AC4AV: A Flexible and Dynamic Access Control Framework for Connected and Autonomous Vehicles

Qingyang Zhang, *Graduate Student Member, IEEE*, Hong Zhong, *Member, IEEE*, Jie Cui, *Member, IEEE*, Lingmei Ren, and Weisong Shi, *Fellow, IEEE*

***Abstract*—Sensing data plays a pivotal role in connected and autonomous vehicles (CAVs), enabling CAV to perceive surroundings. For example, malicious applications might tamper this life-critical data, resulting in erroneous driving decisions and threatening the safety of passengers. Access control, one of the promising solutions to protect data from unauthorized access, is urgently needed for vehicle sensing data. However, due to the intrinsic complexity of vehicle sensing data, including historical and real time, and access patterns of different data sources, there is currently no suitable access control framework that can systematically solve this problem; current frameworks only focus on one aspect. In this article, we propose a novel and flexible access control framework, `AC4AV`, which aims to support various access control models, and provide APIs for dynamically adjusting access control models and developing customized access control models, thus supporting access control research on CAV for the community. In addition, we propose a data abstraction method to clearly identify data, applications, and access operations in CAV, and therefore is easily able to configure the permits of each data and application in access control policies. We have implemented a prototype to demonstrate our architecture on NATS for real-time data and NGINX for historical data, and three access control models as built-in models. We measured the performance of our `AC4AV` while applying these access control models to real-time and historical data. The experimental results show that the framework has little impact on real-time data access within a tolerable range.**

***Index Terms*—Access control, connected and autonomous vehicle (CAV), data security, system security.**

Manuscript received March 6, 2020; revised May 29, 2020 and July 18, 2020; accepted August 9, 2020. Date of publication August 17, 2020; date of current version January 22, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61872001, Grant 62011530046, and Grant U1936220; in part by the Open Fund of Key Laboratory of Embedded System and Service Computing (Tongji University), Ministry of Education under Grant ESSCKF2018-03; in part by the Natural Science Foundation of Shandong Province under Grant ZR2018BF014; and in part by the Open Fund for Discipline Construction, Institute of Physical Science and Information Technology, Anhui University; and in part by the Excellent Talent Project of Anhui University. *(Corresponding author: Hong Zhong.)*

Qingyang Zhang, Hong Zhong, and Jie Cui are with the School of Computer Science and Technology and Anhui Engineering Laboratory of IoT Security Technologies, Anhui University, Hefei 230039, China (e-mail: zhongh@ahu.edu.cn).

Lingmei Ren is with the School of Computer Science, Shenzhen Institute of Information Technology, Shenzhen 518172, China.

Weisong Shi is with the Department of Computer Science, Wayne State University, Detroit, MI 48202 USA.

Digital Object Identifier 10.1109/JIOT.2020.3016961

I. Introduction

With the fast development of sensing, communication, and artificial intelligence technologies, connected and autonomous vehicles (CAVs) have attracted a great deal of attention from industry and academia [1]–[3]. Several autonomous driving systems or commercial products have been released in the industry, such as the Google Waymo [4] vehicle, the Tesla Autopilot system, and the Baidu Apollo platform [5]. With the liberation from driving, increasingly more applications, especially various third-party applications envisioned by [6], will be installed into future CAVs, as supplements to other three kinds of applications, i.e., advanced driver-assistant system (ADAS), real-time diagnostics, and in-vehicle infotainment, to enrich the ride experience. Note that some applications are cross-cutting because they fall under more than one category. However, all of they utilize vehicle sensing data, sensed by a plethora of diverse sensors, to realize their functions. For example, the ADAS leverages the data of installed cameras, light detection and ranging (LiDAR), radio detection and ranging (radar), as well as vehicle status captured from controller area network (CAN) to perceive the surroundings and an attack detection application to access the in-vehicle sound data captured by the microphone as the input of its speech recognition [7]. That means the sensed life-critical data is not only used as the input of ADAS but it is also used by various third-party applications. The malicious applications may pre-empt the limited computing, memory, storage, and network resources to perform their purpose after obtaining data from the CAV system, which will affect the safety of CAVs. Some malicious third-party might tamper data, leading to wrong decisions on driving, even threatening personal and public safety.

In prior researches, the access control technique [8]–[10] is used to protect data from malicious applications by rejecting unauthorized access, which is a security technique regulating who or what can view or use resources in a computing environment. Fig. 1 illustrates the usage of the access control technique in CAV. However, most current researches on CAVs focus on the implementation of autonomous driving vehicle prototypes, including hardware, autonomous driving algorithm [11], and platform [2], [5], [12], and the access control framework enabling the function of applying access control technique to vehicular data is lacking in these researches.

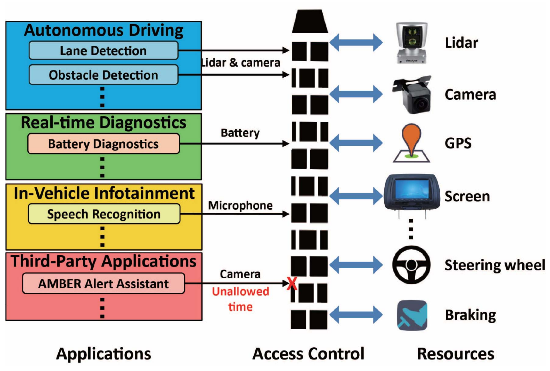


Fig. 1. Function of an access control system.

Hence, the first thing is to build an access control framework for future CAV. However, the future CAV, with various applications, vehicular data and different access patterns on different vehicular, is more complex, thus no one existing access control framework could be applied on CAV, directly. But existing mechanisms used by some operating systems [13]–[15] provide some experiences for referencing.

Typically, different access control models are suitable for different scenarios with different characteristics [16]. For example, attribute-based access control (ABAC) models with high confidentiality are widely used in a cloud-based storage scenario [17]. However, its computing resource costs are high and it might not be suitable for CAV data. In the CAV area, both performance and security are of high priority. Thus, how to choose suitable access control models for different vehicular data is an open problem since no one knows the effects after applying one access control model. But, in any case, some characteristics in the system level should be supported that make the implementation of access control models be more easy and flexible. First, some novel access control models might be proposed for applications, especially for as yet unforeseen applications. Thus, an open-access control framework is required for access control research. Second, the access control framework should be fine-grained. Taking the permits of the steering wheel as an example, it only could be controlled by ADAS applications, but could be monitored by many other applications. Third, access control should be dynamically changed with the context of CAV and system status. For example, in the application proposed in [7], considering the user privacy, the permit to access inside video should be dynamically gained and revoked depending on the recognition of a "help" signal from an inside squeal voice. Finally, the framework should support applying different access control models to the same data with different grains or different data.

However, the design of access control architecture for the future is challenging as it must fulfill the above requirements while it must meet the intrinsic complexity of vehicular data, including historical and real time, and access patterns of different vehicular data. Furthermore, there exist various vehicular data and applications in one CAV, resulting in another challenge when developing such an access control framework for future CAV. Generally speaking, an access control framework should know and identify `which` application is accessing `which` vehicular data with `which` access operation. Here naming is a problem, especially for supporting of fine-grained and dynamic access control, an easy to read and organized naming mechanism is important so that researcher and user could easily set access permissions for different applications, data, and operations.

To tackle the aforementioned issues, in this article, we first introduce the data generated and stored in CAVs and its access patterns based on our observations. Then, we introduce designed access control architecture and data abstraction method to identify application, data, and operation. The proposed framework serves as the access control part of our previous work, open vehicular data analytics platform (OpenVDAP) [6], which is a full-stack edge supported platform that includes a series of heterogeneous hardware and software solutions. We also implement an access control framework prototype based on the proposed architecture, which responds to queries for access actions and records these actions for future auditing. The contributions are summarized as follows.

1) This article is the first to define the data access control problem in CAVs. According to the observations on several CAV platforms, we introduce the characteristics of data and access pattern in emerging CAVs, in terms of real-time data and historical data, while different access patterns are applied. Moreover, different and some new access control models are required.
2) We propose a three-layer access control architecture to protect data on CAVs from unauthorized access. The designed architecture supports fine-grained and dynamic access control, and it is extensible with APIs to assign customized access control models implemented by others, as well as respond to external access actions, resulting in easily extending to meet other access patterns.
3) We demonstrate the proposed design through a prototype framework and evaluate it using different access control models. Experiment results show that our framework has a low impact on real-time data and a high but tolerable impact on historical data, which could be solved by periodically caching application information. Furthermore, we also test our framework on an experimental CAV platform implemented based on OpenVDAP architecture and HydraOne [18], which indicates that our framework could run on platforms with different hardwares.

The remainder of this article is organized as follows. We introduce the access control problem of CAVs in Section II. The designed access control framework is presented in Section III, followed by its instantiated framework in Section IV. Section V shows the results by leveraging video data acquisition for analysis as a case study. We review related works in Section VI. Finally, we conclude this article in Section VII.

II. Problem Statement

The data in CAVs are important since it affects the decision of autonomous driving algorithms and has implications for passenger privacy. How to protect data from unauthorized

TABLE I
REAL-TIME DATA AND HISTORICAL DATA IN CAVS

Category	Examples	Storage location	Potential access pattern
Real-time	GPS, video, Radar, LiDAR cloud point, engine load, etc.	Pub/sub System	Pub/sub
Historical	GPS, traffic data, and metadata of unstructured data.	Database	Web service
	Video, LiDAR cloud point data.	File system	Web service

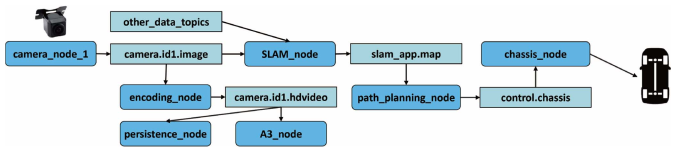


Fig. 2. Example data flow of a real-time camera data.

access in CAVs is a big challenge. In this section, we first introduce vehicular data access patterns. Then, the requirement of CAV's access control framework supporting different access control models is analyzed. Finally, we formally present the problems of designing an access control framework for future CAVs.

A. Data Access Pattern in CAV

As observed in [19], there are four categories of applications, consisting of ADAS [11], [20], real-time diagnostics [21], in-vehicle infotainment, and third-party application [19], [22]. The data accessed by these applications could be classified into two categories, real-time data and historical data, based on the observation of several CAV platforms, which will be accessed with different access patterns. Thus, we will analyze the data access pattern. Table I lists the storage locations and potential access patterns of these data.

Real-Time Data: The main requirements for real-time data access are low latency and one-to-many communication since different applications might access the same real-time data at the same time. Most existing CAV solutions utilize normal or modified versions of the robot operating system (ROS) [1], which provides the publish/subscribe pattern for different applications of CAVs. Taking Apollo as an example, which is an open-source CAV platform, including hardware reference, system, software, and autonomous driving algorithms, it modified ROS as an underlying system and utilizes message-based communications (publish/subscribe pattern) to deal with one-to-many communication (shared memory technique to reduce the latency of data transmission after version 3.5). In academia, OpenVDAP [6] also utilizes a message-based architecture to enable communications of real-time data between devices and applications.

Fig. 2 illustrates an example data processing flow of real-time camera data under the publish/subscribe pattern. The camera pushes raw images to the topic of *camera.id1.image*, and several processing nodes subscribe this topic while the encoding node encodes images into videos for persisting camera data into the file system as historical video data, and the SLAM node analyzes images for autonomous driving. The path planning node (also an autonomous driving application) subscribes the output of the SLAM node and publishes the control data for chassis control.

Historical Data: For historical data, current CAV solutions persist real-time data using the ROS built-in function, which directly saves data as ROS packages. However, it is not a good way for future CAVs. A simple way is storing structured data (e.g., GPS data) into a database and unstructured data (e.g., video) in the file system. In this case, the application could inquire about the structured data from the database directly, or inquire about the storage path of the unstructured data from the database; then it could access the file in the file system.

However, it is insecure to provide a database interface for a CAV on the road. Thus, a centralized manager is needed. To this end, Zhang *et al.* proposed a module, driving data integrator (*DDI*) in the OpenVDAP platform [6], [23], to automatically collect and store relevant context information on the vehicle and the Internet. The application could inquire data from this service. In addition, we need to note that the unstructured data could be accessed through the file system (paths are queried from DDI) or through the DDI service as a more secure method.

B. Access Control

The access control technique aims to protect data and resource in a computer system from unauthorized access. Typically, it includes several concepts, such as access control framework and access control model. The former one captures access actions in an application system or operating system, and apply one of access control models to authenticate the access actions. As mentioned before, different types and access patterns of data exist in CAV, and different applications are willing to manage their data in different ways. Especially, for these as yet unforeseen applications, some novel access control models might be proposed.

Based on the characteristics and requirements of the applied scenario, various access control models have been proposed, such as role-based access control (RBAC), identify-based access control (IBAC), and ABAC [24]. For example, the historical battery information under the IBAC model could be shared with the ones who have the identity certificate issued by the car maker. Meanwhile, access control models founded upon fine-grained and attribute-based encryption (ABE) could secure data and prevent unauthorized access (without right attributes), which we will introduce in Section IV. Thus, how to support several access control models thus provide a flexible and suitable choice for CAV and CAV application developers is still a big challenge.

Moreover, the context of CAV is also important for access control. For instance, third-party applications are prohibited from accessing network resources due to insufficient network bandwidth. Or, third-party applications are prohibited from accessing camera resources to avoid privacy leak, when the CAV is in a special location.

Problem Statement: Current access control frameworks usually focus on only one type of data. For example, access control frameworks in the operating system, messaging system, and Web service system focus on access of file systems, topics, and HTTP requests, respectively. Thus, to protect data with a suitable access control models in CAVs with various applications, data, data access patterns, a systematical, and flexible access control framework is needed, which is enabled to face possible changes occurred in the future, first. Here, several barriers must be solved as follows.

1) How to design a framework to authenticate these access actions from different data sources with different access patterns?
2) How to enable the supporting different access control models in one access control framework, and also enable the development of new access control models?
3) How to uniquely identify various data in access actions and access control models from different data sources?
4) How to dynamically make decisions on access actions based on current vehicle status, including location, computing resource, network resource, as well as supporting different access control models for different data?

III. System Architecture

We have introduced the motivations and goals for the access control framework in future CAVs, and now we will present our design. First, we will introduce some concepts in an access control framework, followed by the security and threat model. Then, we primarily focus on introducing the proposed access control framework for future CAVs.

A. Definition

A traditional access control framework will authenticate one `subject` whether it has the permission of one type of `operation` to one `object`. Thus, an access action could be described as a tuple {`subject, object, operation`}. In detail, the descriptions of `object`, `subject`, and `operation` in our CAV-specific framework are as follows.

Subject: Various applications, including native applications (e.g., ADAS applications) and third-party applications, are installed on CAVs, enhancing the ride experience and public safety, and they need to access and analyze data sensed by CAVs, thus becoming `subjects`.

Object: The `objects` refer to the data in CAVs, e.g., real-time data and historical data. It is easy to understand, and most applications analyze sensed data (as `objects`) for autonomous vehicles. Additionally, remote data, such as road conditions or weather data from remote cloud servers or other vehicles are also included. Furthermore, the application data is also the `object` in CAVs, since some applications might share/require results from collaborative applications.

Operation: As mentioned above, there exist several data sources in CAVs, including publish/subscribe system for real-time data, Web-based service for structured data, and file system for unstructured data. Thus, the operations defined in access actions currently include `subscribe/publish`, `get/post/delete`, and `create/read/write/delete`, respectively.

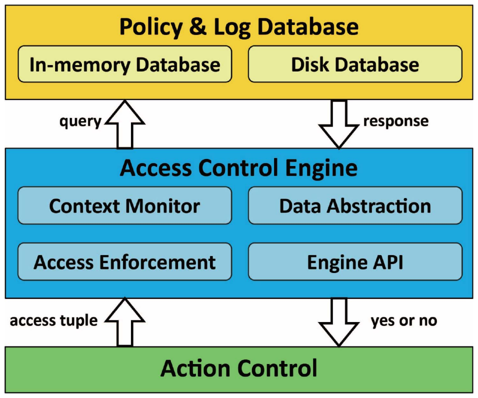


Fig. 3. Architecture of AC4AV.

In this article, we aim to build an access control framework supporting different access control models with dynamic adjustment. Thus, we add a segment `extra` to that tuple, which is used to store additional information. In the following sections, the tuple is defined as follows:

$$\{\text{subject, object, operation, extra}\}. \quad (1)$$

B. Security and Threat Model

The malicious applications in our threat model always try to subscribe to the data-related topics in the Pub/Sub system, query historical data from the database or file system. Typically, applications are isolated utilizing the container technology, and they cannot access others' memory to obtain data, such as subscribed data and the authentication information. In this article, we do not consider the leakage of authentication information and it could be secured by other approaches, such as secure storage. Additionally, the trusted execution environment is also promising to provide the isolation of applications, by executing the part of one application in a hardware-assistant environment, so that the running application can be protected from not only other applications but also the operating system and even hypervisor. Moreover, our `AC4AV` also can be executed in that trusted execution environment, such as Intel Software Guard Extensions (SGX) or AMD Memory Encryption Technology [25], which could significantly reduce the attack surface.

C. Architecture of `AC4AV`

The proposed access control architecture for CAVs is as shown in Fig. 3, which consists of three components: 1) *Access Control Engine*; 2) *Action Control*; and 3) *Policy & Log Database*. The *Access Control Engine* authenticates `operations` and responds `yes` or `no` to the *Action Control* component, which performs as a hook to capture access actions. The *Policy & Log Database* stores all data of `AC4AV`, such as the configuration file and the access action record, in a hierarchical mode while frequently used data is stored in the in-memory database with high-speed access.

Access Control Engine: In the *Access Control Engine*, we introduce four major components: 1) *access enforcement*; 2) *context monitor*; 3) *data abstraction*; and 4) *engine API*.

Since the goals of our access control system include dynamic access authentication, the *context monitor* component is used to collect the system's status information, such as CPU utilization and GPS, enabling an access control policy with dynamic features. For example, the A3 application would gain the access permissions of the historical video data with the specific location and time ranges.

As discussed in Section II, real-time data are different from historical data in terms of their access patterns, and different data identifying methods are applied by different systems. Thus, we designed the *data abstraction* component, which provides the function of identity conversion. It converts the identities of `objects` and `subjects` in captured tuples from different data sources and the internal identities to each other, based on a data abstraction method. Benefiting from this component, third-party applications also could identify their own data using an easy-to-read description method when they implement customized access control policies. The data abstraction method is introduced in Section IV.

To improve the expandability of our `AC4AV`, we designed a component, named *Engine API*, with a series of APIs. The provided functions are multifold: 1) to support a customized policy, a series of APIs for customized policy configuration are provided for applications to a submitted policy file. Moreover, several implemented access control models are provided for `AC4AV` so that applications could assign different access control models to protect their data; 2) to support auditing, a series of APIs are provided for inquiring records of access actions, as well as results; and 3) to configure `AC4AV`, a series of APIs are provided so that the system administrator could configure all components in `AC4AV`. For example, we will not limit the database used by *Policy & Log Database*, so the system administrator could assign it in the configuration file or adjust it through the provided APIs in runtime.

The last component of the *Access Control Engine* is the *access enforcement*, which is a core component, just like an assembler, combining other components to determine the permission to access actions. While a `subject` intends to access the `object` with the `operation`, all the related information of that access action, as shown in tuple `{subject,object,operation,extra}`, will be captured and sent to this component. Then, it will send the segment of the `object` to the *data abstraction* to figure out the internal identification of the accessed data. Then, it will authenticate this action using the specified policy, associated with other factors, defined in the policy and collected by the *context monitor* component. The output of this component indicates whether the `subject` has corresponding permissions.

Action Control Service: To capture access action and deny the action without permissions in different data sources, e.g., message queue system and Web-based service, the *Action Control* service is proposed in our `AC4AV`, mainly implementing two functions, *action capturing* and *action responding*.

The *action capturing* function is that implemented subservice captures all access actions and submits these actions to the *Access Control Engine* with the format as the tuple `{subject,object,operation,extra}`. The *action responding* refers to subservice performing corresponding operations (allowing or denying) based on the responses (`yes` or `no`) from the *Access Control Engine*. Note that the ways for allowing/denying operations will vary depending on the corresponding access method. For instance, an NGINX module will be implemented for capturing all data access queries to the DDI module in OpenVDAP, and it will reject all nonpermission queries with a 403 status code if it receives a `no` from the *Access Control Engine*. Thus, the *Action Control* service will consist of various subservices when implemented.

Policy & Log Database: Vast access action records should be recorded for future auditing, and many policy files should be collected in `AC4AV`. Thus, a *Policy & Log Database* is proposed to store this data. It is a two-layer architecture consisting of two database systems.

The lower layer database is a *disk database*, and it stores all information of `AC4AV`, including access action records, policy files, configuration files, and so on. However, the *disk database* is with a high query latency, and typically, one access control operation requires a quick response. An *in-memory database* is proposed as the upper layer database to reduce the inquiring latencies of these frequently used policies. Once the `AC4AV` starts, the *Access Control Engine* will inquire those policies from the *disk database* and then store the parsed policies to the *in-memory database*. We should note that all the stored data is only allowed to be accessed by the *Access Control Engine*. The reason is that stored data, i.e., access action records, policy files, and configurations of `AC4AV`, are important, especially when auditing in an accident investigation.

IV. Implementation

In this section, we introduce the implementation of our `AC4AV` prototype. First, we introduce the data abstraction method. Second, the prototype implementation of `AC4AV` is illustrated, followed by three access control models that are implemented as built-in models in the *engine API* component.

A. Data Abstraction Method

The naming of `objects` from different data sources are different; thus, how to identify `subject` and `object` is a problem that must be solved in an access control framework. To this end, an easy-to-read data abstraction method is proposed when implementing the prototype of our `AC4AV`, which converts the identities of `objects` and `subjects` in the captured tuples from different data sources and internal identities to each other.

Object: Different data sources have their own methods to identify data. The following equation is an example identity on the Pub/Sub system for the installed front camera, which publishes video data for applications. Thus, one application could obtain real-time video by subscripting such topic

$$\text{camera.id1.channel\_720P.} \qquad (2)$$

In addition, historical data could be accessed through the Web-based service. Taking the historical video data of the foregoing camera as an example, one application could request that data with a specific time period (e.g., $t1$ and $t2$) from the

DDI service. The access URL is as listed in (3). Note that we omitted "`http://`" in the equation due to limited space. In this case, the identity of the historical data is different from the real-time data, shown in (2)

$$ddi/\text{camera?type} = \text{id1\&start} = t1\text{\&end} = t2. \tag{3}$$

To uniformly identify the data in our `AC4AV`, we leveraged uniform resource identifier (URI), which is a hierarchical method. Typically, the `object` can be described in three parts: 1) owner; 2) identifier; and 3) parameters. The identities listed in (2) and (3) can be, respectively, described as the following based on our method:

$$/\text{sys/camera/front/realtime?resolution} = 720 \tag{4}$$
$$/\text{sys/camera/front/history?start} = \text{t1\&end} = t2. \tag{5}$$

Here, `/sys` indicates the owner, the identifier of data is `camera/front/realtime`, and `resolution=720` assigns the resolution of the subscribed video stream, while a real-time video source might publish several streams to the system with different resolutions. Similarly, the historical data sensed by the same sensor can be identified by replacing `realtime` with `history` and setting some parameters, such as start and end timestamps. Based on this method, the `object` can be uniformly identified in the `data abstraction` component, regardless of the `object` in the publish/subscribe system or in the DDI service.

Subject: In addition, we also use this method to identify `subjects` (applications). For example, the application A3 [19] could be defined as `/com/qyzhang/A3?v=1.0`, while `com/qyzhang` is the owner and `A3` is the identifier. Moreover, a group label can be defined to identify a set of applications thus providing a simpler way to manage applications in an access control policy. For example, `group:autonomous-services` is used to refer all services related to autonomous driving. Note that this group label should be created as a system-level configuration through the *engine API* component.

B. Implementation

We implement a prototype of `AC4AV` based on the proposed architecture. Most parts of the implemented prototype are programmed using Golang language, except the *Action Control* subservice on NGINX and the ABE algorithm, which is implemented using C/C++ language. In addition, we have tested our prototype on two different platforms. One is a normal desktop with the Intel CPU, and another is our HydraOne platform [18] with the NVIDIA Jetson TX2 processor, which is an indoor experimental research and education platform for CAVs based on OpenVDAP architecture [6].

Action Control: The *Action Control* service consists of various subservices, implemented to capture all access actions from different systems. In this prototype, we implement two subservices. One is embedded in a publish/subscribe system for real-time data and another is embedded in a Web server for historical data, as shown in Fig. 4.

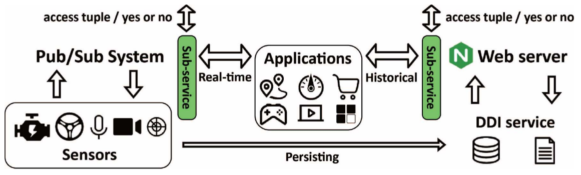


Fig. 4. Implemented subservices for the *Action Control* service.

The chosen publish/subscribe system for our `AC4AV` prototype is NATS [26], which is a simple, high-performance open-source messaging system that provides multilanguage clients, such as Python, Java, C, etc. When recompiled NATS receives a subscription request, the implemented module will capture all information about this action and send this information to the *Access Control Engine* for authentication. If the response is `no`, it will reject this subscription action. In our prototype, the identity of a `subject` is captured according to the socket's port used by the application once the application connects to the NATS. The `object` is the subscribing or unsubscribing topic. In addition, we also modify the protocols of NATS so that the applications can attach extra information in `PUB`, `SUB`, and `UNSUB` protocols for authenticating under different access control models.

Applications could inquire the structured data and metadata of unstructured data from the DDI service, hosted on NGINX [27] in our prototype, which is a popular and high-performance tool to implement a Web server. Benefiting from the expansibility of NGINX, we implemented a subservice of the *Action Control* service using the C++ language, registered to `NGX_HTTP_ACCESS_PHASE` on NGINX. Similar to the subservice in the recompiled NATS, the implemented NGINX module will collect the access information from an HTTP request received by NGINX to the DDI service. Then, the information is sent to the *Access Control Engine*, and the HTTP query is rejected with a 403 Forbidden HTTP status code if the response is `no`.

Access Control Engine: For the *Access Control Engine*, it is implemented as a RESTful Web service. The principle of the *data abstraction* component has been described in the previous section. We implemented this component and provide internal functions for other components, i.e., *access enforcement* component. Note that we use an external storage system, i.e., Redis, to store mapping relationships for identity conversion, in our prototype, which causes extra communications on inquiring identities. The reason is to improve the scalability, considering the increase in the number of mappings. For the *context monitor* component, it periodically obtains and caches system running status from underlying system interfaces, e.g., CPU utilization, as well as some privacy information about vehicle status from the modified NATS, such as GPS data. The *engine API* component allows the system administrator to configure the access control framework and third-party applications to update their configuration. Listing 1 illustrates the interfaces we implemented in our prototype. The `UpdateConfigure` interface allows the system administrator to update system's parameter, and the `InquirePolicy` interface allows the system administrator or third-party application to obtain policies corresponding to the data. Note that

```
/* Update configuration of access control system */
func (*EAPIs) UpdateConf(key string, value string) error
/* Search a policy */
func (*EAPIs) InquirePolicy(data string) (Policy,error)
/* Create a policy */
func (*EAPIs) CreatePolicy(config_json string) error
/* Update a policy */
func (*EAPIs) UpdatePolicy(config_json string) error
/* Delete a policy */
func (*EAPIs) DeletePolicy(config_json string) error
/* Inquire logs */
func (*EAPIs) InquireActionLogList(query string) string
```

Listing 1. Interfaces implemented in the prototype.

```
{ "version": 1.0,
 "data": "camera/front/realtime?fps=25",
 "owner": "system",
 "allow": [
   {   "operation": "read",
       "parameters": {
           "encode": "h264",
           "width": 1920,
           "height": 1080 },
       "access_model": {
           "type": "external",
           "service": "https://localhost:9999/auth"}
   },{ "operation": "read",
       "parameters": {
           "encode": "h264",
           "width": 1280,
           "height": 720 },
       "limit": [ "CPU":"max 0.5" ],
       "access_model": { "type": "ABAC"}
   },{ "operation": "read",
       "parameters": {
           "encode": "raw",
           "width": 3840,
           "height": 2160 },
       "access_model":{
           "type": "ACCL",
           "applications": ["group:autonomous"] }
   }]
}
```

Listing 2. Example of the access control policy.

we allow the system administrator to view all policies and the application to view its own data's policies. To support auditing, we also implemented the InquireActionLogList interface for inquiring access action records with a set of query conditions. In addition, the remaining three interfaces allow for the management of policies. Similarly, we only allow an application to manage its own policies. Moreover, to reduce the attack surface, only the necessary APIs are open to management. Furthermore, some authentication approaches should be considered when calling these APIs. For example, as mentioned before, a hardware-assisted trusted execution environment could ensure that application codes are not modified and provide local attestation. Thus, secure channels between our AC4AV and other applications may be established [28].

The *access enforcement* component implements the function of the receiving access actions, authenticating permission, and responding to the access actions. To describe a policy, JavaScript Object Notation (JSON) is used to represent the policy's features, which is a lightweight data-interchange format. Listing 2 shows an example policy. The parameter segment is used to describe the data. Here, we assign three access control models to data with different parameters of resolution, while autonomous driving applications could access 4k video data based on the defined access control model. Note that the h264-encoded video is the output of the encoding_node in Fig. 2 (camera.id1.hdvideo) and the raw one is the output of the camera_node_1 (camera.id1.image), while we assume camera_node_1 and encoding_node are the system services for the camera 1. The access control models will be introduced in the next section. The limit segment determines the limitations of data on the vehicle status, i.e., CPU, memory, and network. Moreover, we can move the same parameter to the data segment, like the "fps=25." In addition, the data owner could authenticate access actions on its own data, by setting type to "external" in the access_model segment. Typically, the external component must be implemented with an interface to accept the tuple and can be used to achieve dynamic access control. Thus, the *access enforcement* component will forward the request to the assigned external link defined in the service segment.

Policy & Log Database: We implemented this service based on two types of databases. For the disk database, MongoDB [29] is used to store all data, including policies, access action records, and configurations. Every access action with the result and related parameters (such as used system status) is recorded by the *Access Control Engine* and stored in MongoDB. The used in-memory database in our prototype is Redis [30], which is an open-source database and provides key-value storage. Redis will cache these frequently used policies, such as the policies of parts of real-time data and created by running applications, and use the data segment in JSON files as keys. Furthermore, as mentioned above, the identity mapping relationships are also cached in Redis, while those are also persisted on the disk database.

C. Access Control Model

In this section, we illustrate several implemented access control models used for different applied scenarios in this article: access control capability list (ACCL)-based discretionary access control (DAC) model, IBAC model, and ABAC model. The tuple captured by our AC4AV could be used to develop a new access control model.

ACCL-Based DAC Model: An ACCL-based DAC model restricts the access to the regulated objects through ACCLs, which define the set of subjects and their operation permissions. Here, the ACCL-based access control model is a simple DAC model. The model applied to 3840×2160 video data in Listing 2 is an example of the ACCL-based DAC model. Once the *access enforcement* component receives an access action with the object assigned to ACCL model, it checks whether the subject is included in the list defined by the applications segment. This model is appropriate for the scenario that has clearly known all subjects who will access the data so that it could list exhaustively in the policy.

IBAC: An IBAC model restricts access to the regulated objects through the identity of the subjects, which has several ways to implement. In our prototype, we use a certificate-based approach. The data requester (object) should obtain the certificate issued by the data owner using a private key and attaching the certificate when inquiring the data. Only the certificate is valid, the subjects could access

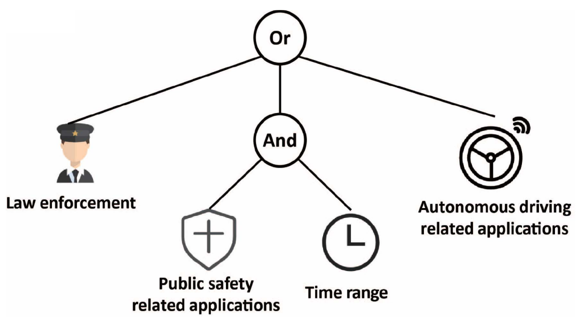

Or
And
Law enforcement
Public safety related applications
Time range
Autonomous driving related applications

Fig. 5. Example of the access structure.

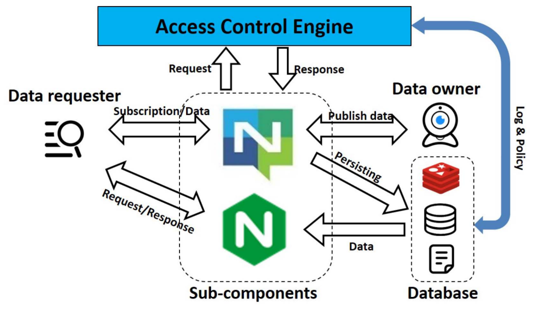


Fig. 6. Implementation of our experiment.

the data. This model is flexible when the owner could make allowed `objects` be part of the issued certificate. Thus, the applications developed by the same company with different application names could access the data without multiple authorizations.

ABAC Model: In our prototype, an ABAC model restricts access to the regulated `objects` through encrypting the data of the `subjects`. Thus, the data can be public for all applications but encrypted by related attributes, defined in an access structure. Only the data requesters, who have the related attributes, can decrypt the data from ciphertexts. Once the *access enforcement* component receives an access action assigned to the ABAC model, it directly allows the request. Fig. 5 illustrates an example of an attribute-based access structure for real-time video data captured by onboard cameras. It determines that the one with the attribute of "law enforcement" or the attribute of "autonomous driving applications" could access real-time video data, as well as the one which has the attribute of "public safety application" could access the data in a special time range.

V. Performance Evaluation

In this section, we first evaluate the response time of access actions, in terms of the topic subscription on NATS and HTTP query forwarded by NGINX. Next, we evaluate the performance of permission updating and revoking, i.e., the effective time interval of the policy going into effect.

A. Experimental Setup

We simplify the environment and assume there are five types of entities, as illustrated in Fig. 6, including data requester, data owner, message queue system, Web service, and `AC4AV`. To evaluate the performance of our `AC4AV` as well as the impacts of different access control models, we set up four cases based on three access control models: 1) ACCL; 2) Local Cert; 3) Remote Cert; and 4) ABAC. In Local-Cert case, the certificate verifying is performed in `Access Control Engine`, and the Remote-Cert case uses a third-party application to verify certificates. In addition, the ACCL and ABAC cases have similar processes in `Access Control Engine`. In an ACCL case, it will check whether the requester is defined in the list and the ABAC case will allow all access actions. When evaluating the performances, four vision-based ADAS algorithms are running to simulate background workloads [31].

The computing unit used in the experiment is with an Intel Core i5-7400 Processor @ 3.0 GHz (in performance mode). We evaluate the performance of our `AC4AV` with concurrent users, where each user sends one query per second and the duration time is set to 10 s. The number of users is ranged from 50 to 1000. The reason of the maximum of users with 1000 is that it is probably higher than the number of expected installed applications in one CAV and is enough to perform a stress test. Additionally, the application keeps HTTP connections alive and reconnect, automatically, which enables the connection reuse to avoid establish a new connection with latency. We should note that all connections are within the CAV, and also work well even the Internet is disconnected. The symmetric cryptography used to encrypt and decrypt is the advanced encryption standard (AES) with the 128-b security level. The certificate is formatted as X.509, including the elliptic curve (ECC)-based public key (P-256 curve, the default in Golang). It is noted that some lightweight cryptographic algorithms [32], [33] could reduce the latency, including certificate and attribute verifying, with the same security level, and we only evaluate our `AC4AV` with some open-source algorithms and standard algorithms.

B. Performance

First, we evaluated the performances of our `AC4AV`. The baseline here is the scenario without an access control technology. We counted the extra latency, in terms of certificate verifying, Redis querying, communication (including queueing time), and other remaining time (as processing latency).

1) Response Time of Access Action: Fig. 7 illustrates the latencies of access actions on the NATS with different access control models. The ACCL and ABAC cases have almost the same response time, thus we merge them in one bar, and they have lowest response time. The Remote-Cert case has the highest response time since it has additional communications between the `AC4AV` engine and third-party authority. The results show that our `AC4AV` framework could handle the authentication on data access actions in real time, and the latencies are decreased as the increasing of concurrent data requesters. It is because that the part of threads awakening is saved. In our experiments, an ECC-based certificate verification takes from 108 μs (with 1000 requesters) to 130 μs (with 50 requesters). The Redis time here includes data abstraction operation and policy querying cost, and takes 210 and 100 μs, respectively, for 100 and 1000 requesters.

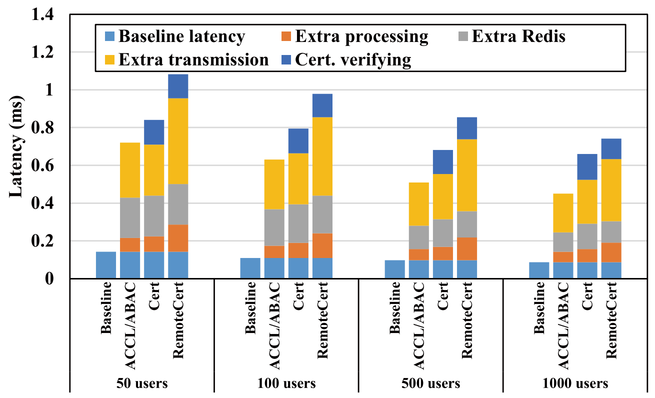


Fig. 7. Access action response time for real-time data with different access control models.

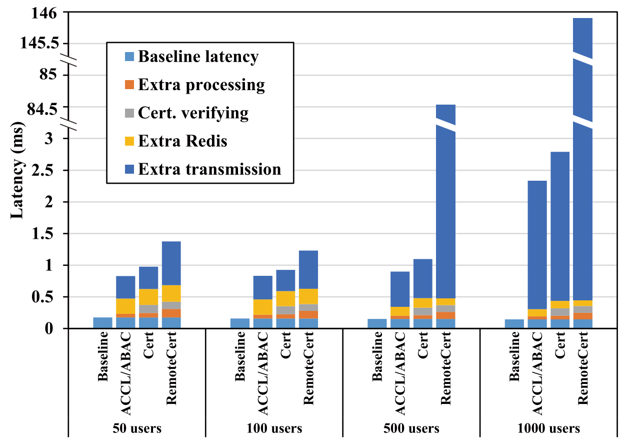


Fig. 8. Access action response time for historical data with different access control models.

Fig. 8 illustrates the performance of responding to access actions captured by Nginx. Similar to the response time of actions captured by the message queue system, the ACCL and ABAC cases have the lowest response time. However, when the number of concurrent requesters reach to 500, the RemoteCert case cannot handle action authentication in real time, and all cases are affected when the number is 1000. In our experiments, the real processing time plus Redis querying and certificate verification are low, but the thread is waiting when a synchronous request is sent, which wastes system resources and results in a long in-coming request queueing time. In addition, such intensive access requests (i.e., 1000) might be infrequent in CAVs. We should note, to be noninductive for a third-party application while inquiring data, the subcomponent of Nginx finds out the HTTP sender by matching the incoming port with system information stored in `/proc`. It costs up to 13 ms. Thus, a cache is built to save searching time to several microseconds for these reused HTTP clients, which use the same port to communicate with the same host. In our experiment setting, all requesters use reuse-enabled HTTP clients, thus the averaged processing time is low in results.

Insight 1: Different access control models causes different extra overhead with different security levels and an asynchronization optimizing (i.e., using callback for communications) should be considered to avoid request queueing.

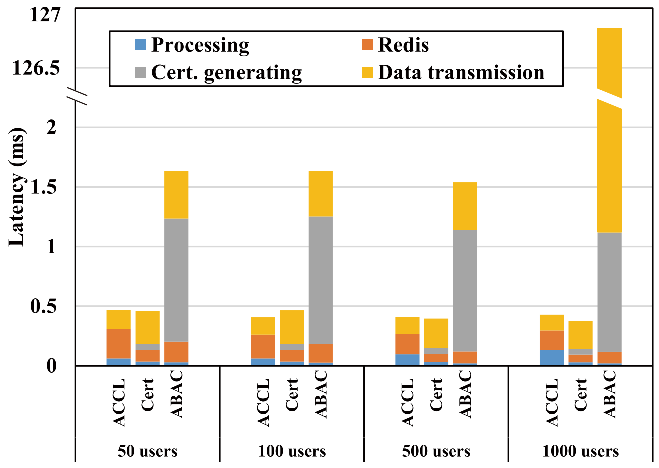


Fig. 9. Time for permission updating with different access control model.

2) Policy Updating and Revoking: In particular, when a malicious third-party application is detected, its permissions should be revoked as quickly as possible. The interval, between the time of sending permission updating/revoking request and entry-into-force time, is the key metric.

For permission updating, the data requester requests the permission of one object from the data owner through the API of *Access Control Engine*. For the ACCL case, the *Access Control Engine* responds to the request directly in our experiments. For other cases, it must notify the data owner for further processing, i.e., generating certificate in certificate-based models, or generating/updating keys in the ABE case. Fig. 9 illustrates the performance of permission updating. The ACCL and Cert cases have similar latency. The ABAC case has a larger latency due to additional data transmission time with the third-party application and the generating of ABE attribute keys also costs much time than other cases, resulting in that the ABAC case cannot respond to concurrent requests very well. Note that the generating time of attributes is positively related to the number of attributes. In our experiment, it randomly generates up to ten attributes for each requester.

For permission revoking, the processes of the ACCL and Cert cases are the same, thus we only show the ACCL case here. For these real-time data, the engine should notify the message queue system to unsubscribe that subscription, so that the ending time is when the subscription is unsubscribed. However, there exists an exception that the data owner only needs to update the encryption key for data with a new access structure or update part of attributes of applications while applying an ABAC model to the data. In this case, the time of generating a new encryption key is the ending time for the ABAC case. Additionally, we also measure the time consumption of generating attributes for applications.

The results of the ACCL case on permission revoking in Fig. 10(a) show that our AC4AV framework could perform permission revoking in real time, even in a concurrent environment. Fig. 10(b) shows the results in the ABE case. Here, we use the access structure in the form of a complete binary tree. The results show that the latency increases as the number of access structure layers increases. It means that permission revoking is a time-consuming operation in an ABAC

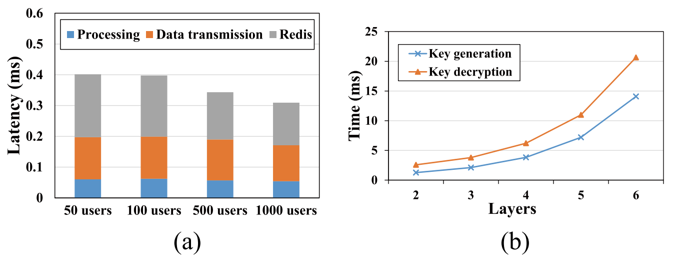


Fig. 10. Time for permission revoking with different access control model. (a) ACCL model. (b) ABAC model.

model. Moreover, if a new access structure, which enables unrevoked applications to decrypt the encryption key and disable the revoked applications, cannot be constructed, the data owner must update keys for all unrevoked applications, which costs much time. Although the revocation operation time is costly, the ABAC model still has obvious advantages in data confidentiality and convenience, especially avoiding to encrypt data for each data requesters.

Insight 2: The security and performance should be traded off. The model with high computing latency leads to a poor performance in a high concurrency scenario, which should be avoided.

C. Case Study: Video Analytics for CAVs

Finally, we use a video analytics application in CAVs as the case studies to measure the impact on one data stream of our AC4AV. The measured application is the lane detection. It is a basic but essential component of the ADAS system for CAVs [31], which processes video data and detect lanes of the road thus the ADAS could make the vehicle stay inside the lane markings. Hence, we measure the performance of real-time data when applying our AC4AV with different access control models.

After subscribing to the live video, the application could obtain the video stream from NATS. For ACCL case and Cert-based cases, they have no difference in video data transmission and processing, and they are the same with the case without the access control mechanism. Only the authentication at the subscription stage is different, which was measured before, thus we only show the results of the baseline (without access control mechanism) and ABAC case. The difference between these two cases is that the former one does not encrypt the video data and the latter one encrypts the video data using AES while the AES key is encrypted by the ABE algorithm. We also measured the performance with two common video formats, raw video data, and H264-encoded video data, while both of them could be obtained from camera devices.

Here, we select round 5-s experimental data as shown in Fig. 11. The raw video with the AES encryption case has the highest latency and the raw video without the AES encryption case has the lowest latency. It is because that the encrypting and decrypting of raw video data cost much time (i.e., 50 ms). Both of the H264-based cases have almost the same latencies. In our case of video data, the transmitted data size of raw video cases is about 140× larger than the H264-encoded cases. To

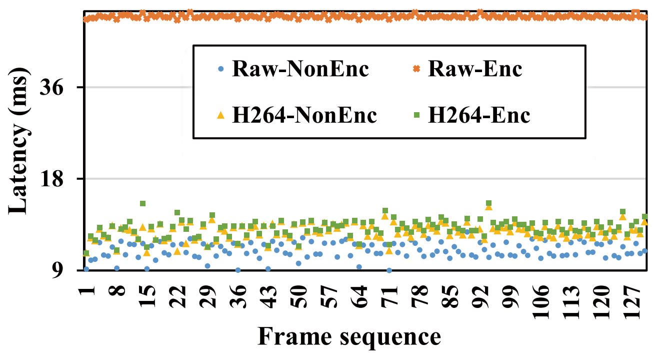


Fig. 11. Latency over time of a round 5-s real-time data.

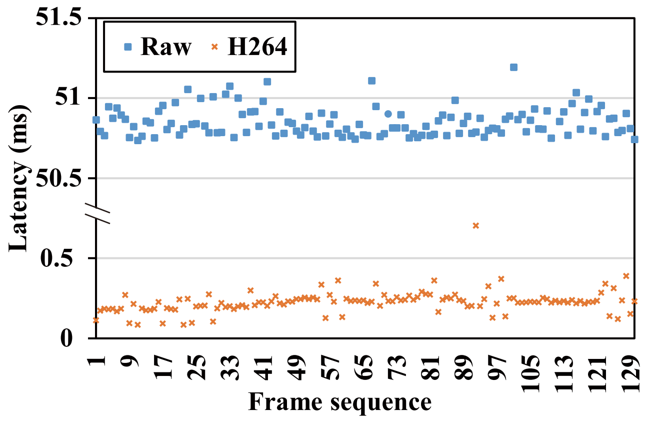


Fig. 12. Encrypting and decrypting impacts on latency over time of a round 5-s real-time data.

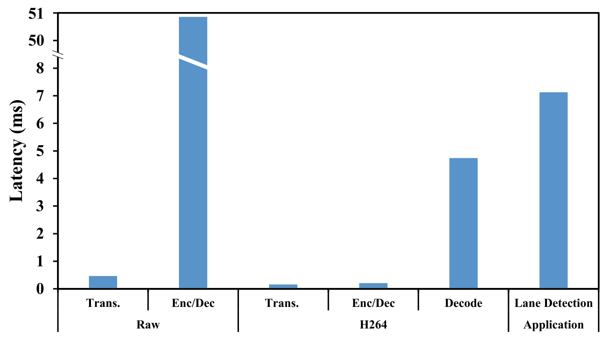


Fig. 13. Latency on different operations.

clearly show that, we also present the impacts of encrypting and decrypting as shown in Fig. 12.

Fig. 13 shows the detailed time consumption on different operations in our experiments, while the lane detection-related operations are the same for all cases. Here, the encryption and decryption cause much time and thus it is suggested to encode the video while encryption is applied. We should note that we do not take video encoding time in count, since some cameras could provide the H264-encoded video data, directly, by utilized inside hardware encoder, which is more effective than software-based encoding. The lane detection we used is a traditional computer vision-based algorithm, which costs less time than most deep learning-based algorithms. For example, we also evaluate the performance of Yolo3-tiny, which could detect objects (e.g., obstacles) in video, and it costs around one second for each frame.

Insight 3: The internal data transmission latency is low and large volume data should be compressed before encrypted transmission, unless the time of compression and decompression will be higher than encryption and decryption.

D. Summary

Typically, the fastest possible action by a human driver takes 100–150 ms [34]. Faster action (i.e., within 100 ms) than human drivers should be preformed by CAVs with better safety [35]. In addition, similar industry standards are published by Mobileye [36] and Udacity takes this timing requirement into the design specifications. In this case, the latencies while applying our framework and access control models should also be limited to 100 ms. Based on the experiments, the latencies incurred by our framework are tolerable for CAVs, while most of the cases are less than 2 ms under a reasonable concurrency, except the ABAC model case for video data. Generally, it is unavoidable that more computation is required to ensure the security of data. Thus, although it is a tradeoff problem, our framework could capture the access actions in CAVs, enable different access control models to different data, and has tolerable overhead.

VI. Related Work

CAV as an emerging research direction has attracted a great deal of attention from industry and academia. In this section, we discuss the related work in the following primary areas: 1) CAV system and platform and 2) access control for CAVs.

CAV System and Platform: Currently, some CAVs are published, such as Google Waymo, Denso Tesla, Baidu Apollo, and PerceptIn DragonFly Pod. Traditional automakers have also released their CAV plans. General Motors has prepared a car without a steering wheel or pedals, and is asking DOT permission to deploy it on road [37]. BMW has been testing CAV on public roads for several years. Most CAVs are usually equipped with heterogeneous computing devices [2], [12], [38], such as GPU-based Nvidia PX, DSP-based TDA from Texas Instruments, FPGA-based Cyclone V of Altera, and ASIC-based EyeQ5 from MobilEye, while Nvidia PX platform is a leading GPU-based solution for CAVs. Building on these heterogeneous computing devices, a real-time operating system will be installed for these latency-sensitive autonomous driving algorithms, e.g., QNS and VxWorks. Currently, most existing autonomous driving systems and platforms utilize a message passing architecture for communication. For example, Apollo was built based on a modified version of the ROS [39]. In addition, Wang *et al.* [18] developed an indoor experimental CAV platform, HydraOne, based on the ROS, which is an implementation of OpenVDAP [6]. However, due to the limitation of ROS on performance, as well as reliability and security issues, current ROS is not suitable for the production deployment of CAVs unless automotive-grade standards are met. Thus, some works are proposed. Tang *et al.* proposed the PerceptIn Operating System for low-speed CAVs, which is based on Nanomsg, a socket library providing several common communication patterns, including messaging. In addition, Baidu has established cooperation with some traditional automakers to build CAV products based on its Apollo, which was developed based on the modified version of ROS.

Access Control for CAVs: The access control technique has been applied to many areas, such as cloud stroage [40], IoT [41]–[43], smart health [44], as well as CAVs [45]. However, the scheme proposed by Habib *et al.* [45] only dealt with the access control problem of data sharing between different CAVs. As far as we know, there is not any research focusing the data sharing inside CAV. As mentioned above, the current CAV system is a closed system, which cannot support third-party applications, or can only install these in the in-vehicle infotainment system, such as Android Auto with limitation of accessing vehicle sensing data. Thus, we mainly discuss access control on the ROS in this section. The ROS is an open-source framework for robots, including a collection of tools and libraries. Some works have studied the security of ROS [46]. SROS is an official version of ROS, including a set of security enhancements, such as securing all socket transport within ROS and access control mechanism [47], [48]. The provided access control model is similar to our ACCL model, which only allows one topic to be published/subscribed by special publishers/subscribers defined in a policy file [13]. However, it cannot deal with all access control requirements mentioned before. Similar to ROS, most existing message queue systems only provide the access control mechanism of the ACCL model. Ferraiolo *et al.* [49] proposed a novel architecture and framework for access control policy specification and enforcement, which is a general access control architecture and could support several access control models. However, deploying such access control architecture into CAV requires much more research, such as implementing access action monitors (i.e., subcomponents of our *Action Control*), and it is not easy to use without APIs. The CAV has a certain similarity with the mobile phone to some extent, such as personality. Android as a popular operating system has provided an access control framework, and to improve the flexibility, some works have been presented. Shebaro *et al.* [14] implemented a context-based access control systems for mobile devices, while the privileges of an Android application could be granted or revoked based on the specific context of the user, such as location. However, current mobile operating systems cannot meet the requirements of CAVs, due to the characteristics of CAVs on existing multiple access patterns as mentioned before.

VII. Conclusion

In this article, we investigated the characteristics of data and access patterns, as well as the difficulties of designing and implementing an access control framework in the CAV scenario. To tackle these problems, we designed and implemented a three-layer access control framework to authenticate access actions of real-time data and historical data on different systems, which supports fine-grained and flexible access control models and is extensible with several APIs, enabling configuring access control policy to application and implementing customized access control models. Then, we implemented a prototype that could capture real-time data access actions on the publish/subscribe system and historical data access actions on the Web service. In addition, three access control models are implemented as built-in models, and third-party developers could utilize those directly or apply their own access control models through APIs. Finally, we demonstrated our framework by evaluating the performances in the cases of applying different access control models to vehicle sensing data. The results show that our framework has a tolerable impact on access actions.

The current version of our framework has low overhead for access actions, and several improvements could be performed in future studies to further reduce the latency. We will design caching modules for different framework components. For instance, frequently used policies and identity mapping relationships could be cached in the *Access Control Engine* component thus no Redis access is requested. Moreover, we will try to analyze the requirements of access control models in CAVs and design suitable access control models. Furthermore, benefiting from the expandability of our AC4AV, we will implement more *Action Control* subservices to face changes in future access patterns, e.g., sharing memory mode for real-time data.

Acknowledgment

The authors are very grateful to the anonymous referees for their detailed comments and suggestions regarding this article.

References

[1] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proc. IEEE*, vol. 107, no. 8, pp. 1697–1716, Aug. 2019. [Online]. Available: https://doi.org/10.1109/JPROC.2019.2915983

[2] J. Wei, J. M. Snider, J. Kim, J. M. Dolan, R. Rajkumar, and B. Litkouhi, "Towards a viable autonomous driving research platform," in *Proc. IEEE Intell. Veh. Symp. (IV)*, Gold Coast, QLD, Australia, Jun. 2013, pp. 763–770. [Online]. Available: https://doi.org/10.1109/IVS.2013.6629559

[3] M. Cebe, E. Erdin, K. Akkaya, H. Aksu, and S. Uluagac, "Block4forensic: An integrated lightweight blockchain framework for forensics applications of connected vehicles," *IEEE Commun. Mag.*, vol. 56, no. 10, pp. 50–57, Oct. 2018. [Online]. Available: https://doi.org/10.1109/MCOM.2018.1800137

[4] (2019). *Waymo*. Accessed: Aug. 8, 2019. [Online]. Available: https://waymo.com/

[5] *Apollo: Autonomous Driving Solution*, Baidu Inc. Beijing, China, 2017. Accessed: Dec. 27, 2017. [Online]. Available: https://apollo.auto/index.html

[6] Q. Zhang *et al.*, "OpenVDAP: An open vehicular data analytics platform for CAVs," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Vienna, Austria, Jul. 2018, pp. 1310–1320. [Online]. Available: https://doi.org/10.1109/ICDCS.2018.00131

[7] L. Liu, X. Zhang, M. Qiao, and W. Shi, "Safeshareride: Edge-based attack detection in ridesharing services," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Seattle, WA, USA, 2018, pp. 17–29. [Online]. Available: https://doi.org/10.1109/SEC.2018.00009

[8] R. S. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE Commun. Mag.*, vol. 32, no. 9, pp. 40–48, Sep. 1994. [Online]. Available: https://doi.org/10.1109/35.312842

[9] F. Li *et al.*, "Cyberspace-oriented access control: A cyberspace characteristics-based model and its policies," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1471–1483, Apr. 2019. [Online]. Available: https://doi.org/10.1109/JIOT.2018.2839065

[10] Q. Lyu, Y. Qi, X. Zhang, H. Liu, Q. Wang, and N. Zheng, "SBAC: A secure blockchain-based access control framework for information-centric networking," *J. Netw. Comput. Appl.*, vol. 149, Jan. 2020, Art. no. 102444. [Online]. Available: https://doi.org/10.1016/j.jnca.2019.102444

[11] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Santiago, Chile, Dec. 2015, pp. 2722–2730. [Online]. Available: https://doi.org/10.1109/ICCV.2015.312

[12] S. Liu, J. Tang, Z. Zhang, and J. Gaudiot, "Computer architectures for autonomous driving," *Computer*, vol. 50, no. 8, pp. 18–25, 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.3001256

[13] R. White, H. I. Christensen, G. Caiazza, and A. Cortesi, "Procedurally provisioned access control for robotic systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Madrid, Spain, Oct. 2018, pp. 1–9. [Online]. Available: https://doi.org/10.1109/IROS.2018.8594462

[14] B. Shebaro, O. Oluwatimi, and E. Bertino, "Context-based access control systems for mobile devices," *IEEE Trans. Depend. Secure Comput.*, vol. 12, no. 2, pp. 150–163, Mar./Apr. 2015. [Online]. Available: https://doi.org/10.1109/TDSC.2014.2320731

[15] R. Wang *et al.*, "Spoke: Scalable knowledge collection and attack surface analysis of access control policy for security enhanced android," in *Proc. ACM Asia Conf. Comput. Commun. Security*, 2017, pp. 612–624. [Online]. Available: https://doi.org/10.1145/3052973.3052991

[16] S. Ravidas, A. Lekidis, F. Paci, and N. Zannone, "Access control in Internet-of-Things: A survey," *J. Netw. Comput. Appl.*, vol. 144, pp. 79–101, Oct. 2019. [Online]. Available: https://doi.org/10.1016/j.jnca.2019.06.017

[17] S. Roy, A. K. Das, S. Chatterjee, N. Kumar, S. Chattopadhyay, and J. J. P. C. Rodrigues, "Provably secure fine-grained data access control over multiple cloud servers in mobile cloud computing based healthcare applications," *IEEE Trans. Ind. Informat.*, vol. 15, no. 1, pp. 457–468, Jan. 2019. [Online]. Available: https://doi.org/10.1109/TII.2018.2824815

[18] Y. Wang, L. Liu, X. Zhang, and W. Shi, "Hydraone: An indoor experimental research and education platform for CAVs," in *Proc. 2nd USENIX Workshop Hot Topics Edge Comput. (HotEdge 19)*. Renton, WA, USA, 2019. [Online]. Available: https://www.usenix.org/conference/hotedge19/presentation/wang

[19] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Distributed collaborative execution on the edges and its application to amber alerts," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3580–3593, Oct. 2018. [Online]. Available: https://doi.org/10.1109/JIOT.2018.2845898

[20] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 6526–6534. [Online]. Available: https://doi.org/10.1109/CVPR.2017.691

[21] M. Amarasinghe *et al.*, "Cloud-based driver monitoring and vehicle diagnostic with OBD2 telematics," in *Proc. 15th Int. Conf. Adv. ICT Emerg. Regions (ICTer)*, Colombo, Sri Lanka, Aug. 2015, pp. 243–249. [Online]. Available: https://doi.org/10.1109/ICTER.2015.7377695

[22] B. Qi, L. Kang, and S. Banerjee, "A vehicle-based edge computing platform for transit and human mobility analytics," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, San Jose, Ca, USA, Oct. 2017, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/3132211.3134446

[23] L. Liu, X. Zhang, Q. Zhang, A. Weinert, Y. Wang, and W. Shi, "Autovaps: An IoT-enabled public safety service on vehicles," in *Proc. 4th Workshop Int. Sci. Smart City Oper. Platforms Eng.*, 2019, pp. 41–47. [Online]. Available: http://doi.org/10.1145/3313237.3313303

[24] W. Ding, Z. Yan, and R. Deng, "Privacy-preserving data processing with flexible access control," *IEEE Trans. Depend. Secure Comput.*, vol. 17, no. 2, pp. 363–376, Apr. 2020. [Online]. Available: https://doi.org/10.1109/TDSC.2017.2786247

[25] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A comparison study of intel SGX and AMD memory encryption technology," in *Proc. 7th Int. Conf. Hardw. Archit. Support Security Privacy (HSAP'18)*, Jun. 2018, pp. 1–8. [Online]. Available: https://doi.org/10.1145/3214292.3214301

[26] *NATS—Open Source Messaging System*, NATS, San Francisco, CA, USA, 2019. Accessed: Sep. 25, 2019. [Online]. Available: https://nats.io/

[27] *NGINX | High Performance Load Balancer, Web Server, and Reverse Proxy*, NGINX Inc, San Francisco, CA, USA, 2019. Accessed: Sep. 15, 2019. [Online]. Available: https://www.nginx.com/

[28] P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. Freiling, and I. Verbauwhede, "Hardware-based trusted computing architectures for isolation and attestation," *IEEE Trans. Comput.*, vol. 67, no. 3, pp. 361–374, Mar. 2018.

[29] *The Most Popular Database for Modern Apps | MongoDB*, MongoDB, Inc, New York, NY, USA, 2019. Accessed: Sep. 30, 2019. [Online]. Available: https://www.mongodb.com/

[30] *Redis*, Redis Labs, Mountain View, CA, USA, 2019. Accessed: Sep. 17, 2019. [Online]. Available: https://redis.io/

[31] M. Maestre. (2019). *Lane Detection Module Using C++ and OpenCV*. Accessed: Oct. 2, 2019. [Online]. Available: https://github.com/MichiMaestre/Lane-Detection-for-Autonomous-Cars

[32] X. Yao, Z. Chen, and Y. Tian, "A lightweight attribute-based encryption scheme for the Internet of Things," *Future Gener. Comput. Syst.*, vol. 49, pp. 104–112, Aug. 2015. [Online]. Available: https://doi.org/10.1016/j.future.2014.10.010

[33] J. Zhang, J. Cui, H. Zhong, Z. Chen, and L. Liu, "PA-CRT: Chinese remainder theorem based conditional privacy-preserving authentication scheme in vehicular ad-hoc networks," *IEEE Trans. Depend. Secure Comput.*, early access, Mar. 11, 2019, [Online]. Available: https://doi.org/10.1109/TDSC.2019.2904274

[34] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, no. 6582, pp. 520–522, Jun. 1996. [Online]. Available: https://doi.org/10.1038/381520a0

[35] S.-C. Lin *et al.*, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2018, pp. 751–766. [Online]. Available: https://doi.org/10.1145/3173162.3173191

[36] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," 2016. [Online]. Available: http://arxiv.org/abs/1610.03295.

[37] *Meet the Cruise AV: The First Production-Ready Car With No Steering Wheel or Pedals*, General Motors, Detroit, MI, USA, 2018. Accessed: Jan. 25, 2018. [Online]. Available: http://media.gm.com/media/us/en/gm/home.detail.html/content/Pages/news/us/en/2018/jan/0112-cruise-av.html

[38] L. Wang, Q. Zhang, Y. Li, H. Zhong, and W. Shi, "Mobileedge: Enhancing on-board vehicle computing units using mobile edges for CAVs," in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Tianjin, China, 2019, pp. 470–479.

[39] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, vol. 3, 2009, p. 5.

[40] S. Xiong, Q. Ni, L. Wang, and Q. Wang, "SEM-ACSIT: Secure and efficient multiauthority access control for IoT cloud storage," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 2914–2927, Apr. 2020.

[41] R. Schuster, V. Shmatikov, and E. Tromer, "Situational access control in the Internet of Things," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2018, pp. 1056–1073. [Online]. Available: http://doi.org/10.1145/3243734.3243817

[42] T. Khalid *et al.*, "A survey on privacy and access control schemes in fog computing," *Int. J. Commun. Syst.*, p. e4181, Oct. 2019. [Online]. Available: https://doi.org/10.1002/dac.4181

[43] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1594–1605, Apr. 2019. [Online]. Available: https://doi.org/10.1109/JIOT.2018.2847705

[44] Y. Zhang, D. Zheng, and R. H. Deng, "Security and privacy in smart health: Efficient policy-hiding attribute-based access control," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2130–2145, Jun. 2018.

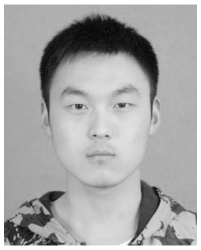
[45] M. A. Habib *et al.*, "Security and privacy based access control model for Internet of connected vehicles," *Future Gener. Comput. Syst.*, vol. 97, pp. 687–696, Aug. 2019. [Online]. Available: https://doi.org/10.1016/j.future.2019.02.029

[46] B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, and P. Schartner, "Security for the robot operating system," *Robot. Auton. Syst.*, vol. 98, pp. 192–203, Dec. 2017. [Online]. Available: https://doi.org/10.1016/j.robot.2017.09.017

[47] B. Breiling, B. Dieber, and P. Schartner, "Secure communication for the robot operating system," in *Proc. Annu. IEEE Int. Syst. Conf. (SysCon)*, Montreal, QC, Canada, Apr. 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1109/SYSCON.2017.7934755

[48] R. White, H. I. Christensen, and M. Quigley, "SROS: Securing ROS over the wire, in the graph, and through the kernel," 2016. [Online]. Available: http://arxiv.org/abs/1611.07060.

[49] D. Ferraiolo, V. Atluri, and S. Gavrila, "The policy machine: A novel architecture and framework for access control policy specification and enforcement," *J. Syst. Archit.*, vol. 57, no. 4, pp. 412–424, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762110000251

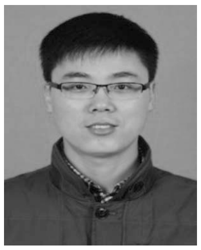


Qingyang Zhang (Graduate Student Member, IEEE) received the B.Eng. degree in computer science and technology from Anhui University, Hefei, China, in 2014, where he is currently pursuing the Ph.D. degree.

His research interests include edge computing, computer systems, and security.

Hong Zhong (Member, IEEE) was born in Anhui, China, in 1965. She received the Ph.D. degree in computer science from the University of Science and Technology of China, Hefei, China, in 2005.

She is currently a Professor and a Ph.D. Supervisor with the School of Computer Science and Technology, Anhui University, Hefei. She has over 120 scientific publications in reputable journals, such as the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, the IEEE TRANSACTIONS ON BIG DATA, and the IEEE INTERNET OF THINGS JOURNAL, academic books, and international conferences. Her research interests include applied cryptography, IoT security, vehicular *ad hoc* network, cloud computing security, and software-defined networking.



Jie Cui (Member, IEEE) was born in Henan, China, in 1980. He received the Ph.D. degree from the University of Science and Technology of China, Hefei, China, in 2012.

He is currently a Professor and a Ph.D. Supervisor with the School of Computer Science and Technology, Anhui University, Hefei. He has over 100 scientific publications in reputable journals, such as the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, and the IEEE INTERNET OF THINGS JOURNAL, academic books, and international conferences. His current research interests include applied cryptography, IoT security, vehicular *ad hoc* network, cloud computing security, and software-defined networking.

Lingmei Ren received the Ph.D. degree from the Department of Electronic and Information Engineering, Tongji University, Shanghai, China, in 2016.

She was a Visiting Scholar with Wayne State University, Detroit, MI, USA, in 2012. She is currently with the School of Computer Science, Shenzhen Institute of Information Technology, Shenzhen, China. Her main research interests include fall detection, human behavior recognition, wireless health, and edge computing.

Weisong Shi (Fellow, IEEE) received the B.S. degree in computer engineering from Xidian University, Xi'an, China, in 1995, and the Ph.D. degree in computer engineering from the Chinese Academy of Sciences, Beijing, China, in 2000.

He is a Charles H. Gershenson Distinguished Faculty Fellow and a Professor of computer science with Wayne State University, Detroit, MI, USA. His research interests include edge computing, computer systems, energy efficiency, and wireless health.

Prof. Shi is a recipient of the National Outstanding Ph.D. dissertation Award of China and the NSF CAREER Award. He is an ACM Distinguished Scientist.