2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

Private Drivers Identification based on users' routine

Gianpiero Costantino and Ilaria Matteucci
IIT-CNR
Pisa, Italy
gianpiero.costantino@iit.cnr.it
ilaria.matteucci@iit.cnr.it

Davide Micale
Dip. di Matematica e Informatica
University of Catania
Catania, Italy
davide.micale@phd.unict.it

Giuseppe Patanè
Park Smart Srl
giuseppe.patane@parksmart.it

***Abstract*—This paper presents Private Secure Routine (PSR) as a paradigm with two main objectives: i) identify drivers depending on their habits/routine and ii) keep private drivers' data. We implemented PSR exploiting the secure Multi-Party Computation (MPC) technique against a honest-but-curious attacker model. Moreover, we evaluated PSR by establishing its accuracy in combination with other existing research works based on machine learning techniques. Evaluation of PSR is performed on different test-beds, considering single-owner and two-owners identification.**

***Keywords*-Driver identification, Privacy, Machine learning**

I. Introduction

Intelligent Transport System (ITS) integrates information and communication technologies into road transportation and offers applications [1] to users such as road safety, traffic efficiency and services.

In this paper, we focus on *authorized drivers identification*. An *authorized driver* for us is a family member, or customers of a car sharing company that is licensed to use the car, i.e., he/she signed an agreement or got explicit authorization to use the car. Driver identification may lead several advantages to the driver such as, customized services, possible discounts, e.g., related to insurance, and it can also be exploited to discourage car theft: in 2020 New York City and Los Angeles have seen a soaring of car thefts [2].

This paper proposes Private Secure Routine (PSR) paradigm for drivers identification that distinguishes authorized drivers of a car from the others in a privacy preserving way. PSR identifies drivers using cars' sensors data gathered, for instance, using OBD-II [3] interface or directly from the CAN bus [4] of the vehicle. Through the data sharing within an ITS architecture, PSR is able to build an accurate model of authorized drivers. PSR exploits the secure Multi-Party Computation (MPC) technique to guarantee that drivers' data are exchanged in a privacy preserving way. We experiment the Private Secure Routine paradigm on two different test-beds and the results are quite promising. As far as we know, PSR is the first algorithm that identifies drivers maintaining private the models and the data of vehicles. PSR exploits the computation units of the RUs so vehicles do not need powerful devices on board.

In literature there are other solutions based on machine learning and neural network techniques for driver identification. Martinelli et al. compared different Decision Trees algorithms on dataset Θ using all features on the research [5] and using only the six best features in [6]. Authors obtained up to 99,2% of precision and recall using J48. Uvarov et al. [7] verified how accurate can be driver identification models using only public sensors' data available with every OBD-II dongle. Authors best result is 79% of accuracy using RF in multi-driver identification whereas on the owner identification problem authors obtained 99% of accuracy.

The paper is structured as follows: Next section presents the background about Neural Network and §III describes the reference architecture on which Private Secure Routine works and depicts the attack models. §IV presents the main steps the Private Secure Routine algorithm at design level while §V at implementation level. §VI reports the results of the PSR experiments and §VII concludes the paper.

II. Neural Network at a glance

Neural Networks (NNs) [8] are computing systems that tries to resemble a human brain to perform a specific task. They are composed of simple process units, *neurons*, that resembles the human neurons to constitute a network. Each neuron can receive input information as weighted signal from other neurons of the network through links. The weighted inputs of a neuron are combined with an extra element called *bias*. Weights and bias are called *parameters*. The output of the neuron is the result of the *activation function* that limits the range of output values of the neuron.

The structure of a NN is called *network architecture* and describes how the neurons are arranged and linked. Networks are organized in layers and are distinguished by the number of layers in *single-layer network*, i.e., network with only one computational layer, and *multi-layer* that has an input layer, an output layer and one or more intermediate layers known as *hidden layers*. The input layer is not a computational layer because neurons belonging to this layer does not perform any computation.

Hence, to define a NN architecture we need to define the number of layers, number of neurons of each layer, the links between neurons, the combiner function and the activation functions of the neurons. Then, the parameters of NN are established through a learning process [9] starting from the definition of a dataset. A dataset is a collection of data, represented by means of a table, that contains samples regarding the task. The rows of the table are called *instances* and the columns are the *features* of each instance. The dataset is split into two partitions: the *training dataset* and the *test dataset*. *Training dataset* represents all instances adopted to make experience and evolve the model. Instead, instances of a *test dataset* is employed to verify how much accurate the model is in doing the predictions.

The learning phase recalls the following five steps: i) The parameters of NN are randomly initialized; ii) The network is run with data of the training set as input; iii) Using a *loss function*, the output of the network, i.e., the predictions are compared with the expected answer, i.e., the labels, obtaining the *loss* value. This indicates how bad the network is to make predictions. iv) The parameters are corrected according to the loss value by using an *optimizer* [10] that searches the best parameters minimizing the loss value. The correction is limited by a *learning rate* multiplier. v) The steps from ii) to iv) are repeated depending on training preferences, e.g., training accuracy.

To make the training faster, the training set is split in *batches*. Each optimization step is executed on instances of a batch: once an optimization step is applied to all training data, an *epoch* is completed. The epochs [11] are the number of times the training data are processed.

III. Private Secure Routine

Private Secure Routine (PSR) is a paradigm to identify authorized drivers belonging to the same vehicle in a privacy-preserving manner. Private Secure Routine is built on top of the Secure Routine (SR) paradigm [12]. The advantages of PSR with respect to SR are twofold:

- PSR distinguishes among several authorised drivers depending on their routine. While SR is able to identify only one authorized driver for a target vehicle, i.e., the owner of the vehicle, PSR identifies more than one authorized driver for a target vehicle.
- PSR guarantees that information about drivers and vehicles are exchanged in a privacy-preserving way.

A. Private Secure Routine infrastructure

The PSR paradigm takes as input all pieces of information about drivers and vehicles circulating in the infrastructure and generates models of each driver in each vehicle using *Secure Multi-Party Computation* (MPC) [13].

Secure Multi-Party Computation is a cryptography technique that involves n parties, where each party i holds the input x_i, and all participants want to compute a function $f(x_1, x_2, ..., x_n)$ maintaining private each party input. The function f in a secret sharing scheme is randomly split into n secrets, named *shares*, in such a way that certain subsets of shares can be used to reconstruct the secret and others reveal nothing about it.

We model the transportation infrastructure, which we define as Private Secure Routine infrastructure, on three different layers that communicates one another (Figure 1).

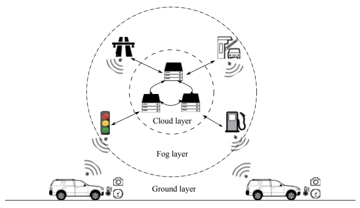

Figure 1: Communication Layers in the PSR infrastructure

The *Ground layer* is composed by *drivers* and *vehicles*. Modern vehicles are equipped with devices such as the *In-Vehicle Infotainment* (IVI) system able to collect, store and communicate information generated by car sensors. While a driver uses the vehicle, this one requests and stores pieces of information about the driver during its usage. Such data are collected via OBD-II [3] or the CAN bus [4] protocol, which transport the data related to vehicle's sensors. Using these data, a *dataset* is generated. It is represented as a table in which all instances referred to a driver are collected. An instance is made of timestamp recorded with the following template: (day, month, year, hour, minute, second and day of the week) and pieces of information about vehicle's components, e.g., oil engine temperature, throttle position, speed and so on. Note that, each vehicle belonging to this layer has an unique identifier "ID", which is represented as a random string, that we employ as data addressing during the model training phase that we show in Section IV.

The *Fog layer* contains all the *Roadside Unit*s (RUs) able to collect and process information coming from the ground layers. RUs are the computational party acting as a bridge between peers of the ground and cloud layer and between vehicles connected to different RUs. In addition, RUs are in charge of share data among entities in a privacy-preserving way by running the MPC technique. Still in this layer, RUs are involved by vehicles to retrieve needed information to train and create drivers' model as described in Section IV. Furthermore, as for vehicles even roadside units have a single identifier "ID" that we use as data addressing during the model training phase.

The *Cloud layer* provides cloud storage resources maintaining a database for the association between drivers and vehicles. Moreover, cloud resources send to RUs the labels for the training of models and also they store roadside units and vehicles public keys.

In this work, we consider the cloud and the vehicles as untrusted entities: their intent is to obtain the data of other vehicles. Instead, RUs are trusted entities, i.e., Trusted Third Party (TTP). If one or more RUs are compromised, the vehicles' data and models are kept private by using MPC. Moreover, we assume that all communications among layers happen through secure channels. This will overcome possible active attacks. Also, we use an asymmetric cryptography protocol [14]. The cloud layer publishes the public keys of all the RUs to the Internet.

B. Attack Model

The cornerstone of Private Secure Routine are the pieces of information about drivers that circulate among the entities belonging to one of the three layers of the depicted scenario. Thus, a possible attacker can play the following attacks:

- Honest-but-Curious (HBC): Also known as Passive Attack; an attacker may exploit the information legitimately gleaned by capturing information exchanged among the three layers of the infrastructure, but he/she will not perform any malicious activity to harvest it.
- Fully Malicious (FM): Also known as Active Attack; an attacker is able to change drivers' information to alter the capability of PSR to identify the drivers. So, the attacker strategy is to succeed in at least one of the following attacks: i) *Impersonation attack* in which attacker forges or alters driver's information that are considered valid by the recipient; II) *Sniffing attack* where the attacker reads the content of any messages exchanged among the layers.

IV. Model Generation

To identify drivers in a vehicle, PSR creates a model able to identify each driver of a target vehicle that circulates on the PSR infrastructure. Model generation depends on different situations that can occur and involve both drivers and vehicles. Hereafter, we highlight three different scenarios and for each of them, we explain how PSR identifies drivers in the infrastructure presented above.

In the PSR paradigm we know that drivers' routines are not fixed, they can evolve over time [15], e.g., a mother who takes for the first time her child to school. Also, drivers' style can be different in particular situations, e.g., heavy rain, snowfall, and so on. If driver identification should fail, a vehicle can use a traditional authentication method e.g., password, voice recognition, and so on, as a fallback method to identify the driver. When a fallback method, out of scope of this paper, is used, a vehicle labels the recent sensors' data as belonging to the authenticated driver. Even if traditional authentication methods fail, sensors' data are labelled as belonging to a non-authorized driver. Moreover, the PSR paradigm transparently authenticates the driver, i.e., PSR continuously compares the actual driving style with its model. Each vehicle periodically issues a new training model to improve the learning of new routines and driving styles.

A. A new vehicle joins the PSR infrastructure

When a driver d_i with her vehicle v_i join the PSR infrastructure for the first time, the creation of a new model is needed to identify d_i. To do this, our PSR requires that the vehicle collects data from its sensors related to driver d_i. Upon the collection of these data, the vehicle v_i generates two labels: one label l_i tied to driver d_i and another label l_{other} to consider data for other drivers different from d_i that may use vehicle v_i. The collected data will be sent to the RUs that will be in charge to train the model for v_i.

Hereafter, we detail how data are collected, shared with the PSR infrastructure and how the model is trained.

Model Initialization: A vehicle does not share the labels directly with the cloud layer. So, v_i first contacts the nearest roadside unit to share its labels and its ID (Step 1 in Figure 2). When the RU receives l_i, l_{other} and the vehicle's ID from v_i, it may forward the labels and the ID to the cloud layer that can grab sensitive information belonging to v_i, e.g., the geographical position of the nearest RU may provide indication about the vehicle location. To avoid this, the roadside unit close to v_i sends the labels and the ID to an intermediary RU randomly chosen (Step 2). Only now, the intermediary forwards the labels and the ID to the server (Step 3) at cloud layer. When labels and the ID are received at the cloud layer, they are stored on a local repository, for instance a database. From now on, v_i and d_i are part of the PSR infrastructure. However, v_i is not yet ready to identify d_i since the model has not been trained.

The vehicle requests the training of its model: The vehicle v_i sends a request to train its model to identify d_i. This step is obtained through a request that is sent to the nearest RU, (Step 4). This one forwards the training request to an intermediary RU, (Step 5). This RU sends the training request at the cloud layer (Step 6). It, then, identifies the label to use, i.e., l_{other}. Note that label l_{other} is identified but not yet sent to RU.

The cloud layer requests the training for v_i's model: To train v_i's model, the l_{other} label is sent from the cloud layer to other vehicles involved in the PSR infrastructure. So, l_{other} is sent, first, to the intermediary roadside unit (Step 7) that forwards the label to the RU closest to v_i (Step 8) and sends l_{other} alongside with the v_i's ID (Step 11) to other RUs within a certain radius, whose size is not relevant here. Finally, each RU forwards the l_{other} label to their connected vehicles together with the IDs of the roadside units involved in the training (Step 12). Note that, the label l_i is not sent because there are no vehicles that have data belonging to the driver d_i. Hence, the only vehicle that has the label l_i is the vehicle v_i itself. At step 12, v_i receives the IDs of the RUs involved in the training.

Vehicles send their shares to train v_i's model: To train v_i's model, each vehicle belonging to the PSR infrastructure must share its collected data sensors. First, each vehicle,

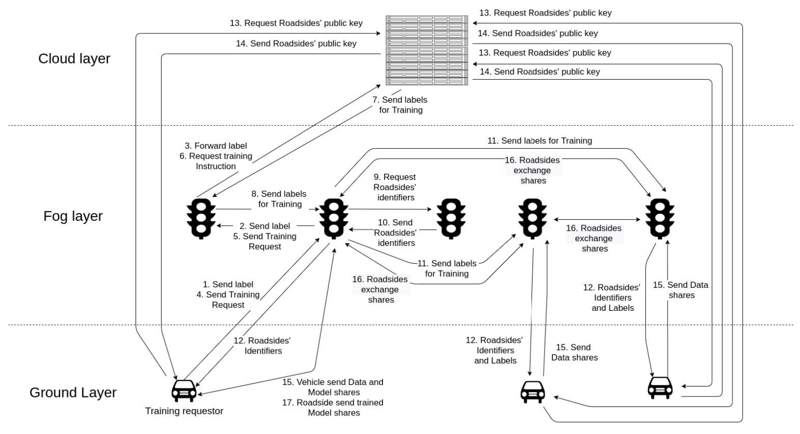
Cloud layer
Fog layer
Ground Layer
Training requestor
13. Request Roadsides' public key
14. Send Roadsides' public key
13. Request Roadsides' public key
14. Send Roadsides' public key
13. Request Roadsides' public key
14. Send Roadsides' public key
7. Send labels for Training
3. Forward label
6. Request training Instruction
11. Send labels for Training
16. Roadsides exchange shares
9. Request Roadsides' identifiers
8. Send labels for Training
2. Send label
5. Send Training Request
10. Send Roadsides' identifiers
16. Roadsides exchange shares
11. Send labels for Training
16. Roadsides exchange shares
1. Send label
4. Send Training Request
12. Roadsides' Identifiers and Labels
15. Send Data shares
12. Roadsides' Identifiers
12. Roadsides' Identifiers and Labels
15. Send Data shares
15. Vehicle send Data and Model shares
17. Roadside send trained Model shares

Figure 2: The PSR Model Generation Workflow

except v_i, labels its sensors' data with the l_{other} label. The sensors' data of vehicle v_i are already labelled. Then, each vehicle obfuscates its data splitting them in shares. Next, each vehicle sends these shares to the nearest RU. Since a roadside unit may reconstruct the original data with all the shares received, the vehicle must encrypt the shares so that only the right RU can have them in plaintext. In the PSR infrastructure, we keep shares secrecy using an asymmetric cryptography protocol, e.g., OpenPGP [16]. OpenPGP is a protocol that provides confidentiality and authentication of messages and data files [17]. Hence, a vehicle, before sending each share to a RU, must obtain the public key of the recipient roadside unit (Step 13) from the cloud layer. Once the corresponding RU public key is retrieved (Step 14), the vehicle uses the key to encrypt the share before sending it to the nearest RU (Step 15). Always at Step 15, the vehicle v_i sends its encrypted shares to the nearest RU. Then the vehicle sends all encrypted shares, the nearest RU will forward them to the other RUs belonging to the infrastructure to train the model (Step 16).

Training v_i's model: All roadside units, involved in the training, decrypt the shares and train v_i's model using the data received from all vehicles in the previous steps.

The vehicle v_i receives the updated model: At the end of training, the roadside units send to v_i the shares containing the new model of d_i. To do this, each RU encrypts the shares with the public key of v_i and sends the resulting shares to the vehicle. From now on, v_i is able to identify d_i and distinguishes her from other drivers (Step 17).

B. A new driver joins the PSR infrastructure

This is the case when a new driver d_i is identified on a vehicle v_i already present in the PSR infrastructure. The vehicle v_i has already a model trained for another driver d_j, with $d_j \neq d_i$, and sensors' data used for d_j are labelled as l_j. Driver d_j must authorize the new driver. The procedure to give d_i the permission to drive the vehicle v_i is out of scope. The vehicle v_i must update the current model to identify also the new driver d_i. To perform this task, we proceed similarly to the previous scenario and the following steps occur.

Model Initialization: The vehicle v_i generates the label l_i. Labels l_{other} and l_j already exist on vehicle v_i since they were created for driver d_j. Then, the vehicle sends the label l_i and its ID to the nearest RU (Step 1). This one forwards the label l_i and vehicle's ID to an intermediary RU randomly chosen (Step 2). Finally, the cloud layer receives l_i and the ID from the intermediary RU and stores them on a database (Step 3). Now, the new driver d_i is part of the transportation infrastructure. However, the vehicle v_i is not yet able to identify d_i since its model has not been trained.

The vehicle requests the training of its model: In this phase, the vehicle v_i sends to the nearest RU a training request (Step 4). The RU forwards this request to an intermediary RU (Step 5) and, then the intermediary sends the request to the cloud layer (Step 6).

The server request the training for v_i's model: The cloud layer sends a training request to all vehicles on the PSR infrastructure within a certain radius. Cloud layer knows the label of driver d_i from Step 3 and the label l_j due to past training for driver d_j. As next step, from the cloud layer the labels l_{other}, l_j and the ID of vehicle v_j are sent to the intermediary RU, (Step 7) which provides to the closest roadside unit of vehicle v_i the labels and the ID (Step 8).

Roadside Unit searches vehicles with authorized drivers in common with v_i: This is a new step with respect to the previous training scenario. We introduce this step since it may happen that driver d_j may be a driver of another vehicle v_j different from v_i. So, we need to consider this situation by collecting also data coming from other vehicles. In this phase, the RU r_i, to which v_i is connected, looks for the vehicle v_j and contacts it through its nearest roadside unit, which we label as r_j. The PSR infrastructure uses a Distributed Hash Table (DHT) [18] to keep track of the nearest roadside unit of each vehicle present in the PSR infrastructure. A DHT provides a lookup table, distributed between peers of a network, to quickly locate data. The Roadside Units are peers of the network. The lookup algorithm uses a key to locate the peer using {key, value} pairs. In particular, the key is the v_j's ID hash and value represents the ID of the nearest Roadside Unit to v_j (Step 9). The RU r_i receives the ID of r_j closest to vehicle v_j (Step 10). Then, r_j receives from r_i the labels l_{other}, l_j and the IDs of v_i and v_j (Step 11). Always at Step 11, r_i sends the label l_{other} and the ID of v_i to the other roadside units. The roadside unit r_j forwards the labels to vehicle v_j and the IDs of the roadside units involved in the model training (Step 12). Also at the same step, each roadside unit sends the label l_{other} and the RUs' identifiers to other vehicles of the PSR infrastructure. Finally, r_i sends only the identifiers to v_i, at the same Step 12.

Vehicles send their data to train the v_i's model: Now, each vehicle shares the data collected from its sensors with the RUs to train v_i's model. First, the vehicle v_j labels with l_j sensors' data that belong to driver d_j. All remaining data not belonging to d_j are labelled as l_{other}. Vehicle v_i sensors' data are already correctly labelled. The other vehicles label their sensors' data as l_{other}. Then, all vehicles get the RUs' public key from the cloud layer (Steps 13 and 14). Then, each vehicle of the PSR infrastructure sends the data, through encrypted shares, to its nearest roadside unit for training (Step 15). Then, these shares will be exchanged among the RUs before the training session (Step 16).

Training v_i's model: Each roadside unit decrypts the received shares and trains the model.

The vehicle v_i receives the updated model: When the training is concluded, the roadside units send to v_i the encrypted shares of the new model exploiting the closest roadside unit r_i (Step 17). Then, the vehicle decrypts and combines all received shares to obtain the new model. From now on, vehicle v_i is able to identify d_i and d_j.

C. Driver identification on a vehicle of the infrastructure

This scenario considers the case in which a driver d_i has already a model in a vehicle v_i and we wish to identify the driver into a different vehicle v_j. Since, data related to driver d_i already exist in the transportation infrastructure, we can directly train her corresponding model. However, even if the model of vehicle v_j is trained using previously d_i driving sessions in v_i, the driver may use a driving style completely different, e.g., a city car is driven in a different way than a sports car. The fallback authentication allows PSR to register the different driving style to later update v_j's model.

Model Initialization: To train the new model, RUs use driver d_i's past data collected by other vehicles of the platform. The resulting model allows vehicle v_j to identify driver d_i. First, vehicle v_j generates a random label l_i for driver d_i. The label l_{other} already exists so it is not necessary to create a new one. Then, vehicle v_j shares the new label alongside with its ID using the nearest roadside r_j (Step 1). Upon label and ID reception, the roadside unit chooses randomly an intermediary RU and forwards the label and the ID to it (Step 2). The intermediary RU forwards the label l_i and the ID to the cloud layer that stores them (Step 3).

Request to train v_j's model: The cloud layer knows v_j has already a model for a driver d_j, with $d_j \neq d_i$. Assuming that driver d_j is already known on the vehicle v_j with the label l_j, v_j's model should include also driver d_i in addition to d_j. Driver d_j could be also known in the vehicle v_k, with $v_k \neq v_j$. Next, a training request for v_j's model with the labels l_i, l_j, l_{other} and the IDs of vehicles v_j, v_i and v_k is sent to the intermediary RU (Step 7). This sends the labels and the IDs to nearest RU r_j (Step 8).

Searching vehicles that already know d_j and d_i: The roadside unit close to v_j looks for those vehicles that already know one or both drivers d_i and d_j. In Step 9, the roadside unit r_j looks for the other RUs that are close to vehicles v_i and v_k to be able to contact them. Then, in Step 10 the roadside unit r_j receives the "IDs" of r_i and r_k respectively nearest to v_i and v_k. On Step 11, r_i receives the labels l_{other}, l_i and the "ID" of vehicles v_i and v_j while r_k receives the labels l_{other}, l_j and the "ID" of vehicles v_k and v_j. At the same step, r_j sends to the other RUs the label l_{other} and the "ID" of vehicle v_j. At Step 12, the roadside units r_i and r_k share the labels respectively with v_i and v_k and the identifiers of RUs involved in the training. At the same time, all the roadside units send the label l_{other} and the IDs to other vehicles of the PSR infrastructure, except v_j that receives only the identifiers.

Vehicles send their data to train v_j's model: Each vehicle sends its data to the RUs involved in the training: vehicle v_i labels with l_i sensors' data that belong to driver d_i and, similarly, vehicle v_k labels with l_j sensors' data that belong to driver d_j. All remaining sensors' data of vehicles v_i and v_k get the label l_{other}. Also, the other remaining vehicles of the infrastructure label their data with l_{other}. The data of vehicle v_j are already correctly labelled, hence no changes are needed. Each vehicle retrieves the public keys of the RUs involved in the training (Step 13 and 14). Then, the vehicle splits data in shares, encrypts the shares with the public keys retrieved in the previous step and sends them to the vehicle's nearest roadside unit for training (Step 15). Such shares are then shared among the RUs (Step 16).

Training v_j's model: All the RUs involved in the training receive the shares, decrypt them and train the model.

Vehicle v_j receives the updated model: At the end of training, the RUs send to v_j the encrypted shares of the new model through r_j, Step 17. Here, v_j decrypts the shares and combine them to obtain the model. From this moment on, the vehicle v_j is able to identify d_i, d_j.

V. Private Secure Routine Implementation

The Private Secure Routine paradigm is implemented by using PySyft framework, a Python library that implements the secure Multi-Party Computation (MPC) for private training of Neural Networks [19]. The framework maintains both parameters of the model and the dataset private.

The PySyft implementation of MPC is secure against the honest-but-curious adversaries [19] but can not guarantee security against active attackers. Some parties could exchange their shares and potentially reconstruct the original values.

We aim to evaluate the impact of MPC on the accuracy results. Hence, we simulate the chain of computations of the parties belonging to the PSR infrastructure without the bottleneck of communications over the network. To do this, we simplify the behaviour of both instances of ground and cloud layer. At ground layer, vehicles generate the dataset and label the instances. At cloud layer, the server sends requests to vehicles to trigger the labelling activity. In our implementation, we employ three roadside units, two of them train the neural network, while the third RU acts as the *Crypto-Store*, i.e., it provides the necessary elements for the computation in the MPC environment [20]. All parties are *Virtual Workers*, i.e., they run on the same computer.

A. Dataset Generation Algorithm

The dataset generation involves the situation in which vehicles generate their dataset to train the vehicle v's model in the different training scenarios described in §IV. Once the dataset is created, vehicle labels the dataset instances (ins) depending on the experiment we are going to setup.

Listing 1 shows how the dataset of the experiments are generated. The algorithm uses the label "Other" to every instances with a label not present in $drivers$, i.e., list of the authorized drivers: if each vehicle is owned by only one driver, d, the list $drivers$ contains only d. In case each vehicle is owned by more than one driver, e.g., two drivers, d_1 and d_2, the list $drivers$ contains these two labels.

Listing 1: Dataset generation

```
function generate_owner_dataset($drivers$, $ins$)
$ins_{labelled}$ ← for each instance in $ins$ set label
    "Other" to instances not made by one of the
    drivers in $driver$
return $ins_{labelled}$
```

The generated dataset has to be shared with the RUs to train the models. Each vehicle creates a private dataset by generating the *shares* (Listing 2) of both training set (85% of instances) and test set (the remaining 15%). The function receives in input the workers' references, one for each RU: the virtual workers that train the model and the crypto provider that setups the Function Secret Sharing algorithm.

The training set and test set are split in batches (Listing 2, lines 5 and 8) with size equal to 1024. To better train the network, the labels are converted into the *one-hot encoding* [10] (Listing 2, lines 6 and 9). A one-hot encoding is a vector of length equal to the number of possible labels, e.g., in case of two owner, the vector size is three: d_1, d_2 and "Other". In Private Secure Routine, the vehicles must create vector of size much bigger than the number of drivers of v: for instance suppose that the one-hot vector is of size $number\_owners\_drivers$ + 1, where the first components of the vector represent the owners of the vehicle and the last component represents the other drivers. A malicious vehicle could easily set its past driving instances has belonging to one of the legit v's owner, e.g., set all instances in the first component as if they belong to the first owner. In case the vector size is too big and the components that represent the legit owners are chosen randomly, the attacker does not know which components represent the legit owners. Hence, she is not able to create fake instances belonging to a legit owner. Then, vehicle creates the shares of training and test set for each virtual worker (Listing 2, lines 7 and 10).

Listing 2: Generation of dataset shares - Vehicle

```
1  function generate_private_dataset($ins_{labelled}$,
       $workers$, $crypto\_provider$)
2  ($ins_{train}$, $ins_{test}$) ← choose randomly 85% of
       instances as training and the remaining as
       testing from $ins_{labelled}$
3  ($x_{train}$, $y_{train}$) ← separate features of the $ins_{train}$
       instances from the respective label
4  ($x_{test}$, $y_{test}$) ← separate features of the $ins_{test}$
       instances from the respective label
5  $loader_{train}$ ← split $x_{train}$ and $y_{train}$ in batches.
       Data are shuffled
6  $loader\_one\_hot_{train}$ ← labels in $loader_{train}$ are one
       hot encoded
7  $loader\_private_{train}$ ← create shares of labels and
       features in $loader\_one\_hot_{train}$, distributing
       them between $workers$ with the help of
       $crypto\_provider$
8  $loader_{test}$ ← split $x_{test}$ and $y_{test}$ in batches
9  $loader\_one\_hot_{test}$ ← labels in $loader_{test}$ are one
       hot encoded
10 $loader\_private_{test}$ ← create shares of labels and
       features in $loader\_one\_hot_{test}$. Return the
       pointers to the shares
11 return ($loader\_private_{train}$, $loader\_private_{test}$)
```

B. Dataset preparation for training

Once RUs have received the private dataset of each vehicle, they are in charge of running the secure multi-party computation technique. Train dataset of each vehicle are combined in a single dataset (Listing 3, line 2). We concatenate the datasets of different vehicles to run the *average imputation*. The procedure handles the, eventually, missing values with the average value of each feature. Also, neural networks require that training dataset is randomly shuffled for an optimal training [10]. Without the dataset concatenation, the average imputation should be run separately for each vehicle dataset, obtaining different averages. Also, at training phase, the neural network should sequentially train each vehicle training dataset. Finally, managing a single training dataset is simpler than managing multiple training datasets. Similarly, test datasets are combined too (line 3).

The average imputation in the training set replaces missing values with the average value of each feature, whereas in the test set null values are filled with the same averages obtained from the training set [21]. First, we calculate the average value of each feature in $loader\_private_{train}$ (line 4). Then, we fill null values with the respective average (line 6) and create new feature columns to keep track of the rows that have null values for each feature (line 5) and the columns of the rows with null values and we apply the average imputation in the test set (lines 7 and 8).

Next step is the normalization of training and test set to speed up the process of model training [22]. We use the *Min-Max Normalization* procedure: for each feature f the dataset values are transformed from the range $[min^f_{loader\_private_{train}}, max^f_{loader\_private_{train}}]$ to the range $[min, max]$ arbitrary using the equation:

$$v_{new} = \frac{v_{old} - min^f_{loader\_private_{train}}}{max^f_{loader\_private_{train}} - min^f_{loader\_private_{train}}}(max-min)+min \tag{1}$$

We choose min equal to -1 and max equal to 1. In Listing 3 line 9, we get the minimum and maximum value for each feature in $loader_{train\_filled}$. Then, we apply the Equation 1 to normalize the $loader_{train\_filled}$. Similarly, we normalize the $loader_{test\_filled}$ using the same $[min^f_{loader_{train\_filled}}, max^f_{loader_{train\_filled}}]$.

Listing 3: Preparation of dataset for training - RUs

```
1  function training_preparation_dataset($array\_loader\_private_{train}$, $array\_loader\_private_{test}$)
2  $loader\_private_{train}$ ← concatenate loaders in $array\_loader\_private_{train}$ in a single loader
3  $loader\_private_{test}$ ← concatenate loaders in $array\_loader\_private_{test}$ in a single loader
4  $averages$ ← calculate average of each feature in $loader\_private_{train}$
5  $loader_{train\_cols\_na}$ ← add a column for each feature, each row contains $TRUE$ if the value is null, $FALSE$ if the value is present
6  $loader_{train\_filled}$ ← fill null values in $loader_{train\_cols\_na}$ with the average in $averages$ of the respective feature
7  $loader_{test\_cols\_na}$ ← add a column for each feature in $loader\_private_{test}$ with null value, each row contains $TRUE$ if the value is null, $FALSE$ if the value is present
8  $loader_{test\_filled}$ ← fill null values in $loader_{test\_cols\_na}$ with the average in $averages$ of the respective feature
9  $scaler$ ← for each feature in $loader_{train\_filled}$ get minimum and maximum values
10 $loader_{train\_normalized}$ ← normalize $loader_{train\_filled}$ in range (-1, 1) according to $scaler$
11 $loader_{test\_normalized}$ ← normalize $loader_{test\_filled}$ in range (-1, 1) according to $scaler$
12 return ($loader_{train\_normalized}$, $loader_{test\_normalized}$)
```

C. Model Generation Algorithm

Each vehicle v creates a new model as described in Listing 4. As a first step, vehicle creates the initial model, (line 2). To optimize the training speed, we adopt a multi-layer Neural Network with three layer architecture: two linear hidden layers with *dropout* [23], as method of regularization, and the *rectified linear unit* (ReLU) [10] as activation function defined as follows:

$$ReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2}$$

The first hidden layer is made of 128 neurons and the second hidden layer contains 64 neurons. The output layer is linear and its size depends on the experiment.

Once the model is created (line 2), the vehicle v generates the shares for the model (line 3). Such shares are sent to RUs in a privacy preserving way. To train the model is required an optimizer that adjusts the model parameters at each epoch to reduce the loss and to increase the accuracy [10]. Then, v defines the optimizer and its parameters (line 4). The model and the optimizer parameters are converted from float to *fixed precision* (line 3 and 5) according to PySyft requirements [24]. Fixed precision represents values with two components: an integer, i.e., the coefficient and the position of the radix point, i.e., the exponent. A value is represented as $value = coefficient * 10^{exponent}$ [25]. Having a low value for the exponent allows RUs to speed up the training but it reduces the accuracy. In the experiments, we keep 3 decimals from the value. Once the model is split into shares and optimizer parameters are converted in fixed precision, the vehicle sends the shares to all the RUs.

Listing 4: Model generation algorithm - Vehicle

```
1 function initialize_model_shares($workers$, $crypto\_provider$, $lr$)
2 $model$ ← create Neural Network
3 $model\_private$ ← create shares of $model$, distributing them between $workers$ with the help of $crypto\_provider$
4 $optim$ ← define the Stochastic Gradient Descent optimizer with the learning rate $lr$
5 $optim\_private$ ← convert the optimizer parameters in fixed precision
6 return ($model\_private$, $optim\_private$)
```

D. Model Training Algorithm

Roadside units are in charge of training the model (Listing 5). As a first step, RUs search the best *Learning Rate* (LR) for the model running a known algorithm [26] that returns shares of the learning rate. The size of the LR influences how much the optimizer adjusts the parameters at each epoch.

The smaller is the LR the more are the epochs necessary for training. On the other hand, the training may never converge to a good accuracy if LR is too high [10].

Once the LR is found, the roadside units send the learning rate shares to vehicle v. The vehicle initializes again the model and the optimizer according to the LR found as seen in Listing 4. From line 5 to 14, roadside units train the model. We set the number of epochs of training (equal to 2 in our experiments). For each epoch i, the parameters of the model change and the accuracy is calculated. Hence, the roadside units must return to the vehicle the most accurate model. First, we define the variables that will contain the best model with the correlated accuracy (lines 5 and 6). For each epoch, we train the model with the training dataset, the optimizer and the loss function MSE (line 8) [27].

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(\hat{y}_i - y_i)^2 \tag{3}$$

where: n is the number of predictions; $\hat{y}_i$ is the i-th value predicted by the NN; y_i is the i-th actual value.

Then, the accuracy of making predictions over the test set is calculated by comparing the results with the correct labels. Next, the stored best accuracy is compared with the last obtained accuracy (line 10). In case there is no best accuracy stored, the current accuracy and the current model are stored (lines 11 and 12). In case the current accuracy is greater than the stored one, the current accuracy and the current model are saved. Otherwise, they are discarded. After the model is trained for all the epochs, the RUs send the shares of the best model and the relative accuracy shares to v (line 15). The vehicle combines the shares and obtains the model trained and the accuracy. Note that the RUs do not need to disclose the accuracy values to make the comparisons. Only v will know how well the model performs.

Listing 5: Model training algorithm - RUs

```
1  function find_lr($workers$, $crypto\_provider$,
       $loader_{train\_normalized}$, $model\_private$, $optim\_private$)
2  $lr\_private$ ← find the best learning rate to train
       $model$ using the training dataset
       $loader_{train\_normalized}$, the optimizer $optim\_private$
        and the loss function MSE
3  return $lr\_private$
4  function train_model($workers$, $crypto\_provider$,
       $loader_{train\_normalized}$, $loader_{test\_normalized}$,
       $model\_private$, $optim\_private$)
5  $best\_accuracy\_private$ ← NULL
6  $best\_model\_private$ ← NULL
7  for $epoch$ = 1 to 2 then:
8  train $model\_private$ using the training dataset
       $loader_{train\_normalized}$ and the optimizer
       $optim\_private$ and the loss function MSE
9  $accuracy\_private$ ← make predictions using the
       values of features in $loader_{test\_normalized}$ and
       calculate the accuracy shares using the
       aforementioned predictions and the label in
       $loader_{test\_normalized}$
10 if $best\_accuracy\_private$ is NULL OR
       $best\_accuracy\_private$ < $accuracy\_private$ then:
11 $best\_accuracy\_private$ ← $accuracy\_private$
12 $best\_model\_private$ ← $model\_private$
13 end if
14 end for
15 return $best\_model\_private$, $best\_accuracy\_private$
```

VI. Private Secure Routine Evaluation

We compare PSR paradigm with Secure Routine [12] and the work in [6], hereafter denoted with M. We experiment PSR on two examples: 1) single owner identification and 2) two owners identification. Then, we evaluate the NN of PSR comparing it alongside an architecture trained without the application of secure multi-party computation. This is because PySyft with MPC is time consuming due to the fact that parties need to exchange several messages within the training phase and that PySyft uses the CPU instead of the GPU that is not yet supported.

A. Experiments

To evaluate PSR, we performed eight experiments on two datasets: Θ [28] and Ψ [29] that have in common 9 features (Table I). The experiments run on a Virtual Machine with an Intel(R) Xeon(R) Gold 6140M using 8 threads, 32 GB of RAM and Ubuntu 18.04 as OS. Our experiments compare PSR with Secure Routine and M on the *Accuracy* metric.

$$Accuracy = \frac{CP}{CP + WP} \tag{4}$$

where CP is the number of *Correct Predictions* and WP is the number of the wrong ones.

1) Single Owner identification (SOI): We aim at identifying if a target instance belongs to the vehicle's owner.

As first experiment, we compare PSR with Secure Routine and M on the Ψ dataset. We select the best feature set in PSR. Since MPC is time consuming, we trained PSR only for two epochs (Listing 5). Secure Routine achieves the best results (Table II(a)). PSR is 6,05% of accuracy lower in comparison with Secure Routine and 4,67% less than M. Note that M do not use this dataset in their research, so we replicated their experiments to establish the accuracy. Despite PSR model being trained only two epochs, it scores an high accuracy keeping private the used data.

We repeat the same experiment on Θ. We use the same feature selected in [12]. The work of M adopted the dataset Θ but they do not calculate the accuracy, so we replicates their experiments to retrieve the accuracy. Table II(b) shows that PSR obtains 89,96% of accuracy, SR achieves the best accuracy. PSR obtains 9,87% of less accuracy than SR.

To measure the impact of MPC on the accuracy of PSR, we evaluate the PSR network comparing the results of the same neural network trained for 2 and 4000 epochs but without the support of MPC.

Then we consider the dataset Ψ. Table III(a) shows that PSR without MPC scores 99,91% of accuracy while PSR with MPC has lost 2,58% of accuracy in comparison with

Table I: Common features description

Feature	Description	Example	Unit
Throttle_pos	Percentage of throttle opening	25%	%
Short_term_fuel_trim_bank_1	Percentage of ratio air/fuel in the first bank of cylinders instantaneous	-3,00%	%
MAF	The air flow mass in the engine	8,12g/s	g/s
Engine_RPM	Number of revolutions per minute crankshaft makes	2100RPM	RPM
Speed	Speed of the vehicle	55km/h	km/h
Timing_advance	Percentage of crankshaft rotation when spark plug fires in advance	0,423%	%
Engine_coolant_temp	Temperature of the coolant/antifreeze liquid mix of engine	90C	C
engine_runtime_minutes	Minutes elapsed from engine start	39m	minutes
engine_runtime_second	Seconds elapsed from engine start	10s	s

(a) SOI in Ψ

PSR	SR	M
93,79%	99,84%	98,46%

(b) SOI in Θ

PSR	SR	M
89,96%	99,83%	99,62%

Table II: Accuracy comparison of Driver Identification paradigms for SOI test bed

the model trained in plain for 2 epochs. This loss of accuracy may be caused by the conversion of input data and model parameters in fixed precision required by PySyft.

Moving on Θ, Table III(b) shows PSR scores slightly lower (0,04%) than the PSR without MPC with two epochs.

(a) Single Owner Identification in Ψ.

PSR 2 epochs	PSR 2 epochs (no MPC)	PSR 4000 epochs (no MPC)
93,79%	96,37%	99,91%

(b) Single Owner Identification in Θ.

PSR 2 epochs	PSR 2 epochs (no MPC)	PSR 4000 epochs (no MPC)
89,96%	90%	97,77%

Table III: Accuracy comparison of PSR for SOI test-bed

2) Two Owners Identification (TOI): Here, we test PSR in case a vehicle is owned by two drivers. Neither Secure Routine nor M were designed and work with two owners identification. Since we are not able to replicate the work in M to test this scenario, we compare PSR only with a slightly modified version of SR able to manage also this case. Once again SR achieves the best result, i.e., 99,69% (Table IV(a)). PSR obtains a respectable 87,51%, i.e., 12,18% less than the same model trained 2 epochs with plain data.

Then, we repeat the experiment also on the other dataset. Table IV(b) indicates that SR obtained an average precision of 99,71%. In comparison, PSR loss 19,63% of accuracy. This results depends on the NN poorly trained, i.e., trained

(a) TOI in Ψ

PSR	SR
87,51%	99,69%

(b) TOI in Θ

PSR	SR
80,08%	99,71%

Table IV: Accuracy comparison of Driver Identification paradigms for TOI test-bed

(a) TOI in Ψ.

PSR 2 epochs	PSR 2 epochs (no MPC)	PSR 4000 epochs (no MPC)
87,50%	92,31%	99.88%

(b) TOI in Θ.

PSR 2 epochs	PSR 2 epochs (no MPC)	PSR 4000 epochs (no MPC)
80,08%	80,04%	95,21%

Table V: Accuracy comparison of PSR on TOI test-bed

for only 2 epochs. An higher number of epochs will increase the accuracy as shows the table V(b), but more on that later.

Also in this case, we evaluate the impact of MPC on PSR. Let us consider the dataset Ψ. Again PSR without MPC achieves an high score, (99,88%), Table V(a). With MPC, PSR loses 4,86% comparing with the same model trained two epochs with plain data.

Moving on the dataset Θ, PSR obtained a better result than the public trained on 2 epochs, i.e., 0,04% more accurate. The best result is obtained by the PSR without MPC fully trained (95,21%).

VII. Conclusion

In this paper, we have presented the Private Secure Routine (PSR) as a novel privacy-preserving paradigm able to identify drivers in an ITS infrastructure. PSR is based on secure multy-party computation. In particular, the PSR paradigm aims to provide a solution to identify more than one driver into the same vehicle and to do it in a privacy preserving way. We evaluated the accuracy of PSR in comparison with two research works present in literature. Although the goal of PSR is on privacy-preserving and multi-owner identification, it obtained elevate accuracy values when compared with the other two research works.

As future work, we will aim to extend PSR to improve its accuracy considering also other privacy preserving techniques.

REFERENCES

[1] "ETSI", "Intelligent transport systems (its); vehicular communications; basic set of applications; definitions," 06 2009, URL:https://www.etsi.org/deliver/etsi_tr/102600_102699/102638/01.01.01_60/tr_102638v010101p.pdf.

[2] S. M. Nir, "Here's Why Car Thefts Are Soaring (Hint: Check Your Cup Holder)," 01 2021, URL:https://www.nytimes.com/2021/01/06/nyregion/car-thefts-nyc.html [retrieved: 07, 2021].

[3] "The OBDII Home Page", "Obd-ii background," URL:http://www.obdii.com/background.html [retrieved: 07, 2021].

[4] International Organization for Standardization, "Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling," https://www.iso.org/standard/63648.html [retrieved: 07, 2021], 2015.

[5] F. Martinelli, F. Mercaldo, V. Nardone, A. Orlando, and A. Santone, "Who's driving my car? a machine learning based approach to driver identification," 01 2018, pp. 367–372.

[6] F. Martinelli, F. Mercaldo, A. Orlando, V. Nardone, A. Santone, and A. K. Sangaiah, "Human behavior characterization for driving style recognition in vehicle system," *Computers & Electrical Engineering*, vol. 83, p. 102504, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045790617329531

[7] K. Uvarov and A. Ponomarev, "Driver identification with obd-ii public data," in *2021 28th Conference of Open Innovations Association (FRUCT)*, Jan 2021, pp. 495–501.

[8] S. S. Haykin *et al.*, "Neural networks and learning machines," 2009.

[9] R. Rojas, *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer-Verlag, 1996.

[10] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 2020, https://d2l.ai [retrieved: 07, 2021].

[11] J. Torres, *First Contact with Deep Learning, practical introduction with Keras*. Watch this space, 7 2018, URL:https://torres.ai/first-contact-deep-learning-practical-introduction-keras/ [retrieved: 07, 2021].

[12] D. Micale, G. Costantino, I. Matteucci, G. Patanè, and G. Bella, "Secure routine: A routine-based algorithm for drivers identification," in *VEHICULAR 2020, The Ninth International Conference on Advances in Vehicular Systems, Technologies and Applications*, 10 2020, pp. 40–45.

[13] O. Goldreich, "Secure multi-party computation," *Manuscript. Preliminary Version*, 03 1999.

[14] W. Stallings, *Network Security Essentials: Applications and Standards*, 6th ed. Pearson, 2016.

[15] I. Lavie, A. Steiner, and A. Sfard, "Routines we live by: from ritual to exploration," *Educational Studies in Mathematics*, vol. 101, no. 2, pp. 153–176, Jun 2019, URL:https://doi.org/10.1007/s10649-018-9817-4.

[16] A. Ulrich, R. Holz, P. Hauck, and G. Carle, "Investigating the openpgp web of trust," in *Computer Security – ESORICS 2011*, V. Atluri and C. Diaz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 489–507.

[17] H. Finney, L. Donnerhacke, J. Callas, R. L. Thayer, and D. Shaw, "OpenPGP Message Format," RFC 4880, Nov. 2007, URL:https://www.rfc-editor.org/rfc/rfc4880.html [retrieved: 07, 2021].

[18] H. Zhang, Y. Wen, H. Xie, and N. Yu, *Distributed Hash Table*, 01 2013.

[19] T. Ryffel, D. Pointcheval, and F. Bach, "Ariann: Low-interaction privacy-preserving deep learning via function secret sharing," 2020.

[20] "OpenMined", "Smpc protocols explanation," URL:https://github.com/OpenMined/PySyft/blob/PySyft/syft_0.2.x/examples/tutorials/advanced/SMPC_Protocols_Explanation.ipynb [retrieved: 07, 2021].

[21] M. Kuhn and K. Johnson, *Applied Predictive Modeling*, 01 2013.

[22] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*, ser. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011. [Online]. Available: https://books.google.it/books?id=pQws07tdpjoC

[23] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[24] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," 2018.

[25] "OpenMined", "precision.py - pysyft source code," URL:https://github.com/OpenMined/PySyft/blob/d811ef1e91e5e2c84fbbf1edf61e6983380b4d16/syft/frameworks/torch/tensors/interpreters/precision.py\#L19 [retrieved: 07, 2021].

[26] L. N. Smith, "Cyclical learning rates for training neural networks," 2017.

[27] K. Das, J. Jiang, and J. Rao, "Mean squared error of empirical predictor," *Annals of Statistics*, vol. 32, 07 2004.

[28] HCRL, "Driving dataset," URL:http://ocslab.hksecurity.net/Datasets/driving-dataset [retrieved: 07, 2021].

[29] C. A. d. S. Barreto, "OBDdatasets," 2018, URL:https://github.com/cephasax/OBDdatasets/blob/master/masterDegreeResearch/dailyRoutes.csv [retrieved: 07, 2021].