in the computation or to the service provider itself. Whereas this also means that the service provider loses direct access to data protected via SMPC (in this case, the speed of each user taken singularly), it still keeps the ability to make statistical inferences on said data.

5.1.2. Implementation

We created a sample scenario using the EasySMPC [10] framework. Recall that this is a Tier-1 compliant app, hence, it meets the following requirements:

- The application features a PET to protect data processing in one or more application features. In this case, the chosen PET is Secure Multi Party Computation (SMPC), to avoid the submission of driving data to other parties to get the average speed of all involved vehicles;
- The application features an interface in which the user can choose if they want to use the PET. In this case, there is a switch that the user can use to select whether they want to use SMPC;
- The application features an information paragraph that explains how data processing changes if the user decides to apply the PET. In this case, the paragraph is shown on top of the screen, as we can see in Figure 2.

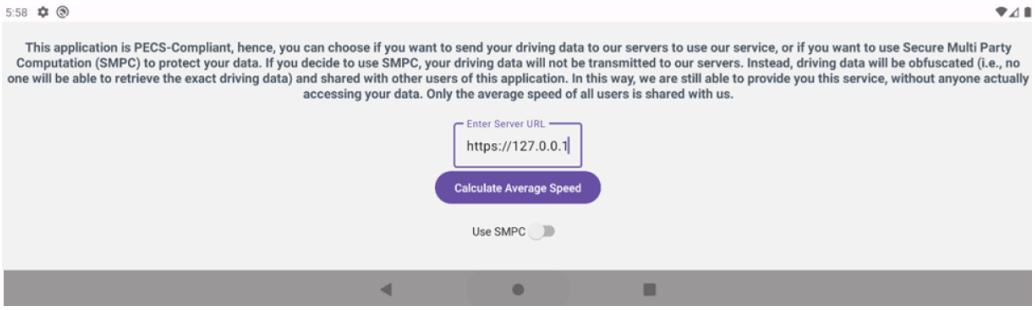


Figure 2: The Main Screen of the "Average Speed" Application.

EasySMPC [10] is designed for secure multi-party computation and to be accessible to non-technical users. Its client and server executables are available on GitHub, no coding skills are required thanks to a user-friendly GUI. In "automatic mode", clients exchange messages via an HTTP-based micro-service that forwards data shares to all participants. A command line version also exists, retaining

automatic mode functions. However, since our application needs to run in an Android setting, we adapted EasySMPC for Android to integrate with PECS. Porting EasySMPC to Android involved code refactoring because, despite both using Java, Android's libraries differ from standard Java ones. Thus, core functionalities were rewritten with Android libraries for secure computation.

EasySMPC uses Arithmetic Secret Sharing as outlined by Demmler et al. [11], enabling secure computation by distributing secret shares of `1` bits. These shares are designed so that their `sum mod 21` reveals the secret, preventing retrieval without all shares. In multiparty scenarios, Arithmetic Secret Sharing supports secure function computation. Demmler et al. [11] also describe a setup where clients generate input shares, split between two non-colluding servers for computation, which then send results back to clients.

EasySMPC supports addition on shares, using a two-step process. Clients first exchange secret shares, summing them into intermediate shares, then repeat the process to reconstruct the result. For client data exchange, we use EasyBackend, EasySMPC's server component,[5] which supports message exchange via HTTP-based micro-services. This software runs on a dedicated server and requires Java, Spring, Keycloak, Maven, and PostgreSQL. In a PECS-compliant production setting, the entity hosting EasyBackend would be the service provider.

When it comes to using our Tier-1 PECS-compliant application, the user simply chooses if they want to activate SMPC or not. Then they type the EasyBackend server's address (in a real production environment, this would be automatically pulled from a configuration file) and then they start the computation. Our tests show that our Android porting of EasySMPC is slower than the official EasySMPC CLI tool, possibly due to computational constraints of the devices in which the Android app runs, but all computations concluded successfully. All source code can be found in the PECS repository.[6]

5.2. A Tier-2 Compliant App: "Engine Faults"

Engine Faults is a test application that tries to infer if the engine is currently experiencing one of the three malfunctions it is designed to recognise. To do so, the application implements Federated Learning, using the EngineFaultDB [12] as the training dataset. The network we use for the purpose is the same that is discussed in the EngineFaultDB paper.

5.2.1. Federated Learning Basics

In normal machine learning approaches, data are gathered within a so-called "dataset" that is then sent to (or aggregated by) a server that subsequently performs the training process. Hence, all data are stored within a single entity, which poses risks related to data breaches and unauthorised access. Furthermore, all data need to be transferred to the server, which increases the attack surface for adversaries, as the transmission of data becomes another possible vector for stealing them.

Federated Learning approaches, instead, work in the opposite way: data are kept within the device that generates them, and the information that is transmitted is the updated model (or, rather, its updated weights) that results from a local training process [13]. This represents a paradigm shift in how the training process works and in how data must be protected. With this approach, the training happens within the device that generates the data, hence, there is no need to transfer data anymore. This has three main implications:

- Data is now scattered between all devices that participate in the Federated Learning process, hence, it will be increasingly harder for an adversary to violate all devices and steal data used for the training;
- Communications only contain the representation of what was learned from the data, and not the data themselves, hence, an adversary that is able to steal the content of the model weight's transmission is not able to infer any data from it;

[5] See https://github.com/easy-smpc/easy-backend
[6] See https://github.com/NGI-TRUSTCHAIN/PECS/tree/dev/pecso/PECSO-SMPC

- The computational burden for the training process is shared between all entities that participate in the federated learning, instead of being entirely managed by the centralised server.

Hence, the service provider only has to provide a base model to be trained, which can also be pre-trained, so that the Federated Learning process only serves the purpose of enhancing the model, instead of making it functional from scratch, as this can be hard to achieve in a pure federated learning environment. Federated Learning also has the advantage of contrasting the so-called concept drift, if the characteristics of the task that the model tries to solve slightly change over time.

This is not surprising if we look more closely at how Federated Learning works in practice:

1. Each client gathers data for the training, within an internal dataset, or by generating or reading data at runtime;
2. Once the clients and the server are ready for the training, the server sends to the clients the current weights of the model, and a new training *round* begins;
3. Each client trains the model locally, for a certain number e of *local epochs*. An *epoch*, in the machine learning jargon, indicates the training of a model on all training samples in the dataset;
4. Once the client terminates this activity, it sends the weights of the trained model to the server;
5. The server collects all results from the clients for the current training round, and aggregates them to update the weights of the model;
6. If the goal number of rounds r has been reached, the training ends. If r has not been reached, the process is restarted from step 2.

The parameters e and r are usually chosen by the server (hence, by the service provider's developers) within the so-called *federated learning strategy*. This means that the number of rounds and of local epochs, along with the details of the model, will change from implementation to implementation [13].

The reader may have noticed that the details of the models updated from a client are never used by the server "as is", but they are aggregated with the results obtained by the other clients. This means that the total improvement of the model performance for each round is (at most) the best result obtained by a single client, which in the average case means that the model will only make a smaller step towards the optimal solution of the problem. Intuitively, if the model is not pre-trained, a large number of rounds (or a large amount of data) is needed to make the model converge towards an optimal solution.

5.2.2. Implementation

We created a sample scenario where an app collects engine data to determine if there are any issues or if the engine is working normally. Because driving style data could potentially violate user privacy, this scenario demonstrates how federated learning can be used to avoid sharing data with service providers. Recall that this is a Tier-2 compliant app, hence, it meets the following requirements:

- The application features a PET to protect data processing in one or more application features. In this case, the chosen PET is Federated Learning, to avoid the submission of driving data to a server to get the engine status;
- The application features an interface in which the user can choose if they want to use the PET or not. In this case, there is a switch that the user can use to select whether they want to use Federated Learning or not;
- The application features an information paragraph that explains how data processing changes if the user decides to apply the PET. In this case, the paragraph is shown in the main screen;
- The application implements the AIDL interface for listening to the PECSi communications containing the latest user policy. In this case, for testing purposes on systems not connected to a car, we developed another component that generates a policy and sends it via the AIDL interface;
- The application stores the received policy locally, and extracts permissions given by the user. In this case, our application tries to retrieve the policy when the user updates the engine status;

- The application verifies if the policy allows the use of a permission, before using or requesting it. In this case, our application looks for permission to read engine data stored on the device.

The Federated Learning part is managed using Flower, a FL framework [14]. Flower supports multiple architectures and offers various examples on its GitHub repository to help start projects across different environments. We started with *Flower Android Client Example with Kotlin and TensorFlow Lite 2022*.[7] Instead of using the original case study, we altered the source code to create the mock application to detect engine problems. We employed the EngineFaultDB [12] dataset for training and testing our model. We implemented a TensorFlow Lite version of the neural network featured in EngineFaultDB for our federated learning setup. This network processes 14 car sensor parameters (such as engine RPM, speed, and fuel consumption rate) with a single hidden layer of 9 nodes, outputting a label from 0 to 3. A 0 indicates the proper function of the engine, while other values represent different malfunctions. We verified in another work that the TFLite version of the model still has a good performance in both training and inference, perfectly in line with other non-quantised versions of the model [6].

When it comes to using our Tier-2 PECS-compliant application, the user chooses if they want to activate FL or not. If they decide to activate it, the FL settings page becomes available and the user can navigate into it. Then, the user can type the address of the server and their ID, and then they can load the dataset in memory. In a real production environment, the ID and the address of the FL server would be hardcoded in a configuration file of the application, while data used for training could be read at runtime from sensors. Finally, the user can start the training process, which usually lasts a few seconds. In our test, we were able to successfully complete FL tasks with one and two participants. The source code of the application can be found in the PECS repository.[8]

6. Conclusions

In this paper, we briefly introduced the Privacy Enrooted Car Systems (PECS) project and thoroughly discussed PECSo, the component that is responsible for user data obfuscation within automotive applications. We first clarified that PECSo is not a single piece of software but a compliance framework for automotive applications. The compliance framework is divided into three levels, becoming increasingly more restrictive on how much the application must comply with the PECS specifications. In the third tier, applications receive privacy policies from the PECSi component via an AIDL interface, so that user choices regarding which data to use can be respected, and implement Privacy Enhancing Technologies (PETs) to protect personal data used within the application and segregate the execution of said PETs to an open-source module that can be publicly assessed and checked. In this way, the application itself never accesses the actual personal data, but only processed what is output by the PET. We then showed two application prototypes we tested on a real car that featured Secure Multi-Party Computation and Federated Learning as PETs. Future work in this field might include the investigation of additional ways to protect AIDL transactions from Man-in-the-Middle attacks and developing a Tier-3 PECS-compliant application to give developers a baseline to build their application.

Acknowledgments

This work was funded by the EU in the framework of the NGI TRUSTCHAIN project, grant N.101093274.

Declaration on Generative AI

The authors used ChatGPT and Grammarly in order to do: grammar and spelling check, paraphrase and reword. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

[7] See https://flower.ai/docs/examples/android-kotlin.html
[8] See https://github.com/NGI-TRUSTCHAIN/PECS/tree/dev/pecso/PECSO-FL

References

[1] G. Bella, G. Castiglione, S. Esposito, M. Raciti, S. Riccobene, Not sure your car withstands cyberwarfare, in: 2024 IEEE International Workshop on Technologies for Defense and Security (TechDefense), IEEE, 2024, pp. 154–159.

[2] G. Bella, G. Castiglione, S. Esposito, M. G. Mangano, M. Marchetti, M. Maugeri, M. Raciti, S. Riccobene, D. F. Santamaria, Privacy-enrooted car systems (pecs): Preliminary design, in: EAI SmartSP 2024 - 2nd EAI International Conference on Security and Privacy in Cyber-Physical Systems and Smart Vehicles November 7-8, 2024 New Orleans, United States of America. To App. to, 2025.

[3] J. Walter, B. Abendroth, T. von Pape, C. Plappert, D. Zelle, C. Krauß, G. Gagzow, H. Decke, The user-centered privacy-aware control system pricon: An interdisciplinary evaluation, in: Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES '18, Association for Computing Machinery, New York, NY, USA, 2018. URL: https://doi.org/10.1145/3230833.3233269. doi:10.1145/3230833.3233269.

[4] M. D. Pesé, J. W. Schauer, M. Mohan, C. Joseph, K. G. Shin, J. Moore, Pricar: Privacy framework for vehicular data sharing with third parties, in: 2023 IEEE Secure Development Conference (SecDev), 2023, pp. 184–195. doi:10.1109/SecDev56634.2023.00032.

[5] H. Elmimouni, E. Shusas, P. Skeba, E. P. S. Baumer, A. Forte, What makes a technology privacy enhancing? laypersons' and experts' descriptions, uses, and perceptions of privacy enhancing technologies, in: I. Sserwanga, A. Goulding, H. Moulaison-Sandy, J. T. Du, A. L. Soares, V. Hessami, R. D. Frank (Eds.), Information for a Better World: Normality, Virtuality, Physicality, Inclusivity, Springer Nature Switzerland, Cham, 2023, pp. 229–250.

[6] M. Maugeri, M. I. Paolo Morana, S. Esposito, G. Bella, User-empowered federated learning in the automotive domain, in: 2024 IEEE International Conference on Blockchain (Blockchain), 2024, pp. 669–674. doi:10.1109/Blockchain62396.2024.00098.

[7] J. I. Choi, K. R. B. Butler, Secure multiparty computation and trusted hardware: Examining adoption challenges and opportunities, Secur. Commun. Networks 2019 (2019) 1368905:1–1368905:28.

[8] F. Bayatbabolghani, M. Blanton, Secure multi-party computation, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 2157–2159. URL: https://doi.org/10.1145/3243734.3264419. doi:10.1145/3243734.3264419.

[9] A. Acar, H. Aksu, A. S. Uluagac, M. Conti, A survey on homomorphic encryption schemes: Theory and implementation, ACM Computing Surveys (Csur) 51 (2018) 1–35.

[10] F. N. Wirth, T. Kussel, A. Müller, K. Hamacher, F. Prasser, Easysmpc: a simple but powerful no-code tool for practical secure multiparty computation, BMC Bioinformatics 23 (2022).

[11] D. Demmler, T. Schneider, M. Zohner, Aby - a framework for efficient mixed-protocol secure two-party computation, in: Network and Distributed System Security Symposium, 2015.

[12] M. Vergara, L. Ramos, N. D. Rivera-Campoverde, F. Rivas-Echeverría, Enginefaultdb: A novel dataset for automotive engine fault classification and baseline results, IEEE Access 11 (2023) 126155–126171. doi:10.1109/ACCESS.2023.3331316.

[13] L. Li, Y. Fan, M. Tse, K.-Y. Lin, A review of applications in federated learning, Computers & Industrial Engineering 149 (2020) 106854. URL: https://www.sciencedirect.com/science/article/pii/S0360835220305532. doi:https://doi.org/10.1016/j.cie.2020.106854.

[14] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, et al., Flower: A friendly federated learning research framework, arXiv preprint arXiv:2007.14390 (2020).