future internet

MDPI

Article

FIVADMI: A Framework for In-Vehicle Anomaly Detection by Monitoring and Isolation †

Khaled Mahbub [1,*], Antonio Nehme [1], Mohammad Patwary [2], Marc Lacoste [3] and Sylvain Allio [3]

[1] College of Computing, Birmingham City University, Birmingham B5 5JU, UK; antonio.nehme@bcu.ac.uk
[2] Digital Innovation & Solution Centre, University of Wolverhampton, Wolverhampton WV1 1LY, UK; patwary@wlv.ac.uk
[3] Department of Security, Orange Labs, 38240 Meylan, France; marc.lacoste@orange.com (M.L.); sylvain.allio@orange.com (S.A.)
* Correspondence: khaled.mahbub@bcu.ac.uk
† This article is a revised and expanded version of a paper entitled "Towards an Integrated In-Vehicle Isolation and Resilience Framework for Connected Autonomous Vehicles", which was presented at VEHICULAR 2020, Porto, Portugal, 18–22 October 2020.

Abstract: Self-driving vehicles have attracted significant attention in the automotive industry that is heavily investing to reach the level of reliability needed from these safety critical systems. Security of in-vehicle communications is mandatory to achieve this goal. Most of the existing research to detect anomalies for in-vehicle communication does not take into account the low processing power of the in-vehicle Network and ECUs (Electronic Control Units). Also, these approaches do not consider system level isolation challenges such as side-channel vulnerabilities, that may arise due to adoption of new technologies in the automotive domain. This paper introduces and discusses the design of a framework to detect anomalies in in-vehicle communications, including side channel attacks. The proposed framework supports real time monitoring of data exchanges among the components of in-vehicle communication network and ensures the isolation of the components in in-vehicle network by deploying them in Trusted Execution Environments (TEEs). The framework is designed based on the AUTOSAR open standard for automotive software architecture and framework. The paper also discusses the implementation and evaluation of the proposed framework.

Keywords: AUTOSAR; ECU; isolation; resilience; system security

Citation: Mahbub, K.; Nehme, A.; Patwary, M.; Lacoste, M.; Allio, S. FIVADMI: A Framework for In-Vehicle Anomaly Detection by Monitoring and Isolation. *Future Internet* **2024**, *16*, 288. https://doi.org/10.3390/fi16080288

Academic Editors: Stefano Rinaldi and Alan Oliveira De Sá

Received: 1 July 2024
Revised: 30 July 2024
Accepted: 2 August 2024
Published: 8 August 2024

1. Introduction

Modern vehicles integrate many electronic and mechanical subsystems to provide added-value services such as driving assistance and entertainment functions to drivers and passengers. *Electronic Control Units (ECU)* are the major components that coordinate such subsystems. Most often, ECUs are driven by software and need to communicate with each other and with the outside environment (e.g., external services or other vehicles). While the communication among ECUs is enabled by the *In-Vehicle Network (IVN)*, and communication with the external environment can be achieved using various technologies such as Vehicle-to-Road (V2R) Infrastructure and Vehicle-to-Everything (V2X) connections [1]. While both types of communication enable safety-critical decision-making, in-vehicle communication requires special attention. This is mainly due to the use of *Controller Area Network (CAN)*, the predominant in-vehicle communication protocol [2].

The design and development of CAN did not take IVN cybersecurity and protection into account and vehicles were considered as closed systems [3–7]; this assumption was reasonable when the automated functions in vehicles were not safety critical. However, the seamless integration of information technology into modern vehicles, capable of performing critical functions, transformed them into open systems with a large attack surface [5,8]. In-vehicle communication based on CAN is not protected against attacks, such as in-vehicle

Spoofing, Eavesdropping, DoS, Replay, Injection etc. [4–6,8–10]. Moreover, the adoption of new technologies in the automotive domain is opening new safety and security challenges. For example, the advent of new generations of ECUs virtualized as lightweight execution environments (e.g., VMs, containers) on different types of virtualization platforms (e.g., OKL4 microvisor, Proteus hypervisor, ETAS STA-HVR) may face system-level isolation challenges such as side-channel vulnerabilities.

The development of IVN security countermeasures is hindered by various factors. The application of sophisticated attack detection mechanisms requires significant amounts of computing power and therefore cannot be used in vehicles due to the limited ECU processing capabilities [11–13]. Moreover, the implementation of IVN security mechanisms must ensure that the normal functionalities of the vehicle are not impacted [14]. Improvement of security and resilience of autonomous vehicles has attracted significant attention in the literature. A prominent example are approaches inspired by the principle of "security by design" practiced in the software industry [15–18] and stating that existing secure development processes should be adapted and incorporated in the design and development of autonomous vehicles. These approaches, however, may not be readily acceptable to the manufacturer as they may require a major redesign of the vehicle or of its components. Various techniques have been applied to perform real-time IVN detection of threats and attacks including signature based [2,14], machine learning based [19,20] and frequency based techniques [7,21,22]. These approaches, however, may not be effective due to the low processing power of the CAN network and the lack of uniform semantics of CAN messages across different car makes and models.

This paper is an extended version of our position paper [23] and discusses the design, implementation and evaluation of a Framework for In-Vehicle Anomaly Detection by Monitoring and Isolation (FIVADMI). More specifically, FIVADMI supports the detection of anomalies when ECUs exchange information on a CAN bus based IVN. FIVADMI adopts a holistic approach that enables detection of anomalies in a compromised vehicle at the communication level (e.g., Injection and DoS attacks in the CAN bus) as well as at the application level (e.g., OBD level, such as anomalies in the acceleration or coolant temperature of a compromised vehicle). For the detection of runtime anomalies, FIVADMI relies on an external rule engine (CLIPS [24]) to detect novel attacks without any significant change in the FIVADMI implementation. By analysing the detected anomalies, FIVADMI produces certificates that reflect the safety and the stability of the current state of the vehicle. The implementation of FIVADMI is based on the AUTomotive Open System ARchitecture (AUTOSAR) open standard, version R22-11,for automotive software architecture and framework. Hence FIVADMI inherently offers scalability, reusability and interoperability across the product lines from different Original Equipment Manufacturers (OEMs) [25]. FIVADMI also ensures isolation of ECUs on the IVN by deploying each ECU within a Trusted Execution Environment (TEE), TEE support is being provided by the OpenEnclave open source project [26].

This paper is structured as follows. Section 2 provides an overview of in-vehicle Electric/Electronic (E/E) Architecture, in-vehicle threats and existing approaches to mitigate in-vehicle threats. Section 3 discusses the FIVADMI design principles and architecture. Section 4 describes the implementation of the main components of FIVADMI. Section 5 presents the experimental results on the evaluation of our implementation. We conclude in Section 6 discussing the limitations of the current implementation and possible future improvements.

2. Related Work

In this section, we briefly provide an overview of in-vehicle *Electric/Electronic (E/E) Architecture* to highlight different communication patterns in a vehicle. We then focus on the state of the art of in-vehicle threats. Finally, we present the existing approaches to mitigate in-vehicle security threats.

2.1. In-Vehicle E/E Architecture

An *Electronic Control Unit (ECU)* can be considered as an embedded computer in vehicles that controls one or more electrical system or subsystems. Typically, different mechanical components (e.g., sensors and actuators) are attached to ECUs. In common scenarios, ECUs receive input from sensors, other ECUs and On-Board Units (OBUs) and control various functionalities by managing the actuators. Modern vehicles incorporate a large number of ECUs. The ECUs generally communicate with each other by means of communication buses, where buses may vary depending on different attributes such as speed, capacity, etc. Some of the commonly used buses are (i) Controller Area Network (CAN), (ii) FlexRay and (iii) CAN with flexible data rate (CAN FD). In-vehicle buses are interconnected following different architectural patterns depending on the complexity, bandwidth and real-time requirements of the vehicle [16,27–29]. In the following, we briefly present the most common in-vehicle network architectures.

Hierarchical with Central Gateway E/E Architecture—As shown in Figure 1, in this class of architecture, ECUs are installed into different buses interconnected by a *central gateway*. The hierarchy comes from the assignment of functions to buses with different bandwidths following their real-time and size requirements [27]. The central gateway is responsible to manage the inter-ECU communications and to convert data from one bus format to another bus format [16,27,29].

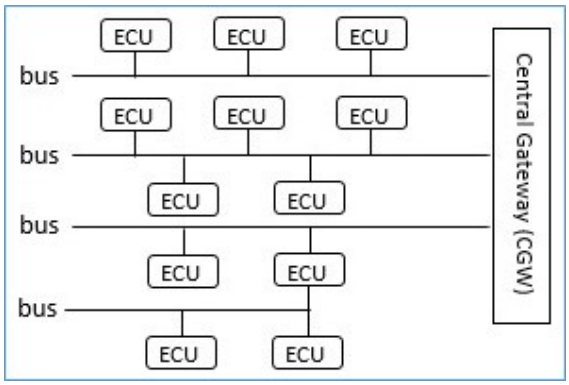

Figure 1. In-Vehicle Hierarchical with Central Gateway E/E Architecture [28].

Domain-based E/E Architecture—As shown in Figure 2, in this class of architecture, ECUs are grouped into different *domains* according to similarity of functionalities [28]. For example, all the ECUs that relate to the powertrain control of the vehicle can be put in one domain. While the functionalities of one domain may be for entertainment and comfort (e.g., audio, telephony), others perform safety-critical functions (e.g., lane keeping, stability control) [1]. All the ECUs in a domain are connected to the same communication bus. The activities of each ECU in a domain are controlled by a single domain controller. Domain controllers are connected through a *common gateway* to enable communication among ECUs belonging to different domains [16,27,29].

Single Bus E/E Architecture—As shown in Figure 3, in this class of architecture all the ECUs are connected to a *single bus*. This pattern is suitable for a less complex architecture when a small number of ECUs are used in a vehicle [6,16,27,29].

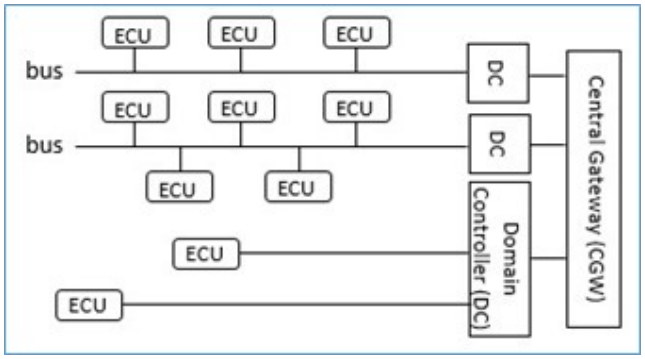

Figure 2. In-Vehicle Domain-based E/E Architecture [28].

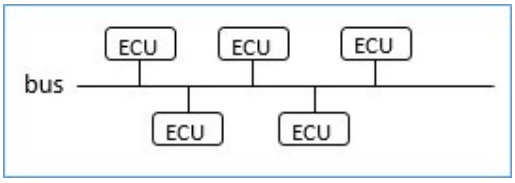

Figure 3. In-Vehicle Single Bus E/E Architecture [13].

2.2. In-Vehicle Threats

Research in the area of automotive safety has demonstrated that modern automobiles are vulnerable to a wide range of security attacks and may endanger the safety of the passengers and drivers. Possible threats can be classified in two broad categories: (i) Physical Threats and (ii) Cyber Threats [6,15,18,30,31].

2.2.1. Physical Threats

In-vehicle spoofing:*In-vehicle spoofing* refers to attacks where an attacker pretends to have another identity and influences the vehicle's operation by providing false or modified data [2,6,15,31].

Denial of Service: In *Denial of Service (DoS)* attacks, an attacker manages to compromise a component (e.g., ECU, Sensor etc.) and inhibits the regular operation of the IVN by various means such as the intentional flooding of the network with messages [1,2,15,18].

Side-Channels: In *side-channel attacks*, the adversary collects information from a hidden channel that leaks physical or logical parameters of a system and analyses the collected information to establish patterns that can extract private information. The hidden channel may already be available in hardware (such as in a processor's cache implementation), or is created using hardware and software techniques [32–35]. A wide range of side-channel attacks have been studied. Some of these, related to the automotive domain, are mentioned here. The variation of CAN bus signals can be analysed to break encrypted and authenticated in-vehicle communication protocols (e.g., poor key provisioning to explore additional vulnerabilities). CAN bus signal variation can be measured in different ways, for example, (a) access the wires directly using a high-precision oscilloscope, (b) a compromised ECU connected to the CAN bus at the input and a modified CAN controller capable of sampling the bus at a high frequency [34,35]. Even stronger key provisioning protocols for CAN bus (e.g., Pns-CAN [36]) can be exploited following this approach [34]. *Cache-based side-channel attacks* exploit the key-dependent cache access patterns of cryptographic applications. Typically, this approach uses a spy process to intentionally create cache contentions with the cryptographic application. The contentions are then analysed to infer cache access patterns, which are in turn associated with likely key values to extract the secret key processed by the cryptographic application [33,37]. Side channel attacks are even applicable to Trusted Execution Environment (TEE) [38–41]. For example, Intel SGX (version 1 and 2) suffers from *Page Fault Attacks* due to the sharing of memory, managed by the OS, by multiple

enclaves [39,42]. Intel SGX is also vulnerable to *interface-based side-channel attacks*, which can be exploited to infer the information of enclave input data, i.e., the observable enclave interface (ECALL/OCALL) invocation patterns (e.g., interface parameter sizes and invocation delay etc.) determined by different enclave input data [43].

2.2.2. Cyber Threats

Eavesdropping: The attacker may sniff and store messages transmitted among different components (e.g., between ECUs) of a vehicle, or between a vehicle and the outside world. In *passive eavesdropping*, the attacker may exploit stored messages to achieve various goals such as extracting critical information (e.g., location or route information of a vehicle) to launch further attacks. In *active eavesdropping*, the attacker modifies the content of intercepted messages or creates new messages in different ways such as injection of malicious messages, delay of the transmission of messages, and/or reorder the transmission of messages, to produce a malicious effect on the operation of the vehicle [2,18,44].

Denial of Service: As for in-vehicle *DoS attacks*, the attacker may cause disruption in authorized communication channels by applying different techniques such as jamming wireless medium or external facing sensors [2,30,45,46].

Replay Attack: In a *replay attack*, an attacker may record authenticated transmitted messages. Subsequently, the valid messages are retransmitted to gain entry to the vehicle or to perform illegitimate operations [15,18,31,47].

Code Modification and Code Injection: The attacker may gain unauthorised access to some components (e.g., IVN, ECUs) of a vehicle through other types of attacks (e.g., replay attack) and inject harmful code in ECUs to produce malicious effects or perform malicious modification to ECU code [2,18].

Packet Fuzzing: The attacker continuously sends invalid data to a component (e.g., ECU) to trigger error or fault condition, with an intention that the error condition may expose security loopholes that can be exploited to carry out further attacks [2,6].

2.3. In-Vehicle Threat Detection and Mitigation Approaches

A significant body of work in the literature focuses on the improvement of resilience of autonomous vehicles. The measures covered in these works to counter the threats and attacks on autonomous vehicles can be classified in 3 broad categories, which are (i) Proactive Defences, (ii) Active Defences and (iii) Passive Defences [2,15–17,31].

2.3.1. Proactive Defences

Proactive defences to improve autonomous vehicle security is underpinned by the "security by design" principle practiced in the software industry [15,17]. Examples include: public key encryption to encrypt messages exchanged between ECUs through the CAN bus [16,27]; or hash-based message authentication to ensure integrity of messages during in-vehicle communication [18]. Security objectives/requirements should be identified during the system analysis phase and appropriate control features should be planned during the system design phase to meet the identified security objectives. The compliance to security guidelines and a static testing of security policies and components can lead to certifications. These approaches are invasive in nature, which require a major redesign of the vehicle or its components and may not be readily acceptable to the manufacturer. Moreover, security measures implemented following these approaches may not be able to handle new type of threats that have been discovered after a vehicle has been designed and manufactured [48].

2.3.2. Active Defences

Active defence recommends near real-time encounter of adversaries as they occur. For example, continuous monitoring can be applied to check the security health conditions of the autonomous vehicles and take adequate remediation actions (e.g., reconfigure attack targets and improve tactics to have better control when the attack occurs) [49]. In this sense, real-time monitoring enables the certification of applications in safety critical systems [50].

For example, the concept of watermarking may be used to detect in real-time replay attacks and spoofing attacks in the in-vehicle network [31]. A known private signal is inserted in the system as watermark. The system output is then compared with the expected output based on the known dynamics of the system. Once monitoring detects an anomaly, Kalman Filter functions are applied to determine the ideal state of the attack target. Existing IVN intrusion detection systems can be categorised as (i) Signature-Based, (ii) Anomaly-Based and (iii) Hybrid [2,13,14,47,51,52].

Signature-Based Detection: These approaches use information about attacks (*signatures*) as a pattern that characterize known threats, and compare signatures against observed events to identify possible attacks [47]. Various mechanisms can be used to derive attacks signatures, e.g., state transition analysis, expert systems, description scripts, etc.

Anomaly-Based Detection: These approaches recommend continuous monitoring of the activities of a system, checking against a reference model (e.g., profile of the system). Alarms are raised if deviation from the reference model is observed [51]. Various mechanisms can be applied to derive the reference model of the system, such as:

- Artificial Intelligence (AI) and Machine-Learning (ML) Based: ML methods typically attempt to learn patterns in the data, and can be supervised or unsupervised [19,20,53]. Various models are possible, e.g., Deep learning [19,20].
- Frequency Based: Given that the majority of ECUs in a vehicle communicate at a regular interval, this approach utilizes known CAN packet frequency between packet sequences to detect anomalies [7,21,22].
- Statistical Based: These approaches compare the currently observed statistical profile (e.g., difference of mean, median, mode of a variable or multiple variables of interest) of the system against a previously determined statistical profile of the system [51,54,55].

Hybrid-Based Approach: combines several IDS techniques (e.g., signature-based and anomaly-based detection approaches). Since IVN attacks take different forms, a single approach is not enough to detect all attack types. Hybrid approaches have the potential to maximize IVN attack detection [2,13,14,52].

In addition, several approaches enable to detect side channel attacks related to the automotive domain.

- Side channel attacks at the physical layer can be prevented at the hardware level by applying various techniques, for example, by varying the voltage level in the bus during transmission (e.g., by using multiple transceivers for each node connected to the bus) [34].
- Cache-based attacks can be detected by quantifying contentions in shared resources, associating these contentions with processes (e.g., a victim process), and issuing a warning at runtime whenever the contentions reach a "suspicious" level [37].
- Interface-based attacks through TEE (SGX) side channels can be detected by continuously tracking the SGX interface events (ecall/ocall) and determine anomaly [50,56].

Work related to side channel attacks range from discussions and demonstrations of side channel related vulnerabilities [39,42,57–59], mitigations of these attacks with countermeasures at the hardware and software levels [42,58,60], and detections of suspicious activities suggesting attempts for side channel attacks [61–63]. While the first category reveals security flaws that often result in reporting vulnerabilities followed by patches from the software vendors [59] or other countermeasures found the literature (second category) [42,60], the third category very often uses hardware and software based counters or other accessories (oscilloscope, CPMs) that can reflect attempts for a side channel attack [64,65]. It should be noted that existing approaches for active defence may not be easily applicable in the automotive domain. For example, signature based detection and machine learning based detection are not easy for the CAN network due to the low processing power of the network. Moreover, despite CAN being an open protocol, variation in the specification of transmission of CAN frames in different makes and model hinders to develop a uniform solution [14,66].

2.3.3. Passive Defences

Passive defence mechanisms mainly focus on detecting, responding and recovering from a security attack once it occurs. This type of defence is suitable to mitigate malwares and code injection and modification techniques. Therefore, it does not support adaptive security mechanisms [16,49,67].

3. FIVADMI: A Framework for Improving In-Vehicle Isolation and Resilience

This section presents an overview of a Framework for In-Vehicle Anomaly Detection by Monitoring and Isolation (FIVADMI) which supports the detection of IVN anomalies by monitoring the data packets exchanged among ECUs. The framework also certifies the IVN state by analysing the monitoring results. In the following sections, we discuss the FIVADMI design principles, introduce the architecture of the framework, discuss the monitoring and certification schemes, and provide some implementation considerations.

3.1. FIVADMI Design Principles

The major objective of FIVADMI is to improve *in-vehicle resilience* by (1) identifying *anomalies* in *real time* in the IVN operation, and (2) taking mitigation actions for the identified anomalies. FIVADMI aims to deal with different issues that may arise due to the application of future generation ECUs (e.g., virtualized as lightweight execution environments such as VMs and Containers) within the vehicle. More specifically, FIVADMI aims to ensure *system-level isolation* of ECUs and provide protection against *side channel attacks* to secure communications. Moreover, FIVADMI is designed to be *scalable, interoperable* and *reusable*.

Resilience: FIVADMI adopts the *active defence* approach to improve in-vehicle resilience, i.e., security properties related to the communication among ECUs are continuously monitored to detect security threats. Proactive defence is not suited to handle common adaptive security requirements in the cyber and automotive domains: it recommends to design control features to meet security objectives at system design time, which are then integrated into the system life-time. However, this approach cannot address new types of threats once the system has been developed. Passive defence is also not suitable for safety-critical systems, like autonomous vehicles, as that approach detects attacks once they have occurred. Therefore, FIVADMI takes actions to mitigate the impact of and gracefully recover from detected security threats. In our context, recovery consists of rolling back (or forward) to a stable state to overcome an intrusion [68]. A full description of the rolling back process is out of the scope of this paper. Also, FIVADMI is based on an external rule engine for real-time detection of anomalies. That approach makes FIVADMI non-invasive, and is most suitable for the automotive domain.

Isolation and side-channel protection: FIVADMI ensures isolation of ECUs in the in-vehicle network by deploying each ECU within a *Trusted Execution Environment (TEE)*. FIVADMI also deploys a component to detect *side channel attacks* in a vehicular context.

Scalability, interoperability, reusability: FIVADMI is based on the *AUTOSAR* open standard for automotive software architecture [25]. This choice gives an assurance of scalability, interoperability, and reusability.

3.2. FIVADMI Architecture

Figure 4 shows the reference architecture for IVN threat detection and mitigation. FIVADMI assumes an in-vehicle *domain-based E/E architecture*. The major components of the architecture are described next.

Trusted Execution Environment (TEE): Trusted Computing gave birth to a number of industry initiatives to specify *isolated execution environments* in the main processor [38,41]. TEE provides security features such as isolated execution, integrity of applications executing with the TEE, and confidentiality of application assets. Several hardware vendors have embedded hardware technologies to support TEE implementations, including AMD PSP [69], ARM TrustZone [70], EVITA Hardware Security Module (HSM) [71] and Intel SGX Software Guard Extensions [44]. The Trusted Platform Module (TPM), based on Trusted

Computing Group (TCG) [56] specifications, is a dedicated processor to secure hardware. In this work, we explore how to apply TEE as execution environments for ECUs to ensure isolated and secure communications between ECUs.

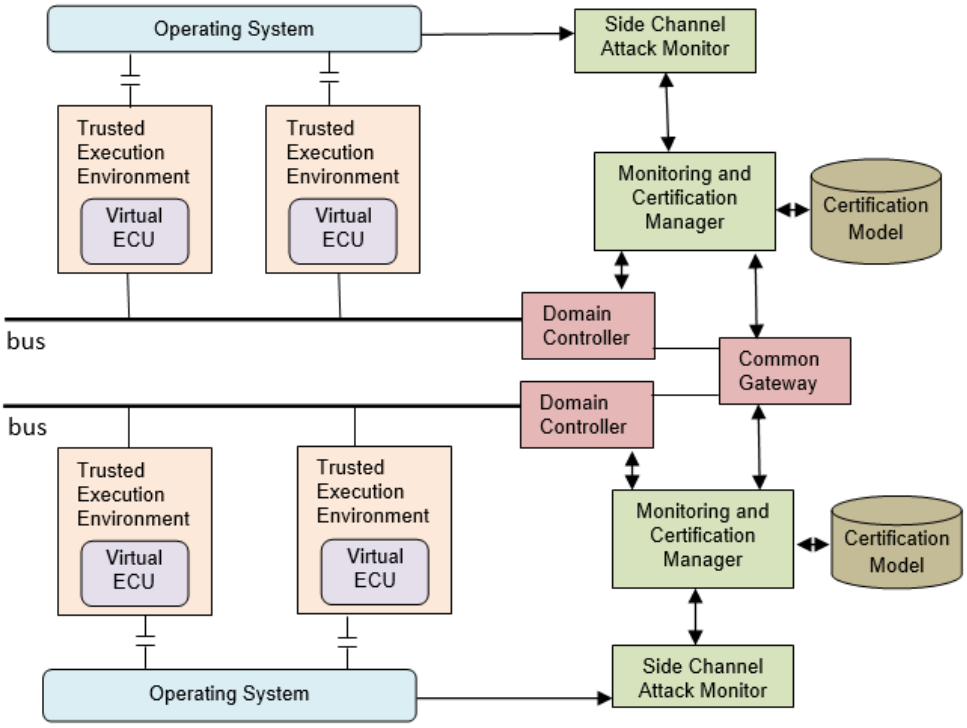

Figure 4. Reference Architecture of FIVADMI.

Side Channel Attack Monitor (SCAM): TEEs remain vulnerable to side channel attacks [42,58]. This component focuses on the runtime detection of side channel attacks (e.g., SGX interface based attacks) that are relevant in a vehicular context. Through the monitoring of Hardware Performance Counters, this component flags suspicions rowhammer and cache and covert side channel attacks.

Monitoring and Certification Manager (MCM): This component performs real-time monitoring of security properties related to IVN components (e.g., ECUs) to detect security threats. It applies hybrid techniques (including frequency-based and statistical-based approaches, and deep packet inspection) to detect threats. Based on the validity of the security properties, this component also maintains the certificates that certify the valid state of ECUs.

To implement the architecture, FIVADMI adopts the AUTomotive Open System ARchitecture (AUTOSAR) open standard. The AUTOSAR consortium was formed by major automotive OEMs like BMW, Ford, and Daimler Chrysler to standardize the automotive software architecture and framework, thereby facilitating scalability, reusability and interoperability across product lines from different OEMs [25]. By adopting AUTOSAR in the implementation, FIVADMI inherits the corresponding benefits of that standard (e.g., scalability, reusability and interoperability).

In AUTOSAR, ECU software is abstracted as a layered architecture built on top of the underlying micro-controller hardware [72]. As shown in Figure 5, this architecture is composed of three layers: namely *Basic Software (BSW)*, *Runtime Environment (RTE)*, and *Application Layer*.

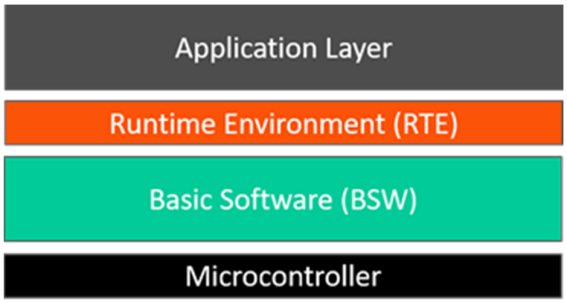

Figure 5. Overview of Software Layers High-Level View [72].

Basic Software Layer (BSW): this layer provides core system functionalities. As shown in Figure 6, this layer has 3 sub-layers, (i) *Micro-controller Abstraction Layer (MCAL)*—contains internal drivers modules that access the micro-controller and internal peripherals directly, (ii) *ECU Abstraction Layer (ECUAL)*—interfaces the drivers of the MCAL and offers an API for accessing peripherals and external devices, thus making higher software layers independent of the hardware layout, and (iii) *Services Layer (SL)*—provides top level services (e.g., operating system, communication, management, and memory services) to application software components.

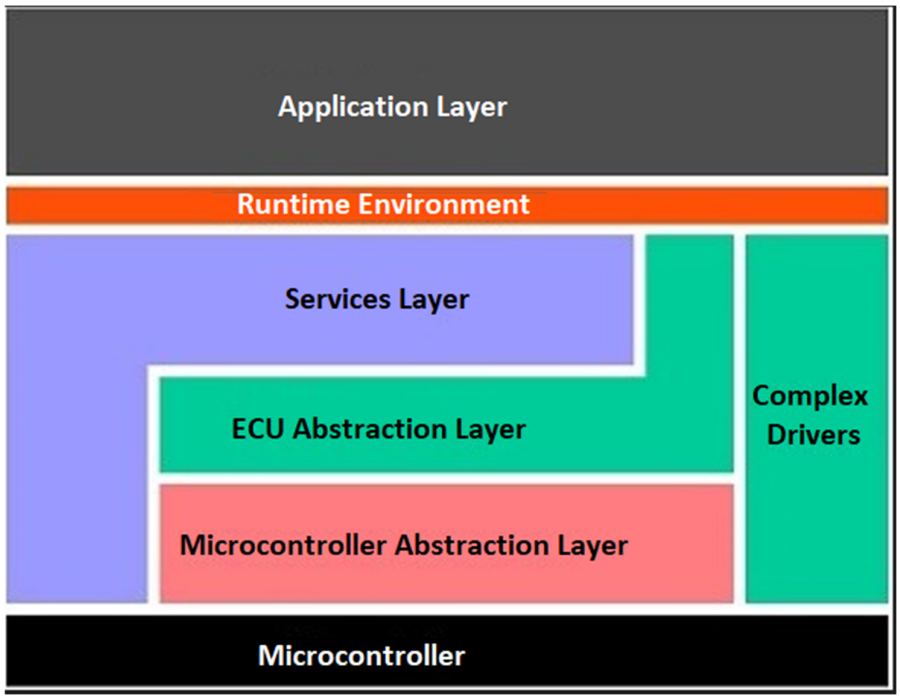

Figure 6. Overview of Software Layers Refined view [72].

Run-Time Environment (RTE): this layer provides communication services to application software, acting as a bridge between applications and the BSW layer.

Application Layer: this layer mainly consists of Software Components (SWC) interconnected to other SWCs and BSW modules. This layer adopts a component-based style of architecture that enhances scalability and re-usability of SWCs.

Figure 7 shows the mapping of FIVADMI architecture with the AUTOSAR architecture. The left side (yellow box) of Figure 7 shows the deployment of virtual ECUs within a TEE following the AUTOSAR architecture. The right side of Figure 7 shows the Domain Controller, Monitoring & Certification Manager and Side Channel Attack Monitor components in FIVADMI, developed as Software Components (SWCs) in the application layer of the AUTOSAR architecture. These components reside within a trusted virtual ECU and collect

the data transmitted among the virtual ECUs within the vehicle. The collected data is used for monitoring the properties related to different ECUs.

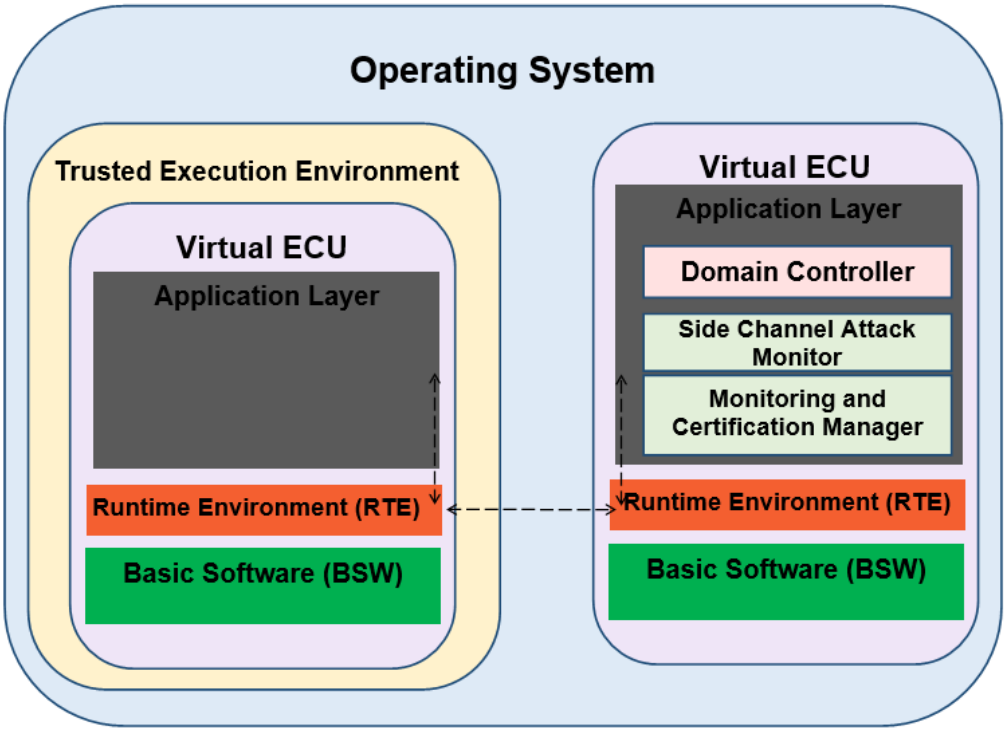

Figure 7. Mapping of FIVADMI and AUTOSAR Architecture.

3.3. Monitoring and Certification Manager

This component performs real-time monitoring of *security properties* and of *behavioural properties* related to IVN components (e.g., ECUs) to detect anomalies. *Security properties* are used to detect threats and attacks at the in-vehicle communication level such as Injection attacks or DoS attacks. *Behavioural properties* are used to detect behavioural anomalies of the vehicle apparent at the application or OBD level, such as anomalies in vehicle acceleration or in coolant temperature. Based on the monitoring results, this component also produces and maintains the certificates of the validity state of individual ECUs and the overall IVN.

The UML activity diagram shown in Figure 8 presents high level view of the FIVADMI monitoring and certification process. The Monitoring and Certification Manager collaborates with a Rule Engine and with the IVN. The Monitoring and Certification Manager receives the regular data exchanged among ECUs and forwards them to the Rule Engine to verify the rules corresponding to the security and behavior policy. In the sequel, security properties and behavioural properties are denoted as security rules and as OBD rules respectively.

At first, the Monitoring and Certification Manager creates the data structures to store runtime data received from the IVN. It then instructs the Rule Engine to create new instances for the OBD rules, with the corresponding data structures inside the Engine.

When the Monitoring and Certification Manager receives data from the IVN, it instructs the Rule Engine to create instances of security rules and to execute all rules against the received data. After this step, the Monitoring and Certification Manager collects the rule execution results and analyses them. Based on the monitoring results, the Monitoring and Certification Manager produces a new certificate or updates the existing one. More detailed descriptions of the monitoring scheme and of the certification schemes are presented in Section 4.3.

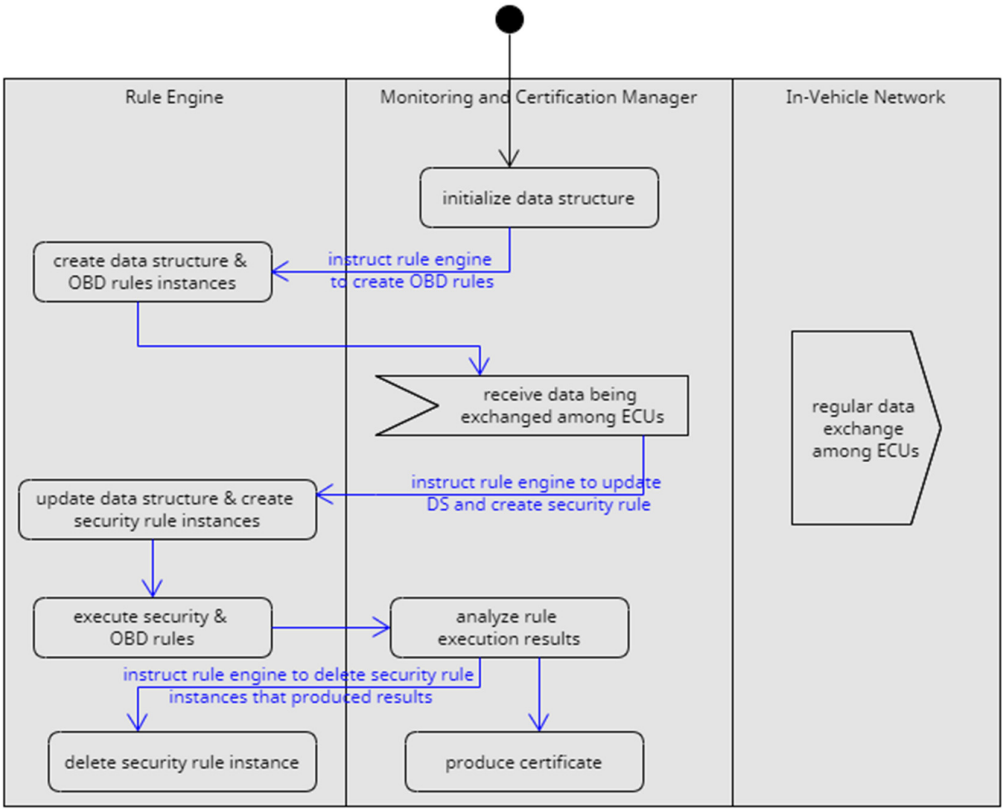

Figure 8. Overview of Monitoring and Certification Process.

3.4. Certification Model

In FIVADMI, the Monitoring and Certification Manager and the Side Channel Attack Monitor perform real-time monitoring of security properties related to ECUs and of behavioural properties of the vehicle to detect anomalies. These components then produce a certificate to certify the valid state of individual ECUs and of the overall IVN. Figure 9 shows a high-level structure of the corresponding certificate.

The main elements of the Certificate structure are the following.

CertificateID: Represents the unique identifier of a generated certificate during monitoring.

MonitoringResultAggregator: Contains the aggregation of monitoring results produced by monitoring different components (e.g., ECUs, CAN bus) of the IVN. This element contains the following sub-elements:

- AggregationTime: Represents the time of aggregation.
- Duration: Specifies time span between monitoring results considered for aggregation (setup by external config file).
- ToMList: Contains a list of TargetOfMonitoring considered for aggregation.
- AggregationRule: Defines how monitoring results should be aggregated. For instance, results with numerical values can be aggregated by applying statistical methods (setup by an external configuration file).
- AggregationResult: Stores the aggregation result.

TargetOfMonitoring (ToM): Represents a component (e.g., ECU, CAN bus) that is monitored to identify security threats associated with it. This element has the following sub-elements:

- ToMType: Type of component (e.g., ECU, CAN bus) to be monitored.
- ToMID: Unique IVN component identifier.
- MonitoringRule: Security property related to this component that is monitored.

- MonitoringEvidenceAggregator: Contains aggregation of results by monitoring the MonitoringRule related to this component. This element contains the following sub-elements:
    a. AggregationTime: Represents the time of aggregation.
    b. Duration: Specifies time span between which in-vehicle network data were considered for monitoring (setup by external configuration file).
    c. AggregationRule: Defines how monitoring results should be aggregated. For instance, results with numerical values can be aggregated by applying statistical methods (external configuration file).
    d. AggregationResult: Stores the aggregation result.

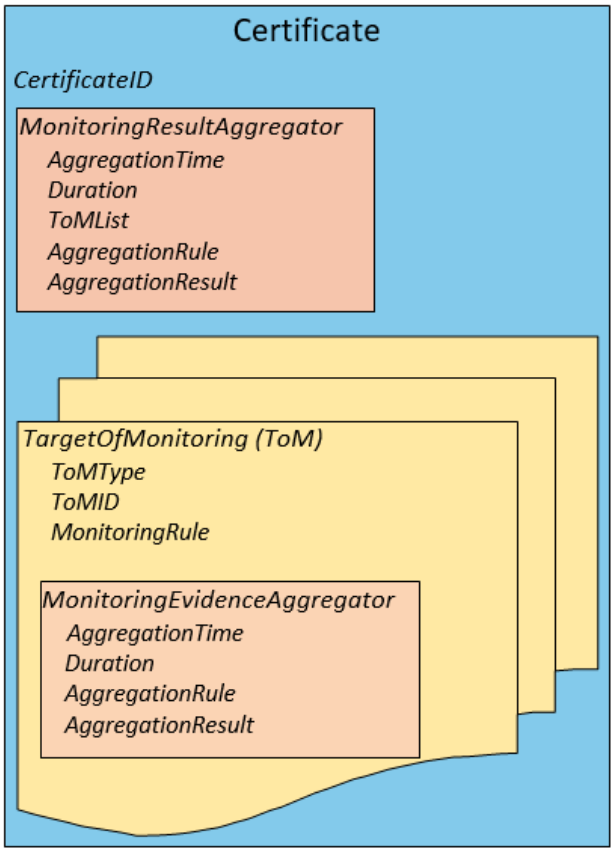

Figure 9. Overview of Certification Model.

3.5. Side Channel Attack Monitor

Side channel attacks are generally "noisy" [73], meaning that they leave a trace that enables flagging suspicious activities. Cache based side channel attacks, including flush and reload, prime and probe and evict and time are proven to be effective approaches that jeopardise the confidentiality of execution in an enclave [42]. The majority of side channel detection approaches use Hardware Performance Counters (HPCs) which are special registers used for performance monitoring [74]. HPC based tools reveal run time based software and hardware events including CPU cycles and cache hits and misses [61,63]. Cache Hits and Misses are among the top key indicators of side channel related activities [61–63], and different patterns of abnormalities in these indicators reflects different attack strategies [62]. Many of the detection approaches in the literature use L1, L2 and L3 Cache attributes to detect Cache based side channel attacks [61,63,75]. Machine learning based approach could be an option for similar classifications due to the capability of these approaches to recognise complex patterns from different inputs [61]. Simple rules can also be used for less complex classifications. In FIVADMI we adopt rule-based approach as it conforms with the low

processing power of in-vehicle Network and ECUs. Figure 10 gives an overarching view of the design that the Side Channel Attack Monitor in FIVADMI adheres to; through the readings on the hardware performance counters, the classification tier uses classification rules to flag suspicions of side channel activities.

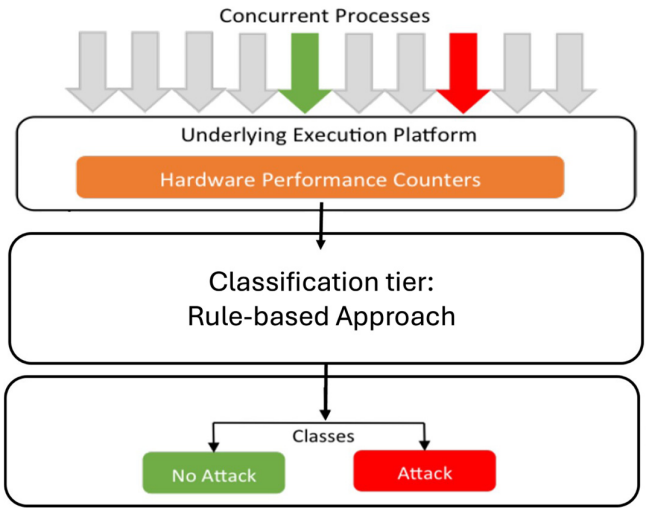

Figure 10. Overview of HPC Based Detection Approaches for Side Channel Attacks.

4. Implementation of FIVADMI

This section presents the implementation details of FIVADMI. The first part highlights the specification of our Trusted Execution Environment. This is followed with a detailed explanation of the technology and algorithms used for our monitoring scheme.

4.1. Trusted Execution Environment (TEE)

Open Enclave (OE) is an open source project to develop applications in a *trusted execution environment*. OE enables to build trusted programmes called enclave applications that interact with untrusted applications, known as *host applications* [26]. Through SGX-enabled operations and support of an Intel CPU with SGX capabilities, enclaves ensure the confidentiality of execution and protection of data [26]. Figure 11 gives an overview of the OE architecture. To execute the code of an enclave, a host application, known to the enclave, calls the interface of the functions in the enclave; these calls are Enclave Calls (ECalls). Enclaves can also call untrusted functions and services provided by the operating system through an interface of the host application; these calls are Outside Calls (OCalls) [76]. ECalls and OCalls are listed as trusted and untrusted functions respectively in a configuration file before the enclave is built; Edger8r, a tool part of the Intel SGX SDK, generates the interfaces specified between the host and the enclave using an Enclave Definition Language (EDL). Interactions are restricted to the specified interfaces throughout the lifecycle of the enclave [77].

In FIVADMI, an enclave is part of the application layer of the ECU. It hosts OBD messages essential to define its functionality. In our prototype, the data hosted within the enclave includes data recorded from real vehicles and used for testing as well as data generated at runtime to simulate real-life behaviour of in-vehicle traffic. While the dynamic data is generated at runtime, the static data is stored in encrypted form. Decryption occurs through a call from the host bound to the enclave.

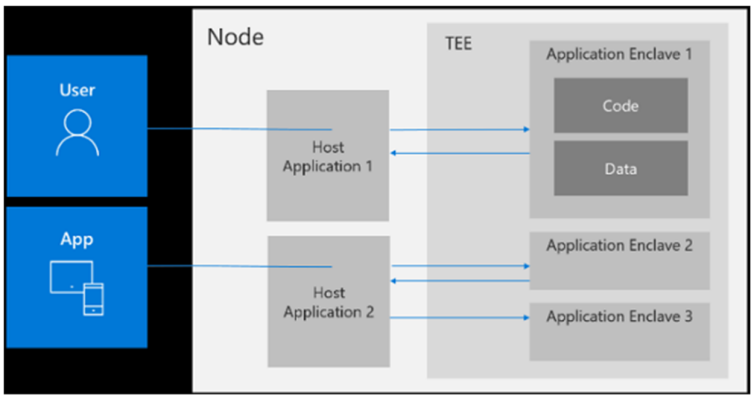

Figure 11. Overview of the Open Enclave Architecture.

4.2. Side Channel Attack Monitor

HexPADS [62,78] is an open-source implementation that uses Perf and covers detections for a variety of side channel attacks including cache-based attacks. HexPADS uses cache hits, cache misses and page faults as HPC attributes; these attributes can be accessed through 'Windows Performance Monitor' [79]. We adopted the algorithm of HexPADs (Figure 12) to detect Cache Side Channel attacks and used Microsoft Excel (version 2016) to apply the formulas on the readings. Determining the thresholds for the side channel detection algorithm in our environment was done through studying the changes in the values of the monitored HPC attributes when a side channel attack is performed. To study this change, we recorded the values of these attributes for two cases: the first with an ECU is running and the second with the same ECU running in parallel with a script that performs a side channel attack. For each case, we collected the data for 300 min equally spread over 15 iterations. For the attack, we used an open-source project that performs a simple Flush + Reload, a cache-based side channel attack [80]. The approach includes two phases: a profiling phase to locate a memory address that is the most engaged during the execution of the victim application, followed with an exploitation phase the targets that memory address to perform the Flush + Reload [81]. The project includes an attacker application, and the executable of a Virtual ECU is the victim application. The algorithm in Figure 12 shows our side channel detection algorithm, adopted from HexPADS, with the threshold obtained from studying the cache activity in our environment.

```
Side_Channel_Detection()
miss_rate = (\\Hyper-V_Server_Name\Memory\Cache Bytes) /
            (\\Hyper-V_Server_Name \Memory\Memory\Cache Faults/sec)
fault_rate =  (\\Hyper-V_Server_Name \Memory\Cache Faults/sec [cur]) /
              (\\Hyper-V_Server_Name \Memory\Cache Faults/sec [prev])
if (
1.49 < miss_rate < 10.4 and
0.00029 < cache_miss < 0.00125 and
1.1245 < fault_rate < 2.8
) cache_attack_detected ();
```

Figure 12. The Algorithm for Cache Side Channel Detection.

Looking at the figure, the algorithm relies on threshold values of hardware performance counters that give early indications of side channel attacks. The phase of identifying these thresholds enables the algorithm to adapt to different operating systems for the ECUs, leading to increasing the applicability of the proposed approach to autonomous vehicles using chips from a variety of OEM. This algorithm is effective and lightweight, enabling its deployment on ECUs that have very limited computational resources. The simplicity of the rule-based approach makes it suitable for state-of-the-art ECUs where the on-chip memory is limited to 5 MB as indicated by a list compiled by NXP [82].

4.3. Monitoring Scheme

In FIVADMI, the Monitoring and Certification Manager uses a predefined set of security rules and of OBD rules. To specify the high-level rules, we use a variant of Event Calculus (EC) [83,84], which is a first order temporal logic language. The choice of event calculus as the specification language has been motivated by the need for expressing the rules in terms of events and temporal constraints, which are the driving factors of our monitoring scheme. In EC, the behaviour of a system is specified in terms of *events* and *fluents*. An *event* is something that occurs at a specific instance of time and may change the state of a system. A *fluent* is a signifier of a system state (e.g., the value of a variable). The occurrence of an event is signified by the predicate ***Happens (e, t)***, which signifies that an event e occurs at some time *t*. An event may initiate or terminate a fluent specifying part of the state of a system. In EC, these effects are represented by the predicates ***Initiates (e,f,t)*** and ***Terminates (e,f,t)***, respectively. The former of these predicates signifies that a fluent *f* starts to hold after the event *e* at time *t*. The latter predicate signifies that a fluent *f* ceases to hold after the event *e* occurs at time *t*. An EC specification may also use the predicates ***Initially (f)*** and ***HoldsAt (f,t)***. The former of these predicates signifies that the fluent *f* holds in the beginning of the operation of a system and the latter predicate signifies that the fluent *f* holds at time *t*.

In the rest of this section, we present the EC rules used by the Monitoring and Certification Manager. We then discuss the algorithm that coordinates the monitoring process.

4.3.1. Rules for Security Properties

Figure 13 shows rules expressed in EC, to monitor injection attack.

IR1. Initially(CAN_Data(dID, dMsg, dTS))

IR2. Happens(CAN_Frame(cID, cMsg), cTS) ∧ **HoldsAT**(CAN_Data(dID, dMsg, dTS), cTS) ∧ cTS ≠ dTS ≠ −1 ∧ dID ≠ −1 ∧ cID ≠ −1 ∧ dID = cID ⇒ (cTS – dTS) < 50

IR3. Happens(CAN_Frame(cID, cMsg), cTS) ∧ **HoldsAT**(CAN_Data(dID, dMsg, dTS), cTS) ∧ cTS ≠ dTS ⇒ **Initiates**(CAN_Frame(cID, cMsg), equalTo(dID, cID), cTS) ∧ **Initiates**(CAN_Frame(cID, cMsg), equalTo(dMsg, cMsg), cTS) ∧ **Initiates**(CAN_Frame(cID, cMsg), equalTo(dTS, cTS), cTS)

Figure 13. EC Rules—Injection Attack Detection.

Rule IR1 signifies the creation of the fluent *CAN_Data(dID, dMsg, dTS)* with empty values at the beginning of the monitoring process. This fluent will be used to store CAN frame source ID, the message contained in the CAN frame and the time stamp of receipt.

Rule IR3 is used to update the fluent *CAN_Data*. Each time a CAN frame is received (signified by the predicate *Happens(CAN_Frame(cID, cMsg), cTS)*), the values of *dID*, *dMsg* and *dTS* in the fluent *CAN_Data* are updated which is signified by the predicates *Initiates(CAN_Frame(cID, cMsg), equalTo(dID, cID),cTS)*, *Initiates(CAN_Frame(cID, cMsg), equalTo(dMsg, cMsg), cTS)* and *Initiates(CAN_Frame(cID, cMsg), equalTo(dTS, cTS), cTS)*.

Rule IR2 is used to detect IVN injection attacks. FIVADMI applies a frequency based analysis approach. By design, most CAN messages are transmitted at regular frequencies. These frequencies are nearly constant, even during transitions between vehicle driving modes [2,7,22]. FIVADMI checks the difference of arrival times between two successive CAN frames from the same source. If the difference is less than a predefined threshold, it raises an alarm as an injection attack might have occurred. The CAN frame that has arrived (signified by the predicate *Happens (CAN Frame(cID, cMsg), cTS))* contains the ID and time stamp of the current CAN frame. The fluent CAN_Data (signified by *HoldsAT (CAN Data (dID, dMsg, dTS), cTS))* contains the ID and time stamp of the previous frame. The difference of arrival time stamps of two successive CAN frames from the same source is checked. The rule checks that both CAN frames have valid data and different time stamps. The rule also checks that both frames are from the same source and that the difference

between time stamps is less than a threshold (e.g., 50). If these conditions are violated, then an alert is raised.

4.3.2. Rules for Behavioural Properties

Figure 14 shows rules expressed in EC, to detect anomalies in vehicle speed. However, it should be noted that similar rules can be specified to detect anomalies in coolant temperature, ambient temperature, and intake air temperature.

SR1. Initially(OBD_Data(dID, dSpd, dTS))

SR2. Happens(CAN_Frame(cID, cSpd), cTS) ∧ **HoldsAT**(OBD_Data(dID, dSpd, dTS), cTS) ∧ cTS ≠ dTS ≠ −1 ∧ cSpd ≠ -1 ∧ dSpd ≠ −1 ⇒ ((cSpd – dSpd)/(sTS – dTS)) < 0.2

SR3. Happens(CAN_Frame(cID, cSpd), cTS) ∧ **HoldsAT**(OBD_Data(dID, dSpd, dTS), cTS) ∧ cTS ≠ dTS ≠ −1 ∧ cSpd ≠ -1 ∧ dSpd ≠ −1 ⇒ ((cSpd – dSpd)/(sTS – dTS)) > −0.2

SR4. Happens(CAN_Frame(cID, cSpd), cTS) ∧ **HoldsAT**(OBD_Data(dID, dSpd, dTS), cTS) ∧ dTS < cTS ⇒ **Initiates**(CAN_Frame(cID, cSpd), equalTo(dID, cID), cTS) ∧ **Initiates**(CAN_Frame(cID, cSpd), equalTo(dSpd, cSpd), cTS) ∧ **Initiates**(CAN_Frame(cID, cSpd), equalTo(dTS, cTS), cTS)

Figure 14. EC Rules—Vehicle Speed Anomaly Detection.

Rule SR1 signifies the creation of the fluent *OBD_Data(dID, dSpd, dTS)* with empty values at the beginning of the monitoring process. This fluent will be used to store CAN frame source ID, the vehicle speed reading contained in the CAN frame and the time stamp of receipt.

Rule SR4 is used to update the fluent OBD_Data. Each time a CAN frame is received (signified by the predicate *Happens(CAN Frame(cID, cSpd), cTS)*), the values of *dID*, *dSpdg* and *dTS* in the fluent OBD_Data are updated which is signified by the predicates *Initiates(CAN Frame(cID, cSpd), equalTo(dID, cID),cTS)*, *Initiates(CAN Frame(cID, cSpd), equalTo(dSpd, cSpd), cTS)* and *Initiates(CAN Frame(cID, cSpd), equalTo(dTS, cTS), cTS)*.

Rules SR2 and SR3 are used to detect anomalies in acceleration and deceleration respectively. This is done by calculating acceleration (or deceleration) of the vehicle as follows:

$$Acc = \frac{(\mathrm{v2} - \mathrm{v1})}{(\mathrm{t2} - \mathrm{t1})} \quad (1)$$

where v2 and v1 are vehicle speeds at time points t2 and t1 respectively. If the calculated acceleration (or deceleration) is greater (or lower) than an acceptable threshold, an alarm is raised.

According to the rule SR2, the CAN frame that has arrived (signified by the predicate *Happens (CAN_Frame (cID, cSpd), cTS)*) contains the speed value and time stamp of the current CAN frame. The fluent OBD_Data (signified by *HoldsAT (OBD_Data(dID, dSpd, dTS), cTS)*) contains the speed value and time stamp of the previous frame. The rule checks that both current CAN_Frame and the previous CAN_Frame have valid speed value and different time stamps; the acceleration is computed. An alert is raised if the acceleration is above a threshold.

Rule SR3 checks the anomaly in deceleration, which can be explained in a similar way as the Rule SR2.

4.3.3. Monitoring Algorithm

Runtime monitoring of security and behavioural rules in FIVADMI is carried out by a Rule Engine. The current implementation of FIVADMI is based on the CLIPS [24] rule engine: in typical usage, CLIPS maintains a set of *facts* and a set of *rules*, where rules operate on facts. In CLIPS, a fact is a piece of information (e.g., variable), and a rule takes the form similar to *IF (condition) THEN (action) ELSE (action)* statements in a procedural language.

CLIPS uses an improved form of the well-known Rete algorithm [85] to match rules against facts. Rules written in CLIPS can be considered as data-driven programs, the facts being the data that trigger execution of rules via the inference engine. The inference engine then decides which rules should be executed and when.

EC rules described in Sections 4.3.1 and 4.3.2 are easily transformed into corresponding CLIPS specification. More specifically, *fluents* in EC are transformed into *facts* (e.g., the fluent *CAN_Data(dID, dMsg, dTS)* is transformed into a *fact* called CAN_Data to contain id, message, and timestamp. Similarly, the fluent *OBD_Data(dID, dSpd, dTS)* is transformed into a fact called OBD_Data and EC rules are transformed into CLIPS rules. At boot-time, instances of these rules are created in CLIPS. At runtime, the Monitoring and Certification Manager receives CAN frames from the IVN, retrieves relevant data from each frame, and stores these data into facts in CLIPS. Security rules and OBD rules are then run against these facts.

Figure 15 shows the algorithm that drives the monitoring process in FIVADMI. The monitoring algorithm is divided into three methods.

```
monitoring()
1       setup fact CAN_Data in CLIPS
2       setup fact OBD_Data in CLIPS
3       setup OBD rules in CLIPS
4       while true
5           receive a CAN_Frame
6           security_rules_check(CAN_Frame)
7           obd_rules_check(CAN_Frame)

security_rules_check(CAN_Frame)
1   extract ID, msg and ts from CAN_Frame
2   update CAN_Data slots in CLIPS

3   for each type of threat to be detected
4       create fact in CLIPS, where fact name ends with _ID+ts
5       create rules in CLIPS, where rule name ends with _ID+ts

6   instruct CLIPS to execute each rule
7   collect rule execution output from CLIPS
8   if a rule has produced a result
9       store the result for further analysis
10      remove the rule and fact from CLIPS

obd_rules_check(CAN_Frame)
1   if CAN_Frame contains OBD related data
2       update OBD_Data slots in CLIPS

3   instruct CLIPS to run each rule
4   collect rule execution output from CLIPS
5   if a rule has produced a result
6       store the result for further analysis
```

Figure 15. Monitoring Algorithm.

Method monitoring(). This method starts the monitoring process. Facts are created in CLIPS (lines 1–2). The corresponding OBD rules are also created (line 3). The loop in line 4 drives the monitoring process: when a *CAN Frame* arrives, the security rules and the OBD rules are monitored, using methods *security_rules_check()* and *obd_rules_check()* respectively.

Method security_rules_check(). Upon receipt of a CAN frame, the ID, message, and time stamp of the frame are extracted. Fact *CAN Data* in CLIPS is updated (lines 1–2). Then, security rules and corresponding facts are created for each type of threat to be detected (lines 3–5). Fact and rule names are made unique by appending the CAN ID and time stamp. This enables checking security rules for two successive CAN frames coming from

the same source. CLIPS executes the security rules and collects the execution results (line 6–7). According to the dynamics of CLIPS rule engine, if a security rule triggers a result (e.g., alert or absence of alert) at any time point, that rule will continue to produce that result for the subsequent time points, which will be considered as a false positive or false negative. Therefore, security rules that have produced a result are removed from CLIPS. The result is saved for further analysis during the production of certificates. (line 9).

Method obd_rules_check(). Upon receipt of a CAN frame, the OBD related data (e.g., speed, coolant temperature, intake air temperature and ambient temperature) is extracted and checked. Fact *OBD_Data* is updated accordingly using that data, executing the corresponding rule. The end of the method is similar to the algorithm for checking the security rules.

Monitoring results are aggregated over a period of time and used to produce certificate, following the certificate model presented in Section 3.4. In FIVADMI, a JSON data structure is used to construct the elements of the certificate. JSON is chosen for its lightweight processing and low memory footprint compared to older data structures. Using JSON, a certificate gets updated at the end of each monitoring cycle, spanning over a configurable period of time, during the certification process of a vehicle. Figure 16 shows a skeleton structure with the different elements of a certificate.

```
{
  "Certificate ID": [integer],
  "MonitoringResultAggregator":{
    "Aggregation Time": [date and time],   /* time of the last aggregation */
    "Duration": [duration], /* timespan of a monitoring cycle */
    "ToMList": [List of Strings],  /* list of ToMs considered for aggregation */
    "Aggregation Rule": "Statistical",
    "Aggregation Result": [colour],  /* status of certificate as colour code e.g. Red,
                                                          Amber, and Green */
    "TargetofMonitoring(ToM)":{
      "ToMType": ['CAN' || 'ECU'],
      "ToMID": [CANbus channel ID || ø],   /* reflects the monitored CAN bus
                                  channel ID when injection rules are monitored */
      "MonitoringRule": [String],   /* name of the monitored rule*/
      "Monitoring Evidence Aggregator": {
        "Aggregation Time": [date and time],
        "Duration": [duration],
        "Aggregation Rule": [String],  /* aggregation method for a specific ToM */
        "Aggregation Result": [String]   /* result of the aggregation */
      }
    },
    "TargetofMonitoring(ToM)": { ... }
}
```

Figure 16. Certificate Generated by FIVADMI.

5. Results

We carried out a set of experiments to evaluate the implementation of FIVADMI. In this set of evaluation experiments, an environment is set up that includes three running ECUs connected via the same CAN bus: one ECU is programmed to send and another to receive CAN frames. The third ECU performs the monitoring and certification functions. This setup is used for the evaluation of anomaly detection, the resilience of our implementation of FIVADMI, and the Side Channel Monitor in our framework. The rest of this section discusses the evaluation of our implementation of FIVADMI: Section 5.1 demonstrates the anomaly detection functionality of our framework and shows the detected violations with sudden accelerations; Section 5.2 tests the resilience of our implementation through the use of a fuzzer for the generation of CAN packets; Section 5.3 covers the evaluation of our Side Channel Monitor and discusses the results that we obtained.

5.1. Evaluation of the Anomaly Detection

We used data from real vehicles, found on Github [86], to evaluate our framework. The dataset contains readings for OBD data including speed and temperature related attributes covered by our rules. The dataset covers readings from different drivers and vehicles. Using an OBD encoder that we developed, we generated CAN frames for each record in the dataset for the evaluation. When the dataset is used for the testing of our monitoring rules, the result of the inspection with the speed rules shows a close to zero violation rate for drivers on the same vehicle and for the same driver on different vehicles. We adopted a similar approach to that of Taylor et al. [22] to introduce anomalies based on this real data: we raised the rate of the OBD messages. For instance, when the rate of the messages is 10×, the change of speed in testing data is 10× that of real data.

Figure 17 shows the change in the violation rate (*number of violation alerts/total number of alerts*) for Speed and Intake Air Temperature ToMs in two vehicles. The results show that, while the alerts for speed anomaly increase with the rate of speed change, the temperature rules remain steady. This is due to the minimal change of the temperature related data in the evaluated vehicles.

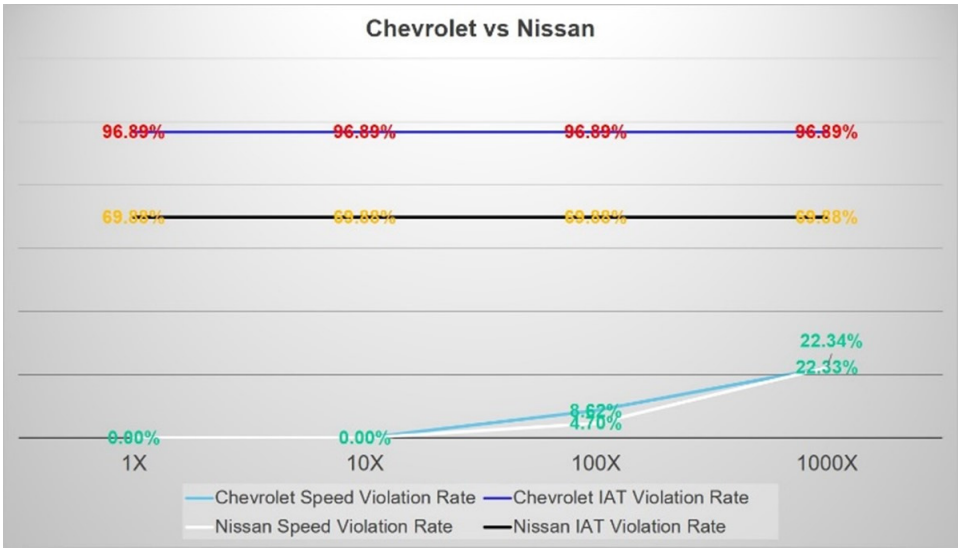

Figure 17. Violation Rates in Speed and IAT from different vehicles.

The high observed anomaly ratio in the graph results from the configuration of a threshold value in the monitoring rules. For the purpose of this evaluation, the value of the threshold was chosen to reflect coloured labels implemented as part of the FIVADMI framework. In the example above, the Certification Manager is configured to assign a Green label when the anomaly rate for a ToM is less than 25%, Red when greater than 75%, and Amber otherwise.

To maximise the adoption of our framework, the monitoring and certification rules are configurable to adapt to the specification of different vehicles in terms of frequency of OBD messages from different ECUs, or acceptable speed shifts within a particular timeframe, among other specifications. Similarly to the case described above, real data derived from different drivers using the same vehicle reflect similar results for speed and temperature rules. Figure 18 shows the violation rate alerts for speed rules for different drivers. The observed difference in the violation rates seems due to the drivers' behviaours: in this case, Driver 3 achieved the most steady driving pattern.

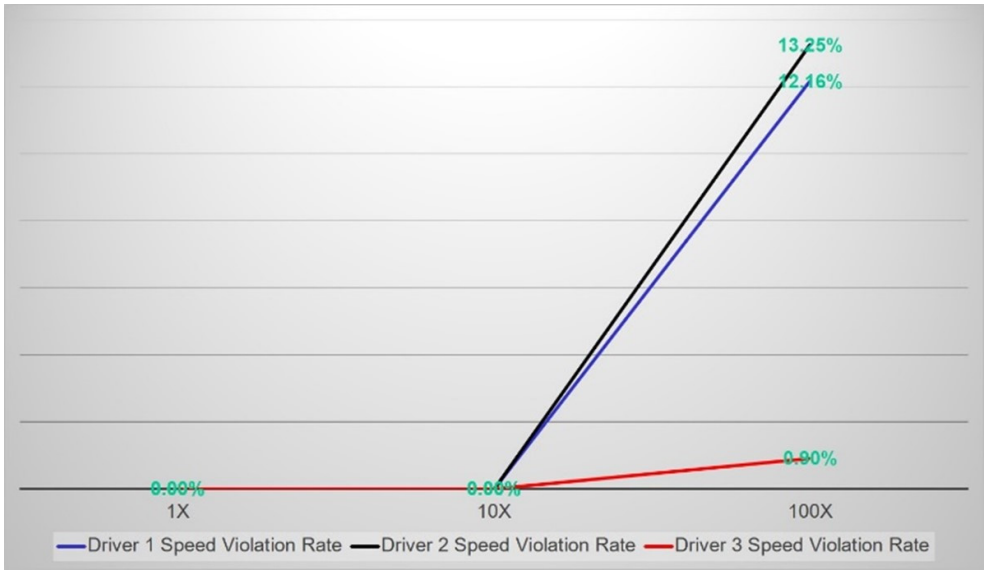

Figure 18. Violation Rates for Speed for different drivers with the same vehicle.

5.2. Evaluation of the Resilience of FIVADMI

To test the resilience of the framework, a fuzzer [87] was used to generate CAN frames. In order to demonstrate the applicability of our framework in realistic settings, we configured the fuzzer to produce CAN frames of up to 128 bits per frame and 100 frames per second. This conforms with the size of a real CAN frame and CAN bit rates [88].

FIVADMI was run over a period of two days with CAN frames generated by the fuzzer to verify that these frames do not trigger any malfunction that will disrupt the functionality of the tool [6]. The generated certificate includes the following ToMs: 'Injection', 'Speed', 'Intake Air Temperature', and 'Coolant'. This proves the diversity of CAN frames generated by the fuzzer. The Targets of Monitoring and the Aggregation results are updated in the certificate at the end of each monitoring cycle (every 1 min) to include the results of the inspection of newly received CAN frames with the monitoring rules. Figure 19 shows the certificate generated by FIVADMI after a two-day evaluation for CAN frames generated using the fuzzer, reflecting the resilience of our implementation. The Aggregation rules in the parent element of the JSON structure in the certificate describe, in natural language, the rule followed to label the certificate with a colour. In the certificate below, 7 ToMs are covered. Four ToMs reflect less than 25% of violations (ToM Injection for IDs 801, 101, 502 and Speed ToM). Two ToM have more than 75% of violations (ToM ID 702 and Coolant ToM). The violation rate of the IAT ToM is 66%. The monitoring ECU is configured to assign equal weight to the ToM to determine the colour of the certificate; given that 4/7 of the ToMs are Green, 1/7 is Amber and 2/7 are red, aggregation result defaults to Amber as the colour code of the certificate as per our Aggregation Rule (reflected in the certificate). We should note that the Aggregation Rule is configurable which enables assigning more weight to a particular Target of Monitoring that has a more critical functionality than the others.

```
{
    "DynamicCertificate ID":       6981,
    "MonitoringResultAggregator":   {
          "Aggregation Time":     "Fri Apr 05 16:13:28 2024 Until Sun Apr 07 16:15:00 2024",
          "Duration":     "1 minute",
          "ToMList":      "Injection Attacks Speed Coolant IAT ",
          "Aggregation Rule":      "The certificate colour in Aggregation Result reflects the colour of 75% of the monitored ToMs. Red
reflects> 75 % alerts, Amber reflects>= 25 % and Green reflects < 25 %. The Default is Amber if 75% of the monitored ToMs do not
have the same colour.",
          "Aggregation Result":   "Amber",
          "TargetofMonitoring(ToM) Injection- All CANIDs":       {
              "ToMType":      "CAN",
              "ToMID":        0,
              "MonitoringRule":       "Injection",
              "Monitoring Evidence Aggregator":       {
                "Aggregation Time":     "Sun Apr 07 16:15:00 2024",
                "Duration":     "1 minute",
                "Aggregation Rule":     "Ratio of Violations/Total",
                "Aggregation Result": "This is the global tally for all CanIDs: Non-violation-Injection 2161107 Violation-Injection: 1520
ratio: 0.000703" }
            },
            "TargetofMonitoring(ToM) Injection":    {
                "ToMType":      "CAN",
                "ToMID":        502,
                "MonitoringRule":       "Injection",
                "Monitoring Evidence Aggregator":       {
                  "Aggregation Time":     "Sun Apr 07 16:15:00 2024",
                    ... .... .... ..... .... ....
                  "Aggregation Result": "This is the global tally 502: Non-violation-Injection 299566 Violation-Injection: 0 ratio: 0.000000"}
            },
            "TargetofMonitoring(ToM) Injection":    {
                "ToMType":      "CAN",
                "ToMID":        702,
                "MonitoringRule":       "Injection",
                "Monitoring Evidence Aggregator":       {
                    "Aggregation Time":     "Sun Apr 07 16:15:00 2024",
                      ... .... .... ..... .... ....
                    "Aggregation Result":   "This is the global tally 702: Non-violation-Injection 0 Violation-Injection: 519 ratio: 1.000000"}
            },
            "TargetofMonitoring(ToM) Speed":        {
                "ToMType":      "CAN",
                "ToMID":        0,
                "MonitoringRule":       "Speed",
                "Monitoring Evidence Aggregator":       {
                      "Aggregation Time":     "Sun Apr 07 16:15:00 2024",
                      .. .. .. .. .. .. .. .. ..
                      "Aggregation Result":   "this is the Violation Ratio for Speed 0.000000"
                }
            },
            "TargetofMonitoring(ToM) Coolant":      {
                "ToMType":      "CAN",
                "ToMID":        0,
                "MonitoringRule":       "Temperature",
                "Monitoring Evidence Aggregator":       {
                      "Aggregation Time":     "Sun Apr 07 16:15:00 2024",
                      .. .. .. .. .. .. .. .. ..
                      "Aggregation Result":   "this is the Violation Ratio for Coolant 1.000000"
                }
            },
            "TargetofMonitoring(ToM) IAT":  {
                "ToMType":      "CAN",
                "ToMID":        0,
                "MonitoringRule":       "Temperature",
                "Monitoring Evidence Aggregator":       {
                      "Aggregation Time":     "Sun Apr 07 16:15:00 2024",
                      .. .. .. .. ... ... .. ... .. .. .. ...
                      "Aggregation Result":   "this is the Violation Ratio for IAT 0.666667" }
                }
        }
}
```


Figure 19. Certificate Produced by FIVADMI for Fuzzer-Generated CAN Frames. (Shortened).

5.3. Evaluation of the Side Channel Monitor

To validate and evaluate the side channel attack monitoring of FIVADMI, we tested the implementation on 10 simulations with a side channel attack targeting the running ECU and 10 simulations when the ECU runs without a side channel attack. Once again to demonstrate the applicability of FIVADMI in realistic settings, we created ECU based on an AUTOSAR implementation [89], that emulates ARM Versatile boards [90] following

the AUTOSAR specification. It should be noted, as our implementation was successfully deployed and executed within a virtual ECU, the framework may have negligible overhead when deployed in realistic settings. Nevertheless, one possible approach would be to deploy FIVADMI in a dedicated ECU to avoid any impact that FIVADMI may have on the regular operation of a vehicle.

Each of these simulations is scheduled for 20 min, and the collected data from the performance counter is studied after the execution with our side channel monitor. Out of the 20 experiments shown in Figure 20, our side channel successfully classified 14 cases (highlighted in green); the wrong classifications (highlighted in red) were equally distributed between false positives and false negatives. This reflects a 70% rate of successful classification of side channel attacks.

	Average Cache Miss	Average Miss Rate	Average Fault Rate	Side Channel Monitor
Test 1- no side channel	3.692959388	0.001058073	1.35911589	Potential CSC
Test 1- with side channel	9.031194208	0.00080766	1.669852028	Potential CSC
Test 2- no side channel	2.603675757	0.003452236	1.038720038	No CSC
Test 2- with side channel	1.647990089	0.000287526	1.093565782	No CSC
Test 3- no side channel	2.324732048	0.000677458	1.41072794	Potential CSC
Test 3- with side channel	3.095794624	0.000782499	3.502029676	No CSC
Test 4- no side channel	2.007809217	0.000322078	1.239943491	Potential CSC
Test 4- with side channel	3.658074418	0.001369777	1.080590695	No CSC
Test 5- no side channel	2.617426274	0.000959456	1.097969442	No CSC
Test 5- with side channel	3.936086973	0.000649898	1.337122628	Potential CSC
Test 6- no side channel	36.35587412	0.008156259	1.050129016	No CSC
Test 6- with side channel	2.920694191	0.000789582	1.836302948	Potential CSC
Test 7- no side channel	4.141978089	0.000786453	1.093210493	No CSC
Test 7- with side channel	2.537116942	0.000405075	1.626305734	Potential CSC
Test 8- no side channel	3.661386161	0.000581728	1.11351984	No CSC
Test 8- with side channel	2.537355182	0.000381015	1.538567296	Potential CSC
Test 9- no side channel	1.435622136	0.000370213	1.008042818	No CSC
Test 9- with side channel	1.988664212	0.000467379	1.335418026	Potential CSC
Test 10- no side channel	1.715506501	0.000329966	1.034488379	No CSC
Test 10- with side channel	3.528514569	0.000938211	2.779346177	Potential CSC

Figure 20. Classification Performed Through SCAM.

Our implementation has a performance advantage over HexPADS [78], the Linux based implementation that SCAM is based on. The performance overhead derived from HexPADS' active defense is not applicable to SCAM. Also, while Hex-PADS' implementation uses a buffer to store the HPC reading [78], Windows Performance Monitor in SCAM uses is configured to update a parseable data structure on the disc to minimize the memory usage of the approach. Both SCAM and HexPADS use a statistical classification approach for side channel attacks; both implementations follow a similar process for finding the thresholds used by the classification algorithm.

6. Conclusions and Outlook

This paper discusses the design, implementation and experimental evaluation of a framework to enhance in-vehicle isolation and resilience focusing on anomalies in the communication layer (e.g., injection attack) and application layer (e.g., anomalies in vehicle speed and coolant temperature). The framework follows an active defence strategy to detect the anomalies in real time. The design and implementation of the framework is based on AUTOSAR open standard to enable scalability, reusability and interoperability across the product lines from different OEMs. AUTOSAR based design is chosen as AUTOSAR is the most prevalent software architecture for ECUs in the automotive industry. However, it should be noted that in our implementation the framework is deployed inside virtual ECU, and it collects CAN frames from the CAN bus. Implementation of FIVADMI does not use any system library from AUTOSAR, in that sense our framework is not closely tied with AUTOSAR and can easily be adapted to non AUTUSAR based automotive software architecture (e.g., POSIX based architecture [91]) that is gaining popularity recently.

The next step to enhance the framework would be to react to intrusions on the in-vehicle network and to recover from attacks. This recovery may be rolling back to a stable

state to overcome an intrusion, or to estimate the stable state by applying different techniques. For example, Kalman filtering can be applied to estimate the correct state of a system, where system state is defined by the variables including the variables (*fluents*) monitored by FIVADMI. Kalman filter would estimate the next states of the system independent of the monitoring done by FIVADMI. If deviation in the value of a monitored variable is detected by FIVADMI, the system can assume the estimated value for that variable in the state predicted by Kalman filter.

Author Contributions: Conceptualization—K.M., A.N., M.P., M.L. and S.A.; methodology—K.M., A.N. and M.L.; software—A.N.; validation—K.M., A.N., M.P., M.L. and S.A.; investigation—K.M., A.N., M.P., M.L. and S.A.; writing—original draft preparation—K.M. and A.N.; writing—K.M. and A.N.; supervision, K.M. All authors have read and agreed to the published version of the manuscript.

Funding: This study was funded by Orange Labs, France. Commissioned Research Agreement No. J06171.

Data Availability Statement: The data that support the experiments of this study are publicly available from https://github.com/cephasax/OBDdatasets (accessed on 04 April 2024). The datasets generated through simulation during the study can be made available from the corresponding author on reasonable request.

Conflicts of Interest: Authors have no conflicts of interest to disclose.

References

1. Faezipour, M.; Nourani, M.; Saeed, A.; Addepalli, S. Progress and challenges in intelligent vehicle area networks. *Commun. ACM* **2012**, *55*, 90–100. [CrossRef]
2. Lokman, S.-F.; Othman, A.T.; Abu-Bakar, M.-H. Intrusion detection system for automotive controller area network (can) bus system: A review. *EURASIP J. Wirel. Commun. Netw.* **2019**, *2019*, 184. [CrossRef]
3. Aliwa, E.; Rana, O.; Perera, C.; Burnap, P. Cyberattacks and countermeasures for in-vehicle networks. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–37. [CrossRef]
4. Boudguiga, A.; Klaudel, W.; Boulanger, A.; Chiron, P. A simple intrusion detection method for controller area network. In Proceedings of the 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 23–27 May 2016; pp. 1–7.
5. Gmiden, M.; Gmiden, M.H.; Trabelsi, H. An intrusion detection method for securing in-vehicle can bus. In Proceedings of the 2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), Sousse, Tunisia, 19–21 December 2016; pp. 176–180.
6. Yang, G.; Tang, C.; Jiang, Z.; Liu, X. Towards interpretable and lightweight intrusion detection for in-vehicle network. In Proceedings of the 2022 4th International Conference on Communications, Information System and Computer Engineering (CISCE), Shenzhen, China, 27–29 May 2022; pp. 179–182.
7. Young, C.; Olufowobi, H.; Bloom, G.; Zambreno, J. Automotive intrusion detection based on constant can message frequencies across vehicle driving modes. In Proceedings of the ACM Workshop on Automotive Cybersecurity, Dallas, TX, USA, 27 March 2019; pp. 9–14.
8. Checkoway, S.; McCoy, D.; Kantor, B.; Anderson, D.; Shacham, H.; Savage, S.; Koscher, K.; Czeskis, A.; Roesner, F.; Kohno, T. Comprehensive experimental analyses of automotive attack surfaces. In Proceedings of the 20th USENIX Security Symposium (USENIX Security 11), San Francisco, CA, USA, 8–12 August 2011.
9. Carsten, P.; Andel, T.R.; Yampolskiy, M.; McDonald, J.T. In-vehicle networks: Attacks, vulnerabilities, and proposed solutions. In Proceedings of the 10th Annual Cyber and Information Security Research Conference, Oak Ridge, TN, USA, 6–8 April 2015; pp. 1–8.
10. Wolf, M.; Weimerskirch, A.; Paar, C. Security in automotive bus systems. In *Workshop on Embedded Security in Cars*; Springer: Bochum, Germany, 2004; pp. 1–13.
11. Hoppe, T.; Kiltz, S.; Dittmann, J. Applying intrusion detection to automotive it-early insights and remaining challenges. *J. Inf. Assur. Secur. (JIAS)* **2009**, *4*, 226–235.
12. Schulze, S.; Pukall, M.; Saake, G.; Hoppe, T.; Dittmann, J. On the need of data management in automotive systems. Datenbanksysteme in Business, Technologie und Web (BTW)–13. In Proceedings of the Fachtagung des GI-Fachbereichs" Datenbanken und Informationssysteme" (DBIS), Munster, Germany, 2–6 March 2009.
13. Tomlinson, A.; Bryans, J.; Shaikh, S.A. Towards viable intrusion detection methods for the automotive controller area network. In Proceedings of the 2nd ACM Computer Science in Cars Symposium, Munich, Germany, 13–14 September 2018; pp. 1–9.
14. Dupont, G.; Hartog, J.D.; Etalle, S.; Lekidis, A. Network intrusion detection systems for in-vehicle network-technical report. *arXiv* **2019**, arXiv:1905.11587.

15. Chattopadhyay, A.; Lam, K.-Y.; Tavva, Y. Autonomous vehicle: Security by design. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 7015–7029. [CrossRef]
16. Daimi, K.; Saed, M.; Bone, S.; Robb, J. Securing vehicle's electronic control units. In Proceedings of the Twelfth International Conference on Networking and Services, Lisbon, Portugal, 26–30 June 2016.
17. David, C.; Fry, S. Automotive Security Best Practices. Recommendations for Security and Privacy in the Era of the Next-Generation Car. 2016. Available online: https://motordna.io/static/stickerlook/images/wp-automotive-security.pdf (accessed on 20 June 2024).
18. Poudel, B.; Munir, A. Design and evaluation of a reconfigurable ecu architecture for secure and dependable automotive cps. *IEEE Trans. Dependable Secur. Comput.* **2018**, *18*, 235–252. [CrossRef]
19. Kang, M.-J.; Kang, J.-W. Intrusion detection system using deep neural network for in-vehicle network security. *PLoS ONE* **2016**, *11*, e0155781. [CrossRef] [PubMed]
20. Seo, E.; Song, H.M.; Kim, H.K. Gids: Gan based intrusion detection system for in-vehicle network. In Proceedings of the 2018 16th Annual Conference on Privacy, Security and Trust (PST), Belfast, Northern Ireland, 28–30 August 2018; pp. 1–6.
21. Sanchez, H.S.; Rotondo, D.; Escobet, T.; Puig, V.; Saludes, J.; Quevedo, J. Detection of replay attacks in cyber-physical systems using a frequency-based signature. *J. Frankl. Inst.* **2019**, *356*, 2798–2824. [CrossRef]
22. Taylor, A.; Japkowicz, N.; Leblanc, S. Frequency-based anomaly detection for the automotive can bus. In Proceedings of the 2015 World Congress on Industrial Control Systems Security (WCICSS), London, UK, 14–16 December 2015; pp. 45–49.
23. Mahbub, K.; Patwary, M.; Nehme, A.; Lacoste, M.; Allio, S.; Rafflé, Y. Towards an Integrated In-Vehicle Isolation and Resilience Framework for Connected Autonomous Vehicles. In Proceedings of the VEHICULAR 2020, Porto, Portugal, 18–22 October 2020.
24. Riley, G. Clips: A Tool for Building Expert Systems. 1999. Available online: https://ntrs.nasa.gov/api/citations/19910014730/downloads/19910014730.pdf (accessed on 20 June 2024).
25. AUTOSAR. Autosar History. 2017. Available online: https://www.autosar.org/about/history/note (accessed on 20 June 2024).
26. OpenEnclave. Open Enclave sdk. 2019. Available online: https://github.com/openenclave/openenclavenote (accessed on 21 June 2024).
27. Alam, M.S.U. Securing Vehicle Electronic Control Unit (ecu) Communications and Stored Data. Master's Thesis, School of Computing, Queen's University, Kingston, ON, Canada, 2018.
28. Hoang, D.; Park, S.; Rhee, J. Traffic-effective architecture for seamless can-based in-vehicle network systems. In Proceedings of the 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 19–21 October 2022; pp. 1612–1614.
29. Mundhenk, P. Security for Automotive Electrical/Electronic (E/E) Architectures. PhD Thesis, Faculty of Electrical Engineering and Information Technology, The Technical University of Munich, Munich, Germany, 2017.
30. ENISI. *Cyber Security and Resilience of Smart Cars: Good Practices and Recommendations*; European Union Agency for Network and Information Security (ENISA): Athens, Greece, 2016.
31. Marquis, V.; Ho, R.; Rainey, W.; Kimpel, M.; Ghiorzi, J.; Cricchi, W.; Bezzo, N. Toward attack-resilient state estimation and control of autonomous cyber-physical systems. In Proceedings of the 2018 Systems and Information Engineering Design Symposium (SIEDS), Charlottesville, VA, USA, 27 April 2018; pp. 70–75.
32. Bazm, M.-M.; Lacoste, M.; Südholt, M.; Menaud, J.-M. Side-channels beyond the cloud edge: New isolation threats and solutions. In Proceedings of the CSNet 2017: 1st Cyber Security in Networking Conference, Rio de Janeiro, Brazil, 18–20 October 2017; pp. 1–8.
33. Bazm, M.-M.; Lacoste, M.; Südholt, M.; Menaud, J.-M. Side Channels in the Cloud: Isolation Challenges, Attacks, and Countermeasures. 2017. hal-01591808. Available online: https://inria.hal.science/hal-01591808 (accessed on 20 June 2024).
34. Jain, S.; Wang, Q.; Arafin, M.T.; Guajardo, J. Probing attacks on physical layer key agreement for automotive controller area networks. In Proceedings of the 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Hong Kong, China, 17–18 December 2018; pp. 7–12.
35. O'Flynn, C.; d'Eon, G. Power analysis and fault attacks against secure can: How safe are your keys? *SAE Int. J. Transp. Cybersecur. Priv.* **2018**, *1*, 3–18. [CrossRef]
36. Mueller, A.; Lothspeich, T. Plug-and-secure communication for can. *CAN Newsl.* **2015**, *4*, 10–14.
37. Kulah, Y.; Dincer, B.; Yilmaz, C.; Savas, E. Spy detector: An approach for detecting side-channel attacks at runtime. *Int. J. Inf. Secur.* **2019**, *18*, 393–422. [CrossRef]
38. A Secure Technology Alliance Mobile Council White Paper. *Trusted Execution Environment (TEE) 101: A primer, Version 1.0, 2018*. Available online: https://www.securetechalliance.org/wp-content/uploads/TEE-101-White-Paper-FINAL2-April-2018.pdf (accessed on 15 May 2024).
39. Jiang, J.; Soriente, C.; Karame, G. On the challenges of detecting side-channel attacks in sgx. In Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses, Limassol, Cyprus, 26–28 October 2022; pp. 86–98.
40. Moghimi, D.; Van Bulck, J.; Heninger, N.; Piessens, F.; Sunar, B. Copycat: Controlled instruction-level attacks on enclaves. In Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), Boston, MA, USA, 12–14 August 2020; pp. 469–486.
41. Sabt, M.; Achemlal, M.; Bouabdallah, A. Trusted execution environment: What it is, and what it is not. In Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, 20–22 August 2015; pp. 57–64.

42. Brasser, F.; Müller, U.; Dmitrienko, A.; Kostiainen, K.; Capkun, S.; Sadeghi, A.-R. Software grand exposure: Sgx cache attacks are practical. In Proceedings of the 11th USENIX Conference on Offensive Technologies (WOOT'17), Berkeley, CA, USA, 16–18 August 2017.
43. Wang, J.; Cheng, Y.; Li, Q.; Jiang, Y. Interface-based side channel attack against intel sgx. 2018. Available online: https://arxiv.org/abs/1811.05378 (accessed on 15 March 2024).
44. Ekberg, J.-E.; Kostiainen, K.; Asokan, N. Trusted execution environments on mobile devices. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 4–8 November 2013; pp. 1497–1498.
45. Ohira, S.; Desta, A.; Arai, I.; Inoue, H.; Fujikawa, K. Normal and malicious sliding windows similarity analysis method for fast and accurate ids against dos attacks on in-vehicle networks. *IEEE Access* **2020**, *8*, 42422–42435. [CrossRef]
46. Stotz, J.P.; Bißmeyer, N.; Kargl, F.; Dietzel, S.; Papadimitratos, P.; Schleiffer, C. Security requirements of vehicle security architecture. Deliverable: D1.1, PRESERVE. 2011. Available online: https://www.preserve-project.eu/www.preserve-project.eu/sites/preserve-project.eu/files/PRESERVE-D1.1-Security%20Requirements%20of%20Vehicle%20Security%20Architecture.pdf (accessed on 15 May 2024).
47. Karopoulos, G.; Kambourakis, G.; Chatzoglou, E.; Hern'andez-Ramos, J.; Kouliaridis, V. Demystifying in-vehicle intrusion detection systems: A survey of surveys and a meta-taxonomy. *Electronics* **2022**, *11*, 1072. [CrossRef]
48. Hamad, M.; Nolte, M.; Prevelakis, V. A framework for policy based secure intra vehicle communication. In Proceedings of the 2017 IEEE Vehicular Networking Conference (VNC), Torino, Italy, 27–29 November 2017; pp. 1–8.
49. Thing, V.L.; Wu, J. Autonomous vehicle security: A taxonomy of attacks and defences. In Proceedings of the 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Chengdu, China, 15–18 December 2016; pp. 164–170.
50. Motruk, B.; Diemer, J.; Buchty, R.; Ernst, R.; Berekovic, M. Idamc: A many-core platform with run-time monitoring for mixed-criticality. In Proceedings of the 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering, Omaha, NE, USA, 25–27 October 2012; pp. 24–31.
51. Jo, W.; Kim, S.; Kim, H.; Shin, Y.; Shon, T. Automatic whitelist generation system for ethernet based in-vehicle network. *Comput. Ind.* **2022**, *142*, 103735. [CrossRef]
52. Kumar, M.; Bhandari, A. A review of detection approaches for distributed denial of service attacks. *Syst. Sci. Control Eng.* **2017**, *5*, 301–320.
53. Dalila Ressi, Riccardo Romanello, Carla Piazza, Sabina Rossi: AI-enhanced blockchain technology: A review of advancements and opportunities. *J. Netw. Comput. Appl.* **2024**, *225*, 103858. [CrossRef]
54. Qayyum, A.; Islam, M.H.; Jamil, M. Taxonomy of statistical based anomaly detection techniques for intrusion detection. In Proceedings of the IEEE Symposium on Emerging Technologies, Islamabad, Pakistan, 18 September 2005; pp. 270–276.
55. Sivasamy, A.A.; Sundan, B. A dynamic intrusion detection system based on multivariate hotelling's t2 statistics approach for network environments. *Sci. World J.* **2015**, *2015*, 850153. [CrossRef] [PubMed]
56. TCG. Trusted Computing Group (tcg). 2015. Available online: https://trustedcomputinggroup.org/wpcontent/uploads/TCGStorageArchitectureCoreSpecv2.01r1.00.pdf (accessed on 15 May 2024).
57. Hund, R.; Willems, C.; Holz, T. Practical timing side channel attacks against kernel space aslr. In Proceedings of the 2013 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 19–22 May 2013.
58. Schwarz, M.; Weiser, S.; Gruss, D.; Maurice, C.; Mangard, S. Malware guard extension: Using sgx to conceal cache attacks. In Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2017, Bonn, Germany, 6–7 July 2017; Volume 10327 of Lecture Notes in Computer Science. Springer: Berlin/Heidelberg, Germany, 2017.
59. Van Bulck, J.; Oswald, D.; Marin, E.; Aldoseri, A.; Garcia, F.D.; Piessens, F. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019.
60. Khandaker, M.R.; Cheng, Y.; Wang, Z.; Wei, T. Coin attacks: On insecurity of enclave untrusted interfaces in sgx. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 16–20 March 2020.
61. Mushtaq, M.; Akram, A.; Bhatti, M.K.; Chaudhry, M.; Lapotre, V.; Gogniat, G. Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters. In Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, Los Angeles, CA, USA, 2 June 2018.
62. Payer, M. Hexpads: A platform to detect "stealth" attacks. In Proceedings of the International Symposium on Engineering Secure Software and Systems, London, UK, 6–8 April 2016.
63. Wang, H.; Sayadi, H.; Rafatirad, S.; Sasan, A.; Homayoun, H. Scarf: Detecting side-channel attacks at real-time using low-level hardware features. In Proceedings of the 2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS), Naples, Italy, 13–15 July 2020.
64. Bazm, M.-M.; Sautereau, T.; Lacoste, M.; Sudholt, M.; Menaud, J.-M. Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters. In Proceedings of the 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC), Barcelona, Spain, 23–26 April 2018.
65. Gamaarachchi, H.; Ganegoda, H. Power analysis based side channel attack. *arXiv* **2018**, arXiv:1801.00932.

66. Palanca, A.; Evenchick, E.; Maggi, F.; Zanero, S. A stealth, selective, link-layer denial-of-service attack against automotive networks. In Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, 6–7 July 2017; Proceedings 14; Springer: Berlin/Heidelberg, Germany, 2017; pp. 185–206.
67. Xie, Y.; Zhou, Y.; Xu, J.; Zhou, J.; Chen, X.; Xiao, F. Cybersecurity protection on in-vehicle networks for distributed automotive cyber-physical systems: State-of-the-art and future challenges. *Softw. Pract. Exp.* **2021**, *51*, 2108–2127. [CrossRef]
68. Binun, A.; Bloch, M.; Dolev, S.; Kahil, M.R.; Menuhin, B.; Yagel, R.; Coupaye, T.; Lacoste, M.; Wailly, A. Self-stabilizing virtual machine hypervisor architecture for resilient cloud. In Proceedings of the 2014 IEEE World Congress on Services, Anchorage, AK, USA, 27 June–2 July 2014; pp. 200–207.
69. Huang, A.; Liu, Y.; Chen, T.; Zhou, Y.; Sun, Q.; Chai, H.; Yang, Q. Starfl: Hybrid federated learning architecture for smart urban computing. *ACM Trans. Intell. Syst. Technol. (TIST)* **2021**, *12*, 1–23. [CrossRef]
70. Thornton, S. Globalplatform Trusted Execution Environment and Trustzone. 2017. Available online: https://www.microcontrollertips.com/embedded-security-brief-arm-trustzone-explained/ (accessed on 12 November 2023).
71. Wolf, M.; Gendrullis, T. Design, implementation, and evaluation of a vehicular hardware security module. In Proceedings of the International Conference on Information Security and Cryptology, Seoul, Republics of Korea, 30 November–2 December 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 302–318.
72. AUTOSAR. Autosar: Layered Software Architecture. 2017. Available online: https://www.autosar.org/fileadmin/standards/R4-3/CP/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf (accessed on 4 April 2024).
73. Paundu, A.W.; Fall, D.; Miyamoto, D.; Kadobayashi, Y. Leveraging Kvm Events To Detect Cache-Based Side Channel Attacks In A Virtualization Environment. *Secur. Commun. Netw.* **2018**, *2018*, 1–18. [CrossRef]
74. Akram, A.; Mushtaq, M.; Bhatti, M.K.; Lapotre, V.; Gogniat, G. Meet the sherlock holmes' of side channel leakage: A survey of cache sca detection techniques. *IEEE Access* **2020**, *8*, 70836–70860. [CrossRef]
75. Zhang, T.; Zhang, Y.; Lee, R.B. Cloudradar: A real-time side-channel attack detection system in clouds. In Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, Crete, Greece, 10–12 September 2018; Springer: Cham, Switzerland, 2016.
76. Intel. Intel Software Guard Extensions Developer Guide. 2016. Available online: https://software.intel.com/enus/node/702976 (accessed on 20 March 2024).
77. Intel. Intel Software Guard Extension sdk for linux os. 2023. Available online: https://download.01.org/intel-sgx/latest/linux-latest/docs/Intel_SGX_SDK_Release_Notes_Linux_2.22_Open_Source.pdf (accessed on 15 March 2024).
78. HexHive. Hexpads, a Host-Based, Performance-Counter-Based Attack Detection System. 2018. Available online: https://github.com/HexHive/HexPADS (accessed on 20 April 2024).
79. Microsoft. Overview of Windows Performance Monitor. 2023. Available online: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/cc749154(v=ws.11) (accessed on 10 March 2024).
80. IAIK. Cache Template Attacks. 2019. Available online: https://github.com/IAIK/cachetemplateattacks (accessed on 15 December 2023).
81. Gruss, D.; Spreitzer, R.; Mangard, S. Cache template attacks: Automating attacks on inclusive last-level caches. In Proceedings of the 24th USENIX Security Symposium (USENIX Security 15), Washington, DC, USA, 12–14 August 2015.
82. NXP, i. MX RT Crossover MCUs. 2024. Available online: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus:IMX-RT-SERIES (accessed on 26 July 2024).
83. Mahbub, K.; Spanoudakis, G. Monitoring ws-agreements: An event calculus–based approach. In *Test and Analysis of Web Services*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 265–306.
84. Shanahan, M. *The Event Calculus Explained*; Springer: Berlin/Heidelberg, Germany, 1999.
85. Doorenbos, R. Production Matching for Large Learning Systems. PhD Thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1995.
86. Barreto, C. Obddatasets. 2019. Available online: https://github.com/cephasax/OBDdatasets (accessed on 4 April 2024).
87. CaringCaribou. Caring Caribou, a Friendly Car Exploration Tool. 2018. Available online: https://github.com/CaringCaribou/caringcaribou (accessed on 8 August 2023).
88. Andreica, T.; Curiac, C.-D.; Jichici, C.; Groza, B. Android Head Units vs. In-Vehicle ECUs: Performance Assessment for Deploying In-Vehicle Intrusion Detection Systems for the CAN Bus. *IEEE Access* **2022**, *10*, 95161–95178. [CrossRef]
89. Parai Wang. Automotive Software and Its Tool-Chain. Available online: https://github.com/autoas/as/ (accessed on 26 July 2024).
90. Arm Versatile Boards. Available online: https://www.qemu.org/docs/master/system/arm/versatile.html (accessed on 26 July 2024).
91. Thoen, F.; Smart, K.; Amringer, N. Accelerating Development of Software-Defined Vehicles with Virtual ECUs. White Paper, Synopsis. Available online: https://www.synopsys.com/content/dam/synopsys/verification/white-papers/virtual-ecu-wp.pdf (accessed on 26 July 2024).