CONNECT

CCAM TRUST & RESILIENCE

D5.3:MEC-enabled Orchestrator, Continuous Authorization, Trust Management and DLT-based Secure Information Exchange

Project number:	101069688
Project acronym:	**CONNECT**
Project title:	Continuous and Efficient Cooperative Trust Management for Resilient CCAM
Project Start Date:	1st September, 2022
Duration:	36 months
Programme:	HORIZON-CL5-2021-D6-01-04
Deliverable Type:	OTHER
Reference Number:	D6-01-04 / D5.3 / 1.00 June 30, 2025
Workpackage:	WP 5
Due Date:	February 28, 2025
Actual Submission Date:	June 30, 2025
Responsible Organisation:	SURREY
Editor:	Christopher Newton
Dissemination Level:	PU – Public
Revision:	1.00 June 30, 2025
Abstract:	This deliverable completes the description of the work of WP5 on the MEC-enabled networking technology to support the CONNECT task-offloading needs, the *CONNECT* Distributed Ledger Technology, the use of Verifiable Credentials and the crypto protocols used to ensure the security and privacy of data in the *CONNECT* eco-system. Trusted task offloading, the *CONNECT* Distributed Ledger Technology and the use of attributes for access control and for signing and encrypting failed attestation data have all been described, implemented and measured. All of these extensions work in tandem with the MEC-enabled Orchestrator (described in earlier Deliverables) managing the secure lifecycle management of CCAM services deployed over the continuum.
Keywords:	Trusted Execution Environment, MEC, TEEGuard, CCAM, Security Architecture, Distributed Ledger, Attribute-based Protocols

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or CINEA. Neither the European Union nor the granting authority can be held responsible for them.

Editor
Christopher Newton (SURREY)

Contributors (ordered according to beneficiary numbers)

Nikos Fotos (UBITECH)
Stefanos Vasileiadis (UBITECH)
Thanassis Giannetsos (UBITECH)
Panagiotis Pantazopoulos (ICCS)
Evangelos Kosmatos (ICCS)
Konstantinos Latanis (SUITE5)
Claudio Ettore Casetti (POLITO)
Marco Rapelli (POLITO)
Riccardo Sisto (POLITO)
Alessio Sacco (POLITO)
Francesca Bassi (IRTSX)
Ines Ben Jemaa (IRTSX)
Matthias Schunter (INTEL)
Christopher Newton (SURREY)

Executive Summary

This deliverable presents the final contributions of WP5 to the *CONNECT* project. WP5 has worked on several disparate, but important components of the *CONNECT* system: trusted task offloading, the *CONNECT* Distributed Ledger Technology (DLT) and the use of attributes both for access and for protecting the security and privacy of data within the system.

Trusted task offloading allows compute intensive tasks to be outsourced to the MEC in a trustworthy way. This requires the use of Trusted Execution Environments on the vehicle and the MEC and secure connections between them, together with the necessary trust assesments that are carried out before executing the offloading. A twist on the 5GAA ”Line Change Warning” use case was used to provide a focus for the work and a use case involving video analytics for detecting slow-moving vehicles was developed and assessed. This demonstrated the feasibility of secure offloading to the MEC infrastructure and it will be further showcased in the WP6 slow-moving traffic detection use case.

The CONNECT DLT supports: the deployment of trust models and required trust levels for MEC and CCAM services and the secure storage of failed attestation data. The trust models are supplied as requested by a TAF to support their trust assessments, while the failed attestation data can be retrieved and used for diagnosis and the identification of any vulnerabilities. This diagnosis can then lead to updated trust models and RTLs, which then propagated to the TAFs. The *CONNECT* DLT that was described in D5.2, has been developed and benchmarked and the workflows necessary for trust model deployment and the secure storage and retrieval of failed attestation evidence designed.

The use of attributes allows flexible and fine-grained control of: access to the *CONNECT* DLT and the security and privacy of the data stored there. Access control for the *CONNECT* DLT was implemented using Attribute-Based Access Control (ABAC), while a new protocol for Attribute-Based Signcryption (ABSC) was designed and implemented. This ABSC protocol ensures data confidentiality and authenticity, it was benchmarked stand-alone, but will be implemented and benchmarked as part of the end-to-end testing in WP6.

In conclusion, the *CONNECT* technologies outlined here are ready for integration into real-world CCAM scenarios and this will form part of the remaining work to be done in WP6.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

This deliverable concludes the work on one of the core enablers of the CONNECT framework, one that allows the characterisation of trust in connected and automated vehicles based on data (trustworthiness evidence) from external sources. *Besides verifiability, gathering such evidence necessitates an auditable management mechanism capturing the execution of data sharing agreements in a secure and privacy-preserving manner*. While the former is been achieved through the **CONNECT Trusted Computing Base** and the exposed attestation capabilities [12] (enabling the *zero-trust paradigm* where security of any CCAM actor is bootstrapped through the secure communication and verification of appropriate trustworthiness claims), the latter is realised through the integration of a **Blockchain-based infrastructure mounted with all the necessary security mechanisms (e.g., Access Control and Attribute-based SignCryption) and data models (i.e., mediated through smart contracts)** for managing the characteristics and inherent structures of the data of interest. Essentially, the trustworthiness evidence, produced by a trust source (e.g., the attestation process as an output of the CONNECT Attestation & Integrity Verification TS), which captures an inconsistent state of the target CCAM actor, thus, depicting an incident of risk based on which the trust level will need to be updated.

To this end, D5.3 provides the final design specifications and implementation details of all security controls and data model abstractions that formulate the **CONNECT Distributed Ledger Technology** (CONNECT DLT) manifesting the management of all trust-related information surrounding the operation of the Trust Assessment Framework (TAF): from hosting and enforcing trust policies, comprising the necessary trust model templates capturing the type of trust relationships that need to be assessed, to the secure management of produced trustworthiness reports and evidence based on which a (*negative*) trust decision was reached; i.e., where the Actual Trust Level is deemed lower against the Required Trust Level that characterises the safety-critical operation of the target system. All the above is safeguarded through CONNECT's *crypto agility layer* introducing a novel **Attribute-based SignCryption** scheme as well as a *2-layer* **Access Control** mechanism for controlling that such sensitive data is only processed by authenticated and authorised entities exhibiting the necessary set of attributes - adopting also the identity management approaches and structures currently been proposed as part of the Self-Sovereign Identity (SSI) ecosystem [6]. The whole DLT infrastructure has been integrated and tested, and detailed micro-benchmarking has been performing showcasing that such data sharing agreement protocols can support the TAF operation and further elevate its scalability. This is a pre-cursor to additional tests of the overarching DLT pipeline (with the integration of also the crypto agility layer) to be performed as part of the final round of experiments (documented in the context of D6.2).

After bootstrapping and establishing trust across vehicles, the unlocked data siloes can be ex-

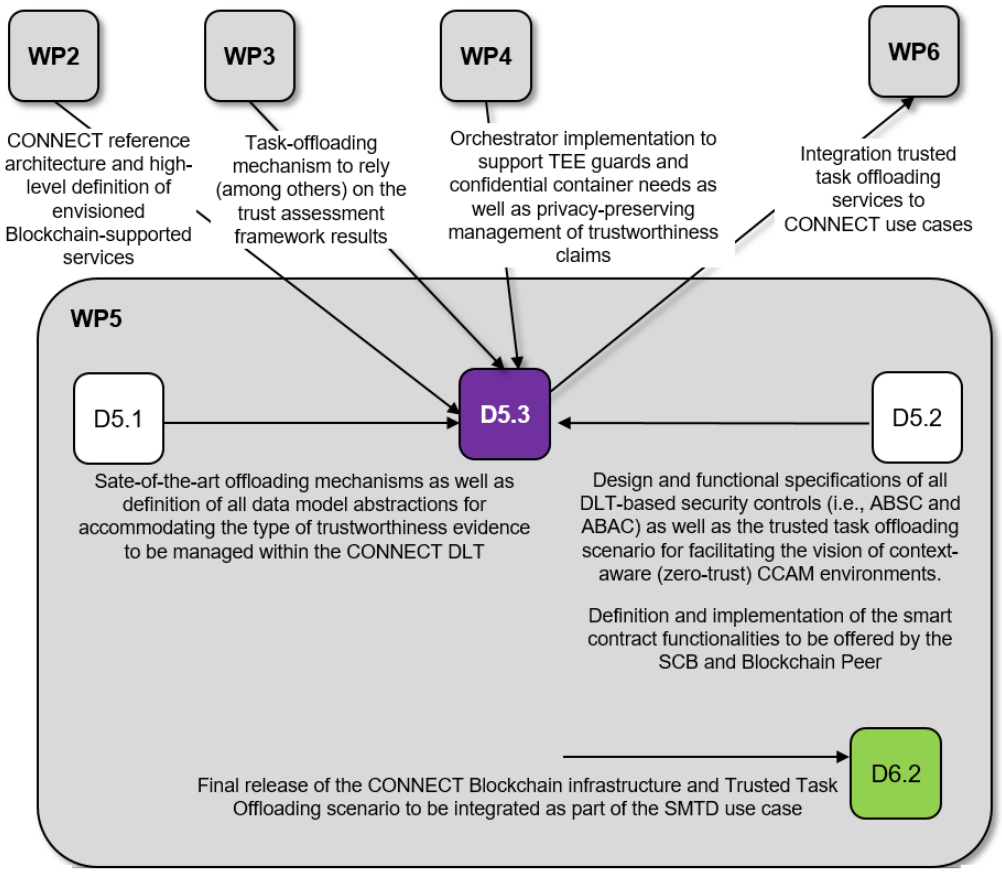

Figure 1.1: Relation of D5.3 with other WPs and Deliverables.

tended from the standalone vehicle domain to safe and secure solutions distributed from vehicles to MEC and Cloud Facilities, thus, facilitating the vision of next-generation trustworthy CCAM environments. In this context, CONNECT has already shown how 5G can act as technological enabler for **fast offloading, through the secure network acceleration offering, distributed processing and low-latency networking [10, 7]**. This "*trusted task offloading*" scenario is been described, designed and scrutinised in the context of this deliverable (and as part of the *Slow-Moving Traffic Detection* use case [8]). The scenario, motivated by a relevant 5GAA automotive use-case [3], serves as a basis for evaluating the way CONNECT can outsource automotive tasks and calculations in a trustworthy manner to the backend MEC infrastructure. The corresponding application is a video analytics service that includes inference tasks (recognizing frames with slow moving vehicles ahead) to be carried out at the infrastructure. Real-world experimentation results for a static OBU, linked over a 5G experimental test-bed to the MEC-side, are detailed and mark one of the very few relevant assessments; this highlights the performance and applicability of the CONNECT secure offloading protocols to enable the establishment of trust relationships and trust calculations in next generation CCAM.

1.1 Relationship with other CONNECT Deliverables

The deliverable presents the work that acts as the solid basis for the finalization of the WP5 implementation work. As such, it draws on CONNECT concepts (i.e., architecture, use-cases formulation) finalised in WP2 as well as on all trust-related mechanisms (designed and imple-

mented in the context of WP3 and WP4) enabling both a) cyber-secure data sharing between data sources in the CCAM ecosystem that had no or insufficient pre-existing trust relationship, and b) outsourcing tasks to the MEC and cloud in a trustworthy way.

In this context, Figure 1.1 depicts the direct and indirect relationships to other tasks and Work Packages (WPs). First, it distils the mechanisms (specified in the context of D5.1 and D5.2 as it pertains to the Blockchain-based architecture and trusted task offloading mechanisms) based on which the overarching concept of *context-aware continuous trust assessment in zero-trust CCAM, extending the stand-alone vehicle domain to safe and security solutions distributed from vehicles to MEC and Cloud facilities,* could be materialised. This vision is enabled by the vehicle's communication with other entities formulating the Vehicle-to-everything (V2X) landscape. The resulting "*CCAM continuum*" paradigm seeks to seamlessly and securely combine the available hardware and software (from the in-vehicle sensors to the MEC virtualised infrastructure) to support the deployment and operation of certified CCAM functions. Whereas the collective consideration (treatment) of the continuum resources presents opportunities for increased performance and a higher degree of automation, the individual Software (SW) and Hardware (HW) infrastructure may exhibit diverse yet dynamic trust states; and when those infrastructures are brought together to make-up the continuum strong trust assessment mechanisms need to also be deployed for asserting to the correct state of all these functional assets. Thus, D5.3 manifested on these designs proceeding with the final implementation and detailed benchmarking as evidence on the applicability of such mechanisms in safety-critical applications.

In addition to its methodological and micro-benchmarking contributions, this work is situated within the wider landscape of ongoing experimentation activities of the overall CONNECT framework towards the evaluation of the end-to-end security controls and trust assessment mechanisms in realistic operational environments. Especially, the "*trusted task offloading*" approach outlined in this deliverable responds to the emerging expectations from both research and regulatory communities for structured, interpretable, and evidence-based trust collaboration and reasoning of the entire CCAM continuum and will be further evaluated in the context of the *Slow-Moving Traffic Detection* use case (D6.2).

1.2 Organization of this Deliverable

This deliverable is organised as follows:

Chapter 2: Trusted Offloading - Implementation and Pipeline Integration Describes the implementation of trusted offloading using a scenario based on the 5GAA "Lane Change Warning" use case. It details the hardware/software setup, including the use of Trusted Execution Environments (TEEs), encrypted communication, and time synchronization between On-Board Units (OBUs) and the cloud capabilities of the Multi-Access Edge Computing (MEC) Architecture. The chapter also introduces a trust model to support secure task delegation.

Chapter 3: Trusted Offloading - Testing and Benchmarking Presents performance benchmarks of the trusted offloading pipeline. It evaluates latency and synchronization accuracy, confirming the viability of the CONNECT mechanisms in real-world conditions.

Chapter 4: The CONNECT DLT - System Overview Explains the architecture and components of the CONNECT DLT, including Blockchain Peers, Chaincode Services, Security Context Broker (SCB), and Off-Chain Storage. It details five data workflows supporting trust model deployment, attestation, and vulnerability management for MEC and CCAM services.

Chapter 5: The CONNECT DLT - Access Protocols Focuses on access control mechanisms using Attribute-Based Access Control (ABAC) and Attribute-Based Signcryption (ABSC). It explains how entities are granted access based on verifiable attributes and how data confidentiality and authenticity are ensured through cryptographic protocols. Benchmarking results show the scalability and efficiency of these mechanisms.

Chapter 6: Implementation and Testing of the CONNECT DLT Architecture Covers the smart contract structures used in the DLT for storing trust policies and attestation reports. It presents detailed benchmarking results for data operations under varying conditions, including batch processing and concurrent access, highlighting the system's robustness and scalability.

Chapter 7: Conclusion Summarises the readiness of the CONNECT technologies for integration into real-world CCAM scenarios. It emphasises the successful implementation of trusted offloading, secure DLT-based data handling, and advanced access control mechanisms.

Chapter 2

Trusted Offloading - Implementation and Pipeline Integration

2.1 Secure Offloading Implementation and Pipeline Integration

2.1.1 The CONNECT Trusted Offloading Concept

Task-offloading allows shifting computation tasks from (typically) low-capability in-vehicle processing units to the powerful (edge) infrastructure. Such operations provide promising ways to offload non safety-critical tasks from the vehicles to accelerate processing. Task-offloading has been identified as an important 'application' for the CONNECT concept (see D2.1).

While the relevant offloading research is broad (see D5.1), it has hardly addressed (and experimentally explored) how to effectively combine the networking technology and distributed trust-anchoring capabilities [22]. This combination, referred to herein as "trusted offloading" is mostly needed because of the emerging connected vehicles paradigm that has strong requirements for trustworthiness.

To serve the purposes of a feasible (in line with the project resources) yet insightful offloading instance that will enable the derivation of real-world evaluation results, we have devised a scenario that is "compatible" with the main objectives of the SMTD use-case (see D6.1). The scenario resembles the 5GAA-introduced "Lane Change Warning" use case (5GAA C-V2X Use Cases and Service Level Requirements Volume I, Oct. 2020 [4]) whereby the vehicle of interest is alerted about a leading/lagging remote vehicle. In our case, the identification of the slow-moving (leading) vehicle is offloaded (and carried-out by dedicated ML software) at the MEC. The necessary computation corresponds to a non-safety critical task that is deemed appropriate to be shifted outside the vehicle, while still employing data captured by the vehicle's sensors.

The proposed offload concept (depicted in Figure 2.1) requires the necessary functionality to be installed and operated at both ends i.e., the vehicle- and the MEC- side. The corresponding pipeline seeks to collect the video signal (i.e., frames) from the vehicle camera capturing the scene of the road-ahead. After the required processing at the vehicle OBU, the relevant data are transmitted (through cellular) to the MEC-hosted applications that leveraging ML capabilities can reach relevant video analytics decisions (i.e., detection/recognition of objects appearing in the vehicle's front-scene).

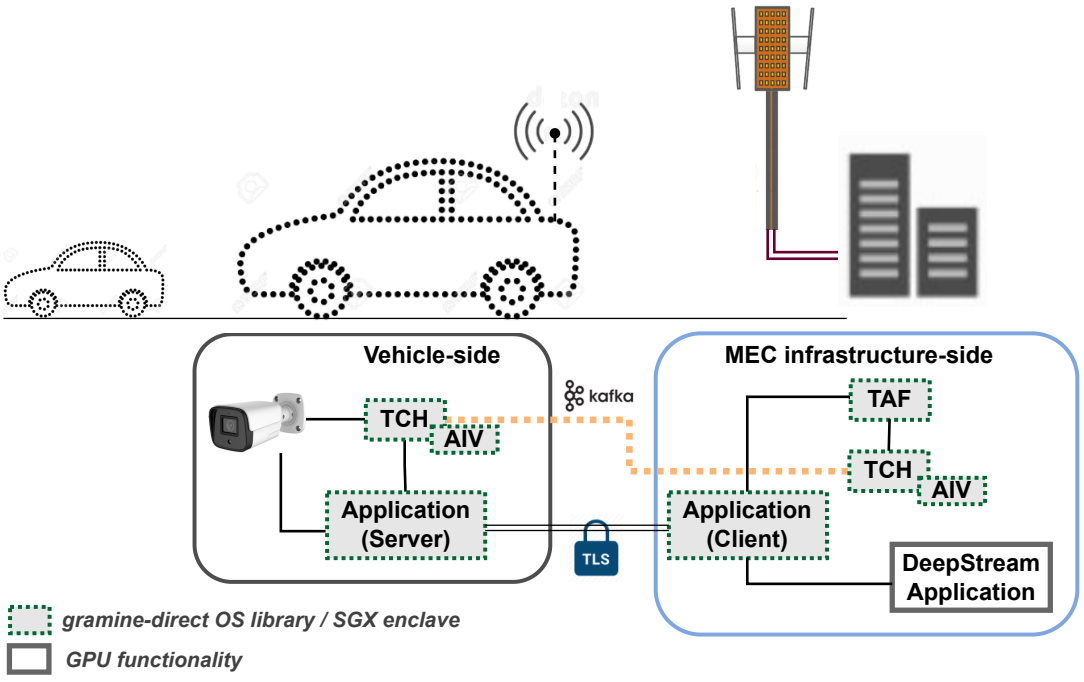

Figure 2.1: Trusted task offloading: vehicle- and infrastructure- side CONNECT software entities

What the CONNECT technology offers to the process amounts to a trusted-computing base which:

- provides an appropriate *trusted execution environment*, whether this is the OS library (*gramine-direct*) or the HW-based enclave (*SGX enclave*), whereby all relevant software applications are running. The exception to this rule is the MEC-hosted inference software which typically relates to GPU capabilities and is assumed to be trusted (see DeepStream application in Figure 2.1).
- instantiates mechanisms to extract relevant *attestation data* and produces claims (see details in [10]) about the operation of the vehicle OBU to serve as input to the CONNECT trust assessment.
- *assesses the trustworthiness* of the operation of the vehicle OBU software by receiving and processing the claims that reach the MEC, where the trust assessment framework (TAF) resides, through encrypted channels.

The CONNECT trusted computing base offers all required functionality to essentially enable the offloading of computation tasks under a zero-trust hypothesis; which is what the 'trusted offloading' seeks to achieve. The basic functionality of the modules involved in the trusted offloading operation is described in Table 2.1.

In what follows, the realization of a static offloading pipeline integrated into the ICCS 5G test-bed is described. The relevant pipeline will act as a basis for the in-lab experimental evaluation of the CONNECT *trusted* offloading capability. A similar setup will be integrated into the vehicle and evaluated/demonstrated in test track conditions.

Table 2.1: Functionality of the modules of the CONNECT trusted offloading pipeline

Module name	Description of functionality
Attestation and Integrity Verification (AIV)	A software module that manages the attestation process (i.e., attestation evidence collection) at the vehicle OBU. A push type of operation is used to feed the TAF which receives initial attestation evidence and is then updated in line with evidence changes.
Trustworthiness Claims Handler (TCH)	A software module that shares all trust-related information between the vehicle OBU and the MEC in a privacy-preserving manner. In the offloading case, the relevant data consists of Trustworthiness Claims (TCs) produced (by the AIV) so as to provide trustworthiness evidence in a measurable (i.e., the comparison between the actual and the required trust level – see relevant definitions in [9]) and verifiable manner. Claims in the offloading case include evidence on the OBU's software properties: secure boot, configuration integrity and access control [9].
Trust Assessment Framework (TAF)	A software framework which, given the offloading trust model (see Section 2.1.5) running at the MEC, is able to evaluate the trust sources of AIV data as trust-worthiness evidence for the OBU operation during offloading.
Application (server)	A custom application running at the vehicle OBU which is responsible to read the camera feed, transcode the video stream and deliver the generated video frames over an encrypted (TLS) mobile channel to the application counterpart, residing at the MEC. The aforementioned communication is realised through a web socket in order to be executed in a TEE environment. In the basic mode is responsible also to communicate with the TCH (through a data broker) and deliver the trustworthiness claims. In piggybacking mode (explained later) the Application (server) embeds the trustworthiness claims inside the video frames before sending data to the MEC.
Application (client)	The offloading application counterpart hosted at the MEC. It is mainly responsible for receiving the video data send by the vehicle OBU, appropriately decode them and prioritise them as a buffered input to the DeepStream application. It also communicates with the TAF (through a data broker) in order to ensure that a specific data source is trusted. In piggybacking mode it is also responsible to retrieve the trustworthiness claims embedded in the video frames and deliver them to the local TCH.
Deepstream Application	DeepStream is a powerful streaming analytics toolkit that processes raw visual data to perform inference tasks (e.g., for object recognition, detection, classification etc.) leveraging deep learning models and frameworks (like TensorFlow, PyTorch) and NVIDIA GPUs. Inference conclusions on the detection of humans and vehicles in the frames captured by the ego-vehicle camera, are derived by the application at the MEC.

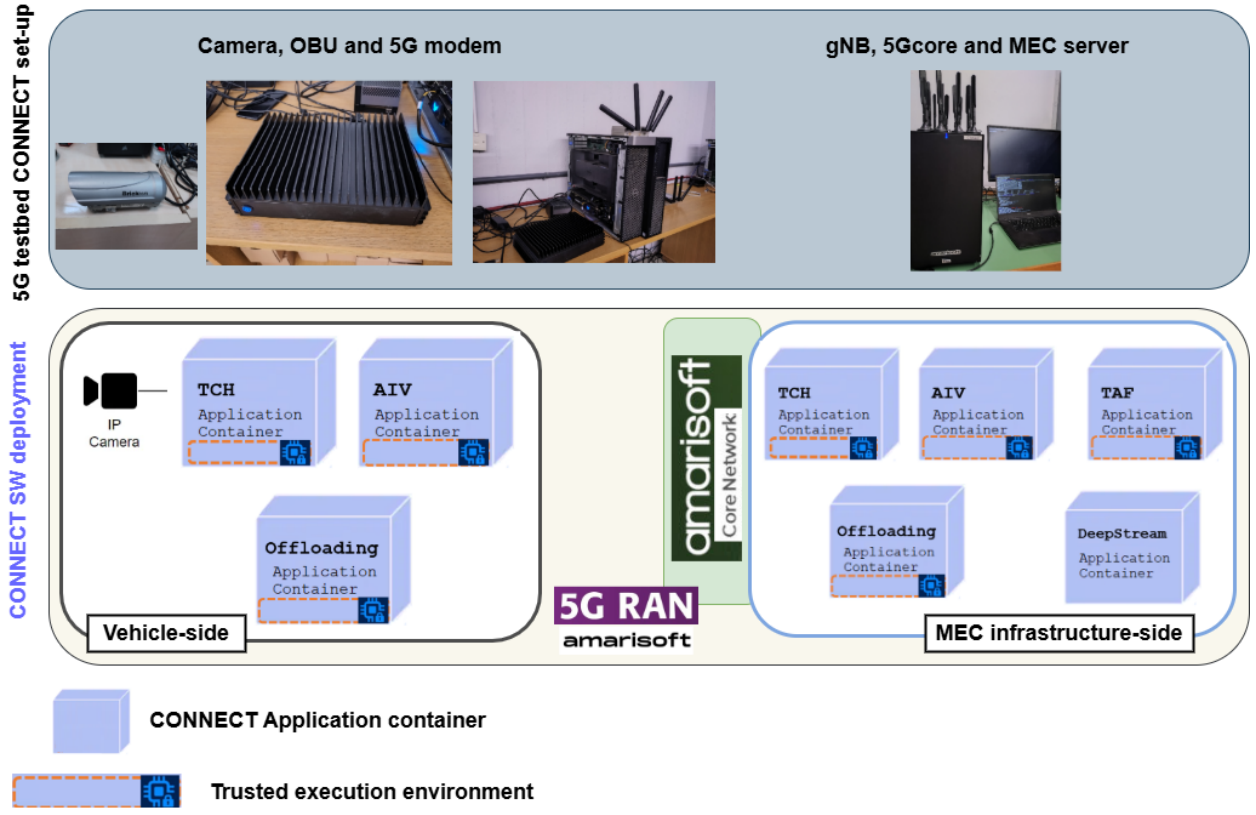

Figure 2.2: The lab set-up of the offloading pipeline and the corresponding deployed software modules

2.1.2 Realization of the (Static) Task-offloading Pipeline: Hardware/Software set-up

The offloading concept and general functionality of the modules has been realised in the testbed of partner ICCS, using real-world devices. The upper part of Figure 2.2 depicts the hardware modules employed to realise the offloading pipeline in the lab. The whole set-up seeks to provide the basis for a static analysis and measurement of the offloading pipeline (see Section 3). Details of the hardware used and the proprietary 5G system technology are shown in Figure 2.3.

On the vehicle side, a server equipped with Intel SGX acts as the on-board unit (OBU), hosting all CONNECT applications. It connects to the 5G network via an external 5G modem, enabling communication with the base station. The MEC side is emulated by an SGX-enabled laptop positioned behind a commercial 5G system provided by Amarisoft. This setup uses an indoor antenna array to access the 5G radio network (see Figure 2.2, top right). The full CONNECT offloading pipeline has been deployed and tested on this setup.

Regarding the application software components realising the offloading in a TEE enviroment there are: a) the Application Server; b) the Application Client and; c) the DeepStream Application.

The Application Server located in the vehicle is implemented in Python and it is responsible to read the video stream from the camera, transcode the video flow and deliver the video frames to the Application Client. In detail, the Application Server receives the camera feed using the RTSP protocol [23]. The video stream is further transcoded using the FFmpeg tool [24] toward an H264 video stream. The H264 video frames when generated are buffered into fixed chunks and further delivered to the Application Client (in the MEC) using a web socket connection. It is important to

emphasise that this connection is further secured through TLS (see paragraph 3.1.2), therefore before exchanging any actual data a TLS handshake between the Application Server and Application Client takes place. In a TEE environment, the Application Server is communicated with the local TCH through a data broker in order to deliver the claims. The frequency of claims exchange between Application Server and TCH is configurable and can be varied between 30ms to a few seconds. In addition, the claims (instead of being delivered to the local TCH) are embedded into the video frames and piggybacked to the Application Client.

The Application Client located at the MEC, is implemented in Python. It is responsible to receive the video frames from the Application Server through the web socket (as explained above), decapsulates from the video frame the trustworthiness claims (if they exist) and further forward the video frames towards the DeepStream Application through a second (different) web socket. Therefore, the Application Client acts as a web socket client toward the Application Server and as a web socket server toward the DeepStream Application. In a TEE environment, the Application Client communicates with the TAF (through a data broker). The Application Client registered with the TAF and further subscribed to it in order to receive any information related with the trustworthiness of the data source. This information is used by the Application Client is order to decide if the received video frames are coming from a trusted entity. In case the entity is not trusted, the video frame is dropped and not delivered to the DeepStream Application. In addition, in the piggybacking mode, the Application Client is responsible to decapsulate the claims from the video frames and deliver the claims toward the local TCH (through a data broker).

The DeepStream Application, implemented in Python, uses Ultralitics as the main library for executing inference. The DeepStream Application is responsible to receive the video frames sent by the Application Client through the web socket, stores the video frames into a buffer and executes inference with a configurable frequency (from a few frames up to hundreds or frames). The DeepStream Application, assumed to be trusted, is not executed in a TEE environment therefore does not communicate with any TCH or TAF instances.

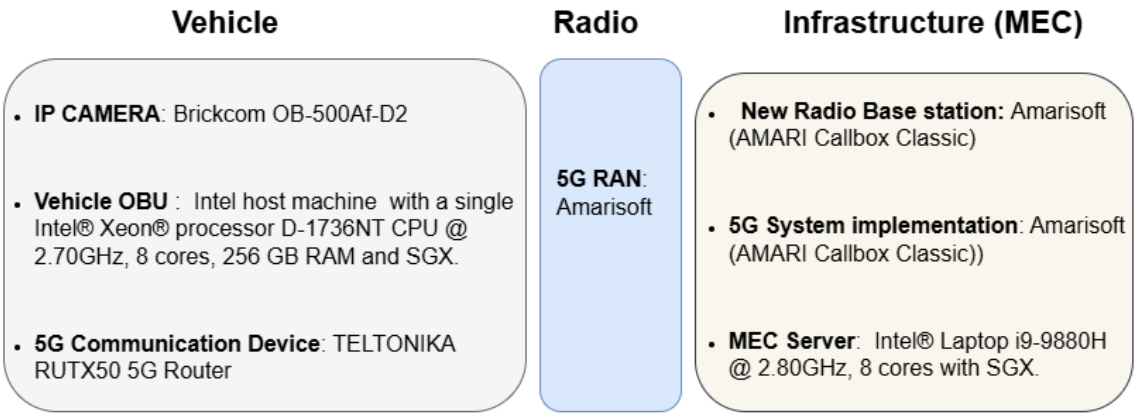

Figure 2.3: CONNECT lab setup: hardware and 5G modules

Preparing the applications to run in isolated environments - i.e., enclaves - we have considered the adoption of Gramine, a guest operating system which allows the seamless execution of applications inside SGX enclaves. Part of the ”graminization” process of an application includes the description of all the software dependencies and configuration properties that characterise the application to be executed within an SGX enclave. This description is encoded in the form of a Gramine Manifest which lists the dependencies and configurations in the form of key-value pairs. Consequently, this allows the sealing process which includes integrity checks issued by

the authoring entity of an application. During the launch of the trusted execution environment, Gramine checks at run-time that the application is booting with the expected set of libraries and configuration parameters.

For the graminization of the applications running both on the vehicle OBU and on the MEC, we have considered the adoption of existing gramine images for Docker environments. This facilitates the seamless management (e.g., spawning, replicating) of enclaves through the docker (or Kubernetes) runtime environment. Specifically, for the realization of the task offloading pipeline we rely on the "gramineproject/gramine:1.7-focal" image available on Docker Hub. Having it as a base image, we derive a final image per application that needs to be running inside enclaves. Inside each final image we place the corresponding executable application including its dependencies, together with the associated Gramine Manifest file. Figures 2.1 and 2.2 present the Gramine Manifest description for the task-offloading application on the vehicle OBU and on the MEC, respectively.

As can be seen in the manifests below, we allocate 4 GB of SGX-protected RAM (i.e., Enclave Page Cache) during the launch of both task offloading applications. In addition to that, from the manifests we observe that both applications require as a software dependency the FFmpeg library [24]. This is important due to the required processing (i.e., encoding//decoding and compression) of the video frames that are (captured by the camera and) sent from the vehicle OBU to the MEC application.

```
libos.entrypoint = "/app/run_client"
loader.log_level = "{{ log_level }}"

loader.env.LD_LIBRARY_PATH = "/lib:/app/_internal/lib"
loader.entrypoint = "file:{{ gramine.libos }}"

fs.mounts = [
  { path = "/lib", uri = "file:{{ gramine.runtimedir() }}" },
  { path = "/app", uri = "file:/app/dist/run_client" },
  { path = "/etc/hosts", uri = "file:/etc/hosts" },
  { path = "/usr/bin/ffmpeg", uri = "file:/usr/bin/ffmpeg" }
]

sgx.debug = true
sgx.enclave_size = "4G"
sgx.max_threads = {{ '1' if env.get('EDMM', '0') == '1' else '64' }}

sgx.trusted_files = [
  "file:/app/dist/run_client/",
  "file:{{ gramine.libos }}",
  "file:{{ gramine.runtimedir() }}/",
  "file:/etc/hosts",
  "file:/usr/bin/ffmpeg"
]

sgx.allowed_files = [
  "file:logs/ws_client.log"
]
```

Listing 2.1: Gramine Manifest Template to run the data collection logic on the vehicle OBU side (NUC) inside a TEE.

```
libos.entrypoint = "/app/run_server"
loader.log_level = "{{ log_level }}"
```

```
3
4 loader.env.LD_LIBRARY_PATH = "/lib:/usr/lib:{{ arch_libdir }}:/usr/{{
    arch_libdir }}:/app/_internal/lib"
5 loader.entrypoint = "file:{{ gramine.libos }}"
6 loader.argv = ["/app/run_server"]
7
8 fs.mounts = [
9   { path = "/lib", uri = "file:{{ gramine.runtimedir() }}" },
10  { path = "/app", uri = "file:/app/dist/run_server" },
11  { path = "/etc/hosts", uri = "file:/etc/hosts" },
12  { path = "/usr/bin/ffmpeg", uri = "file:/usr/bin/ffmpeg" },
13  { path = "/usr/lib", uri = "file:/usr/lib" },
14  { path = "/usr/{{ arch_libdir }}", uri = "file:/usr/{{ arch_libdir }}" },
15 ]
16
17 sgx.debug = true
18 sgx.enclave_size = "4G"
19 sgx.max_threads = {{ '1' if env.get('EDMM', '0') == '1' else '64' }}
20
21 sgx.trusted_files = [
22   "file:/app/dist/run_server/",
23   "file:{{ gramine.libos }}",
24   "file:{{ gramine.runtimedir() }}/",
25   "file:/etc/hosts",
26   "file:/usr/bin/ffmpeg",
27   "file:{{ arch_libdir }}/",
28   "file:/usr/{{ arch_libdir }}/",
29 ]
30
31 sgx.allowed_files = [
32   "file:logs/ws_server.log",
33 ]
```

Listing 2.2: Gramine Manifest Template to run the data processing logic on the MEC side inside a TEE.

2.1.3 Operation of the Task-offloading Pipeline: Sequence of Actions

In this section, we describe the sequence of actions (i.e., communications) that take place during the trusted offloading process between the software entities presented in Section 2.1.1. Notably, this sequence is the same in case of the herein described static (lab) evaluation and the real-world deployment and test-track evaluation that will follow (see D6.2). We recognise two modes of operation as regards to the communication of the trustworthiness claims from the vehicle OBU to the MEC application. The first realises the separate transmission of claims and video data; it uses two different channels[1] to reach the MEC, one for the claims and one for the video frames. An interesting alternative is to send the claims inside the video frame messages, typically in the form of an extended header; a networking technique broadly known as piggybacking [16].

A common phase to the two modes is the bootstrapping process, which is essentially triggered by the MEC. The application at the MEC informs vehicle(s) (6.1.1 message Initialise Offloading)

[1]This case is reflected in Figure 2.1 where a Kafka pub/sub system has been deployed to accommodate the communication of trustworthiness claims from the TCH application in the vehicle to the one residing at the MEC. Video frames on the other hand, are sent to the MEC application through a socket connection over an encrypted cellular channel.

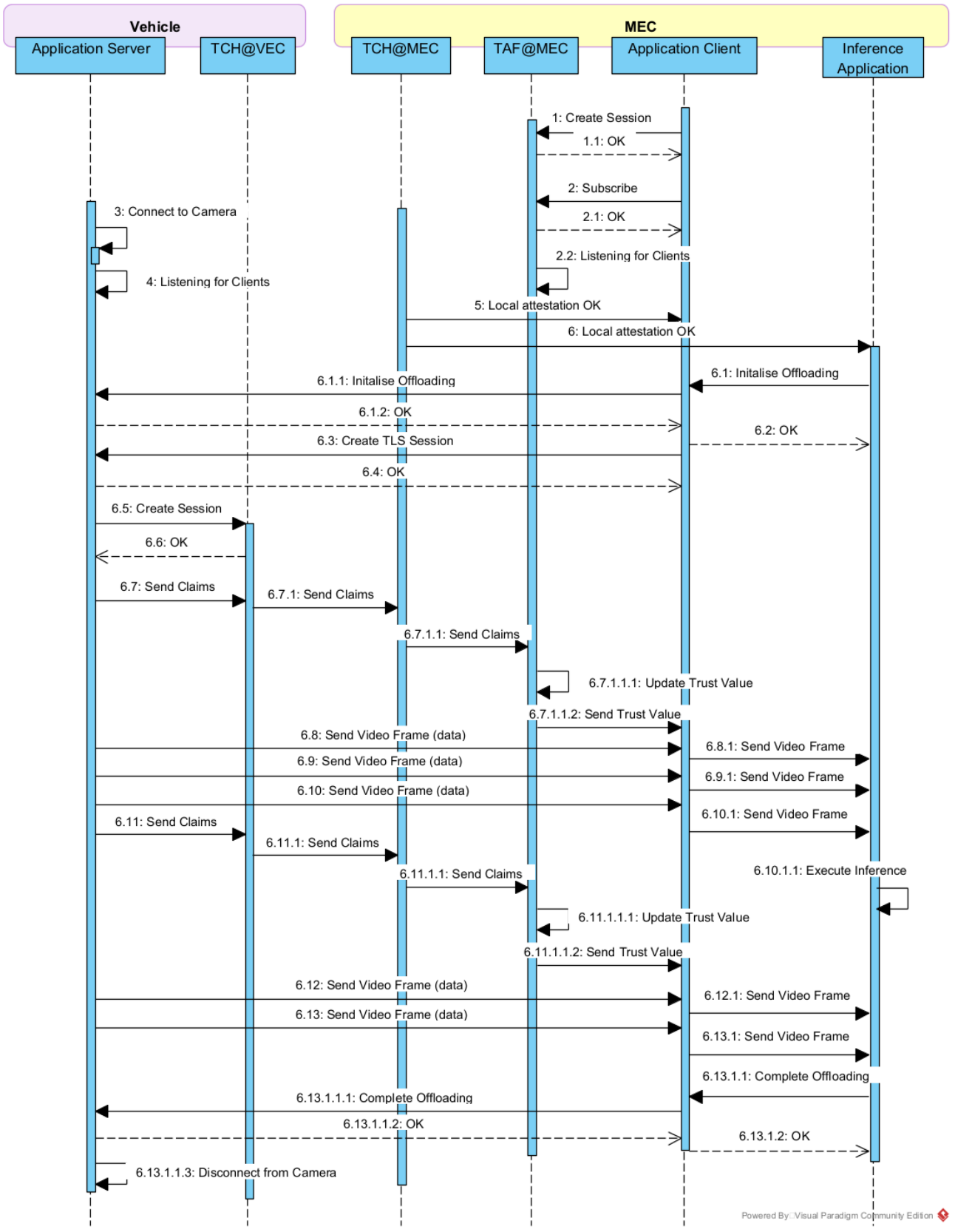

Figure 2.4: Sequence diagram of the operation in which video frames and claims are sent separately

moving within the MEC range that the infrastructure and the hosted applications are capable to receive offloaded tasks (data).

In Figure 2.4, the two 'parallel' communication links from the vehicle to the MEC applications, are depicted. The vehicle TCH instance sends claims (message 6.7.1 Send Claims) directly to the TCH counterpart at the MEC while the offloading application sends video data (6.8 Send Video Frame) to the corresponding MEC application. The MEC TCH forwards the received claims to the local TAF, which subsequently provides the trust assessment result (Send Trust Values) to the application (client). Inference based on the received video data is carried out at the DeepStream application as long as the TAF outcome suggests that the vehicle OBU operation (which provides the video data) is trustworthy.

In Figure 2.5, a single communication link from the vehicle to the MEC appears (6.5 Send Video Frames). Those video frames include the video data together with the trustworthiness claims that have been produced by the vehicle TCH and sent to the vehicle application to be included in the frames. When claims are delivered to the Application Client (on the MEC), the corresponding software processes the data to extract a) the claims that are forwarded to the TAF through the local TCH (6.5.2 Send Claims) and b) the video frames that serve as input to the DeepStream software (6.6.1 Send Video Frames).

On a final note, we comment on the relationship between a video frame and the computed trustworthiness claim. Essentially, the claims refer to the (nominal) operation of the software running at the vehicle OBU rather than offering integrity guarantees for each video frame. As such, we do not assume any direct link between the offloaded video data (at a given time) with the corresponding (computed/collected) claims. Along this line, the claims may be transmitted with different frequencies (in relation to the number of transmitted frames) to support an offloading process which halts if any claim, when fed to the TAF, results in an unfavourable trust assessment. In both modes described above and realised for the derivation of results, the transmission frequency of the claims is a configuration parameter. This parameter affects the traffic overhead potentially added by the offloading pipeline and its evaluation is presented in Section 3.2.

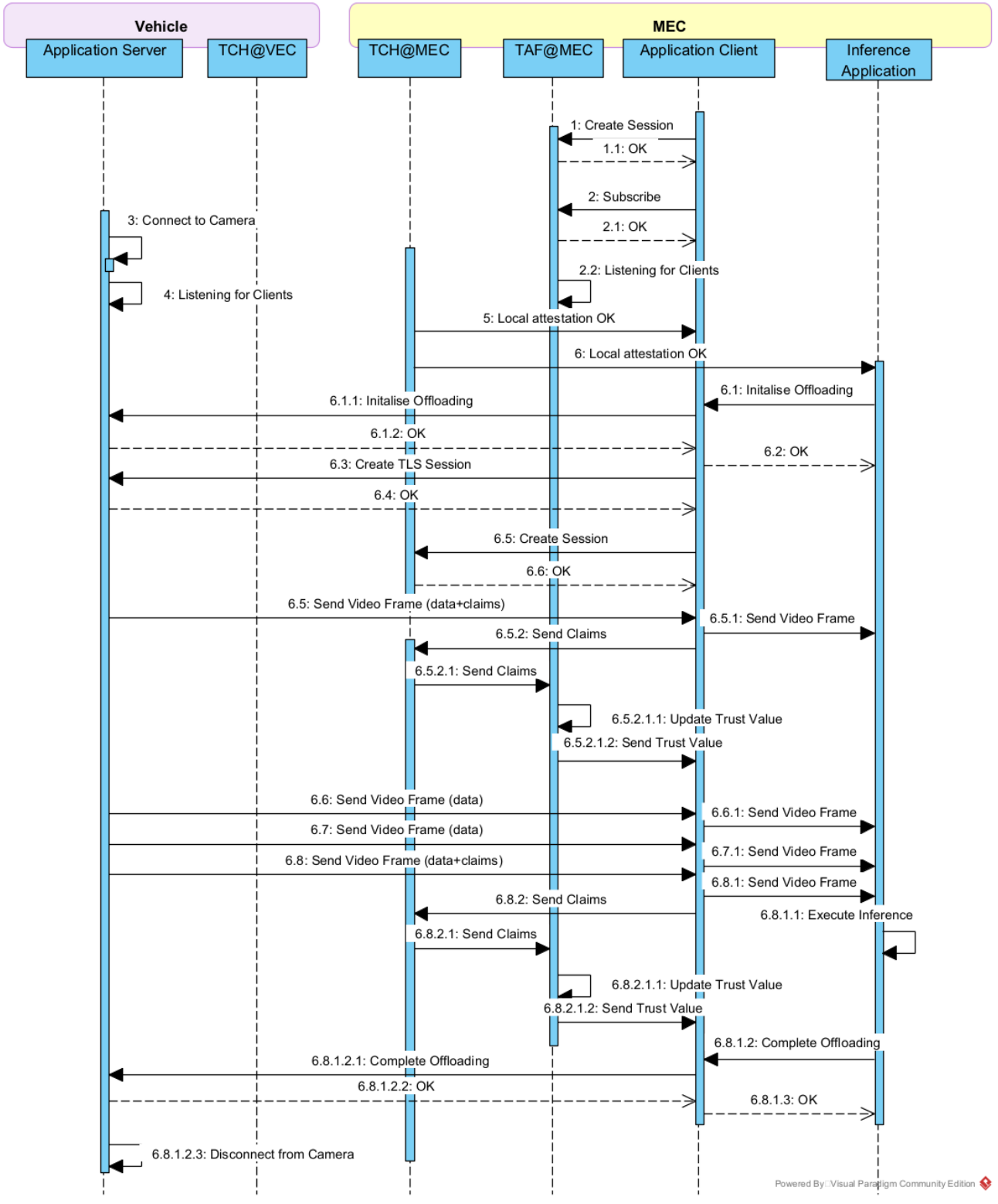

Figure 2.5: Sequence diagram of the operation in which claims are included in the video frames (piggybacking)

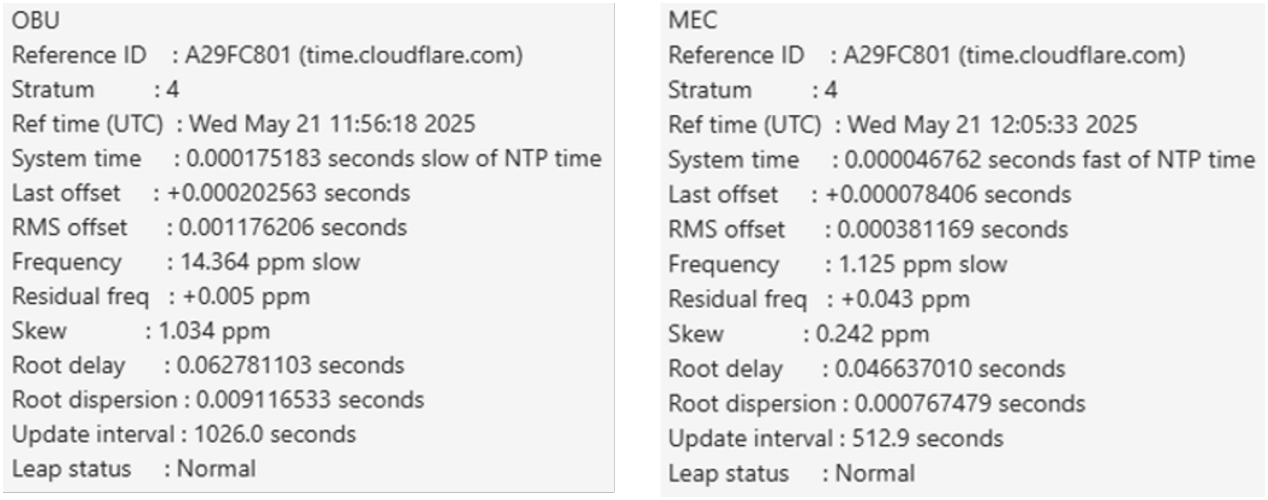

Figure 2.6: Time synchronization results using Chrony in OBU (left) and MEC (right).

2.1.4 On the OBU - MEC Time Synchronisation

Clock synchronization is important for our measurements. Aiming to compute the one-way latency from the vehicle OBU to the infrastructure requires accurate correlation of log files between the two end-points. As such, if the system time between the two devices (communication end-points) is not synchronised, our evaluation experiments which are heavily relying on time values, would suffer from low reliability.

To address the challenge, we have resorted to well-established networking synchronization solutions. The Network Time Protocol (NTP) is a robust and widely used application layer protocol designed to synchronise clocks across network devices [21]. It offers accurate Coordinated Universal Time (UTC) from a time source (i.e., an atomic clock or GPS) enabling network devices (acting as clients) to adopt high-precision time corrections.

The reference implementation of NTP and default sychronisation tool in Linux is ntpd (Network Time Protocol daemon). Ntpd does not scale well in intermittent network connections (such as when using laptops on the move) and congested networks, therefore, in our case we have used the Chrony protocol instance [14]. The latter software is an NTP implementation which synchronises faster and more accurately even in challenging environment (by using hardware timestamping). We deployed and used Chrony in both OBU and MEC, sychronised to a common NTP Server. In order to ensure that both devices are well sychronised, we used the Chrony tracking command which monitors the current status of time synchronization. The results of this command are illustrated in Fig 2.6. The command estimated the Root Mean Square (RMS) differences between the offset samples generated by having the two endpoints (i.e., OBU and MEC server) accessing an NTP synchronization source over Chrony. Chrony periodically sends NTP requests to a time source (time.cloudeflare.come in our case) and for each received response, it calculates a time offset reflecting the difference between the local clock (ie OBU and MEC, respectively) and the time source clock. The RMS is derived by a sample of offsets generated over a certain time period. As illustrated in Fig 2.6, a small time difference is observed both in the vehicle OBU and the MEC. The commands are executed with a time difference of 9 minutes and as depicted the vehicle software exhibits larger RMS offset (1.76msec) compared to the MEC application (0.38msec) which may rely only on wired internet connections to reach the synchronization source. The observed difference as well as the actual offset values are negligible and (as will become evident in the next Chapter) do not actually affect any of our latency measurements.

2.1.5 A Task-offloading Trust Model and Relevant Messages

The main trust assessment need in the offloading case relates to the evaluation of the nominal operation of the software running at the vehicle OBU. This software drives the offloading of any relevant data captured by the vehicle sensors. Along this line, a TAF instance is hosted at the MEC server and is fed by evidence stemming from the vehicle. Appropriate security guarantees for the integrity of transmitted data (from the vehicle to the MEC) are provided by relevant protocols (TLS) that are in place (see Figure 2.2 as well as the experiments of paragraph 3.1.2).

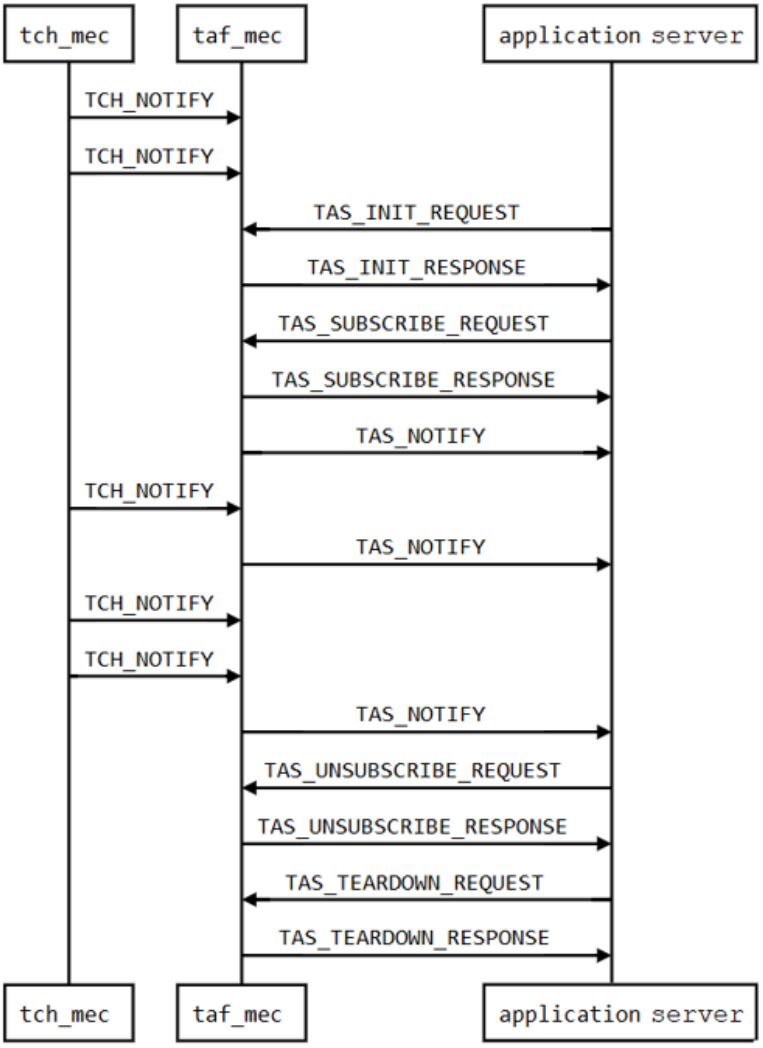

Figure 2.7: Interactions with the TAF for the task offloading scenario.

A dedicated trust model for task offloading has been designed and developed by CONNECT (defined as TO@0.0.1 in D3.3). The involved trust model is relatively simple; it has a single root node representing the MEC, which is modelled as the agent for which the TAF assesses trustworthiness. The offloading software that we are interested in is running there. Furthermore, it contains a single leaf node in the trust graph that represents the vehicle OBU. The trustworthiness handler (TCH) acts as the single trust source which provides the relevant data (i.e., attestation evidence) needed to assess the trust relationship.

Figure 2.7 shows the links between the TAF and the other two CONNECT software modules hosted by the MEC. TAF which keeps listening to the local TCH for incoming guarantees is ini-

tiated by the application server that calls for ”loading” the appropriate trust model. The TAF outcome which is provided as a notification to the application is updated whenever new evidence are available.

Chapter 3

Trusted Offloading - Testing and Benchmarking Measurements

3.1 Evaluation of Security and Trust Mechanisms/Computations

We divide the presentation of our static offloading experimentation results along three axes: a) those that concern the effectiveness of the involved security controls and trust computations; b) those that capture the networking quantities of interest; and c) those that reflect the consumption of resources along the offloading pipeline operations. In what follows, we detail the way we have carried out the relevant experiments and provide comments on the derived results.

3.1.1 Launch of Offloading Applications in a TEE

The first dimension that we aim to evaluate refers to the launch of the task-offloading applications inside a TEE environment. This is translated to the time it takes for the Gramine LibOS (library) to verify the manifest information and allocate the necessary resources to the spawned process, namely the enclave. In our experimentation, we consider the software-based TEE flavour (i.e., using the Gramine-Direct option), and the full-fledged, SGX-backed TEE version; both offered by the Gramine LibOS. Table 3.1 summarizes our measurements of the time that it takes for the task-offloading application which runs on the vehicle OBU to launch inside a TEE. Our results are averages accompanied by the corresponding standard deviation over five independent measurements for each TEE capability.

As expected, it is observed that the SGX-based enclave adds more burden to the launch of the application. Gramine SGX incurs a higher launch overhead compared to Gramine-Direct due to the additional steps required to initialise the hardware-backed Trusted Execution Environment, including enclave creation, allocation of Enclave Page Cache (EPC) memory, cryptographic verification of the enclave's measurement and manifest file as well as enforcement of SGX-specific integrity and attestation checks [13]. We have to also highlight that small differences that appear in the secure management of enclaves instantiated in either the Vehicle OBU or the MEC infrastructure are primarily due to the different compute capabilities of the underlying devices hosting the services (see Figure 2.3).

Table 3.1: Launch time (in ms) of the application running on the vehicle OBU

Launch of the app on the NUC	Hw-based TEE	Sw-based TEE	No TEE
Average	9994.6	586.0	558.0
Standard Dev.	335.1	12.9	15.2

Table 3.2: Launch time (in ms) of the application running on the MEC

Launch of the app on the MEC	Hw-based TEE	Sw-based TEE	No TEE
Average	10080.8	610.6	580.8
Standard Dev.	154.7	13.0	34.0

Table 3.3: Launch time (in ms) of the TAF on the MEC

Launch of the TAF on the MEC	Hw-based TEE	Sw-based TEE	No TEE
Average	6200.0	1006.5	811.0
Standard Dev.	400.0	24.5	24.0

Table 3.3 complements our results with the achievable launch time of the TAF instance which is hosted in the offloading scenario at the MEC. When comparing the measurements across the different applications, we observe two key points. First, given the similarities of the Manifest files for both task offloading applications, especially in the number of trusted files and software dependencies, it is reasonable to observe a similar launch time (i.e., less than 10 seconds). In addition, this controlled/verifiable launch is only performed one time during the boot-up of either the Vehicle OBU or the deployment of the MEC service, thus, its impact is negligible. This is primarily an operation that can be considered as part of the “setup” phase of an application. The instantiation of the TAF occurs only once while the update of the trust models can happen more frequently if there are changes in the trust relationships of the observed objects. But, as detailed in D3.3 [11], this operation is rather efficient contributing to the scalability of the entire trust assessment architecture. Secondly, when looking at the results of Table 3.3, it is clear that the codebase requires more time during the launch of the TAF application. Nevertheless, given the minimal set of additional requirements (i.e., only the application package is required apart from the Gramine runtime environment), Gramine requires significantly less time compared to traditional TEEs (e.g., INTEL SGX, ARM TrustZone) that typically manage the launch of the entire application and not only the codebase, as been performed by Gramine.

3.1.2 Establishment of a TLS-protected Cryptographic Connection

During the task-offloading process, it is critical that the video data extracted from the vehicle are transmitted over a secure and authenticated channel to the inference engine running on the MEC. Hence, a secure connection drawing on Transport Layer Security (TLS) should be established between the two offloading ends: the Application Server running on the vehicle OBU and the Application Client running on the MEC. TLS is a cryptographic protocol offering communications security for the nodes (e.g., Internet nodes) of a computer network [17]. Relying on cryptography and assuming the existence of one asymmetric key pair (associated with a certificate issued by a trusted certification authority) persisted in each end of the offloading process, this protocol

contributes to the establishment of a session key that allows the secure and privacy-preserving exchange of the data used in the task-offloading process.

We evaluated the TLS session establishment time for three scenarios as regards the environment in which the application server and client are running: a) without the presence of any TEE, assuming the offloading application is not using the functionalities offered by CONNECT; b) with the software-based TEE flavour using a Gramine-Direct instance; and c) with the presence of the hardware-based TEE, leveraging SGX-enabled CPU cores.

Table 3.4: TLS session establishment delay performance (in milliseconds) in different trusted execution environments

TLS establishment (in ms)	No TEE	Sw-based TEE	Hw-based TEE
Total TLS establishment delay	60	62	65

Table 3.4 depicts a slight increase in the total time required for the TLS establishment for the Sw-based TEE and Hw-based TEE compared to the 'vanilla' case i.e., no TEE case. The presented time increase across the increasingly complex environments is marginal; we can thus, consider that the TLS connection establishment delay remains practically unaffected by the underlying TEE mechanisms.

3.1.3 TAF Performance Measurements

The TAF application running at the MEC receives the trustworthiness claims (i.e., a type of trustworthiness evidence in the TAF terminology) as input. Each relevant message (i.e., TCH_NOTIFY) from the local TCH triggers the TAF computations. As such, for the evaluation of the TAF process we are interested in the time between the receipt of a new TCH_NOTIFY message and the corresponding update of the computed trust level (Chapter 9 in CONNECT D3.3 [11]) in the trust model instance (see the paragraph 2.1.5), called by the TAF. Note that the frequency of the transmitted trustworthiness claims does not affect the execution time of the TAF instance. In general, we have run experiments with a rate of one claim being transmitted every 1 or every 100 data chunks (i.e., video frames). As already measured and documented in [8] (Section 7.4), the secure measurement and collection of the necessary attestation evidence account for less than 50 milliseconds on average. This refers to the AIV process running inside a hardware-based TEE.

After running the same scenario 10 times, our results shown in Table 3.5 suggest that less than 15 milliseconds is a sufficient time window for the TAF computations. Similar results have been presented in Chapter 6 of the final CONNECT WP3 deliverable. Those results confirm that the runtime performance of the TAF is way below the safety-critical CCAM functions and their typical requirement of 200ms latency [5]. Clearly, this comes as a result of the careful design of the trust model template employed for the offloading case as well as the efficient computations carried out to quantify the trust opinions.

A final interesting observation relates the Hw-based enclave with the TAF operation. The SW-based TEE flavour introduces on average a slight (time) impact on the overall TAF operation, compared to the TEE-free operation. On the contrary, when moving from the Sw-based TEE to the Hw-based one (i.e., TAF running inside an SGX-based enclave), there is a significant increase of almost 35%. As seen in the CONNECT D3.3 (Chapter 6) the more complex the trust model is, the more overhead is introduced in the TAF operation due to the underlying Hw-based TEE.

Based on the previous statement and the results presented here and in D3.3, it becomes obvious that the TAF runtime performance is rather static compared to the trust model and is dependent on the number of trust model instances that need to be created; i.e., one trust model instance is created per trust object (e.g., vehicle) for which a trust relationship needs to be assessed. In fact, when running inside a hardware-based TEE, the performance degradation of the TAF runtime performance gets linearly higher as we consider a higher number of TMIs which is linked to a higher volume of processes been launched as part of the enclavised TAF. For instance, in the case where 100 ATL calculations need to be executed, attributing to 100 new trust objects (vehicles) been observed as part of the TMI, TAF takes approximately $15ms$ to finalise the calculation of all trust decisions resulting to an overhead of 35% compared to when a single ATL calculation is required ($5.25ms$).

Table 3.5: TAF performance (in milliseconds) in different trusted execution environments

TAF Operation (in ms)	TAF with no TEE	TAF in Sw-based TEE	TAF in Hw-based TEE
Average	9.67	10.67	14.33
Standard Dev.	2.36	2.36	3.31

3.2 Evaluation of Networking Performance Metrics

In this Section, we present the evaluation of the offloading pipeline from a networking standpoint. In particular, we are interested in assessing the time required to have a video analytics inference result ready at the MEC, based on the 'offloaded' video data. Furthermore, we explore the throughput values for different offloading pipeline deployments with respect to the launched TEE characteristics.

3.2.1 End-to-End Video Analytics Inference Delay

In our task-offloading scenario, we define the application end-to-end (E2E) delay as the time between the transmission of the first video frame (from the Application Server on the vehicle OBU) and the generation of the inference decision (in the Inference Server on the MEC). Implementation-wise, the Inference Server executes an inference process after having received (i.e., buffered) a number of n video frames. The parameter n is configurable and during the executed trials has been set to 40, 80 and 120 frames. Therefore, the 'total' end-to-end delay time is made up by a sum of a) the time needed for the transmission of all n video frames from the Application Server to the Application Client; b) the time needed for the delivery (buffering) of the video frames towards the Inference Server (i.e., DeepStream application in Figure 2.1) and finally c) the processing of the frames by the ML model and the generation of inference results.

As expected, the E2E delay is affected by several configuration parameters (e.g., selection of n) and network conditions (e.g., radio links). We therefore tried to evaluate the E2E delay under a set of selected scenarios. This allows us to shed light on the way the various configurable parameters exercises influence on the achievable inference delay.

Initially, we evaluated the impact of the underlying network which accommodates the data transfer from the vehicle OBU to the MEC. As depicted in the top-most part of Figure 2.2, the vehicle OBU

is represented in our lab setup by a dedicated mini PC (we call a NUC). We first connect the NUC and the (MEC) infrastructure over an Ethernet local network and then, we substitute it with a 5G new radio and core (as presented in the aforementioned picture). In both cases the full-fledged SGX- backed TEE version is applied, with the parameter n kept constantly to 40 frames. The presented averages are computed over 50 samples of delay values.

Table 3.6: E2E delay (in milliseconds) in different network environments

E2E delay (in ms)	Ethernet LAN	5G Network
Average	1597	2811
Standard Dev.	442	413

As expected, Table 3.6 suggests that the E2E delay is highly affected by the network environment, the results illustrate a high increase in the 5G network case. The 5G links, despite the ideal network conditions (i.e., proximity to the base station) yield an almost double E2E inference delay compared to the wired direct connections. As explained in the next paragraph, the high values of the E2E delays are due to the buffering of video frames before they are fed to the ML application for inference.

A second experiment seeks to evaluate the impact of the TEE type on the E2E delay. For this case a set of experiments were executed under increasing levels of trusted execution environments capabilities: a) without the presence of any TEE; b) using a software-based TEE (Gramine-Direct) and c) launching a full-fledged SGX-backed enclave. In all cases, the experiments were executed in a 5G network environment.

Table 3.7: E2E delay (in milliseconds) in different trusted execution environments

E2E delay (in ms)	No TEE	Sw-based TEE	Hw-based TEE
Average	2399	2686	2811
Standard Dev.	285	418	413

In line with intuition, our results (Table 3.7) illustrate that the application of an increasingly complex TEE poses additional E2E time costs that on average appear 12% (Gramine-Direct) and 17% (SGX) larger than the vanilla case (i.e., offloading with no TEE present). It is important to mention here that the average one way latency for the test-bed 5G network is initially measured to around 16.8ms with a standard deviation of about 1.8ms (estimated based on 100 ping values). Therefore, given our prior results on the latency for launching the pipeline (i.e., server, client and TAF) applications, the TLS session establishment and the time for TAF computations, it becomes clear that the large portion of the observed E2E delay is due to buffering and ML computations rather than the network itself.

Finally, we evaluated the impact of inference frequency parameter on the E2E delay. The inference frequency defines how often the inference process is executed by setting the number of video frames that should be considered (i.e., input to the DeepStream application) in each inference round. Our so-far E2E latency results suggest that buffering incoming video frames at the MEC in order to feed the DeepStream application is an implementation-specific characteristic that adds heavily to the final latency outcome compared to the network (e.g., transmission) or the TEE computations, as described in paragraph 3.1. Along these lines, we look into the impact of selection 40, 80 and 120 video frames as inference frequency values. Note that if values below 40 are selected, the inference process (DeepStream application) fails to identify objects such as the

leading vehicle (see Figure 2.1), or does identify it but with low precision. Therefore, we consider 40 frames as the minimum value and then we double and triple this value in order to evaluate the impact of video frame buffering on the E2E delay. Since the video frame rate in our case is 30fps, the values of 40, 80 and 120 are further translated to a minimum latency due to buffering of 1.33, 2.67 and 4.00 seconds, respectively.

Table 3.8: E2E delay (in milliseconds) for different inference frequencies

E2E delay (in ms)	40 frames	80 frames	120 frames
Average	1615	3236	4810
Standard Dev.	416	517	493

The results in Table 3.8 confirm that the E2E delay is highly affected by the inference frequency; the higher the frequency the larger the inference delay caused by the time to have the required number of video frames collected (buffered) at the DeepStream application. Even for the case of the minimum required frames, the achievable latency remains two orders of magnitude larger than the measured launch time of the CONNECT applications, the TAF response (both evaluated in the previous section) as well as the nominal 5G latency for the one-way delay from a UE device to the MEC. Clearly, optimising the data pipeline to feed the ML models of a video analytics application, is implementation-specific and its exploration remains outside of the CONNECT scope.

3.2.2 Achievable Network Throughput

In order to evaluate the overhead in terms of traffic generated by different trust execution environments, we carried-out a number of experiments measuring the application layer throughput on the uplink direction. The uplink direction is important in our case since the video frames generated in the camera on the vehicle are transmitted locally in the Application Server and then through the 5G test-bed network are delivered to the Application Client on the MEC.

What is important to stress is that the application throughput is not affected by the underlying network but by the TEE related functionalities and their parameters. In this direction we evaluated a set of scenarios for different configurations of the full-fledged SGX-backed TEE version, being the most advanced one. In detail, we evaluated:

1. The case in which no TEE is applied
2. The SGX case in which evidences are also embedded every 100 frames
3. The SGX case in which evidences are embedded every 30 frames (practically this means one transmission of evidences every 1 second (the frame rate is 30FPS)
4. The SGX case in which evidences are embedded in each frame. This means that every 33.3 ms the Application Server sends one video frame together with the trustworthiness evidence

Table 3.9: Application layer throughput (Kb/s) for different TEE configurations

TEE configuration	Average Throughput (Kb/s)	Overhead (%)
No TEE (vanilla)	3182.4	0
TEE (evidences every 100 frames)	3215.04	1.03
TEE (evidences every 30 frames)	3226.32	1.38
TEE (evidences every frame)	3691.68	16.00

The results of Table 3.9 derived by relying on the ifstat Linux utility [1], illustrate that no significant overhead (compared to the vanilla case) is added due to TEE operation when the evidence transmission frequency is kept low (e.g. every 30 frames meaning once per second or lower). In the case we need to receive the trust values in smaller time-scales (e.g., once per frame) the overhead is becoming significant. This also exhibits significant dependency on the bitrate of the streaming video (as shaped by the camera capabilities) and the trustworthiness evidences' size. In our case the streaming video has an average throughput of 2.3 Mbps. One then needs to trade-off the imposed throughput overhead with the trust monitoring needs; and in case of a safety-critical application rather than a video analytics, this trade-off needs to be resolved under particular time constraints [5].

3.3 Evaluation of Resources Consumption

3.3.1 Devices Power Consumption

Our final exploration amounts to experiments that assess the consumption of power, compute and memory resources by the devices used for the setup of the offloading pipeline.

We evaluated the power consumption both in the vehicle side (i.e., the NUC device) and the MEC server during the offloading process for the different TEE capabilities: a) without the operation of any TEE; b) using a software-based TEE (Gramine- Direct); and finally c) availing the full-fledged SGX-backed TEE capability.

Table 3.10: Power consumption (in Watts) in different trusted execution environments

Power consumption (in Watts)	No TEE	Sw-based TEE	Hw-based TEE
NUC - Average	13.0	13.2	15.2
NUC - Standard Dev.	0.1	0.4	3.5
MEC - Average	5.5	5.9	8.0
MEC - Standard Dev.	2.9	3.8	5.6

The results (Table 3.10) demonstrate that the power consumption on both devices (NUC and MEC server) appears to be increasing with higher TEE capabilities. This is mainly due to the extra processes (e.g., employ CPU features to create the protected Enclave Page Cache) required to achieve the SGX security guarantees [13]. In all test scenarios, the average and standard deviation values over 50 samples are calculated. In the MEC case we observed high standard deviation values due to the fact that high spikes are measured at the laptop serving as the MEC server, during the experiments.

3.3.2 Devices Memory and CPU Usage

For our final experiments we are evaluating the impact that the TEE environment has on the memory and CPU usage of the (containerised) applications involved. To better understand the effect on all the application modules involved, we measure both the memory and CPU usage for each of the three containers in which the modules are located: a) Application Server container at the NUC device (i.e., vehicle OBU); b) Application Client container on the MEC server and c) Inference Server on the MEC. We evaluated the relevant consumption under different TEE environments [1]. All the scenarios are executed while utilising the 5G lab test-bed deployed at ICCS (see Figure 2.2) to facilitate the offloading.

Table 3.11: Memory usage (%) for different trusted execution environments

Memory usage (%)	**No TEE**	**Sw-based TEE**	**Hw-based TEE**
Application Server - Average	0.05	0.06	10.1
Application Server - Standard Dev.	0	0	1.6
Application Client - Average	0.07	0.08	13.3
Application Client - Standard Dev.	0	0	0.4
Inference Server - Average	1.4	-	-
Inference Server - Standard Dev.	0	-	-

Looking at the memory usage results in Table 3.11, it becomes clear that the SW-based TEE (gramine-direct) has some effect on the memory used, compared to the vanilla case. In our experiments, the gramine OS library provides some isolation for the offloading application server and client (containers) code at the expense of 20% and 14%, respectively, against the plain containers operations. This is expected, as Gramine's Direct execution mode runs the application directly on the host Linux OS, without the security or performance overhead associated with Intel SGX enclaves. On the other hand, the use of Hw-based TEE flavour (Application running inside an Intel SGX enclave) has a significant impact on the memory usage exceeding the linear trend as we increase the level of the TEE guarantees. Specifically, it goes from 0.05% to 10.1% and from 0.07% to 13.3% for the Application Server and Application Client containers, respectively. The same effects also hold for the CPU utilisation (Table 3.12) in which especially in the Application Server case the compute resource usage is multiplied by five in the SGX case compared to the vanilla case (i.e., no TEE). In addition, the CPU usage of the Application Client is (on average) almost doubled.

Table 3.12: CPU usage (%) for different trusted execution environments

CPU usage (%)	**No TEE**	**Sw-based TEE**	**Hw-based TEE**
Application Server - Average	6.4	8.2	29.5
Application Server - Standard Dev.	3.8	4.8	20.1
Application Client - Average	8.2	8.8	14.9
Application Client - Standard Dev.	4.5	4.7	13.4
Inference Server - Average	30.1	-	-
Inference Server - Standard Dev.	24.1	-	-

[1] As the Inference Server (DeepStream application) is assumed trusted and executed outside of a TEE, values are provided only for the 'No TEE' case.

The significant overhead both in terms of memory consumption and CPU utilization when running applications inside an SGX enclave can be attributed to several architectural and runtime factors. One of the primary contributors to the memory overhead has to do with the configuration in the Gramine Manifest description for each ”enclavised” application. As illustrated in Listings 2.1–2.2, the enclave memory size is configured to 4 GB. This implies that during launch of each enclave, the requested enclave memory is pre-allocated regardless whether it is used by the application or not. In addition, a major factor is the multithreaded nature of the applications combined with their reliance on I/O-intensive operations. In this case, the task offloading applications establish WebSocket channels for communication of the two counterparts and use Kafka clients to interact with other CONNECT components, namely the TAF and TCH. These functions are running continuously and require frequent interaction between the enclave and the untrusted host OS, resulting in expensive transitions (i.e., ECALLs and OCALLs) between the trusted and untrusted environment, further amplifying the CPU usage.

A significant source of additional memory overhead is the set of files and software dependencies provisioned through the Gramine manifest and made available to the enclavised applications. On the one hand, the establishment of a secure and authenticated TLS channel between the vehicle and the MEC introduces memory overhead due to the cryptographic operations involved, as well as the in-memory storage of cryptographic material. On the other, an alternative to in-memory storage is to seal (i.e., encrypt and integrity-protect) this material and keep it on the host. When needed, enclaves securely unseal and load their own cryptographic assets. Nevertheless, both approaches contribute to the elevated memory overhead observed in enclavised application deployments.

Finally, heavy and complex software dependencies provide significant impact on the memory overhead. One such example refers to the use of libraries to receive and process the video streams from the camera sensors. As seen in the Manifest descriptions (Listings 2.1–2.2), this implementation relies on the FFmpeg library which connects to the camera stream and extracts the video workload to be used in the context of the inference task on the MEC. Even though FFmpeg is launched as a separate subprocess (i.e., separate enclave within its own EPC allocated), its execution within an enclave leads to additional memory overhead due to the (enclave) startup cost, memory loading, key management between the parent-children enclaves, and the (potentially limited) EPC space.

In general, there are multiple fine-tuning capabilities and optimization strategies that could help mitigate many of the performance challenges discussed so far. Nevertheless, this evaluation has demonstrated the potential of trusted computing mechanisms to support important operations such as the task offloading scenario, within the safety-critical limitations posed by the CCAM ecosystem.

Chapter 4

The *CONNECT* DLT - System Overview

4.1 Design and Implementation of the Final DLT Release

The *CONNECT* Distributed Ledger Technology (DLT) is an important component within the *CONNECT* framework, enabling the tamper-resistant storage and retrieval of data that is used to establish and maintain trustworthiness between connected vehicles. The data is duplicated across a number of nodes that work together to validate the data being stored and reach a consensus about its accuracy. The *CONNECT* DLT uses a blockchain infrastructure and this is what ensures that the data is tamper-resistant, as blocks cannot be revised later as this would break the chain. The combination of immutability and decentralization of the *CONNECT* DLT ensures the security of confidential vehicle data and facilitates reliable data exchanges for the calculations that are necessary for assessing the operational trustworthiness of vehicles.

The *CONNECT* DLT provides a highly secure repository for failed attestation data[1] and trust policy-related data. This stored information, particularly the failed attestation data, is used by key stakeholders from the Cooperative, Connected and Automated Mobility (CCAM) and Original Equipment Manufacturer (OEM) sectors, to assess and update the trust policies for the MEC and connected vehicles.

The different components of the *CONNECT* DLT, which have been presented in detail in D5.2, are now summarised below.

Blockchain Peer: This accommodates the necessary chaincode for the execution of smart contract processes in the *CONNECT* DLT and manages the Private Ledger and Off-Chain Storage where transaction data are stored.

Chaincode Service: This enables the actual execution of smart contracts for the recording of trust model templates, required trust levels (RTLs) and failed attestation evidence in the *CONNECT* DLT.

Security Context Broker: This constitutes a trusted mediator between the *CONNECT* DLT infrastructure and those components of the *CONNECT* framework that desire to access the *CON-*

[1] If attestation succeeds then the system is configured and working as expected and so there is no need to burden the system with data from these attestations.

NECT DLT. The SCB is responsible for granting access to the *CONNECT* DLT only to authorised entities that have the necessary Verifiable Presentations by using ABAC. It also uses **Attribute-Based Signcryption (ABSC)** to verify the source of failed attestation evidence it receives before it is stored in the *CONNECT* DLT. The SCB operates as a Node.js server and provides REST APIs for the communication with the rest of the entities of the *CONNECT* framework.

API Layer: This manages the interaction between the SCB and the Blockchain Peers and provides the necessary REST APIs that can be invoked by different *CONNECT* entities to access the DLT, upon authorization based on their Verifiable Presentations.

Off-Chain Storage: This enables the storage of the signed and encrypted failed attestation evidence, that because of its large size and/or sensitive nature cannot be directly stored in the Private Ledger of the *CONNECT* DLT.

Private Ledger: This facilitates the storage of trust model templates and RTLs, as well as the database location pointers of the data stored in the Off-Chain Storage. The *CONNECT* DLT enables the operation of various data workflows towards guaranteeing auditability and certifiability of data sharing transactions among the *CONNECT* entities. In this way, the trustworthiness of connected vehicles is safeguarded within the *CONNECT* framework.

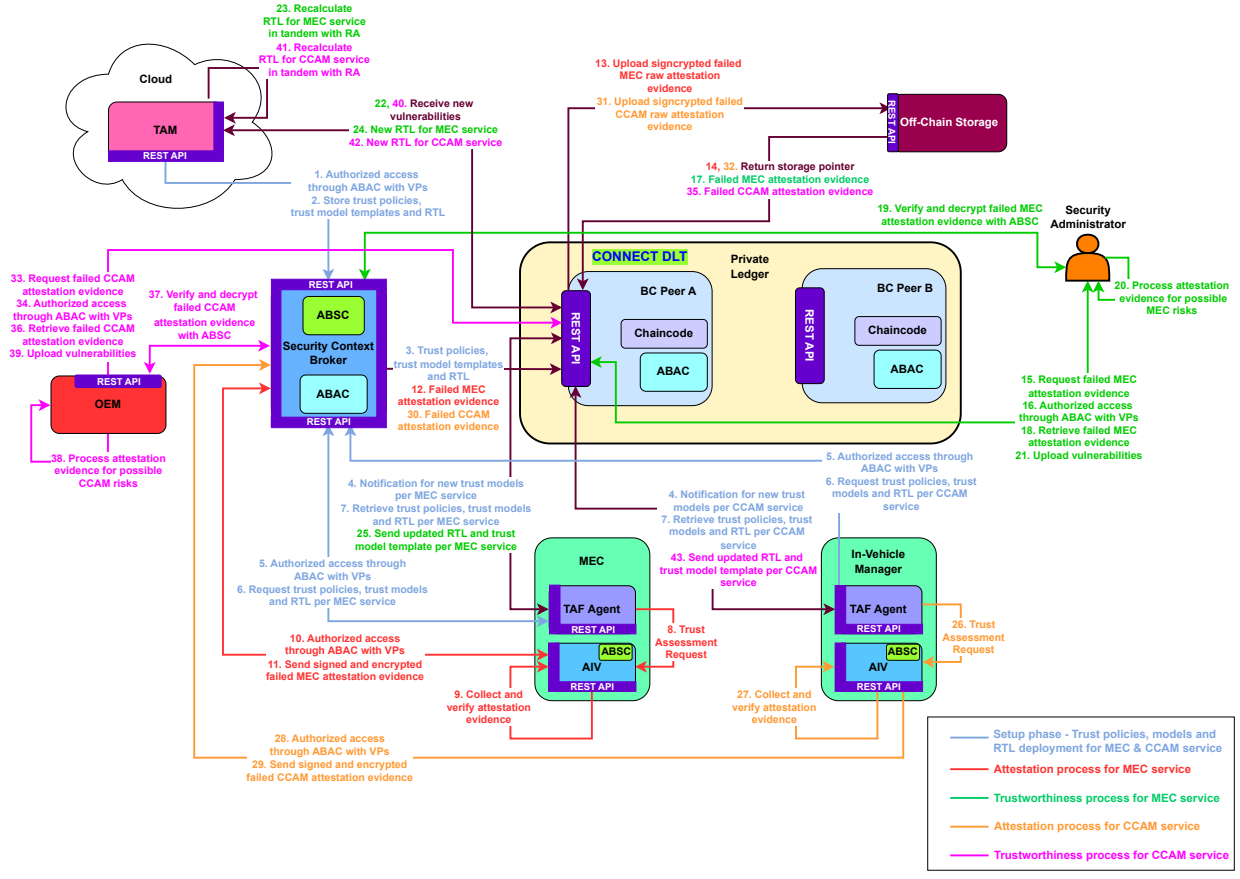

Figure 4.1: Architecture of Final *CONNECT* DLT Release

The five dedicated data workflows of the *CONNECT* DLT are depicted in the Figure 4.1 (with a larger version for printing at the end of Appendix B). Each of the workflows are represented with a different colour to showcase the various phases for establishing trustworthiness in *CONNECT* framework. These phases are described in the next paragraphs.

A. Setup phase – Trust-related Data Deployment for MEC & CCAM Services

1. The Trust Assessment Manager (TAM), the component inside the TAF which orchestrates the overall process of trust assessment, requests access to the *CONNECT* DLT to store trust model templates in the Private Ledger. In this case the TAM has already obtained during the enrollment phase a valid certificate from the Privacy CA, it is authorised by the ABAC Service of the Security Context Broker (SCB) through the check of its Verifiable Presentation (as created using a subset of its Verifiable Credentials) to access the *CONNECT* DLT.
2. Upon authorization, the TAM forwards the trust model templates to the *CONNECT* DLT through the REST APIs provided by the SCB.
3. The trust model templates are stored in the Private Ledger of the *CONNECT* DLT using the corresponding chaincode through the Blockchain Peer.
4. A notification that new trust model templates have been deployed is sent from the involved Blockchain Peer through REST APIs to the TAF Agents of the MEC and the Vehicles.
5. The MEC and the Vehicular TAF Agents, using their Verifiable Presentations, are authorised through ABAC to get access to the *CONNECT* DLT.
6. Upon authorization, the TAF Agents request the newly deployed trust model templates from the *CONNECT* DLT through REST APIs.
7. The trust model templates are retrieved from the Private Ledger of the *CONNECT* DLT by the TAF Agents. Depending on which service is going to be triggered by these trust model templates, either a MEC or a CCAM service, the next data flows are following; B and C for a MEC service, while the data flows are D and E for a CCAM service.

B. Attestation Process for a MEC Service

8. When an application requires it, the TAF Agent of the MEC sends a trust assessment request to the corresponding AIV.
9. The AIV of the MEC collects and verifies the respective attestation evidence for the certain MEC in *CONNECT* framework.
10. In case of failure of the attestation process, the AIV requests to access the *CONNECT* DLT and is authorised through ABAC with the use of its Verifiable Presentations.
11. Upon authorization, the AIV uses ABSC to sign and encrypt the failed MEC attestation evidence and sends it to the SCB through REST APIs.

12. The SCB uses ABSC to verify the signature on the signed and encrypted failed MEC attestation evidence. If the signature is verified, the SCB forwards the signed and encrypted failed MEC attestation evidence to a Blockchain Peer of the *CONNECT* DLT. The Blockchain Peer stores through a smart contract (chaincode) the failed attestation report in the Private Ledger.

13. The signcrypted failed MEC attestation evidence is stored in the Off-Chain Storage through REST APIs.

14. A database pointer for the database location of the failed attestation evidence is sent back to the Blockchain Peer and is stored in the Private Ledger, amending the block of the corresponding data transaction (failed attestation report block) in the Private Ledger of the *CONNECT* DLT.

C. Trustworthiness Process for a MEC Service

15. A Security Administrator can request from the *CONNECT* DLT (through the REST APIs offered by the SCB) the failed MEC attestation evidence in order to check for any potential new threats and vulnerabilities.

16. The Security Administrator is authorised by the ABAC Service of the *CONNECT* DLT using its Verifiable Presentation to get access to the Private Ledger.

17. The Blockchain Peer, using the corresponding database location pointer from the Private Ledger, retrieves the encrypted failed MEC attestation evidence from the Off-Chain Storage.

18. The failed MEC attestation evidence is forwarded from the Blockchain Peer through the SCB to the Security Administrator.

19. The Security Administrator can obtain the necessary attribute-based decryption keys through the *CONNECT* Certificate Authority (CA) and can verify the signature and decrypt the signed and encrypted failed MEC attestation evidence (see Figure 5.8).
    Note:

    (a) In general, any entity that satisfies the ABAC access policy and the ABSC decryption policy can retrieve and decrypt the failed attestation data.

    (b) Decryption of the failed attestation data can be also done on behalf of the requesting entity by the SCB (see Figure 5.7).

20. The Security Administrator processes the raw attestation traces for potential MEC risks.

21. In case new vulnerabilities are identified, the Security Administrator uploads them through the REST APIs of the SCB to the Private Ledger of the *CONNECT* DLT.

22. The new vulnerabilities are retrieved by the TAM through the SCB interfaces (REST APIs).

23. The TAM recalculates the RTL for the MEC service based on these vulnerabilities in collaboration with the Risk Assessment (RA) component.

24. The new RTL and trust model templates for the MEC service are stored in the Private Ledger of the *CONNECT* DLT through the SCB REST APIs.

25. The new RTLs and respective trust model templates are then available to the TAF Agent of the MEC. Following the update, the SCB informs the TAF of the update so that it can update its local copy of the Trust Model.

D. Attestation Process for CCAM Service

26. When an application requires it, the TAF Agent of the In-Vehicle Manager sends a trust assessment request to the corresponding AIV.
27. The AIV of the In-Vehicle Manager collects and verifies the respective attestation evidence for the vehicle in *CONNECT* framework.
28. In case of failure of the attestation process, the AIV requests to access the *CONNECT* DLT and is authorised through ABAC using a Verifiable Presentation.
29. Upon authorization, the AIV uses ABSC to sign and encrypt the failed CCAM attestation evidence and sends it to the SCB through REST APIs.
30. The SCB uses ABSC to verify the signature on the signed and encrypted failed CCAM attestation evidence. If the signature is verified, the SCB forwards the failed CCAM attestation evidence to a Blockchain Peer of the *CONNECT* DLT. The Blockchain Peer stores through a smart contract (chaincode) the failed attestation report in the Private Ledger.
31. The signcrypted failed CCAM attestation evidence is stored in the Off-Chain Storage through REST APIs.
32. A database pointer for the database location of the failed attestation evidence is sent back to the Blockchain Peer and is stored in the Private Ledger, amending the block of the corresponding data transaction (failed attestation report block) in the Private Ledger of the *CONNECT* DLT.

E. Trustworthiness Process for CCAM services

33. An Original Equipment Manufacturer (OEM) can request from the *CONNECT* DLT (through the REST APIs of the SCB) the failed CCAM attestation evidence in order to check for any potential new threats and vulnerabilities.
34. The OEM is authorised by the ABAC Service of the *CONNECT* DLT using its Verifiable Presentation to get access to the Private Ledger.
35. The Blockchain Peer, using the corresponding database location pointer from the Private Ledger, retrieves the encrypted failed CCAM attestation evidence from the Off-Chain Storage.
36. The failed CCAM attestation evidence is forwarded from the Blockchain Peer through the SCB interfaces (REST APIs) to the OEM.
37. The OEM can obtain the necessary attribute-based decryption keys through the *CONNECT* CA and is able to verify the signature and decrypt the failed CCAM attestation evidence (see also Figure 5.8). As noted in item 19 verification and decryption of the failed attestation data could be also done on behalf of the OEM by the SCB (see also Figure 5.7).

38. The OEM processes the raw attestation traces for potential CCAM risks.
39. In case new vulnerabilities are identified, the OEM uploads them through SCB REST APIs to the Private Ledger of the *CONNECT* DLT.
40. The new vulnerabilities are retrieved by the TAM through the SCB interfaces (REST APIs).
41. The TAM recalculates the RTLs for the CCAM service based on these vulnerabilities in collaboration with the Risk Assessment (RA) component.
42. The new RTLs and trust model templates for the CCAM service are stored in the Private Ledger of the *CONNECT* DLT through the SCB REST APIs.
43. The new RTLs and respective trust model templates are then available to the TAF Agent of the vehicle. Following the update, the SCB informs the TAF of the update so that it can update its local copy of the Trust Model.

4.2 Data in the *CONNECT* DLT

The *CONNECT* framework leverages the capabilities of Blockchain towards establishing data workflows that ensure the trustworthiness of connected vehicles, as presented in the Table 4.1 below.

Table 4.1: Data workflows

Data Workflow	Description
Trust policies, trust model templates and RTL	This workflow refers to the trust assessment process that is run by the Trust Assessment Manager (TAM) and results in the generation of the trust policy templates, trust models and RTLs for the MEC/CCAM services and their update upon occurrence of new vulnerabilities. The trust policy templates, trust models and RTLs are recorded in the Private Ledger of the *CONNECT* DLT through smart contracts, along with access policies and ABSC policies.
Attestation data	This workflow pertains to the attestation process that is initiated by a MEC service or a CCAM service and produces an attestation report and attestation raw data regarding the trust level of the MEC/CCAM service. The attestation report is stored in the Private Ledger, while the attestation raw data in the Off-Chain Storage. The storage location pointer of the attestation raw data is also stored in the Private Ledger, through smart contracts to ensure integrity and auditability.

Chapter 5

The *CONNECT* DLT - Access Protocols

5.1 Attributes in the *CONNECT* Framework

Attributes are used in the *CONNECT* framework for access control, generating digital signatures and for encryption. The use of attributes together with the associated polices provides flexibility and makes the system easier to maintain, new requirements can be met by updating the policies being used and, where necessary, issuing new attribute credentials.

5.1.1 Introduction

Entities[1] in the *CONNECT* system have, or are assigned, attributes. These represent what the entity is, or the current state of that entity. Examples might include:

- An employee of a vehicle manufacturer who may have attributes such as: 'employee of BMW', 'engineer', 'working in France', 'security level 2', ...
- An ECU in a vehicle which may have attributes such as: 'manufactured by Infineon', 'have a TEE', 'secure boot', 'attested correctly (10 minutes ago)', ...
- A person with attributes such as: 'works for the vehicle certification agency', 'engineer', 'certified accident assessor', ...

How these attributes are assigned and confirmed (when necessary) will vary from one case to another. For people and some system components verifiable credentials and their combination in a verifiable presentation as described in Deliverable 5.1 [6] will be used. For other components some attributes may be assigned ('has a TEE', for example) while others may be the result of, for example, attestation or control-flow integrity measurements.

Given an entity's attributes we need to decide if they are sufficient to allow access to a system, or to enable particular operations to be carried out. In *CONNECT* these decisions are based on policies, $\mathcal{P}$, if the entity's attributes satisfy the policy then they are able to proceed and to have access, to sign a message or to decrypt some ciphertext.

For example, any employee in the system will have attribute values for at least a subset of: $\{\text{employer}, \text{role}, \text{security\_level}, \text{country}\}$ and a policy set on some data says that it can only be

[1] In this context an entity may be any person or device involved in the protocol.

decrypted by engineers employed by BMW and working in France or Germany. The policy will be:

$$\mathcal{P} = (\text{employer} = \text{BMW} \wedge \text{role} = \text{engineer} \wedge (\text{country} = \text{France} \vee \text{country} = \text{Germany})).$$

An engineer working for BMW in France will satisfy the policy, while an engineer working for Fiat will not, wherever they work. This is clearly very flexible any employee with the correct attributes can decrypt the data.

While convenient for a human interpreter, for automatic processing policies are converted to access trees and monotone span programs. These are now described before giving details of how they are then used to for access control and for protecting data in the system.

5.1.2 Access Trees and Monotone Span Programs

Access trees and monotone span programs are used to express access policies.

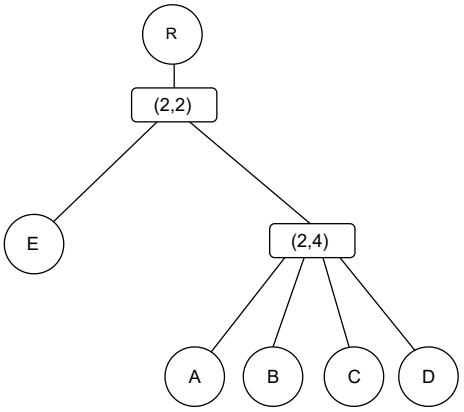

Figure 5.1: Access tree using threshold gates.

Access Trees An example access tree is shown in Figure 5.1 (taken from the example in [19]). At the root of the tree is a root value, R, while at the leaves are values associated with the attributes. How the R value is used will vary depending on the protocol being used, it may be a Boolean value, or a value to be used directly in the protocol. In all cases the root value will only be available if the user's attributes 'satisfy' the tree. In the example the access tree represents the condition that to obtain the value R the user should have attribute E and two of the attributes A, B,C and D. Given values for the attributes A, B, C and D we work up the tree (if the user does not have a value for a leaf attribute it is replaced by $\perp$). As we move up the tree if the children of a node meet the required threshold then that node's value can be passed up the tree, otherwise $\perp$ is passed up instead.

When the tree represents a Boolean circuit, then the attributes and the root value are all Boolean values (0 or 1). For the example shown $R = E \wedge (((A \wedge B) \vee (C \wedge D)) \vee ((A \vee B) \wedge (C \vee D)))$ and this is shown as a 'binary' tree in Figure 5.2.

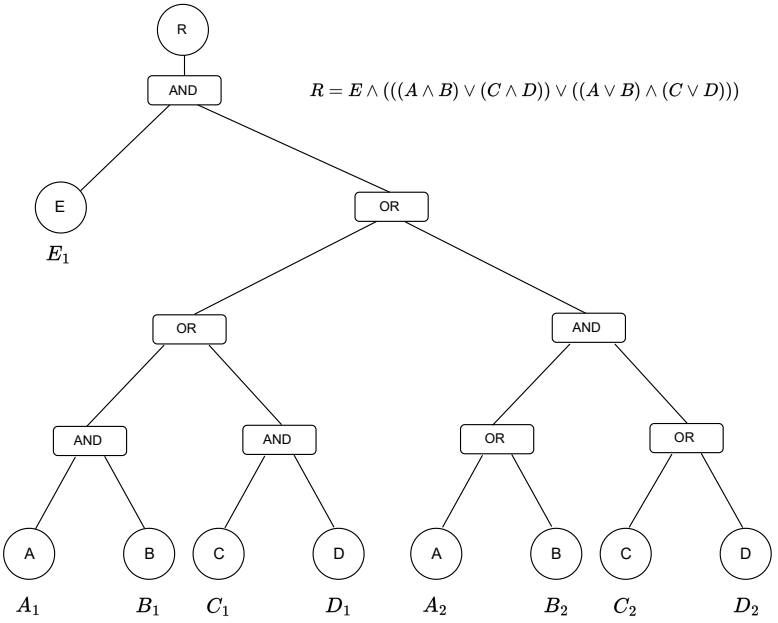

Figure 5.2: Access tree (Figure 3 from [19]).

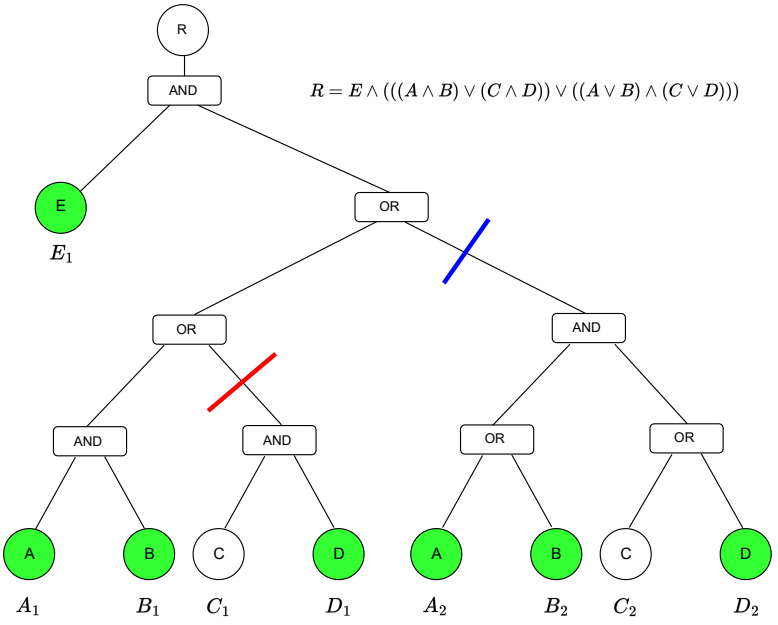

Figure 5.3: The pruned access tree.

As before to obtain the root value, we start at the leaves and work up the tree. For the example, suppose that a user has attributes E, A, B and D, we can find out whether they satisfy the policy by pruning the access tree (see Figure 5.3). we see that (E_1, A_1, B_1), (E_1, A_2, D_2) and (E_1, B_2, D_2) are solutions. The red line shows that we can prune the branch where $C \wedge D$ failed, while the blue line shows where we can prune the tree further as we already know that the condition is satisfied.

Monotone Span Programs Monotone span programs (MSPs) provide an alternative mechanism for attribute based access control [15][2]. An MSP is defined by the tuple $\langle M, \pi \rangle$. $M \in \mathbb{Z}_p^{n_1 \times n_2}$ is a matrix where every row is associated with an attribute and the function π gives the mapping between the attributes and the rows. For a Boolean circuit like that shown in Figure 5.2 there is a direct mapping between the leaves of the access tree and the rows of the MSP matrix.

As described below, a policy is satisfied, if a linear combination of the rows associated with a user's attributes is equal to the vector $(1, 0, .., 0) \in \mathbb{Z}_p^{1 \times n_2}$. As an illustration of the principle involved when using an MSP, consider a vector $\vec{v} = (s, r_1, .. r_{n_2-1})$ where s is a secret and $r_1, .., r_{n_2-1}$ are chosen at random from $\mathbb{Z}_p$. Let the rows of M be M_i with $i \in \{1, .., n_1\}$, then the n_1 elements of $M\vec{v}$ are $M_i\vec{v}$ and these 'hide' s. Let the linear combination of rows associated with a user's attributes that equals $(1, 0, .., 0)$ be $\sum_j \gamma_j M_j$, then $\sum_j \gamma_j M_j \vec{v} = (1, 0, .., 0)\vec{v} = s$ and s can be recovered. The details in the protocols that follow are different, but the underlying idea is the same.

More formally $M \in \mathbb{Z}_p^{n_1 \times n_2}$ and for $r \in [n_1]$, $\pi : r \to \mathcal{P}_\mathcal{A}$. Note that an attribute may be associated with more than one row of M and so the mapping π may not be injective. In this case, we define a further function $\rho(r) = |\{z \,|\, \pi(r) = \pi(z) \text{ with } z \leq r\}|$ to denote the $\rho(r)^{th}$ occurrence of attribute $\pi(r)$.

Given a set of attributes, $\mathcal{S} \subseteq \mathcal{P}_\mathcal{A}$, access is confirmed if the vector $(1, 0, .., 0) \in \mathbb{Z}_p^{1 \times n_2}$ lies in the span of the rows, M_i of M for which $\pi(i) \in \mathcal{S}$. If we let $\mathcal{I} = \{i : \pi(i) \in \mathcal{S}\}$, then we have

$$\sum_{i \in \mathcal{I}} \gamma_i M_i = (1, 0, .., 0)$$

for some constants $\gamma_i \in \mathbb{Z}_p$. Conversely, $\langle M, \pi \rangle$ does not accept $\mathcal{S}$ if there exists a vector $\omega \in \mathbb{Z}_p^{n_2}$ which is orthogonal to all of the rows $\{M_i : i \in \mathcal{I}\}$ and not orthogonal to $(1, 0, .., 0)$.

Note that writing $\bar{\gamma} \in \mathbb{Z}_p^{1 \times n_1}$ and setting $\bar{\gamma}_i = 0$ when $i \notin \mathcal{I}$ an alternative test for access is:

$$\bar{\gamma} M = (1, 0, .., 0).$$

For the example above, the corresponding MSP (obtained using the Lewko-Waters algorithm [18]) is:

$$M = \begin{pmatrix} 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \begin{matrix} E \\ A \\ B \\ C \\ D \\ A \\ B \\ C \\ D \end{matrix}$$

For this MSP we see that, for example, $\pi(3) = B$ and $\rho(3) = 1$. We also have $\pi(7) = B$ and in this case, $\rho(7) = 2$.

[2] Threshold access trees can be routinely converted into MSPs [19], as can access trees based on AND and OR gates.

As for the access tree example, suppose that the user has attributes E, A, B and D. In this case we have $\gamma_4 = \gamma_8 = 0$, using these in the equation gives:

$$\begin{pmatrix}\gamma_1 & \gamma_2 & \gamma_3 & \gamma_5 & \gamma_6 & \gamma_7 & \gamma_9\end{pmatrix} \begin{pmatrix}0 & -1 & 0 & 0 & 0\\ 0 & 0 & 0 & -1 & 0\\ 1 & 1 & 0 & 1 & 0\\ 1 & 1 & 0 & 0 & 1\\ 0 & 0 & -1 & 0 & 0\\ 0 & 0 & -1 & 0 & 0\\ 1 & 1 & 1 & 0 & 0\end{pmatrix} = \begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix}$$

Taking the transpose gives:

$$\begin{pmatrix}0 & 0 & 1 & 1 & 0 & 0 & 1\\ -1 & 0 & 1 & 1 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & -1 & -1 & 1\\ 0 & -1 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 0\end{pmatrix} \begin{pmatrix}\gamma_1\\ \gamma_2\\ \gamma_3\\ \gamma_5\\ \gamma_6\\ \gamma_7\\ \gamma_9\end{pmatrix} = \begin{pmatrix}1\\ 0\\ 0\\ 0\\ 0\end{pmatrix}$$

Re-arranging into row-echelon form gives:

$$\begin{pmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 1 & -1\end{pmatrix} \begin{pmatrix}\gamma_1\\ \gamma_2\\ \gamma_3\\ \gamma_5\\ \gamma_6\\ \gamma_7\\ \gamma_9\end{pmatrix} = \begin{pmatrix}1\\ 1\\ 1\\ 0\\ 0\end{pmatrix}$$

We have seven unknowns and just five equations, extracting the null-space gives the solution as:

$$\begin{pmatrix}\gamma_1\\ \gamma_2\\ \gamma_3\\ \gamma_5\\ \gamma_6\\ \gamma_7\\ \gamma_9\end{pmatrix} = \begin{pmatrix}1\\ 1\\ 1\\ 0\\ 0\\ 0\\ 0\end{pmatrix} + \alpha \begin{pmatrix}0\\ 0\\ 0\\ 0\\ -1\\ 1\\ 0\end{pmatrix} + \beta \begin{pmatrix}0\\ -1\\ -1\\ 0\\ 1\\ 0\\ 1\end{pmatrix}$$

In this case, as the γ–values can only be 0, or 1, possible solutions are:

$$\begin{pmatrix}1\\ 1\\ 1\\ 0\\ 0\\ 0\\ 0\end{pmatrix} (E, A, B), \qquad \begin{pmatrix}1\\ 0\\ 0\\ 0\\ 1\\ 0\\ 1\end{pmatrix} (E, A, D) \quad \text{and} \quad \begin{pmatrix}1\\ 0\\ 0\\ 0\\ 0\\ 1\\ 1\end{pmatrix} (E, B, D)$$

Note.

- Given a set of attributes we just need to find one solution to confirm that the policy is satisfied. So for the example above, the user has attributes E, A, B and D, but attributes E, A and B are enough to either satisfy the access tree (see Figure 5.3) or solve the MSP equation and show that the policy is satisfied. Further solutions are un-necessary.
- Given a binary access tree, we can obtain the monotone span program using the Lewko-Waters algorithm [18]. The access tree together with the attributes of an entity can also be used to check whether a policy is satisfied and to obtain $\bar{\gamma}$, as pruning the tree is often easier than using linear algebra.

5.2 DLT Data Access - Attribute-based Access Control

As described in Section 4.1 the *CONNECT* DLT is an important component of the CONNECT framework. Data stored on the DLT needs to be kept securely and any updates need to be properly authorised. The *CONNECT* DLT is therefore a permissioned DLT and to manage this, the SCB implements a highly granular access control mechanism that facilitates continuous authorization evaluation during data storage and retrieval. As such, the CONNECT framework uses Attribute-Based Access Control (ABAC), a mechanism that grants or denies access based on the attributes of the entity that is requesting access, as is also described in D5.2.

The flexibility of ABAC is particularly useful in dynamic systems like CONNECT, enabling fine-grained control over resource access. For the *CONNECT* DLT the ABAC mechanism uses Verifiable Credentials (VCs) and Verifiable Presentations (VPs). VCs and VPs were described in some detail in Deliverable 5.1 [6]. In short, VCs are signed statements that the holder has some attribute, or set of attributes. The VC is signed by an external authority that is acceptable to the verifier. For a VP the user combines some of their VCs and signs the resulting data together with a challenge from the verifier. The verifier can check the signatures on the credentials and the signature from the user and if all is correct, accepts that the user has these attributes.

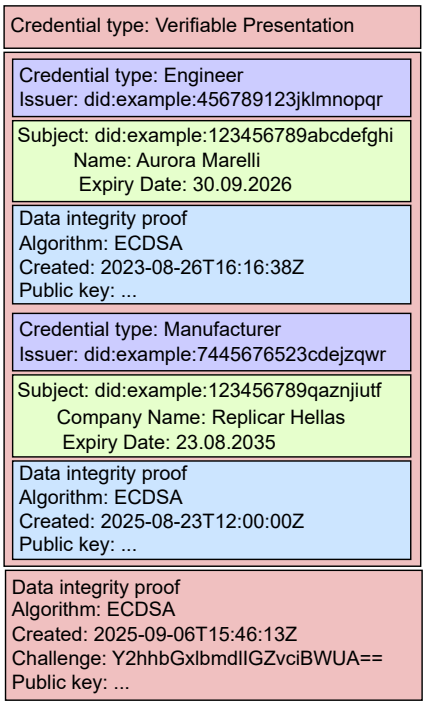

Figure 5.4: A VP for an engineer working for Replicar Hellas.

For example, see Figure 5.4 where a user's Verifiable Credentials (VCs) have been combined into a Verifiable Presentation (VP). Note that there is no restriction on the number of VCs that can be included although, for brevity, only two VCs are shown in the example. The separate VCs serve as the encoded attributes utilized within the ABAC system. When presented, a VP provides a cryptographically verifiable way to confirm an entity's attributes as they request access. The Security Context Broker (SCB) acts as the trusted mediator between the CONNECT DLT and other CONNECT entities, utilizing ABAC to ensure secure and authorized access.

The initial access of a new CONNECT entity (i.e., other CONNECT components or CCAM stakeholders) to the CONNECT DLT private channels requires the submission of its attributes to the Attribute-Based Access Control (ABAC) system. These attributes, encoded as Verifiable Presentations (VPs) and issued by a trusted external authority (for example, an Original Equipment Manufacturer), undergo a verification process to confirm their authenticity. Upon successful validation of these attributes, a request is transmitted to the Blockchain Certificate Authority (CA) for the registration of the CONNECT entity and the issuance of an appropriate certification. If the presented attributes are verified as valid, the Blockchain CA issues a new certificate, which is subsequently stored by the entity. Successful verification of the issued certificate by the SCB results in the entry of the CONNECT entity into the CONNECT DLT and its access to the Private Ledger. This flow of the ABAC authorization process is depicted in the Figure 5.5 below.

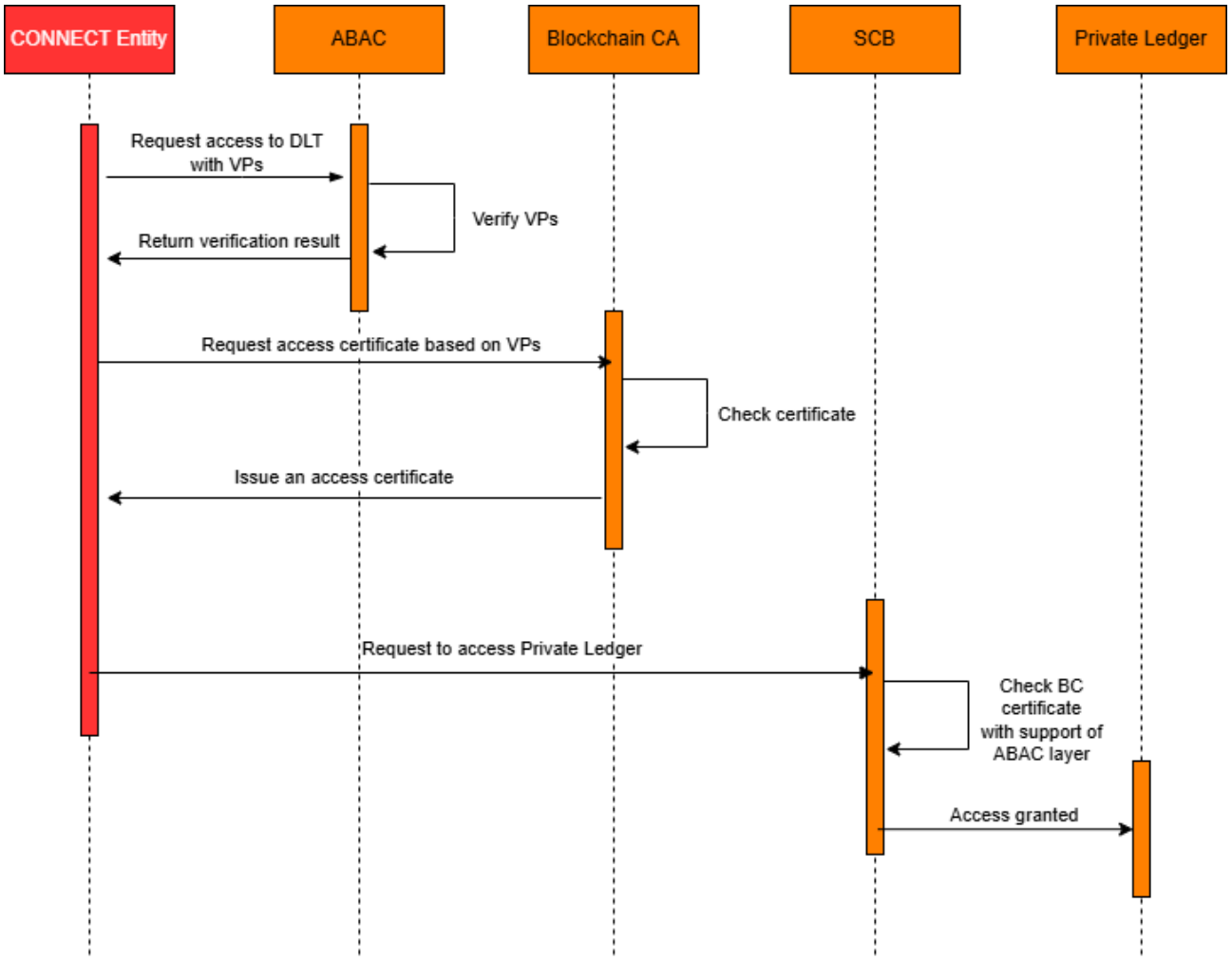

Figure 5.5: Accessing DLT through Attribute-based Access Control

5.3 Attribute-based SignCryption

Attribute-Based SignCryption (ABSC) is designed to provide message authentication and confidentiality in a single protocol and is a combination of Attribute-Based Encryption (ABE) and Attribute-Based Signatures (ABS).

- Attribute-Base Encryption (ABE) extends public-key cryptography to allow more control over who can decrypt a given ciphertext. In traditional public-key cryptography, a message is encrypted for a specific user using the user's public-key. As they hold the private key they are able to decrypt the ciphertext, while others are not. In ABE a user has a set of attributes which control whether they are able to decrypt a given ciphertext, or not. This is clearly more flexible any user with the correct attributes can decrypt the data.
- Attribute-Based Signatures (ABS). in a similar way ABS allows more flexibility in the signing process. Traditionally a signer has a private key that they use to sign the message, the verifier has the corresponding public key which they can use to verify the signature. In ABS a verifier who receives a valid signature knows that the signer has a set of attributes that satisfy an agreed policy.

For the ABE component of our ABSC scheme we use Riepel and Wee's FABEO protocol [20], while for the ABS component we have derived a new FABEO-based scheme.

In general, a system will have some universal set of attributes, $\mathcal{U}$, while policies are defined over some subset of these attributes. In the example in the introduction, $\mathcal{U}$ can be large, however the attributes in $\mathcal{P}$ are much more restricted ($\{\mathrm{BMW}, \mathrm{engineer}, \mathrm{France}, \mathrm{Germany}\}$). As described above, policy $\mathcal{P}$can be defined using an access tree, or as a monotone span program (MSP). If necessary, access trees can readily be converted to MSPs, but not the other way around.

Notation The following symbols and abbreviations apply:

p A prime number.

$\mathbb{Z}_p$ The set of integers in $[0, p-1]$.

$\mathbb{Z}_p^*$ The multiplicative group of invertible elements in $\mathbb{Z}_p$. The set of integers in $[1, p-1]$.

$\mathbb{F}_p$ A finite field of order p.

$\mathbb{G}_1$ An additive cyclic group of order p over an elliptic curve. The curve has points with co-ordinates in $\mathbb{F}_p \times \mathbb{F}_p$.

$\mathbb{G}_2$ An additive cyclic group of order p over an elliptic curve. The curve has points with co-ordinates in $\mathbb{F}_{p^2} \times \mathbb{F}_{p^2}$.

g_1, g_2 generators for $\mathbb{G}_1$ and $\mathbb{G}_2$.

$P + Q$ The elliptic curve sum of points P and Q.

O_E The elliptic curve point at infinity.

$[k]P$ Multiplication operation that takes a positive integer k and a point P on the elliptic curve E as input and produces as output another point Q on the curve E, where $Q = [k]P = P + P + \ldots + P$, i.e., the elliptic curve sum of k copies of P. The operation satisfies $[0]P = [O_E]$ and $[-k]P = [k](-P)$.

$\mathbb{G}^T$ A multiplicative cyclic group of order p.

$\hat{b}$ A pairing function $\hat{b} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}^T$ such that for all $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$, and all positive integers a, b, the equation $\hat{b}(P^a, Q^b) = \hat{b}(P, Q)^{ab}$ holds.

$a \xleftarrow{\$} B$ a is chosen uniformly at random from the set B.

H_0 A cryptographic hash-function, $H_0 : \{0,1\}^* \to \mathbb{Z}_p$.

H_1 A cryptographic hash-function, $H_1 : \{0,1\}^* \to \mathbb{G}_1$.

$X \parallel Y$ is used to mean the result of the concatenation of data items X and Y in the order specified.

Other notation will be introduced as necessary.

5.3.1 The Attribute-Based Signcryption Scheme

To implement this scheme we will combine elements of Attribute-Based Encryption and Attribute-Based Signature schemes derived from FABEO. These schemes are described in Appendix A.

The ABE component of our scheme is based on the FABEO KP-ABE scheme, but for the key generation component we adopt the method that we derived for ABS rather than use the FABEO key generation method directly. In this case, to ensure that different entities cannot collude we bind the attributes (and therefore the keys) to a secret key held by the entity requesting those keys.

In this scheme there will be the following entities:

- Privacy CA – the Privacy CA (CA) generates the system parameters, defines the universe of attributes and manages the policies defined over these attributes. These policies may be defined using access trees, but when used in the FABEO protocols will be converted to monotone span programs (MSPs). Other entities will obtain keys for their attributes from the CA. How they will prove that they actually have the attributes will be done using a separate channel - in *CONNECT* this may be done using verifiable credentials.
- Signcryptor – the entity that is encrypting and signing the data. There will be two policies involved, one for the signing and one for the encryption. The signing policy, $\mathcal{P}_S$, will confirm to the receiver that the data was signed by an entity with the correct attributes, while the decryption policy, $\mathcal{P}_D$, will control who can decrypt the data.
- Receiver – the entity that receives and decrypts the data. They will need attributes that satisfy the decryption policy and to retrieve the data from the DLT they will also need attributes that satisfy the ABAC policy, $\mathcal{P}_A$, as well ($\mathcal{P}_D$ and $\mathcal{P}_A$ may be the same).

Attribute	Description
$engineer$	An engineer working for the vehicle manufacturer or one of their component suppliers.
oem_1	A vehicle manufacturer.
oem_2	A vehicle manufacturer.
...	
ecu_1	An ECU manufacturer.
ecu_2	An ECU manufacturer.
...	
$regulator$	Someone working for the Vehicle Certification Agency.

Table 5.1: Attributes used for Attestation Failures.

Referring to Table 5.1 example policies for reporting on attestation failures could be:

$$\begin{aligned}\mathcal{P}_S &= oem\_1 \wedge ecu\_3 \\ \mathcal{P}_D &= (engineer \wedge oem\_1) \vee (engineer \wedge ecu\_3) \vee regulator \\ \mathcal{P}_A &= (engineer \wedge (oem\_1 \vee ecu\_3)) \vee regulator\end{aligned}$$

These policies can be stored as access trees and converted to MSPs as required (using the Lewko-Waters algorithm [18]). To avoid too may subscripts, in the notes that follow we will represent $\mathcal{P}_S$ as the MSP $\langle M_S, \pi_S \rangle$ and $\mathcal{P}_D$ as the MSP $\langle M_D, \pi_D \rangle$. In the algorithms variables relating to encryption/decyption will have primes (e.g n_1 for the number of rows in M_S and n_1' for the number of rows in M_D).

The certificate authority, CA, chooses the system parameters and their master secret key, msk.

SETUP

1 Choose $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{b}, g_1, g_2)$
2 Choose H_0 and H_1
3 $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$
4 Compute the master public key
$mpk = \hat{b}(g_1, g_2)^\alpha$
5 Store the master secret key $msk = \alpha$
6 **return** $(\mathcal{G}, H_0, H_1, mpk)$

For signing and decryption the same mechanisms are used to bind attributes and for key generation, but with different parameters (i.e. different values for $\langle M, \pi \rangle$, $\mathcal{I}$, SK and PK). As for the ABS protocol above:

BIND-ATTRIBUTES($\mathcal{S}, \langle M, \pi_M \rangle$, SK)

1 $\mathcal{I} = \{i : \pi_M(i) \in \mathcal{S}\}$
2 $h_m = H_0(M)$
3 **for** $i \in \mathcal{I}$
4 $\sigma_i = [\text{SK}]H_1(i \parallel h_m \parallel \pi_M(i))$
5 PK $= [\text{SK}]g_2$
6 **return** BA=($\mathcal{I}$, $\{\sigma_i\}_{i\in\mathcal{I}}$, PK)

To obtain their keys the requestor sends BA to the CA together with proof that they hold the attributes (possible in the form of a verifiable presentation).

KEY-GEN($msk, \langle M, \pi_M \rangle$, BA)

1 Extract $\mathcal{I}, \{\sigma_i\}_{i\in\mathcal{I}}$ and PK from BA
2 $h_m = H_0(M)$
3 $r \xleftarrow{\$} \mathbb{Z}_p$
4 $\bar{v} \xleftarrow{\$} \mathbb{Z}_p^{n_2-1}$
5 $sk_1 = [r]g_2$
6 **for** $i \in \mathcal{I}$
7 Check that $\hat{b}(\sigma_i, g_2) \stackrel{?}{=} \hat{b}(H_1(i \parallel h_m \parallel \pi_M(i)), \text{PK})$ and fail otherwise
8 $sk_{2,i} = [M_i(\alpha \parallel \bar{v})^T]g_1 + [r]\sigma_i$
9 **return** $sk = (sk_1, \mathcal{I}, \{sk_{2,i}\}_{i\in\mathcal{I}})$

The signer binds their attributes, $\mathcal{S}$, to their key, SK, and uses these to generate their signing keys, sk.

$$\begin{aligned} \text{BA} &= \text{BIND-ATTRIBUTES}(\mathcal{S}, \langle M_S, \pi_S \rangle, \text{SK}) \\ sk &= \text{KEY-GEN}(msk, \langle M_S, \pi_S \rangle, \text{BA}). \end{aligned}$$

As in FABEO, $msg \in \mathbb{G}_T$ and it is used to generate a symmetric key that will be used to actually encrypt the data. The SignCryption protocol is:

SIGN-CRYPT$(msg, \mathcal{P}_\mathcal{D}, \langle M_D, \pi_D \rangle, \mathcal{P}_\mathcal{S}, \langle M_S, \pi_S \rangle, \text{BA})$

// Generate the signature
Extract $\mathcal{I}, \{\sigma_i\}_{i \in \mathcal{I}}$ and PK from BA
$\bar{\gamma} =$ CALCULATE-GAMMA$(\mathcal{P}_\mathcal{S}, \mathcal{S})$
$h_m = H_0(M_S)$
$s, t, r_\alpha, \{r_i\}_{i \in [n_1]} \xleftarrow{\$} \mathbb{Z}_p^{n_1+3}$
$\sigma_1 = [\text{SK} \cdot t] sk_1$
$\sigma_2 = \sum_{i \in \mathcal{I}} [\gamma_i \cdot s] sk_{2,i}$
$\sigma_3 = \sum_{i \in \mathcal{I}} [\gamma_i \cdot s] H_1(i \parallel h_m \parallel \pi_S(i))$
$\sigma_4 = [t] g_2$
// Now do the encryption
$h'_m = H_0(M_D)$
for $i' \in [n'_1]$
    $ct_{1,i'} = [t] H_1(i' || h'_m || \pi_D(i'))$
$ct_2 = [t] g_2$
// Note that ct_2 == σ_4
$ct_3 = mpk^t \cdot msg$
$\bar{ct} = (\{ct_{1,i'}\}_{i' \in [n'_1]}, ct_2, ct_3)$
$Y = mpk^{s \cdot t}$
$Z = mpk^{r_\alpha}$
$W = \sum_{i \in [n_1]} [r_i] H_1(i \parallel h_m \parallel \pi_S(i))$
$c = H_0(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \{ct_{1,i'}\}_{i' \in [n'_1]}, ct_3, Y, Z, W, msg)$
$s_\alpha = r_\alpha - s \cdot t \cdot c$
for $i \in [n_1]$
    $s_i = r_i - \gamma_i \cdot s \cdot c$
$\bar{\sigma} = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, c, s_\alpha, \{s_i\}_{i \in [n_1]})$
return $(\bar{\sigma}, \bar{ct})$

The receiver (decryptor) uses their attributes, $\mathcal{S}'$, and key, SK′, to generate their decryption keys, sk'

$$\begin{aligned} \text{BA}' &= \text{BIND-ATTRIBUTES}(\mathcal{S}', \langle M_D, \pi_D \rangle, \text{SK}') \\ sk' &= \text{KEY-GEN}(msk, \langle M_D, \pi_D \rangle, \text{BA}'). \end{aligned}$$

Then to verify the ABSC signature and decrypt the data, the algorithm SIGNCRYPT-VERIFY is used.

SIGNCRYPT-VERIFY$(mpk, \langle M_S, \pi_S \rangle, \mathcal{P}_\mathcal{D}, \langle M_D, \pi_D \rangle, \mathcal{S}', \mathsf{BA}', sk', (\bar{\sigma}, \bar{ct}), msg)$

$h_m = H_0(M_S)$
$Y' = \hat{b}(\sigma_2, \sigma_4)/\hat{b}(\sigma_3, \sigma_1)$
$Z' = mpk^{s_\alpha} \cdot Y^c$
$W' = \sum_{i \in [n_1]} [s_i] H(i||h_m||\pi(i)) + [c]\sigma_3$
$c' = H_0(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \{ct_{1,i'}\}_{i' \in [n_1']}, ct_3, Y', Z', W', msg)$
if $c' \neq c$
    return 0;
else
    // Decrypt
    Check that ct_2 == σ_4, if not **return** 0
    Extract sk_1', $\mathcal{I}'$ and $\{sk'_{2,i}\}_{i \in \mathcal{I}'}$ from sk'
    $\bar{\gamma}'$=CALCULATE-GAMMA($\mathcal{P}_\mathcal{D}$, $\mathcal{S}'$)
    $U = \hat{b}\left(\sum_{i' \in \mathcal{I}'} [\gamma'_{i'}] ct_{1,i'}, [\mathsf{SK}'] sk_1'\right)$
    $V = \hat{b}\left(\sum_{i' \in \mathcal{I}'} [\gamma'_{i'}] sk'_{2,i'}, ct_2\right)$
    $msg = ct_3 \cdot U/V$
    return msg

5.3.2 ABSC in the *CONNECT* Framework

Figure 5.6 shows how ABSC is used in *CONNECT* to securely store failed attestation data on the DLT. The IAM in the vehicle has external access and is just used to pass information to-and-fro.

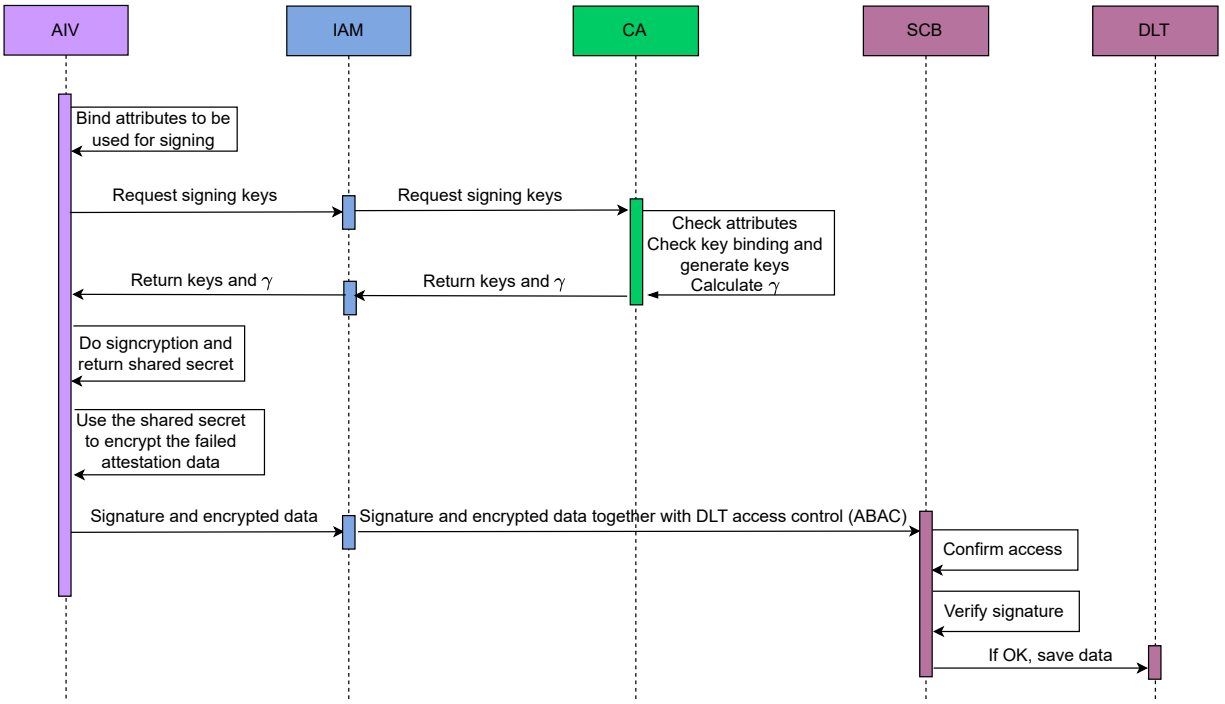

Figure 5.6: The signcryption protocol.

Figures 5.7 and 5.8 show two possible scenarios for verification and the retrieval of the data. In the first figure the SCB does the verification and decryption of the data and it is then supplied

to the requestor. This is done just on the basis of the ABAC access control and this may not be sufficient in some cases. The protocol in Figure 5.8 provides extra security as it is only a requestor with attributes that satisfy the decryption policy that can gain access to the data.

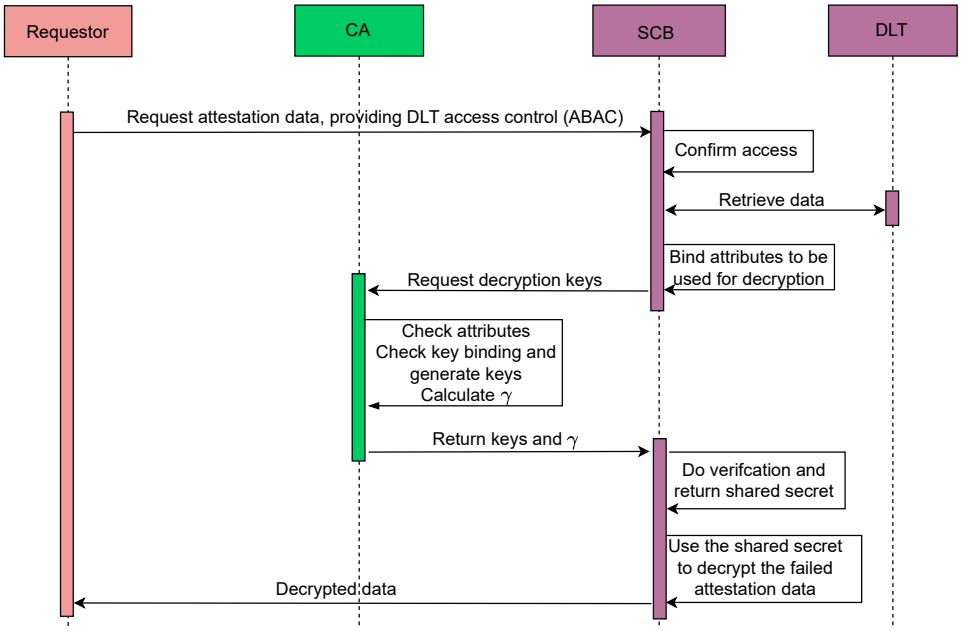

Figure 5.7: Protocol for verification and decryption by the security context broker.

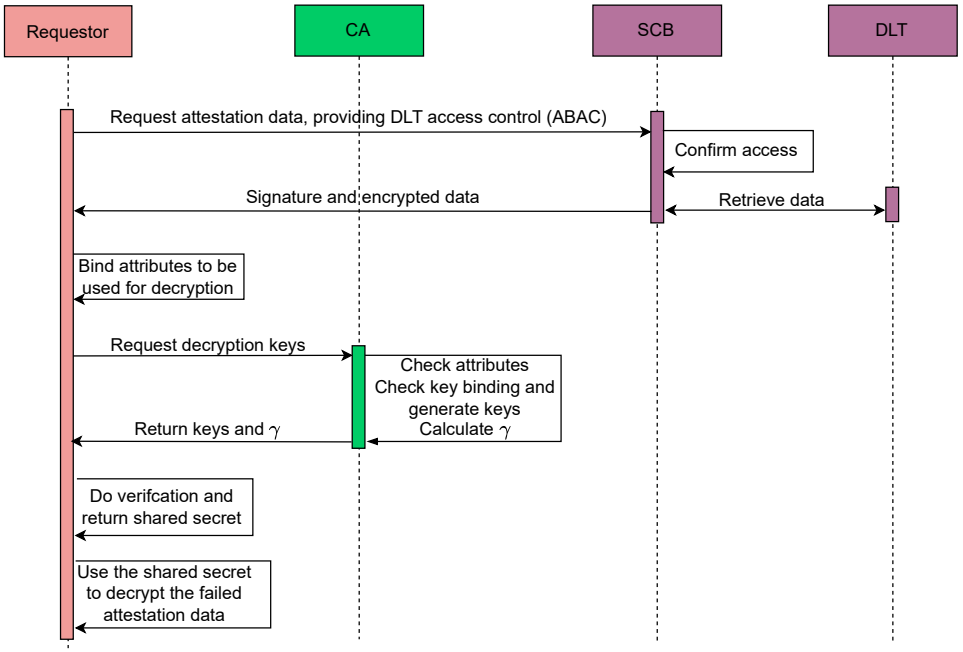

Figure 5.8: Protocol for verification and decryption by the requestor.

5.3.3 Benchmarking Elements of the ABSC Scheme

In this section we benchmark the different components of the ABSC scheme[3]. Timings will depend on the precise details of a given policy, the timings given here are for policies that are simple conjunctions of N_A attributes, this represents the worst case for a policy with N_A attributes as the number of columns in the MSP matrix depends directly on the number of AND gates in the policy, and in addition, for this policy all rows of the MSP matrix are used in the calculations. The benchmarks given here, in Table 5.1, were obtained on a Dell Lattitude 5491 laptop, with an Intel Core i7-8850H Processor (6 Cores, 9M Cache, 2.6GHz) and 32GB of available memory. Timings were obtained using the Linux `clock_gettime` routines with the `CLOCK_PROCESS_CPUTIME_ID` clock. The tests were run on a single core without using Gramine TEE emulation. As discussed previously the Gramine TEE will add some overhead and this will be reflected in the end-to-end measurements reported in D6.2.

The test program uses the MIRACL Core [2] cryptographic library and was compiled using GCC

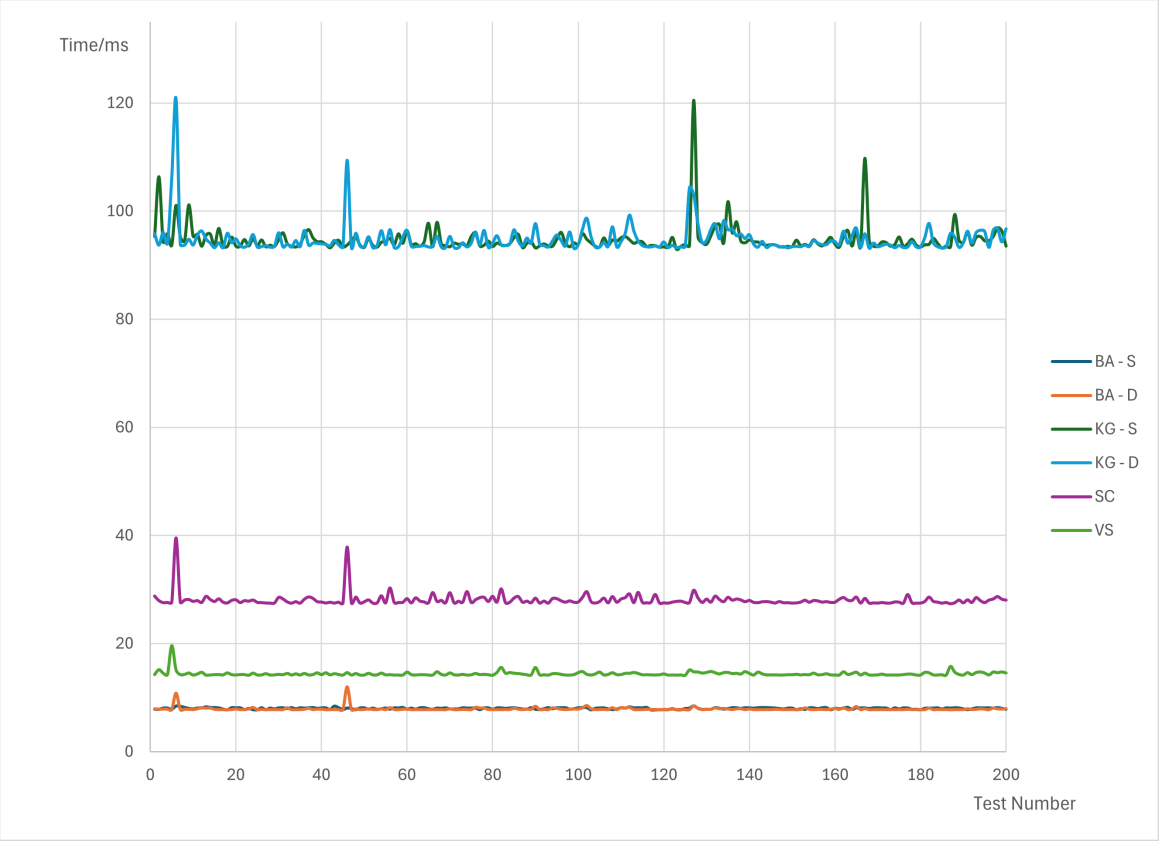

Figure 5.9: The Raw Timing Data for 30 Attributes.

version 11.4.0. The test program uses the FP256BN elliptic curve and runs the whole protocol checking that the signature verifies and that the shared secret is obtained correctly. The test program was run 200 times and the different functions in the protocol timed as outlined above. Figure 5.9 shows one of the timing sets, the traces are: BA-S – BIND-ATTRIBUTES for the signer, BA-D – BIND-ATTRIBUTES for the decryptor, KG-S – KEY-GEN for the signer, KG-D – KEY-GEN for the decryptor, SC – SIGN-CRYPT and VS – SIGNCRYPT-VERIFY.

[3]Benchmarking of the end-to-end protocols shown in Figures 5.6, 5.7 and 5.8 will be described and the results reported in Deliverable D6.2.

The traces are typically noisy at the start and then settle down. There are still odd spikes, presumably due to some system events that disrupt the process. To minimise these disruptions the laptop being used was disconnected from the network and the WiFi was disabled. However, as seen in Figure 5.9, disruptions still occur and so we average over 200 tests, this was enough to get consistent results. The timing averages and standard deviations obtained are given in

Table 5.2: Times Measured for the Different ABSC Functions

Function	Number of Attributes	Average Time (ms)	Standard Deviation
BIND-ATTRIBUTES	5	1.78	0.34
	10	3.01	0.19
	20	5.46	0.22
	30	7.95	0.29
KEY-GEN	5	16.40	1.04
	10	31.98	0.92
	20	63.37	1.67
	30	94.78	2.71
SIGN-CRYPT	5	7.11	0.80
	10	11.21	0.55
	20	19.59	0.86
	30	28.05	1.2
SIGNCRYPT-VERIFY	5	8.58	0.28
	10	9.69	0.31
	20	12.02	0.23
	30	14.40	0.46

Table 5.2. Normally the policies for signing and encryption will be different and different results for BA-S and BA-D would be obtained, similarly KG-S and KG-D would also be different. For these tests the signing and encryption policies were the same and so the two sets of 200 tests for BIND-ATTRIBUTES and for KEY-GEN are essentially the same (as highlighted in the Figures) and so they were combined for the table.

Figure 5.10 shows the results from the table plotted against the number of attributes. As expected from the function definitions the timings scale linearly with the number of attributes. The KEY-GEN function takes much more time than the others as it involves checking pairings for each attribute and the pairings calculations are very expensive. In the protocols, a user only has to call KEY-GEN (and BIND-ATTRIBUTES) once for each policy that they need to use, the keys that they receive can then be stored securely and used as required and so this comparatively heavy cost should not be a problem. The SIGNCRYPT-VERIFY function also requires pairings calculations, but however many attributes are involved there are only three pairings, there is some increase in the time required with the number of attributes, but this is not significant.

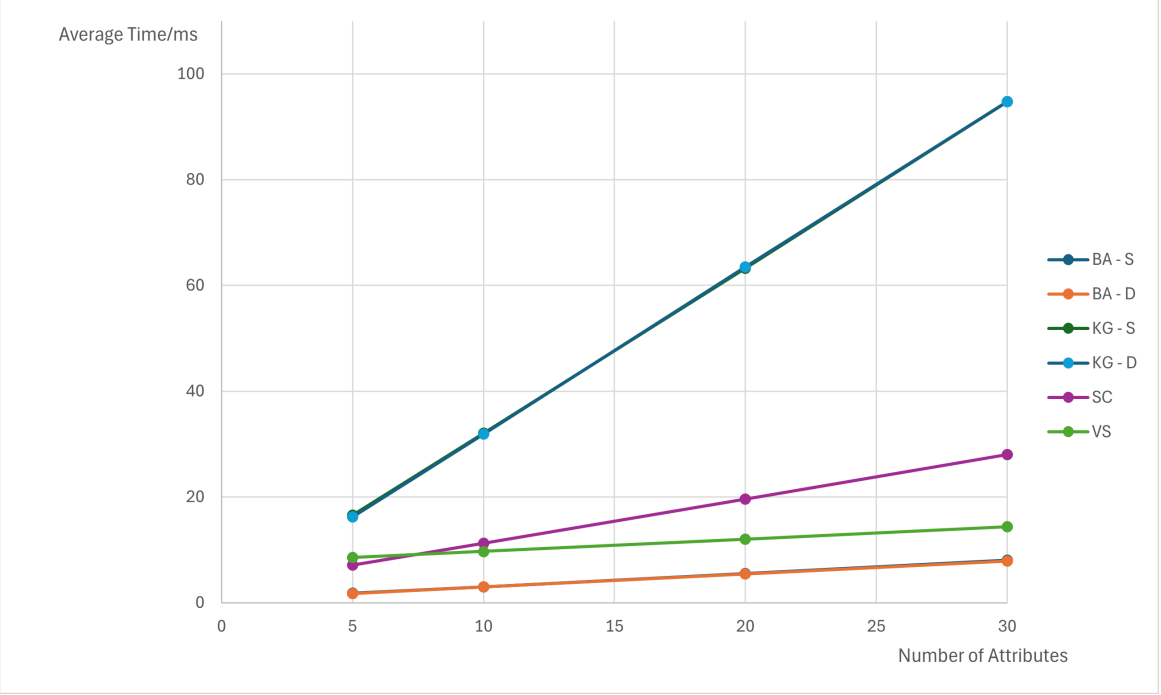

Figure 5.10: Averages vs Number of attributes N_A.

Chapter 6

Implementation and Testing of the *CONNECT* DLT Architecture

6.1 Smart Contracts

To record trust model templates, RTLs and attestation evidence in the CONNECT DLT, the Security Context Broker leverages smart contracts through the employment of chaincode. The data models used in the context of smart contract definition drive the functional specifications in the final release of the CONNECT DLT for the storage of such information. These models are presented in the form of data structures, describing the properties that shall be included in each structure of a smart contract.

The specified data structures along with their fields are defined below with accompanying tables.

Trust Policy Structure This structure contains all required information for the definition of a Trust Policy, comprising the RTL and the trust model for the specific CCAM function of interest. After invoking the `CreateTrustPolicy()` function, the Trust Policy structure is stored in the Private Ledger (see Table 6.1).

Name	Data Type	JSON Schema	Description
PolicyID	`string`	‘json:”policyID”‘	String identifier used as a unique key for denoting a specific policy. Required so that CONNECT entities are able to query for a specific policy, based on the required functionality to be performed.
CreatedAt	`string`	‘json:”createdAt”‘	String identifier including the creation date of the policy. Used for complex queries on the DLT that are based upon creation date of a Policy.
TrustSources	`[]string`	‘json:”trustSources”‘	An array of the trust sources that should be used to provide the necessary trust evidence for the evaluation of a given trust property.

TrustProperties	`[]string`	‘json:”trustProperties”‘	A (possibly many-to-one) mapping of trust sources to the trust properties that that will be used by the TAF when calculating a trust opinion. When setting up a Trust Model a Smart Contract will be deployed to query the DLT and identify the trust properties needed by this PolicyID and hence the trust sources that are required.
TrustModel	`object`	‘json:”trustModel” : { ”name” : `string`, ”version” : `string`, ”description” : `string`, ”type” : `string`, ”url” : `url`, ”hash” : `string` }‘	The TrustModel is uniquely identified by the `name` and `version` fields. The `type` indicates whether the `url` refers to a pre-compiled TrustModelInstance, or a Trust Model Template. If this is a Trust Model Template the `url` will point to a file that specifies all of the relevant Trust Objects, Trust Model Instantiation Policies, Trust Relationships, Trust Methods and Trust Sources that will be used by the TAF when generating Trust Model Instances. Details of Trust Model Templates and their use are given in Deliverable D3.3. The `hash` will be the hash of the file pointed to by the `url` and the `url` itself. This will allow the Trust Model Manager to verify the file that it is receiving.
ModeOfOperation	`string`	‘json: ”modeOfOperation”‘	String identifier denoting the type of communication with the TAF. More specifically, it can be synchronous or asynchronous; in the former case, the TAF is initiating the collection of evidence through a trust assessment request; in the latter case, the TAF is notified automatically of any change in the trust state of the target CONNECT entity.
RequiredTrustLevel	`number`	‘json: ”requiredTrustLevel”‘	The trust level that should be met, or exceeded, by the actual trust level calculated by the TAF.

Table 6.1: Trust Policy Model Structure

Data Structure for Saving a Failed Attestation Report The data structure used by the DLT when recording a failed attestation. The ABSC signature and the encrypted failed attestation data will be written to off-chain storage and so this record only contains a pointer to this storage (see Table 6.2).

Name	Data Type	JSON Schema	Description

ReportID	`string`	‘json:”reportID”‘	A string identifier used as a unique key for the identification of a specific attestation report. Complex queries on the DLT involving multiple reports utilise the reportID field to identify the relevant reports.
TaskID	`string`	‘json:”taskID”‘	A string identifier used to designate the TaskID of the task that was performed, and based on which the attestation report was generated.
RecordedAt	`string`	‘json:”recordedAt”‘	A string identifier including the creation date of the attestation report. Complex queries on the DLT may be based upon this field.
ListOfAccessAttributes	`[]string`	‘json: ”listOfAccessAttributes”‘	This field refers to the list of attributes that a CONNECT entity should possess, in order to be able to read this particular attestation report.
DBPointer	`string`	‘json:”DBPointer”‘	A string identifier, that is used by the SCB to store the location of the failed attestation report in the Off-chain Storage.

Table 6.2: DLT Failed Attestation Report Data Structure

6.2 Microbenchmarking of the DLT Pipeline

Within CONNECT, the Blockchain Infrastructure is designed to fulfill two main roles. The first role is providing TAF configurations for the different applications running on a vehicle, or the MEC. This is done through the deployment of Trust Model Templates (TMT) and Required Trust Levels (RTL). The second key role is the secure storage of failed attestation data that highlight verification failures. Such failures are indicative of unusual or potentially risky behavior, providing critical insights for Security Administrators and OEMs to trace execution flows and discover any vulnerabilities. Identifying vulnerabilities will drive the process for re-establishment of trustworthiness in vehicles and the MEC. Overall, the CONNECT Blockchain serves as a reputation-based mechanism, which is instrumental in establishing trust in the system.

In this deliverable the key Blockchain interactions of the final release of the CONNECT DLT are measured, including data storage and retrieval on the Distributed Ledger Technology (DLT) Infrastructure, integrated with the Attribute-based Access Control mechanisms for secure authorization. The efficiency of these operations is directly impacted by factors such as the total data volume of the DLT (ledger height) and the number of data transactions that are requested simultaneously.

The smart contracts of the final release of the CONNECT DLT for the storage and retrieval of attestation reports and trust model templates have been refined, while the ABAC Service has also been integrated into the CONNECT DLT. It has to be mentioned that in general, based on the performed benchmarking activities, **the ABAC Service adds in each transaction about 0.01 seconds (with a standard deviation of 0.003 seconds) for the authorization of the entity needing to access the CONNECT DLT**. The evaluation of the CONNECT DLT has taken place in an environment with an 8-core 3,2 GHz CPU and 16GB RAM. The Blockchain Infrastructure deployment had a low ledger height with one Blockchain Peer and one Ordering Service. The

writing/read operations for the final release have been performed through Postman client REST API calls that represented CONNECT entities. For the benchmarking of writing/reading of attestation evidence, JSON files of average size of 4 KB have been used, while for trust policy templates, JSON files of about 1 KB have been used.

For this setup, the transaction times for each operation in the CONNECT DLT is presented in Table 6.3, Writing an attestation report is more time consuming as the attestation reports include more data than the trust model templates and this necessitates using the Off-Chain Storage for the attestation evidence.

Table 6.3: Required Time Per Operation

Operation	Average Time (s)	Standard Deviation (s)
Write Attestation Report	2.105	0.073
Read Attestation Report by ID	0.041	0.009
Write Trust Policy	2.080	0.067
Read Trust Policy by ID	0.027	0.008

In the following figures, the intermediate measurements for each operation are described.

For the writing of an attestation report (Figure 6.1), the total time is 2105 ms. The API call from the CONNECT entity to the SCB lasts 17 ms, while the ABAC process called by the SCB lasts 10 ms. The recording of the raw attestation data in the Off-Chain Storage takes 4 ms and the recording of the attestation report and the off-chain database pointer to the Blockchain takes 2074 ms. These 2074 ms are split into 2050 ms for the smart contract operation in the Blockchain and to 24 ms for the actual recording of attestation report and off-chain database pointer on the on-chain database.

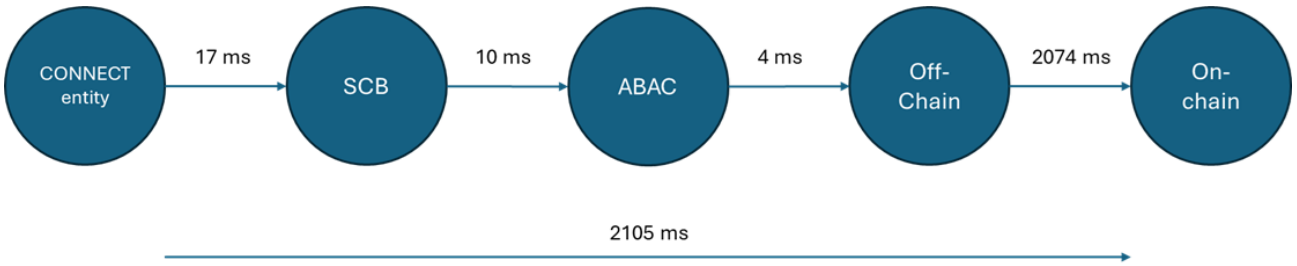

Figure 6.1: Measurements for Writing Attestation

For reading an attestation report (Figure 6.2), the total time is 41 ms. The API call from the CONNECT entity to the SCB lasts 17 ms and the ABAC process called by the SCB lasts 10 ms. The reading of the attestation data and the off-chain database pointer from the on-chain database takes 11 ms, while the reading of the attestation raw evidence from the Off-Chain storage takes 3 ms.

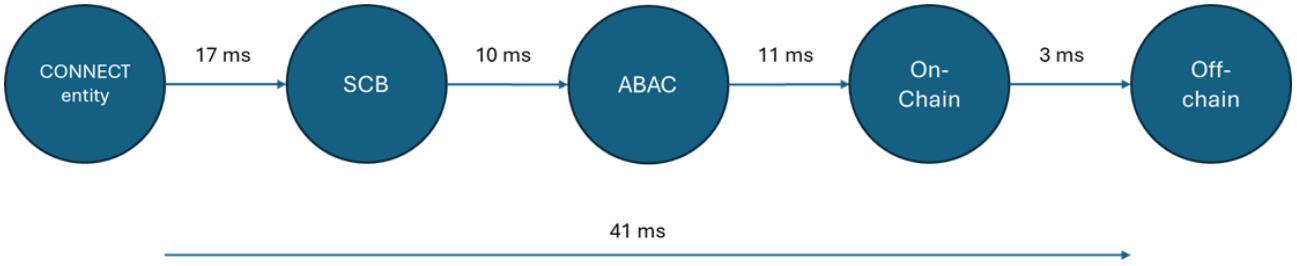

Figure 6.2: Measurements for Reading Attestation

For the writing of a trust policy (Figure 6.3), the total time is 2080 ms. The API call from the CONNECT entity to the SCB takes 16 ms and the ABAC process called by the SCB takes 9 ms. The recording of the trust policy in the Blockchain lasts 2055 ms, which are split into 2050 ms for the smart contract operation in the Blockchain and to 5 ms for the actual recording of trust policy on the on-chain database.

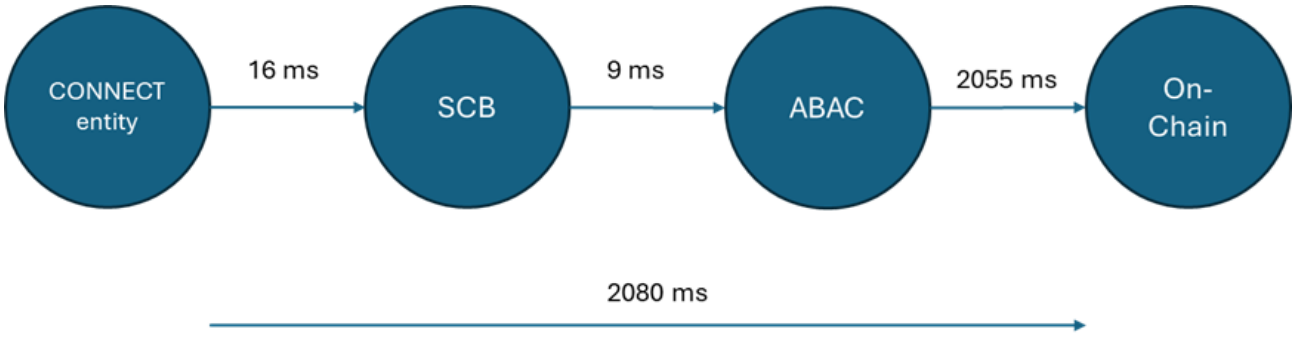

Figure 6.3: Measurements for Writing Trust Policy

For reading a trust policy (Figure 6.4), the total time is 27 ms. The API call from the CONNECT entity to the SCB takes 15 ms, while the ABAC process called by the SCB takes 9 ms. The reading of the trust policy from the Blockchain lasts 3 ms.

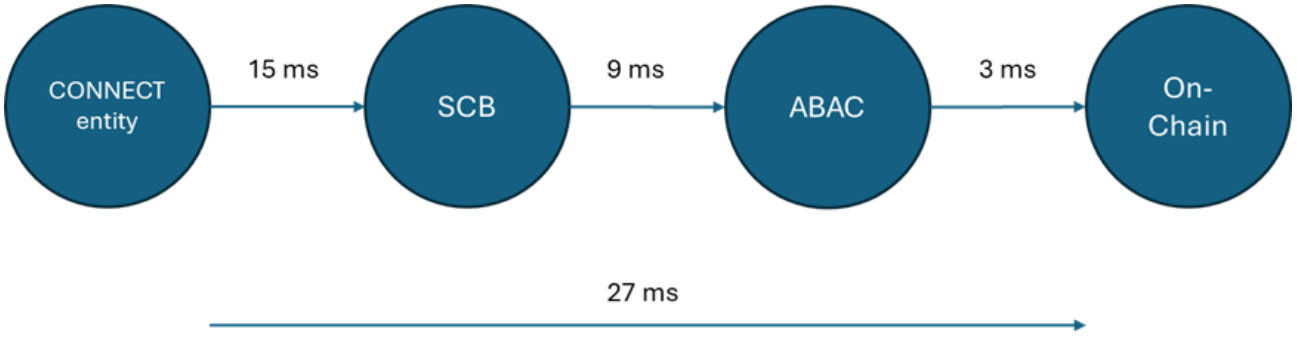

Figure 6.4: Measurements for Reading Trust Policy

In the following Table 6.4, the time that is required for reading and writing different numbers of attestation reports in a single transaction is measured (i.e., running the operation for a different number of times). It can be seen that as the number of attestation reports that are included in a transaction increases, the time that is necessary for the execution of the transaction increases, too. It has to be noted that the required time for the reading/writing of one attestation report at the CONNECT DLT (first row) is slightly higher than in the previous table, since here the first transactions that initialised the connection with the BC have been measured. In addition, in the writing operation, the tables in the Private Ledger and Off-Chain Storage had to be established for the first transaction.

Table 6.4: Required Time for Reading/Writing a Number of Attestation Files

Number of Attestations	Time to read number of attestations (s)	Time to write number of attestations (s)
1	0.043	2.274
10	0.128	20.653
50	0.433	103.496
100	0.671	205.646
500	2.895	1030.981

In the next Table 6.5, the time for writing one attestation report has been measured with different ledger heights in the CONNECT DLT. It is observed that as the number of attestation reports that are stored in the CONNECT DLT increases (i.e., the ledger height is increased), the respective time for writing an attestation report remains almost the same and is not affected by the ledger height in our Blockchain deployment.

Table 6.5: Required Time for Write Action with Different Ledger Height

Blockchain Height	Number of Attestation Reports on the ledger	Time (s)
from 1 to 50	20	2.105
from 50 to 100	70	2.112
from 100 to 200	150	2.110
from 200 to 500	350	2.115
from 500 to 1000	700	2.108

In Table 6.6, simultaneous calls from different CONNECT entities for writing attestation reports (one attestation report per each entity) to the CONNECT DLT have been performed in order to test the behavior of the Security Context Broker and in particular, the maximum number of concurrent calls that the SCB is able to handle. It is noticed that the higher the number of entities' calls, the longer the time is for performing each writing of attestation report in the CONNECT DLT. Nevertheless, the SCB is shown to handle efficiently more than 1000 concurrent calls. It has to be mentioned that the write time for each attestation report is lower here than in the previous tables (e.g., 0.4 seconds here compared to 2.105 in Table 6.3), as another API environment has been used for performing these concurrent CONNECT entities' calls, while the focus has been given to the number of entities making these calls.

Table 6.6: Required Time for Simultaneous Transactions by Different CONNECT entities

Number of Entities	Time to write each attestation (s)
50	0.4
100	0.5
200	0.7
300	0.9
400	1.7
500	2.3
600	2.6
700	3.1
800	3.3
900	3.9
1000	4.5
1100	4.7
1200	5.1
1300	5.6
1400	6.1
1500	6.5

Chapter 7

Conclusion

This deliverable has described the last parts of the CONNECT WP5 work. This work spans three directions and complements the previous WP5 theoretical work and implementation efforts.

The relevance and capability of CONNECT mechanisms to support the task offloading needs has been explored under a scenario of video data shifted from a vehicle computer to the edge infrastructure. Based on this video data, inference is carried out at the infrastructure to determine slow-moving traffic in front of the considered vehicle; a scenario that can be viewed as part of the CONNECT SMTD use-case. The offloading pipeline involved has been instantiated with a dedicated CONNECT trust model to support trust assessment and relevant applications running in trusted environments deployed over real-world hardware. Extended experimentation has shed light on a number of challenges pertaining to the time and resources needed for launching the relevant trusted environments as well as the end-to-end inference latency and bandwidth requirements over a baseline Ethernet network but more importantly over a 5G testbed, utilising commercial 5G radio and core instances.

Two more threads of work are presented in this document, both related to the CONNECT ledger. The design and implementation of advanced attribute-base access control mechanisms providing (fine-grained) access control to the CONNECT ledger based on the attributes that an entity seeking to gain access, exhibits. The final version of the verifiable credentials and presentations that the attribute-based access control employs for privacy-preserving operations is explained. An advanced combination of attribute-based encryption and flexible signatures has been developed and integrated in the CONNECT ledger to ensure message authentication and confidentiality.

Finally, the document explores the smart contracts that can be utilised to access the CONNECT ledger fabric. The required structures to include trust model templates, computed trust level values and attestation evidence are analysed. Individual experiments at the CONNECT DLT level have explored the required time for recording trust policies and attestation reports. Scalability testing regarding the performance of the CONNECT distributed ledger shows its capability to efficiently cope with concurrent calls (reaching the order of magnitude of a thousand) for attestation reports.

The conclusion of WP5 tasks is marked by this final deliverable of the work-package, providing important contributions to the CONNECT body of work. The WP5 work has established the appropriate networking means to support the CONNECT concept and evaluated a static (5G test-bed) instance of its offloading capability. At the same time, WP5 designed, implemented and evaluated the operation of the CONNECT blockchain layer and its attribute-based access control mechanisms as well as its capability to support smart contracts. Complemented by the

contributions of the trust-assessment methodology (WP3) and the enclave technology (WP4), the project provides all required inputs to carry-out its final evaluation round and present exploitable results for the CCAM community.

Appendix A

Attribute-Based Encryption and Attribute-Based Signatures

In this appendix we describe the underlying algorithms that are used to build the *CONNECT* ABSC scheme.

A.1 Attribute-Based Encryption

There are two different formulations of ABE depending on who sets the policy and how this is then used, these are now outlined before describing access trees and monotone span programs and giving details of several ABE schemes.

- **Ciphertext-Policy ABE.** In ciphertext-policy attribute based encryption (CP-ABE) a user is given a set of (private) keys associated with their attributes, while the encryptor incorpoates an access policy into the ciphertext. A user will be able to decrypt the ciphertext, if and only if their attributes satisfy the policy included with the ciphertext.

  While the encryptor has control over the policy and therefore the conditions that must be satisfied for decryption, they have no control over who the decryptor might be. Any current, or future, user with the correct attributes will be able to decrypt the data.

- **Key-Policy ABE.** In key-policy attribute based encryption (KP-ABE) the policy that the decryptor must satisfy is encoded in the keys used to decrypt the ciphertext. The user receives their (private) decryption keys from the key generator who decides what policy must be satisfied for decryption. The encryptor is given the associated public key(s) and uses those the encrypt the data, as for CP-ABE, they get no say in who can subsequently decrypt the ciphertext.

Note. An important property which has to be achieved by both, of these formulations is called collusion resistance. This basically means that it should not be possible for distinct users to ”pool” their secret keys such that they could together decrypt a ciphertext that neither of them could decrypt on their own (which is achieved by independently randomizing users’ secret keys).

A.2 FABEO: Fast Attribute-Based Encryption with Optimal Security

In their paper descibing FABEO [20], Riepel and Wee give protocols for both CP-ABE and KP-ABE although here we focus on KP-ABE as this forms the basis for the encryption component of our ABSC protocol.

Define $\tau = \max_{i \in [n_1]} \rho(i)$, the maximum number of times any single attribute is used in M (in the example: $\tau = 2$, A, B, C and D all appear twice).

KP-ABE The certificate authority, CA, chooses the system parameters and their master secret key, msk.

SETUP

1 Choose $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{b}, g_1, g_2)$
2 Choose $H : \mathcal{U} \to \mathbb{G}_1$.
3 $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$
4 Compute the master public key
$mpk = \hat{b}(g1, g2)^\alpha$
5 Store the master secret key, $msk = \alpha$
6 **return** $(\mathcal{G}, H, mpk)$

The CA decides on the policy to be used for the data and this fixes the MSP, $\langle M, \pi \rangle$. A user has a set of attributes, $\mathcal{S}$, and contacts the CA for their set of decryption keys.

KEY-GEN$(msk, \langle M, \pi_M \rangle, \mathcal{S})$

1 $\mathcal{I} = \{i : \pi_M(i) \in \mathcal{S}\}$
2 $\bar{r}' \xleftarrow{\$} \mathbb{Z}_p^\tau$
3 **for** $j \in [\tau]$
4 $sk_{1,j} = [\bar{r}'_j] g_2$
5 $\bar{v} \xleftarrow{\$} \mathbb{Z}_p^{n_2 - 1}$
6 **for** $i \in \mathcal{I}$
7 $sk_{2,i} = [M_i(\alpha \parallel \bar{v})^T] g_1 + [\bar{r}'_{\rho(i)}] H(\pi_M(i))$
8 **return** $sk = (\{sk_{1,j}\}_{j \in [\tau]}, \{sk_{2,i}\}_{i \in \mathcal{I}})$

The encryptor calculates the ciphertext, using the policy's attribute set, $\mathcal{P}_\mathcal{A}$.

ENCRYPT$(mpk, \mathcal{P}_\mathcal{A})$

1 $s \xleftarrow{\$} \mathbb{Z}_p$
2 **for** $u \in \mathcal{P}_\mathcal{A}$
3 $ct_{1,u} = [s] H(u)$
$ct_2 = [s] g_2$
4 **return** $ct = (\{ct_{1,u}\}_{u \in \mathcal{P}_\mathcal{A}}, ct_2)$

The data is encrypted using symmetric encryption with a key derived from $d = mpk^s$.

If $\mathcal{S}$ satisfies $\langle M, \pi_M \rangle$ there exist constants $\{\gamma_i\}_{i \in \mathcal{I}}$ s.t. $\sum_{i \in \mathcal{I}} \gamma_i M_i = (1, 0, \ldots, 0)$.

DECRYPT($mpk, \mathcal{S}, \langle M, \pi_M \rangle, ct, sk$)

1 $\bar{\gamma} =$ CALCULATE-GAMMA($\langle M, \pi_M \rangle, \mathcal{I}$)
2 $X = \hat{b}(\sum_{i \in \mathcal{I}} [\gamma_i] sk_{2,i}, ct_2)$
3 $Y = \prod_{j \in \tau} \hat{b}(\sum_{i \in \mathcal{I}, \rho(i)=j} [\gamma_i] ct_{1,\pi(i)}, sk_{1,j})$
4 **return** $d = X/Y$

A.3 Attribute-Based Signatures

The Attribute-Based Signatures (ABS) scenario:

1. The certificate authority, CA, generates master secret key and the corresponding master public key. When requested, they also generate attribute keys for a signer using an agreed signing policy, $\mathcal{P}_\mathcal{S} = \langle M, \pi_M \rangle$. Who agrees the policy?
2. A signer has a key pair, (sk, pk), and uses the private key, sk, to bind their attributes to the policy, $\{\sigma_i = [sk] H_1(i \parallel h_m \parallel \pi_M(i))\}_{i \in \mathcal{I}}$, where $h_m = H_0(M)$ and $\mathcal{I}$ is the set of rows in M associated with the signer's attributes. Note that we set $pk = [sk] g_2$ and the binding of an attribute can be verified by checking that $\hat{b}(\sigma_i, g_2) \stackrel{?}{=} \hat{b}(H_1(i \parallel h_m \parallel \pi_M(i)), pk)$.
3. The signer's bound attributes (and confirmation that they legitimately have them) are sent to the certificate authority, CA, together with the public key, pk. The issuer verifies each σ_i using pk.
4. If all the tests pass, the CA issues the signer with with their set of (secret) attribute keys.
5. Given the policy, $\mathcal{P}_\mathcal{S}$, to be satisfied the signer uses their corresponding attribute keys to sign a message and sends the signature to the verifier.
6. The verifier uses the master public key to verify the signature against the policy, $\mathcal{P}_\mathcal{S}$.

Note: As Maji et al. emphasised, it should not be possible to combine keys for one policy with those for another to satisfy a third policy. For example if policy, $\mathcal{P}_1$ is $A \wedge B$ and $\mathcal{P}_2$ is $C \wedge D$ it should not be possible to use keys from these policies to satisfy $\mathcal{P}_3 = A \wedge C$, i.e. no collusion.

The certificate authority, CA, chooses the system parameters and their master secret key, msk.

SETUP

1 Choose $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{b}, g_1, g_2)$
2 Choose $H_0 : \{0,1\}^* \to \mathbb{Z}_p$ and $H_1 : \{0,1\}^* \to \mathbb{G}_1$.
3 $\alpha \stackrel{\$}{\leftarrow} \mathbb{Z}_p^*$
4 Compute the master public key
$mpk = \hat{b}(g_1, g_2)^\alpha$
5 Store the master secret key, $msk = \alpha$
6 **return** $(\mathcal{G}, H_0, H_1, mpk)$

A signing policy is agreed and this fixes the MSP, $\langle M, \pi \rangle$. A signer has a set of attributes, $\mathcal{S}$, and binds these to the policy using SK.

BIND-ATTRIBUTES($\mathcal{S}, \langle M, \pi_M \rangle$, SK)

$\mathcal{I} = \{i : \pi_M(i) \in \mathcal{S}\}$
$h_m = H_0(M)$
for $i \in \mathcal{I}$
    $\sigma_i = [\mathsf{SK}]H_1(i \parallel h_m \parallel \pi_M(i))$
$\mathsf{PK} = [\mathsf{SK}]g_2$
return BA=($\mathcal{I}$, $\{\sigma_i\}_{i\in\mathcal{I}}$, PK)

To obtain their keys the signer sends BA to the CA together with proof that they hold the attributes (possible in the form of a verifiable presentation).

KEY-GEN($msk, \langle M, \pi_M \rangle$, BA)

Extract $\mathcal{I}, \{\sigma_i\}_{i\in\mathcal{I}}$ and pk from BA
$h_m = H_0(M)$
$r \xleftarrow{\$} \mathbb{Z}_p$
$\bar{v} \xleftarrow{\$} \mathbb{Z}_p^{n_2-1}$
$sk_1 = [r]g_2$
for $i \in \mathcal{I}$
    Check that $\hat{b}(\sigma_i, g_2) \stackrel{?}{=} \hat{b}(H_1(i \parallel h_m \parallel \pi_M(i)), pk)$ and fail otherwise
    $sk_{2,i} = [M_i(\alpha \parallel \bar{v})^T]g_1 + [r]\sigma_i$
return $sk = (sk_1, \mathcal{I}, \{sk_{2,i}\}_{i\in\mathcal{I}})$

The ABS signature includes an SPoK of the attribute keys and so does not reveal which attributes were being used.

SIGN($msg, \mathcal{P}_\mathcal{S}, \langle M, \pi_M \rangle, \mathcal{S}, sk_1, \{sk_{2,i}\}_{i\in\mathcal{I}}$)

$\bar{\gamma} =$ CALCULATE-GAMMA($\mathcal{P}_\mathcal{S}$, $\mathcal{S}$)
$\mathcal{I} = \{i : \pi_M(i) \in \mathcal{S}\}$
$h_m = H_0(M)$
$s, t, r_\alpha, \{r_i\}_{i\in[n_1]} \xleftarrow{\$} \mathbb{Z}_p^{n_1+3}$
$\sigma_1 = [sk \cdot t]sk_1$
$\sigma_2 = \sum_{i\in\mathcal{I}}[\gamma_i \cdot s]sk_{2,i}$
$\sigma_3 = \sum_{i\in\mathcal{I}}[\gamma_i \cdot s]H_1(i \parallel h_m \parallel \pi(i))$
$\sigma_4 = [t]g_2$
$Y = mpk^{s\cdot t}$
$Z = mpk^{r_\alpha}$
$W = \sum_{i\in[n_1]}[r_i]H_1(i \parallel h_m \parallel \pi(i))$
$c = H_0(\sigma_1, \sigma_2, \sigma_3, \sigma_4, Y, Z, W, msg)$
$s_\alpha = r_\alpha - s \cdot t \cdot c$
for $i \in [n_1]$
    $s_i = r_i - \gamma_i \cdot s \cdot c$
return $\sigma = (\sigma_1, \sigma_2, \sigma_3, \sigma_4, c, s_\alpha, \{s_i\}_{i\in[n_1]})$

Note that as $\gamma_i = 0$ when $\pi(i) \notin \mathcal{S}$ line 7 can also be written $\sigma_3 = \sum_{i \in [n_1]} [\gamma_i \cdot s] H_1(i \parallel h_m \parallel \pi(i))$.

To verify the ABS signature, the algorithm VERIFY$(mpk, \langle M, \pi_M \rangle, \sigma, msg)$ takes an ABS signature, the access policy $\langle M, \pi_M \rangle$ and a message msg and outputs 0 or 1.

VERIFY$(mpk, \langle M, \pi_M \rangle, \sigma, msg)$

1 $h_m = H_0(M)$
2 $Y' = \hat{b}(\sigma_2, \sigma_4) / \hat{b}(\sigma_3, \sigma_1)$
3 $Z' = mpk^{s_\alpha} \cdot Y^c$
4 $W' = \sum_{i \in [n_1]} [s_i] H(i || h_m || \pi(i)) + [c]\sigma_3$
5 $c' = H_0(\sigma_1, \sigma_2, \sigma_3, \sigma_4, Y', Z', W', msg)$
6 **if** $c' \neq c$
7 **return** 0;
8 **else**
9 **return** 1

Appendix B

List of Abbreviations

Abbreviation	Translation
5GAA	5G Automotive Association
5G	Fifth-generation (of) Cellular Networks Technology
ABAC	Attribute-Based Access Control
ABE	Attribute-Based Encryption
ABS	Attribute-Based Signature
ABSC	Attribute-Based Signcryption
AIV	Attestation and Integrity Verification
API	Application Programming Interface
ATL	Actual Trust Level
bps	Bits per Second
CA	Certificate Authority
CCAM	Connected, Cooperative, and Automated Mobility
CP-ABE	Ciphertext-Policy ABE
CPU	Central Processing Unit
DLT	Distributed Ledger Technology
E2E	End-to-End
ECU	Electronic Control Unit
EPC	Enclave Page Cache
gNB	gNodeB
GPU	Graphics Processing Unit
FFmpeg	Fast Forward Moving Picture Experts Group
FPS	Frame per Second
HW	Hardware
I/O	Input/Output
KP-ABE	Key-Policy ABE
LAN	Local Area Network
MEC	Multi-Access Edge Computing
ML	Machine Learning
MSP	Monotone Span Program

Abbreviation	Translation
NTP	Newtork Time Protocol
NTPD	Network Time Protocol daemon
NUC	Next Unit of Computing
OBU	On-Board Unit
OEM	Original Equipment Manufacturer
OS	Operating System
RA	Risk Assessment
RAM	Random-access Memory
RAN	Radio Access Newtork
RMS	Root Mean Square
RSTP	Real Time Streaming Protocol
RTL	Required Trust Level
SCB	Security Context Broker
SGX	Software Guard Extensions
SMTD	Slow Moving Traffic Detection
SW	Software
TAF	Trust Assessment Framework
TAM	Trust Assessment Manager
TCH	Trustworthiness Claims Handler
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TO	Task Off-loading
TMI	Trust Model Instance
TMT	Trust Model Templates
UE	User Equipment
UTC	Coordinated Universal Time
VC	Verifiable Credentials
VP	Verifiable Presentation

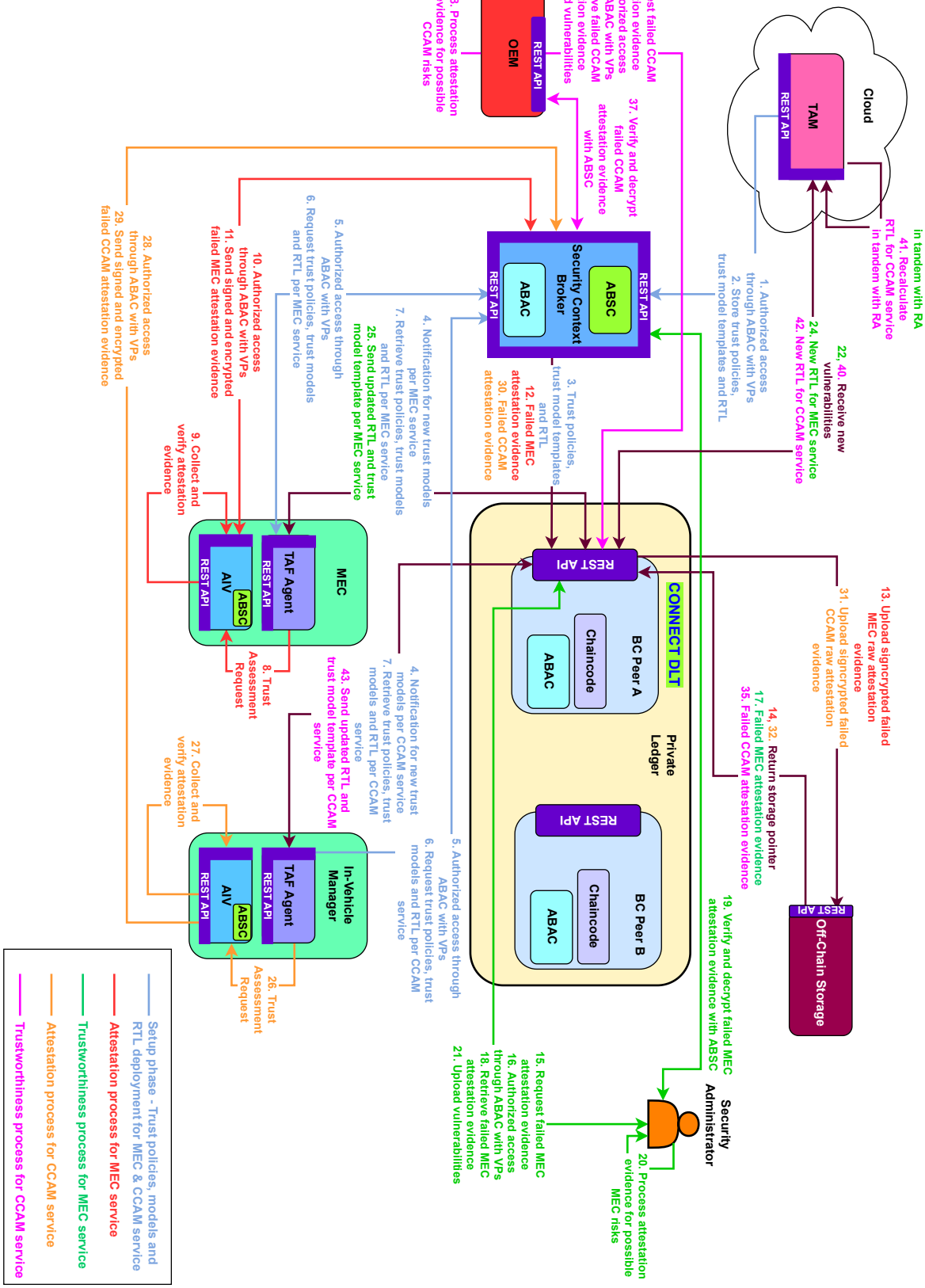

Figure B.1: Larger Figure for the DLT Architecture

References

[1] Linux ifstat. https://man7.org/linux/man-pages/man8/ifstat.8.html. Accessed: 2024-03-30.

[2] The miracl core cryptographic library. https://github.com/miracl/core. Accessed: 2025-06-23.

[3] June) 5G Automotive Association (2024. Safety treatment in connected and automated driving functions – phase 2. https://5gaa.org/content/uploads/2025/02/5gaa-wi-sticad-technical-report.pdf/.

[4] 5GAA Automotive Association. C-v2x Use Cases and Service level Requirements Volume I. Technical report, 2020. Accessed: 2025-06-28.

[5] Guido A. Gavilanes Castillo et al. Latency assessment of an ITS safety application prototype for protecting crossing pedestrians. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5, 2020.

[6] The CONNECT Consortium. Distributed processing and CCAM trust functions offloading & data space modelling. CONNECT Deliverable D5.1, Project 101069688 within HORIZON-CL5-2021-D6-01, November 2023. https://horizon-connect.eu/public-deliverables/.

[7] The CONNECT Consortium. Distributed processing, fast offloading and MEC-enabled orchestrator. Deliverable D5.2, Project 101069688 within HORIZON-CL5-2021-D6-01, Mar. 2024.

[8] The CONNECT Consortium. Integrated framework (first release) and use case analysis. Deliverable D6.1, Project 101069688 within HORIZON-CL5-2021-D6-01, Nov. 2024.

[9] The CONNECT Consortium. Trust & risk assessment and CAD twinning framework (initial version). CONNECT Deliverable D3.2, Project 101069688 within HORIZON-CL5-2021-D6-01, February 2024. https://horizon-connect.eu/public-deliverables/.

[10] The CONNECT Consortium. Virtualization- and edge-based security and trust extensions (first release). CONNECT Deliverable D4.2, Project 101069688 within HORIZON-CL5-2021-D6-01, January 2024. https://horizon-connect.eu/public-deliverables/.

[11] The CONNECT Consortium. Trust & risk assessment and CAD twinning framework (final version). CONNECT Deliverable D3.3, Project 101069688 within HORIZON-CL5-2021-D6-01, June 2025. https://horizon-connect.eu/public-deliverables/.

[12] The CONNECT Consortium. Virtualization- and edge-based security and trust extensions (final release). Deliverable D4.3, Project 101069688 within HORIZON-CL5-2021-D6-01, Mar. 2025.

[13] Intel Corporation. Intel software guard extensions programming reference, Publisher: Intel San Jose, CA, USA, 2014. https://www.intel.com/content/dam/develop/external/us/en/documents/329298-002-629101.pdf.

[14] Amina Elbatoul Dinar, Boualem Merabet, and Samir Ghouali. NTP server clock adjustment with Chrony. In Jyotsna K. Mandal, Somnath Mukhopadhyay, and Alak Roy, editors, *Applications of Internet of Things*, pages 177–185, Singapore, 2021. Springer Singapore.

[15] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 89–98, New York, NY, USA, 2006. Association for Computing Machinery.

[16] Sanjit Kaul, Roy Yates, and Marco Gruteser. On piggybacking in vehicular networks. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, pages 1–5, 2011.

[17] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 429–448, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[18] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 568–588, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[19] Zhen Liu, Zhenfu Cao, and Duncan S. Wong. Efficient generation of linear secret sharing scheme matrices from threshold access trees. Cryptology ePrint Archive, Paper 2010/374, 2010. https://eprint.iacr.org/2010/374.

[20] Doreen Riepel and Hoeteck Wee. FABEO: Fast Attribute-Based Encryption with Optimal Security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2491–2504, New York, NY, USA, 2022. Association for Computing Machinery.

[21] Teemu Rytilahti, Dennis Tatang, Janosch Köpper, and Thorsten Holz. Masters of time: An overview of the NTP ecosystem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 122–136, 2018.

[22] Ahmed Samy, Ibrahim A. Elgendy, Haining Yu, Weizhe Zhang, and Hongli Zhang. Secure task offloading in blockchain-enabled mobile edge computing with deep reinforcement learning. *IEEE Transactions on Network and Service Management*, 19(4):4872–4887, 2022.

[23] Henning Schulzrinne, Anup Rao, Rob Lanphier, Magnus Westerlund, and Martin Stiemerling. Real-time streaming protocol version 2.0. Technical report, RFC7826, 2016.

[24] V. Subhash. *Quick Start Guide to FFmpeg: Learn to Use the Open Source Multimedia-Processing Tool like a Pro*. Apress Berkeley, February 2023. https://www.springerprofessional.de/en/quick-start-guide-to-ffmpeg/23985372.