NTNU
Norwegian University of
Science and Technology

Public Key Infrastructure in Cooperative Intelligent Transport Systems

Sindre Trandum

Master of Science in Telematics - Communication Networks and Networked
Submission date: June 2018
Supervisor: Peter Herrmann, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title:	Public Key Infrastructure in C-ITS
Student:	Sindre Trandum

Problem description:

Vehicles and traffic infrastructure today is rapidly becoming more and more connected, using technology to enable users to utilize transport networks more safely and efficiently. Cooperative Intelligent Transport Systems (C-ITS) describes a domain where smart vehicles and infrastructure can interact and share data directly with each other. The planned introduction of C-ITS on European roads have the potential of revolutionizing how vehicles interact with each other, and how traffic is organized. With vehicles and infrastructure exchanging real-time information, individual drivers and traffic managers can use the information to make better decisions. C-ITS can provide early warning against accidents, as well as an improved situation overview. The ambition is to significantly reduce the number of road traffic accidents, and increase the traffic flow on European roads.

C-ITS requires a high level of trust, as users will exchange traffic-messages, and base their actions on information received from strangers. The need for a way to establish trust and preserve the integrity of the system is evident. This thesis will explore how a Public Key Infrastructure (PKI) can enable C-ITS users to communicate in an authenticated way. A theoretical introduction to the system is presented, followed by a practical implementation of a PKI in a test network.

Responsible professor:	Peter Herrmann, IIK NTNU
Supervisor:	Bjørn Magne Elnes, Aventi

Abstract

Cooperative Intelligent Transport Systems (C-ITS) enable vehicles and infrastructure to exchange information and provide users with an enhanced situation overview. It can provide early warning in potentially dangerous situations, and help increase safety and efficiency on European roads. This thesis explores the use of Public Key Infrastructure (PKI) in a Cooperative Intelligent Transport System, and how it can be used to achieve authentication within the system. The first part of the thesis consists of a theoretical presentation of C-ITS and PKI. The overall structure of the trust hierarchy is discussed before the technical specifications and requirements of the system are introduced.

The second part describes an experimental implementation of a PKI in a C-ITS test network. By creating an ITS certificate hierarchy, and modifying the configuration of the Dynniq ITS units, message authentication was successfully introduced to the network. The certificate hierarchy was generated using both OpenSSL and itssec, where the use of the itssec tool proved successful. Multiple configurations were tested, and the results were analyzed using supplied applications and Wireshark. After the results were verified, the PKI was expanded to include an Authorization Authority. The use of an HTTP file-server and the MQTT protocol were explored in order to handle network authentication and certificate distribution. Both options were assessed as viable candidates for more extensive testing in the future.

Sammendrag

Kooperative Intelligente Transportsystemer (C-ITS) gir kjøretøy og infrastruktur muligheten til å utveksle informasjon for å oppnå et bedre situasjonsbilde. Systemet kan gi brukere advarsler i potensielt farlige situasjoner, og øke både sikkerheten og effektiviteten på europeiske veier. Denne oppgaven utforsker bruken av PKI som sikkerhetssystem i kooperative intelligente transportsystemer. Den første delen av oppgaven gir en teoretisk introduksjon til C-ITS, før den utforsker hvordan man kan introdusere autentisering i nettverket ved hjelp av digitale sertifikater. Den overordnede hierarkiske strukturen av sikkerhetssystemet blir presentert, samt tekniske spesifikasjoner på protokoll-nivå.

Den andre delen av oppgaven er praktisk og består av en implementasjon av PKI i et test-nettverk. Flere fremgangsmåter ble testet, før en vellykket implementasjon ble gjennomført ved hjelp av verktøyet *itssec*. Resultatene ble analysert i Wireshark, og flere tester ble gjennomført for å verifisere funnene. En Authorization Authority server ble også innført i nettverket, og to mulige løsninger ble testet. En HTTP-server og en MQTT-broker ble konfigurert for å se på løsninger for nettverks-autentisering og distribusjon av sertifikater. Begge løsningene ble vurdert som legitime kandidater for videre testing.

Preface

This thesis marks the conclusion of my Master's degree in Communication Technology from the Norwegian University of Science and Technology. The two years I have spent at the university has provided me with both knowledge and memorable experiences, and I will remember my time here as a combination of hard work and great fun.

Working with ITS has been very interesting, as I had little knowledge about the topic beforehand. There is a saying which goes "planning is everything, the plan is nothing." This was true during the work on this thesis, and it taught me something about adapting the plan to the current reality. I would like to thank my responsible professor, Peter Herrman, and my supervisor, Bjørn Magne Elnes, for their support and guidance during this thesis.

Contents

List of Figures **ix**

List of Acronyms **xiii**

1 Introduction **1**

1.1 Scope & Limitations 2

1.2 Aventi 2

1.3 Outline 3

2 Method **5**

3 Cooperative ITS **7**

3.1 Security in C-ITS 9

4 Public Key Infrastructure **11**

4.1 Public Key Cryptography 11

4.2 Digital Signatures 13

4.3 Digital Certificates 15

4.4 Public Key Infrastructure in C-ITS 17

4.5 Public Key Infrastructure Specifications 18

4.5.1 Protocol Layers & Headers 18

4.5.2 Secure Header & Trailer 20

4.5.3 Authorization levels and QoS 22

4.5.4 Threat Analysis 23

4.5.5 Cryptographic Requirements 24

4.6 Privacy 26

5 PKI Implementation **29**

5.1 Introduction 29

5.2 Test Network 30

5.3 Greenflow 34

5.4 Packet analysis 35

6 Authenticated CAM exchange **37**
6.1 Test 0: Enable Secure Mode 38
6.2 Test 1: Secure mode using a certificate hierarchy (OpenSSL) 39
6.3 Test 2: Secure mode using a certificate hierarcy (ITSSEC) 41
6.4 Test 3: Verification of the authentication 51
6.5 Test 4: Intermediate CA 52

7 Authorization Authority **53**
7.1 Authorization Authority 53
7.2 Apache HTTP Server 54
7.3 MQTT Server using TLS 58

8 Discussion & Conclusion **67**
8.1 Discussion 67
8.2 Conclusion 69
8.3 Future work 70

References **71**

Appendices

A OpenSSL Certificate Hierarchy **75**

B Wireshark CAM packet **81**

C AppendixC **83**

List of Figures

3.1 C-ITS Scenario [1] 7
3.2 DENM packet captured in Wireshark: vehicleBreakdown. 8

4.1 Public key message example. 11
4.2 Generic Digital Signature Model [2]. 14
4.3 Digital certificate [3]. 15
4.4 CA hierarchy [4]. 15
4.5 Trust model C-ITS [5]. 17
4.6 Protocols in a frame carrying a CAM payload. 18
4.7 BTP packet structure encapsulated in a lower layer frame [6]. 19
4.8 Security wrapping of CAM packet. 20
4.9 Unsecured GeoNetworking packet structure [7]. 21
4.10 Secured GeoNetworking packet structure [7]. 21
4.11 Digital Signature Operations in the protocol stack [8]. 21
4.12 Examples of ITS threats, attacks and countermeasures [7]. 23
4.13 Worst case scenario / stress test [8]. 24
4.14 NIST Recommended Key Sizes [9]. 25
4.15 Authorization Tickets Framework [10]. 27

5.1 Communication flows. 30
5.2 C-ITS test network. 31
5.3 Kapsch link test application. Three OBUs are visible. 32
5.4 Dynniq map display of ITS-units represented by blue dots in the Ngorongoro lab. 34
5.5 Show response: Feedback 34
5.6 Certificate overview. 35
5.7 CAM packet broadcast from the Dynniq OBU. The OBU is configured as a heavy truck. 35

6.1 CAM section in its-facilities.conf. 37
6.2 Simple PKI hierarchy. 39
6.3 Itssec generation of root key. 42

6.4 Itssec generation of root certificate. 43
6.5 Listing of available certificates. 44
6.6 Verification of hex digest. 44
6.7 Generation of RSU key. 45
6.8 Generation of RSU certificate. 46
6.9 The RSU certificate is not signed. 47
6.10 The RSU certificate is now signed by root. 47
6.11 Using secure copy to deploy the keys and certificates to the devices. . . 48
6.12 Greenflow certificate overview. 49
6.13 Wireshark unsecure CAM message (left) and secure CAM message (right). 50
6.14 Un-signed certificate: State: invalid. 51
6.15 Signed certificate hierarchy. 52

7.1 Authentication Authority Apache Setup. 54
7.2 Certificates hosted on the AA. 55
7.3 MQTT design in the test network. 59
7.4 Mosquitto MQTT broker running on port 1883. 60
7.5 MQTT Mosquitto verbose output. 65

A.1 Verification of the root CA certificate. 77
A.2 Partial view of the OBU certificate. The intermediate CA is listed as the issuer. 79

B.1 The expanded headers in a secure CAM packet from Wireshark. 82

List of Acronyms

AA Authorization Authority.

AT Authorization Ticket.

BTP Basic Transport Protocol.

CA Certificate Authority.

C-ITS Cooperative Intelligent Transport Systems.

CRL Certificate Revocation List.

CSR Certificate Signing Request.

DSA Digital Signature Algorithm.

EA Enrolment Authority.

ECC Elliptic-curve cryptography.

ECDSA Elliptic Curve Digital Signature Algorithms.

ECTL European Certificate Trust List.

EDCA Enhanced Distributed Channel Access.

ETSI European Telecommunications Standards Institute.

HSM Hardware Security Module.

IoT Internet of Things.

ITS Intelligent Transport Systems.

ITS-S Intelligent Transport Systems Station.

LLC Logical Link Control.

MAC Medium Access Control.

MQTT Message Queuing Telemetry Transport.

OBU On-Board Unit.

PKI Public Key Infrastructure.

QoS Quality of Service.

RSA Rivest Shamir Adleman.

RSU Roadside Unit.

SDK Software Development Kit.

TLM Trust List Manager.

TLS Transport Layer Security.

TVRA Threat, Vulnerability and Risk Analysis.

V2I vehicle-to-infrastructure.

V2V vehicle-to-vehicle.

WLAN Wireless local area network.

Chapter 1

Introduction

Cooperative Intelligent Transport Systems (C-ITS) is a communication system designed to allow road users to communicate with other vehicles and infrastructure. The use of intelligent transport systems is aimed to increase the safety and efficiency in traffic. By acting as a traffic management system, the aggregated information from the C-ITS can help optimize the traffic flow in congested areas. By providing situational awareness and decision support to road users, it can help to avoid accidents and decrease the severity of those that occur. 25,670 people lost their lives on EU roads in 2016, with 130,000 recorded as seriously injured [11]. In 2017 the European Transport Security Council (ETSC) published a briefing on C-ITS, stating that new technologies with clear road safety benefits are urgently needed [12].

The European Commission is acting as a coordinating authority in the effort to create both a legal and a technical framework for the introduction of intelligent transport systems on a large scale. An EU strategy on cooperative, connected and automated mobility was adopted in 2016, stating that "talking" vehicles may be deployed on European roads as early as 2019 [13]. Several major car manufacturers participate in the development of C-ITS, through membership in a Car 2 Car Communication Consortium [14]. This joint effort drives the development forward, facilitating the introduction of C-ITS. The US also has ongoing projects working to introduce similar systems, where the Department of Transportation is acting as a coordinating authority. They have proposed making ITS vehicle-to-vehicle (V2V) equipment mandatory in all new vehicles, making the deployment of this technology a clear priority [15].

The messages sent between vehicles may contain information about velocity, acceleration, heading, and position. Messages describing special events, such as accidents, will also be communicated. This makes it very important to verify and authenticate the messages, to avoid distribution of false information in the system, either by accident or malicious exploitation. Security and privacy is consequently a prerequisite for the deployment of a system of this caliber. The European Commission have together with stakeholders in the C-ITS domain decided to adopt a collective security and certificate policy, to facilitate the secure deployment and operation of C-ITS in Europe [5].

The first objective of this thesis is to give a theoretical presentation of how to achieve authentication and message integrity in C-ITS using PKI. Both C-ITS and PKI will be examined, before tying the two together. The second objective is to use a PKI to introduce authentication in a C-ITS test network. Using ITS equipment provided by Aventi Intelligent Communication, a C-ITS communication network will be configured. The primary focus of this thesis will be to create and implement a PKI for this network and enable the units to communicate in a secure and authenticated manner.

1.1 Scope & Limitations

For the first theoretical part, the scope is to provide an introduction to C-ITS and PKI and present the current design of the security infrastructure, as prescribed by the European Commission. There are several use-cases not covered in the policy documents, and one might think of security issues not covered in the available literature. These have been defined as out of scope, as the focus is on understanding the current framework for PKI in C-ITS, rather than picking it apart. For the second part, the objective is to implement a PKI in the C-ITS test network, using the available ITS equipment and available software. The PKI described in the policy documents is very comprehensive, and it is not likely that it can be reproduced in full in this thesis. Therefore, functionality is introduced incrementally, dictated by the overall progress of the implementation. The primary objective is to enable a secure CAM exchange between ITS units, through the use of digital certificates. The secondary objective is to expand the PKI to include entities handling authorization and authentication to the network.

1.2 Aventi

Aventi Intelligent Communications AS is an Oslo based company specializing in, among other things, automation in transport. Their ambition is to be ready to provide solutions for autonomous traffic systems when the first autonomous vehicles appear in commercial traffic on Norwegian roads. Aventi Intelligent Communications, together with the Norwegian Public Roads Administration, NTNU, and SINTEF will over the next five years perform R&D-projects to explore the possibilities and potential for Intelligent Transportation Systems on Norwegian roads. They are the proposing company of this thesis.

1.3 Outline

- **Chapter 2** Describes the methods used for both the theoretical and Practical parts of this thesis.
- **Chapter 3** Provides an introduction to C-ITS.
- **Chapter 4** Opens with an introduction to PKI, before covering the use of PKI in C-ITS.
- **Chapter 5** Covers the C-ITS test network used in the practical part of the thesis.
- **Chapter 6** Covers the process of achieving authenticated CAM exchange in the test network.
- **Chapter 7** Introduces the Authorization Authority and the two solutions (HTTP/MQTT) tested for certificate distribution.
- **Chapter 8** Discusses the results and concludes the report.

Chapter 2 Method

The first phase of this thesis was a literature study on C-ITS and the proposed security infrastructure. It was important to attain a satisfactory understanding of the system before planning any further. The starting point was the policy documents from EU, which outlines the framework and standards to be used in Europe. The most important document in the initial phase was the Security Policy & Governance Framework for Deployment and Operation of European Cooperative Intelligent Transport Systems (December 2017)[5]. For clarification and elaboration on specific key areas, papers discussing the topic were analyzed. The development of ITS has been underway for decades, and the system has been a target for many scientific publications. To ensure the relevance of the information found in the research papers, material published in 2015 or later was prioritized. Standards published by the European Telecommunications Standards Institute (ETSI) were used for documentation on the protocols and analysis of packets.

For the practical part of this thesis, I chose to use the experimental method combined with tests in an incremental process. The experimental method is used because of the deterministic relationship between the inputs to the system, and the produced output. The nature of the system is such that a cause (input) will always lead to the same effect (output). By manipulating variables in the configuration, it is possible to change the output, and later to recreate identical results. These system attributes make the experimental method preferable over other methods. The tests were conducted in order to confirm or refute the success of the implementation of security features. The results of the tests were observed through the use of the web-applications monitoring the system, providing quick feedback. Packet dumps of the C-ITS communication were available on the Kapsch units, enabling analysis of the communication flow in Wireshark.

One objective of the thesis is to implement a PKI in a C-ITS network. The initial state of the system was unauthenticated, meaning the units would accept CAM/DENM messages from all transmitting units in range. To reach the objective, functionality had to be introduced so that authenticated messages were identified and accepted. A series of tests was conducted, aimed at introducing secure CAM exchange to the network. After testing and verifying the secure mode, an Authorization Authority server was configured, and two approaches to the distribution of certificates were further explored. The secure CAM exchange and the Authorization Authority are described in chapters 6 & 7.

Chapter 3 Cooperative ITS

Intelligent Transport Systems (ITS) describes the digital technology used either in vehicles or transport infrastructure. Cooperative Intelligent Transport Systems focus on the communication and interaction between these systems, be it vehicle-to-vehicle (V2V), or vehicle-to-infrastructure (V2I)[12].

The underlying idea of C-ITS is for road users to send and receive valuable information, which can be used to safely and effectively react to external events on the road. Examples include potentially dangerous vehicle movement (collision avoidance), and environmental hazards [16].

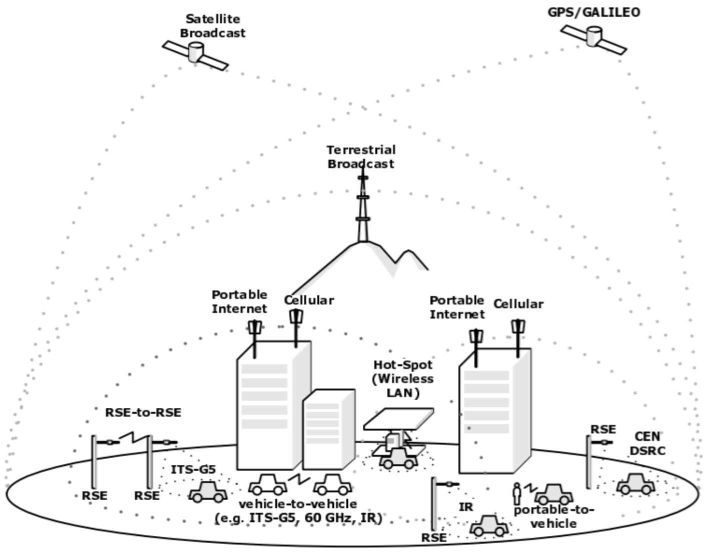
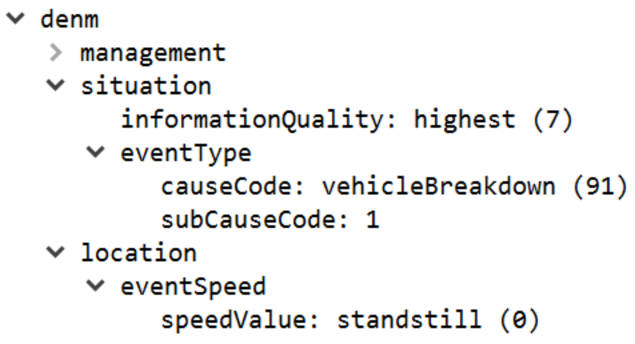


Figure 3.1: C-ITS Scenario [1]

Fig 3.1 illustrates different communication scenarios using various technologies to implement C-ITS. This thesis focuses on the ITS-G5 standard (wireless short-range communications), which uses the 5.9 GHz frequency band [17]. This technology combined with GPS can provide users with a 360 degree overview of similarly equipped vehicles with a communication range of approximately 300 meters.

Two essential components of a C-ITS network are the On-Board Unit (OBU) and the Roadside Unit (RSU). The OBU will be integrated into the vehicle by the manufacturer for new models, or retroactively fitted in older models. This is the unit that will broadcast messages from the vehicle, and act as the user's gateway to the network. The RSUs will be deployed throughout the infrastructure and provide information to the users, as well as collect information about the current traffic situation. The term Intelligent Transport Systems Station (ITS-S) is used to describe units equipped with either an OBU or an RSU. It is a collective term used to describe units in the C-ITS network where further clarification of role is not necessary.

In C-ITS there are two main types of messages transmitted between users, Cooperative Awareness Messages (CAM), and Decentralized Environmental Notification Messages (DENM). CAM messages are used by both OBUs and RSUs, and functions as an underlying information message used by applications and services. They can contain information about vehicle position, and basic vehicle data (acceleration, path history, curvature, vehicle size). Each C-ITS equipped vehicle broadcasts CAM messages continuously, and the sum of these messages form a situational picture of the traffic situation in the near geographical area. RSUs can use received CAM messages to estimate queue lengths at intersections, and provide a real-time image of the traffic. DENM messages function as hazard warnings and are triggered by specific events. They are only sent out for the duration of the triggering event. Examples of DENM events are accidents, emergency vehicle approaching, or roadwork.



```
denm
  management
  situation
    informationQuality: highest (7)
    eventType
      causeCode: vehicleBreakdown (91)
      subCauseCode: 1
  location
    eventSpeed
      speedValue: standstill (0)
```

Figure 3.2: DENM packet captured in Wireshark: vehicleBreakdown.

One scenario where these messages can potentially be lifesaving is in a multi-vehicle collision on the freeway. A user hitting the brakes hard while on the freeway can indicate that an accident is about to take place. This event can trigger the broadcast of a DENM message, warning the drivers coming from behind. When receiving an emergency message, the drivers can be alerted by either a visual message, an audio alarm, or a vibration in the seat. The alert can provide vital time to react for users not yet in visual range of the situation. The warning can quickly be propagated to oncoming vehicles, decreasing the chance of a chain collision. RSUs receiving the warnings can instantly alert authorities of the accident, decreasing their reaction time for deploying emergency services.

3.1 Security in C-ITS

The nature of road transportation is such that any communication network must be flexible and dynamic. The challenge in C-ITS is that any pair of vehicles will not have any prior knowledge about each other. The task of establishing credentials for authentication becomes complex, as strangers must decide whether or not to trust each other in a very short span of time. The V2V communication in C-ITS is one-way, with no handshake procedure to establish trust. Messages received are verified using the attached signature scheme, and either accepted or discarded. It is clear from reading the policy documents that security has been a requirement throughout the design process, and that the use of a PKI architecture has been the primary focus. The documents outline the objectives for the security system, but not how to achieve them. In chapters 6 and 7 potential solutions are explored. The critical element in C-ITS security is to assure the users that the received information comes from another vehicle, and enable them to trust the correctness of that information [18]. Chapter 4 explores how PKI can achieve this.

Chapter 4

Public Key Infrastructure

This chapter opens with a general overview of some of the most important features in a PKI, before focusing on how PKI can be implemented in C-ITS.

4.1 Public Key Cryptography

Public key encryption, also know as asymmetric encryption, is characterized by the use of public/private key pairs. Each user has one or more unique key pairs, where the private key is kept secret, and the public key is available to others. Through the use of cryptographic algorithms, the public key is used to encrypt messages, and the private key is used for decryption. In fig 4.1 Bob knows Alice's public key. He uses it to encrypt a message, and sends the ciphertext to Alice. She then uses the private (secret) key to decipher the message, and receives the plaintext.

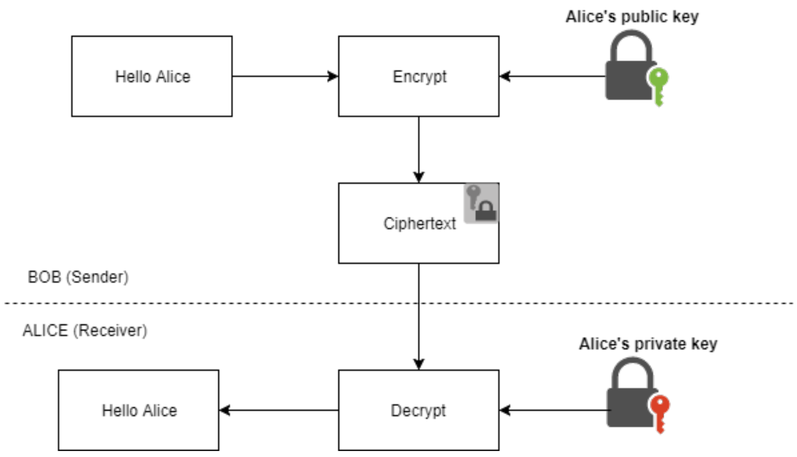


Figure 4.1: Public key message example.

One-way functions

This form of cryptography is possible because of one-way functions. One definition of a one-way function as a function f is [19]:

1. The description of f is publicly known and does not require any secret information for its operation.
2. Given x, it is easy to compute $f(x)$.
3. Given y, in the range of f, it is hard to find an x such that $f(x) = y$.

A one-way function is therefore easy to compute in one direction and infeasible to compute in the other direction [2]. Two examples which are used in public key encryption are:

1. Factoring problem: f(p,q)=pq, for randomly chosen primes p,q.
2. Discrete logarithm problem:

$f(p, g, x) =< p, g, g^x(modp) >$ for g a generator of Z_p^* for some prime p.

Hash functions

Hash functions are a form of one-way functions used to provide message integrity, i.e. detect any tampering or altering of the original message [20]. Cryptographic hash functions are mathematical algorithms that takes input of arbitrary length, and maps it to a fixed-length bit-string. The bit-string is called the hash or digest of the input.
For example: Using the hash function SHA-256, the input '**This secret string**' has the hash value of
'**2eb1e57914a653e2addf59e0d2d43a49fe8adc0273b299ddeb4a71a6fe1c3785**'.
The hash function is deterministic, i.e the same message will result in the same hash. The sender can create a hash value of the message, and attach it to the message. The receiver can perform the same hash function on the message and compare the results. If the hash values are identical, the receiver knows that the message has not been altered.

Trapdoor functions

Some one-way functions are called trapdoor functions [21], due to their property of being easy to compute in one direction, yet difficult to compute in the opposite direction (finding its inverse) without some secret value. The secret value in such functions is called the trapdoor.

This feature is utilized in public key cryptography, by using trapdoor one-way functions to provide confidentiality and authentication for the users. When used for encryption, the public key is used to encrypt the data using a one-way function. The private key contains the trapdoor value, and can find the inverse of the function and decrypt the data.

4.2 Digital Signatures

Since the private key is unique to the user, it can also be used for authentication in the form of a digital signature. To generate a digital signature, the sender encrypts the hash value of the message, using his private key. The recipient receives the encrypted hash value (the digital signature) attached to the message. He then computes the hash value off the message, and decrypts the digital signature using the sender's public key. If the two hash values match, the signature is considered valid. A digital signature is used as confirmation that a message has not been altered and as confirmation of the message sender's identity. The private key of the sender and the message contents are used as input to a cryptographic function to create the digital signature. The receiver can use the public key of the sender to validate the message contents, and to determine whether the message was sent by the claimed sender [2].

Suppose that Alice wishes to generate a signature on a message m [22].

Inputs:

– Alice's private signing key, KS.

– The message m

Output: Signature s = Sig(m, KS).

Suppose that Bob wishes to verify a claimed signature s on the message m.
Inputs:

– Alice's public verification key, KV .

– The message m

– The claimed signature s

Output: A boolean value Ver(m, s, KV) = true or false

Based on the true or false output from the verification algorithm, the receiver can accept or discard the message [22].

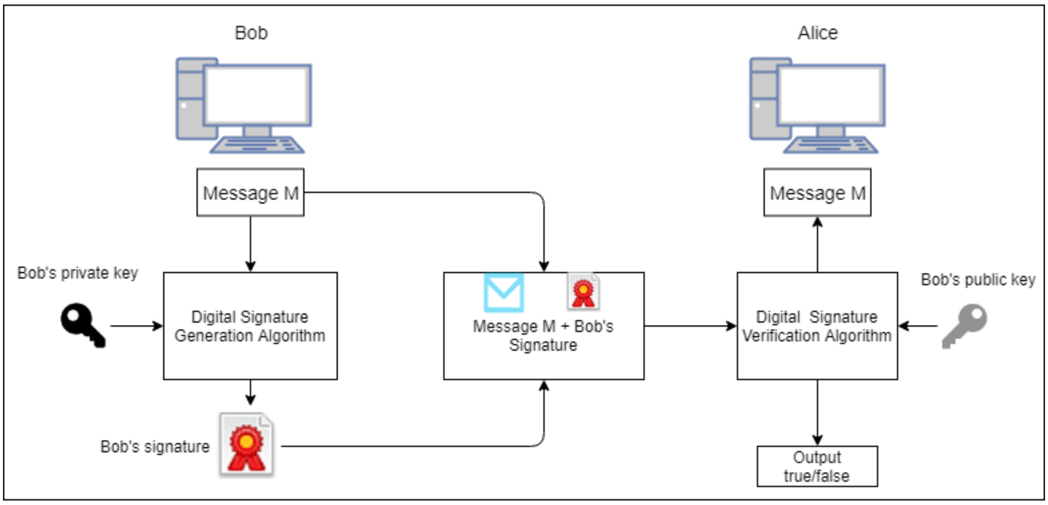

Bob
Alice
Message M
Message M
Bob's private key
Digital Signature Generation Algorithm
Message M + Bob's Signature
Digital Signature Verification Algorithm
Bob's public key
Bob's signature
Output true/false

Figure 4.2: Generic Digital Signature Model [2].

4.3 Digital Certificates

Digital signatures are key components in digital certificates. A digital certificate is analogous to a traditional identification card. They are electronic credentials issued by a CA (Certificate Authority), and they are used to certify the online identities of individual users and organizations [3].

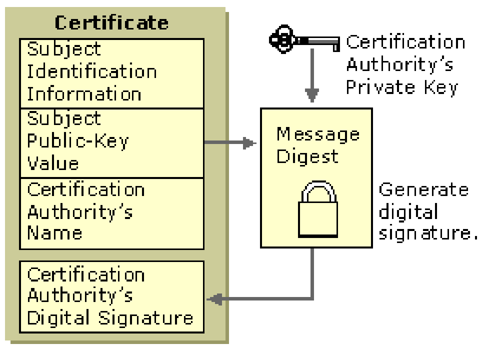

Certificate
Subject Identification Information
Subject Public-Key Value
Certification Authority's Name
Certification Authority's Digital Signature
Certification Authority's Private Key
Message Digest
Generate digital signature.

Figure 4.3: Digital certificate [3].

The use of CAs enable the formation a trust hierarchy, where a few trusted entities (root CAs) on tier 1 signs of on CAs on tier 2, which can issue certificates, or sign off on lower ranking CAs to expand the hierarchy. If a CA on tier three is compromised, the intermediate CA on tier two can revoke its certificate, limiting the potential damage.

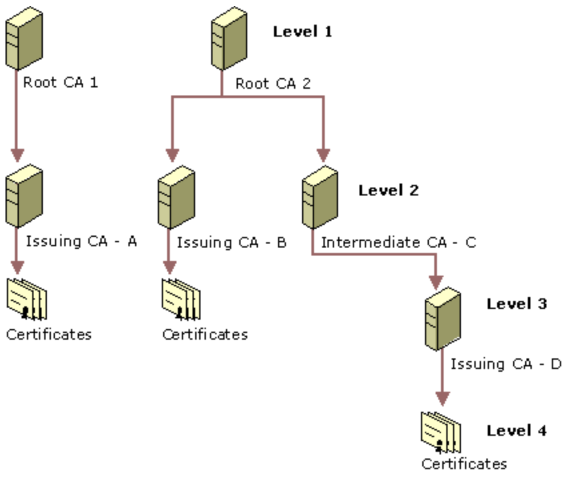


Figure 4.4: CA hierarchy [4].

Each user will have a list of trusted CAs, which it will use to validate other users. A digital certificate will contain the signature of a CA, and if that CA is in the list of trusted CAs, the authentication process can proceed.

Certificate Revocation

When signing a digital certificate, the signing authority always provides an expiration date. Based on the purpose of the certificate, the validity period of a certificate can vary from several years, to a few minutes. If a certificate is deemed not trustworthy before the expiration date, the CA can add it to a Certificate Revocation List (CRL). A CRL is a list of revoked certificates that has been issued, and then later revoked by a CA [23].

In C-ITS, the highest authority regarding trust will be the Trust List Manager (TLM) (fig 4.5). If any sub-tier CAs in the C-ITS PKI are compromised or fails meet the security requirements, the TLM can revoke their verification. The certificates signed by that CA will no longer be trusted.

The certificates used by vehicles will not be revoked by its corresponding CA [5]. These certificates will have a short lifetime, to minimize the risque of unauthorized usage.

4.4 Public Key Infrastructure in C-ITS

Public key infrastructure refers to a set of roles, policies, and procedures used to create, manage, distribute, store, and revoke digital certificates [24].

The C-ITS PKI infrastructure involves several entities with different responsibilities, required to provide the level of trust required in such a system. In the certificate policy for C-ITS, a description of the high level trust model is provided [5].

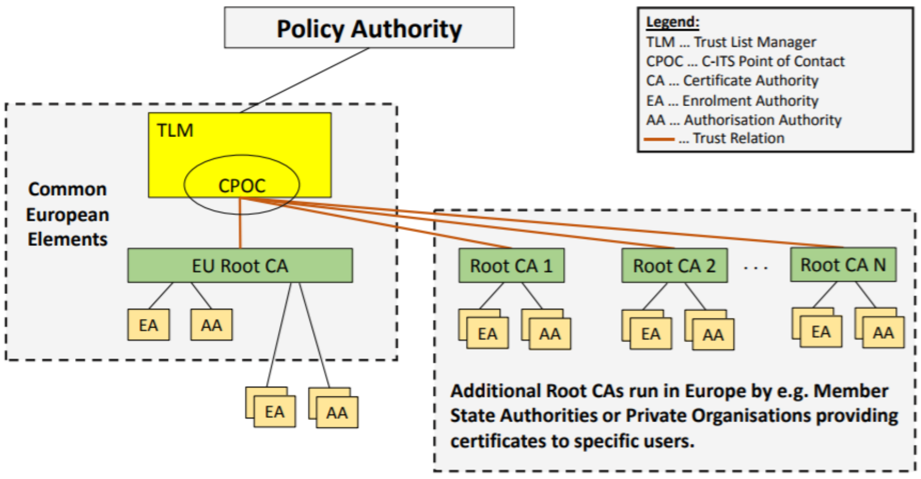

Policy Authority
Legend:
TLM ... Trust List Manager
CPOC ... C-ITS Point of Contact
CA ... Certificate Authority
EA ... Enrolment Authority
AA ... Authorisation Authority
... Trust Relation
TLM
CPOC
Common European Elements
EU Root CA
EA
AA
EA
AA
Root CA 1
Root CA 2
Root CA N
EA
AA
EA
AA
EA
AA
Additional Root CAs run in Europe by e.g. Member State Authorities or Private Organisations providing certificates to specific users.

Figure 4.5: Trust model C-ITS [5].

The ITS authority hierarchy consists of several entities, most notably the AA, EA, and Root CA. The EA (Enrolment Authority) is responsible for issuing a proof of identity authenticating an ITS-S. The ITS-S can use the proof of identity it received from the EA to request services from the AA (Authorization Authority). Each EA/AA hierarchy has a Root CA at the summit. This is the root of trust for all the certificates in that hierarchy. In order to trust an incoming message, the ITS-S must at least have access to the Root certificate of the hierarchy [25]. The trust hierarchy used in the practical part of the thesis will be based on this trust model.

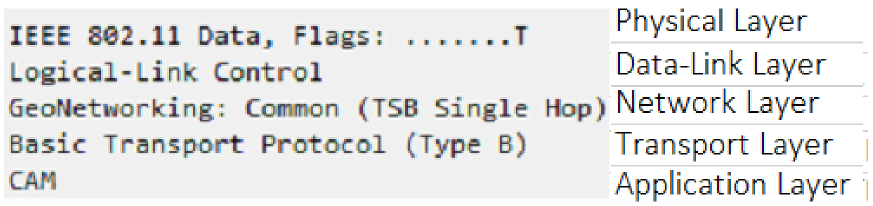
Above the Root CAs in the hierarchy is the Trust List Manager (TLM). The TLM manages the European Certificate Trust List (ECTL), managing the trust relations to all the Root CAs in the EU. The certificate management system for C-ITS will be enormous, and it is not yet determined which entity will have the authority as TLM. For interoperability between all the root CAs in this multinational system, a standardized format for certificates called X.509 has been chosen, as well as a set of cryptographic algorithms that must be supported [5].

4.5 Public Key Infrastructure Specifications

Sections 4.1 and 4.2 have described the framework and qualities of a generic PKI. In a new and complex system such as C-ITS, the design of the PKI must take into account all the attributes and parameters of the specific system. A comprehensive certificate policy have been published by the European Commission [5], outlining the requirements and specifications of PKI in C-ITS. This section provides a more detailed look at those specifications.

4.5.1 Protocol Layers & Headers

The structure of a frame carrying a CAM payload is depicted in 4.6. It is taken from a packet capture file during transmission without authentication.



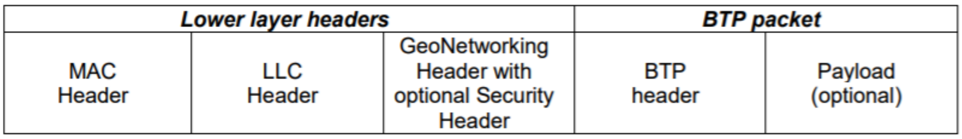
```
IEEE 802.11 Data, Flags: .......T
Logical-Link Control
GeoNetworking: Common (TSB Single Hop)
Basic Transport Protocol (Type B)
CAM
```

Figure 4.6: Protocols in a frame carrying a CAM payload.

The lowest layer in this protocol stack is the physical layer using the IEEE 802.11 standard. This standard is widely used for implementing WLAN communication in computer networks, commonly known as WiFi. To accommodate the requirements in wireless vehicular communication systems, an amendment called 802.11p was introduced. The amendment defines enhancements needed to support ITS, such as communication between high speed vehicles [26]. The second layer is the data-link layer which is divided into two sub-layers; medium access control (MAC) and logical link control (LLC). The MAC sub-layer interfaces with the physical layer and provides flow control and multiplexing. 802.11p uses the Enhanced Distributed Channel Access (EDCA) protocol, which can provide Quality of Service (QoS) for prioritized transmissions [27]. The LLC sub-layer interfaces with the network layer and provides flow control and multiplexing for the logical link. In ITS, these two layers are collectively referred to as the access layer [28]. The ITS-G5 standard covers the technology used in the access layer, designed to support data exchange between mobile stations in ad-hoc mode. The 5,855 - 5,925 GHz frequency band has been allocated to the use of ITS in Europe [28].

The GeoNetworking protocol is the network layer protocol in the stack, providing ad-hoc networking based on geographical addressing. The position input is provided by GPS. The policy documents outlining the use of the GeoNetworking protocol in C-ITS specify that the protocol shall provide secure communication, including authentication, authorization, integrity and non-repudiation. The GeoNetworking protocol supports cryptographic protection through the use of digital signatures and certificates [29].

The Basic Transport Protocol (BTP) provides an end-to-end, connection-less transport service in the ITS ad-hoc network. BTP is a lightweight protocol with a 4-byte header containing information about source and destination ports. The protocol allows services at the ITS facilities layer (CAM/DENM) to access the GeoNetworking protocol [6]. The payload in fig 4.6 is an unauthenticated CAM packet, interchangeable with DENM packets and other facility layer services.



Lower layer headers			***BTP packet***	
MAC Header	LLC Header	GeoNetworking Header with optional Security Header	BTP header	Payload (optional)

Figure 4.7: BTP packet structure encapsulated in a lower layer frame [6].

4.5.2 Secure Header & Trailer

The implementation of authentication and secure transmission is done at the network level, introducing a security header and trailer to the GeoNetworking protocol. The security header holds information used by the security layer when processing the packet, such as *signer_info*, *encryption_parameters*, and *recipient_info*. The trailer contains the information needed to verify the message integrity and authenticity [7]. The structure of both the header and trailer are defined in the standard ETSI TS 103 097 [30].

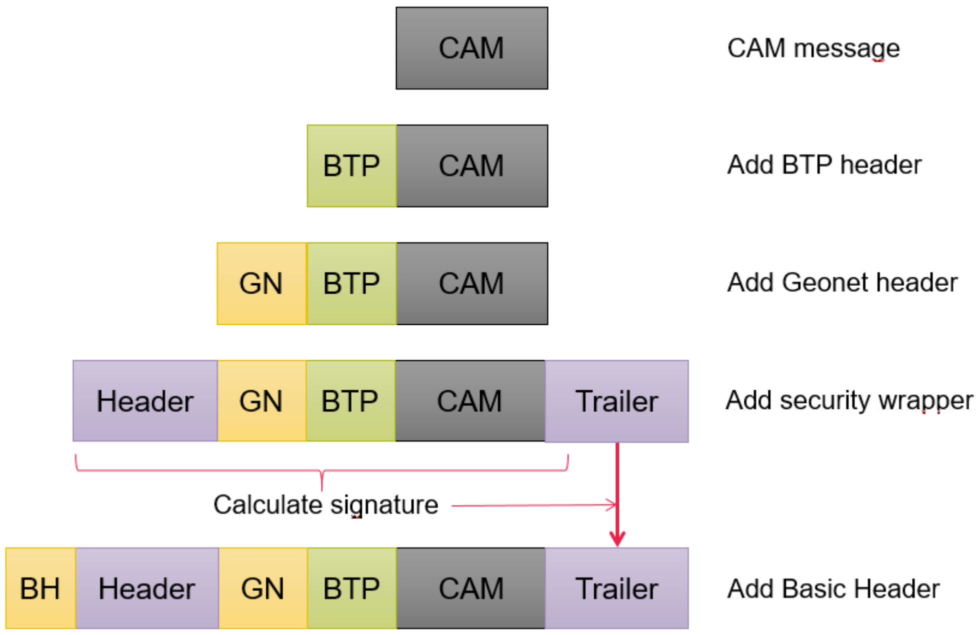
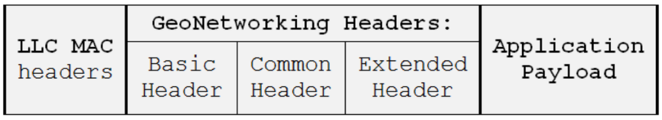


Figure 4.8: Security wrapping of CAM packet.

The signature is calculated after the GeoNetworking header is added, and consists of input from:

- Security header
- GeoNetworking header
- Basic Transport Protocol header
- Payload (CAM/DENM)
- First part of the security trailer

After the signature is calculated, it is stored in the security trailer. The receiver uses the information in the security trailer to authenticate the sender, and verify the integrity of the message. The security wrapper (fig 4.8) is a part of the GeoNetworking protocol, which contains information about the geographical position of the transmitting unit. The header is split into three sub-headers to facilitate the application of the security wrapper. When transmitting in secure mode, the GeoNetworking protocol adds a fourth secure header after the basic header.



LLC: Logical Link Control, MAC: Medium Access Control.

Figure 4.9: Unsecured GeoNetworking packet structure [7].

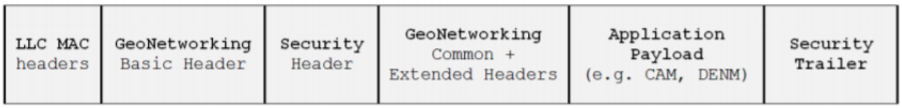

LLC MAC headers
GeoNetworking Basic Header
Security Header
GeoNetworking Common + Extended Headers
Application Payload (e.g. CAM, DENM)
Security Trailer

Figure 4.10: Secured GeoNetworking packet structure [7].

Figure 4.11 shows an overview of the digital signature operations, generation and verification, and how it is handled by the protocol layers. The facilities and application layers are unaware of security measures provided by the lower layers.

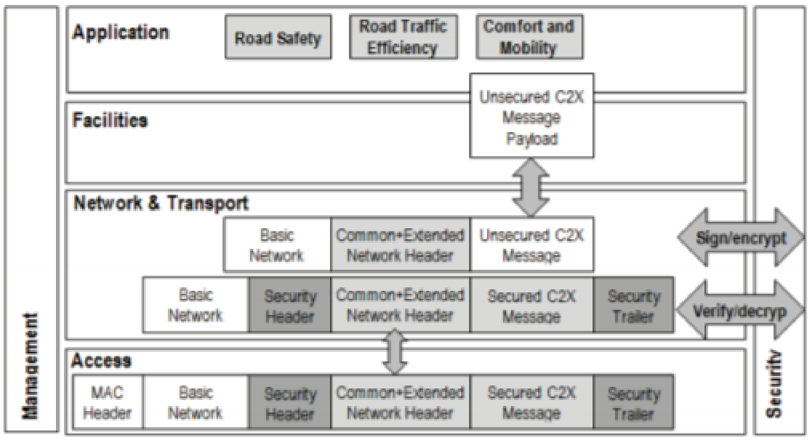


Figure 4.11: Digital Signature Operations in the protocol stack [8].

4.5.3 Authorization levels and QoS

Some applications and services in ITS require a higher level of authorization. This can for example be emergency services requesting the right of way and priority at a traffic light. The certificate determines its user's privileges and permissions to send certain types of messages, using the ITS-Application Identifier (ITS-AID) to indicate the authorization level [31]. The receiver accepts a signed CAM if the certificate is valid and the CAM is consistent with the ITS-AID in its certificate. Time-sensitive transmissions can be prioritized by the access layer using QoS. The use of Enhanced Distributed Channel Access in ITS-G5 introduces priority classes, making it possible for the network to reserve resources for critical transmissions [27].

4.5.4 Threat Analysis

When implementing a security infrastructure in a communications system, it is important to be aware of the potential threats towards the system. The European Telecommunications Standards Institute (ETSI) conducted a Threat, Vulnerability and Risk Analysis (TVRA) for communications in an Intelligent Transport System. The analysis considered vehicle-to-vehicle and vehicle-to-roadside network infrastructure communications services in the ITS [32].
In the report, they focused on five key categories and some of their associated attacks.

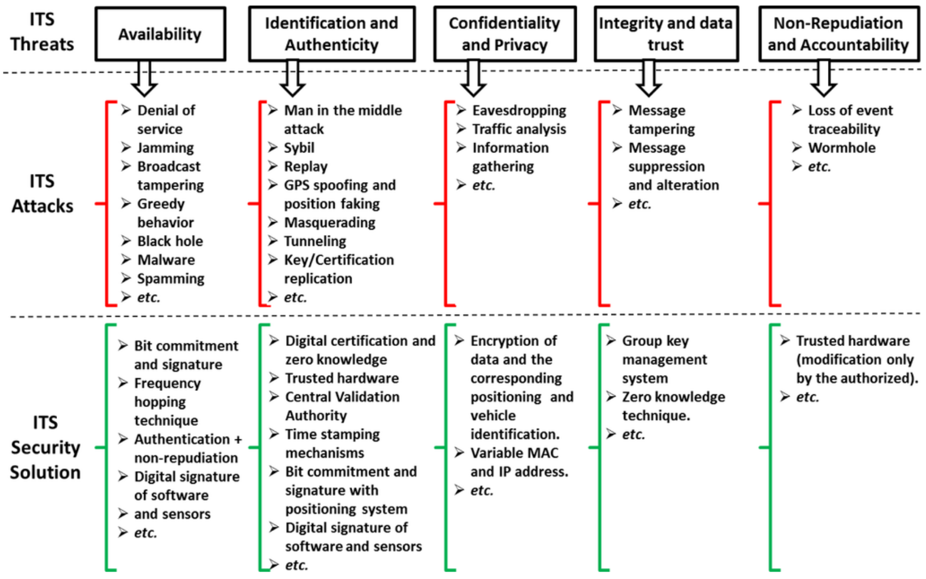

ITS Threats
Availability
Identification and Authenticity
Confidentiality and Privacy
Integrity and data trust
Non-Repudiation and Accountability
ITS Attacks
Denial of service
Jamming
Broadcast tampering
Greedy behavior
Black hole
Malware
Spamming
etc.
Man in the middle attack
Sybil
Replay
GPS spoofing and position faking
Masquerading
Tunneling
Key/Certification replication
etc.
Eavesdropping
Traffic analysis
Information gathering
etc.
Message tampering
Message suppression and alteration
etc.
Loss of event traceability
Wormhole
etc.
ITS Security Solution
Bit commitment and signature
Frequency hopping technique
Authentication + non-repudiation
Digital signature of software
and sensors
etc.
Digital certification and zero knowledge
Trusted hardware
Central Validation Authority
Time stamping mechanisms
Bit commitment and signature with positioning system
Digital signature of software and sensors
etc.
Encryption of data and the corresponding positioning and vehicle identification.
Variable MAC and IP address.
etc.
Group key management system
Zero knowledge technique.
etc.
Trusted hardware (modification only by the authorized).
etc.

Figure 4.12: Examples of ITS threats, attacks and countermeasures [7].

The two most important categories for this thesis are authenticity and integrity. The motivation for implementing a PKI security protocol is to ensure that only authenticated users have access to the ITS resources, and that messages are received correctly. The ITS messages will be broadcast and can be received by any ITS station within range of the signal. The content of a broadcast message is assumed to be of little interest to an eavesdropper, and encryption and decryption of the messages would add costly overhead. Therefore, the CAM and DENM packets are broadcast without confidentiality, with an attached digital signature scheme. The decision not to prioritize confidentiality in the design of the security affects the privacy settings in the system. This is addressed in section 4.6.

4.5.5 Cryptographic Requirements

All ITS stations shall be able to use the private key for signing operations, in addition to verification of received messages. CAM and DENM messages are broadcast, and in congested areas each ITS station will have to generate signatures and verify incoming messages at a very high rate. Fig. 4.13 shows a scenario designed to stress test the capacity of an ITS station. It is a worst case scenario with a traffic-jam on a four-lane intersection, where all the cars in communication range (300 m) are broadcasting ITS-messages.

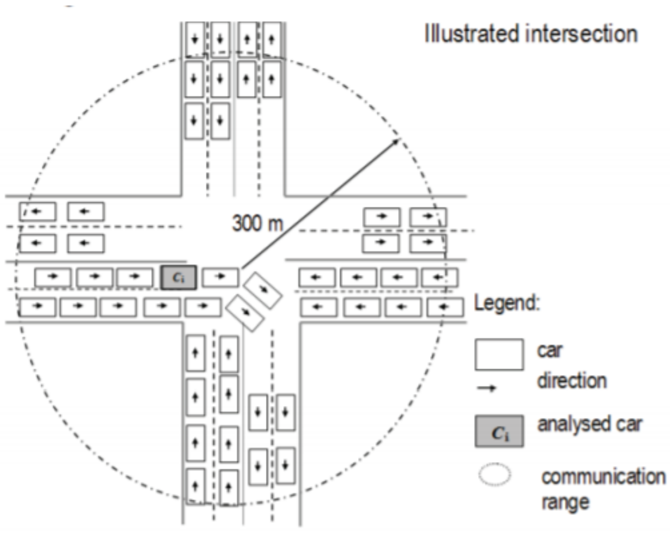


Figure 4.13: Worst case scenario / stress test [8].

In this scenario with 800 cars in range, each generating a CAM message every 300 ms [8], the unit must be able to verify 2400 messages per second. This affects the design of the system, putting hard demands on the hardware and software handling the cryptographic processing. The ITS devices installed in vehicles and infrastructure might have limited computational capacity and memory available. The OBU used in the test-network of this thesis uses an ARMv7 quad core CPU and has 1 GB of RAM available, significantly less than an ordinary laptop computer. It is therefore critical to select effective digital signature schemes, minimizing time spent processing messages.

The processing time of the safety header and trailer $T_{HT}(M)$ (Fig 4.11) is defined as [8]:

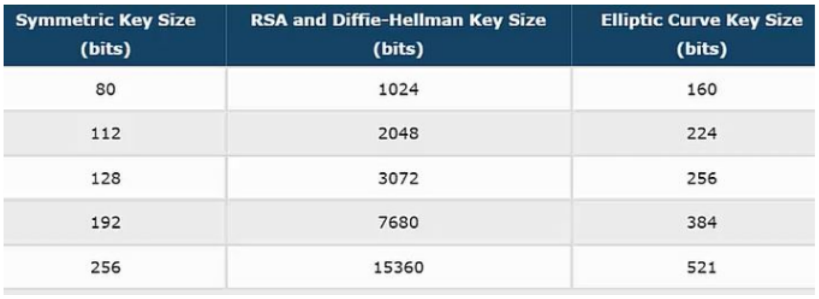
$$T_{HT}(M) = T_{sign}(M) + T_{tx}(Sign_{PrKv}[M]) + T_{verify}(M).$$

– $T_{sign}(M)$ is the attachment of time stamp T.

– $T_{tx}(Sign_{PrKv}[M])$ is the time needed to transfer a signed message including addition of certification data from the CA.

– T_{verify} is the time needed to verify a signature.

Both $T_{tx}(Sign_{PrKv}[M])$ and T_{verify} are affected by the algorithm used in these operations. In section 6.1.4 of the certificate policy document [5], several cryptographic algorithms are specified for use in generating keys and verification. They are all Elliptic Curve Digital Signature Algorithms (ECDSA), a variant of the Digital Signature Algorithm (DSA) which uses elliptic curve cryptography [2]. ECDSA has short key lengths compared to RSA[1] and other DSA schemes. For the same level of security, an RSA key length of 1024 bits corresponds to an ECDSA key length of 160 bits [33]. The short key lengths and computational complexity make ECDSA cryptographic schemes well suited for use in ad-hoc vehicular networks such as C-ITS [8].



Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Figure 4.14: NIST Recommended Key Sizes [9].

[1]The Rivest-Shamir-Adleman (RSA) scheme has since [its publication] reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption [2].

4.6 Privacy

The privacy of the users is a big concern when implementing C-ITS. It's not hard to imagine how a system receiving information about position from vehicles could be used to track individual users. The travel pattern of a private vehicle can reveal a lot of personal information, so naturally people are sensitive about sharing their location at all times. The EU has with the introduction of GDPR [34] signalized an increased focus on user data protection, and privacy is a key point in the design of C-ITS.

CAM and DENM messages contain information which can be linked to position and IDs of users, making it necessary to implement features to preserve the data privacy. One approach to this issue is to introduce pseudonymity, which can be described as the ability to use a resource without revealing the user's identity, but maintaining accountability for the use of the resource [18]. To achieve pseudonymity of ITS-users, the C-ITS PKI is designed to utilize Authorization Tickets (AT). ATs are certificates generated by an Authentication Authority, used by the vehicles to authenticate to the C-ITS network without disclosing sensitive information. Data that can be used to identify a single user is called Personal Identifiable Information (PII), and use of ATs in the PKI shall ensure that PII linking an ITS-station to the real identity of the user is not broadcast during regular use of an ITS-station. The implementation of pseudonymity through ATs is intended to prevent tracking of users, either by the authorities managing the system, or by illicit actors using broadcast information for unauthorized surveillance [10].

This design requires that an ITS unit must have some information available before it can start the process of joining the C-ITS network [25]. This initial information includes, but is not limited to:

- A canonical identifier which is globally unique.
- Public/private key pair for cryptographic purposes.
- Contact information for the EA and AA which will issue certificates for the ITS-S.

In addition, the EA must have access to a permanent unique identifier of the ITS-S, and its public key [25]. How the distribution of this information will be handled is not clear, but one likely scenario is that the manufacturer of an ITS-S, e.g vehicle, will be responsible for providing the required information for the initial authentication.

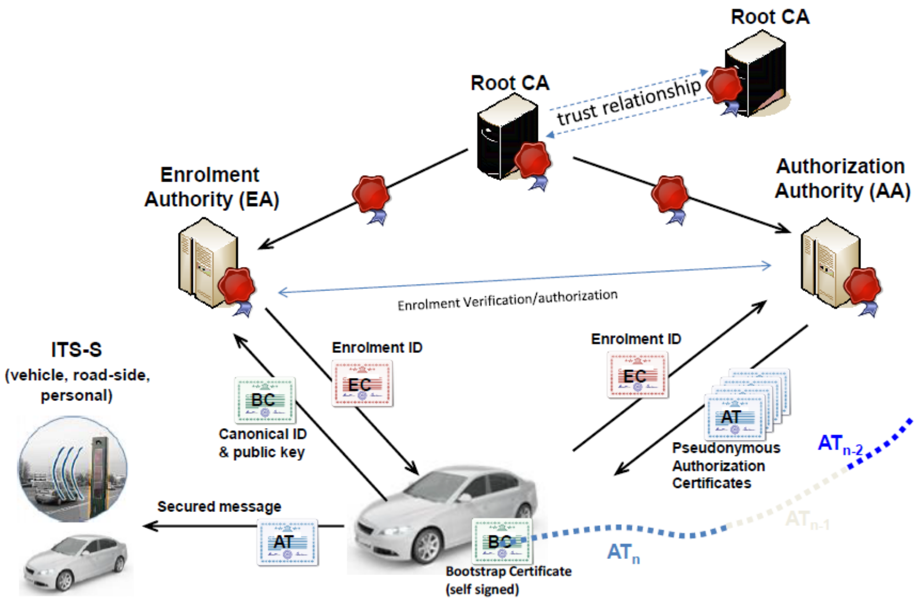

Root CA
Root CA
trust relationship
Enrolment Authority (EA)
Authorization Authority (AA)
Enrolment Verification/authorization
ITS-S (vehicle, road-side, personal)
Enrolment ID
EC
Enrolment ID
EC
BC
Canonical ID & public key
AT
Pseudonymous Authorization Certificates
AT n-2
AT n-1
AT n
Secured message
AT
BC
Bootstrap Certificate (self signed)

Figure 4.15: Authorization Tickets Framework [10].

In C-ITS PKI architecture, it is the Authorization Authority (AA) that provides the vehicles with pseudonymous ATs. Figure 4.15 illustrates the message flow. The Enrolment Authority (EA) authenticates the user, and provides a token in the form of an Enrolment ID. The user uses this token to authenticate himself to the AA, and receives Pseudonymous Authorization Certificates in return. Since the EA and AA are two separate entities, both physically and organizationally, the ATs will not be directly linkable to the real identity of the users.

In the documents describing privacy and security in C-ITS, one solution of how to utilize ATs in practice has been outlined [10]. The general idea is to separate each trip into three unlinkable segments, and to modify the identity of the ITS-station during movement [18].

- The first segment from the start of a trip, i.e. a location relevant to an individual, to the mid segment.
- The mid segment, where location data are anonymous because they cannot be associated to a location relevant to an individual.
- The last segment that connects the mid segment to the end of the trip, i.e. a location relevant to an individual.

This means that during a single trip, a vehicle will use multiple ATs in order to preserve pseudonymity. In the current design, the vehicle manages a pool size of 100 ATs with a validity period of 1 week [10]. The ATs are selected randomly from the pool with equal probability and without replacement. The pool re-starts when it is completely empty. In chapter 7 two solutions for the AA have been explored.

Quick Summary

C-ITS is a vehicular ad hoc network designed to allow the exchange of digital information between vehicles and traffic infrastructure. The network requires security which is introduced through the use of public key cryptography. The public key infrastructure in C-ITS will use digital certificates signed by certificate authorities to enable message authentication and integrity. The security is implemented in the GeoNetworking protocol on the network layer, using a secure header and trailer. The signature scheme is based on Elliptic Curve Cryptography, and privacy is maintained with pseudo anonymous Authorization Tickets.

Chapter 5

PKI Implementation

5.1 Introduction

The purpose of the theoretical presentation of C-ITS in part one was to explore the design and requirements of a C-ITS security system, and illuminate how PKI can help solve security-related challenges. The introduction of intelligent transport systems on EU roads faces many complex issues, and security is one of the most important ones. By presenting the need for authentication in C-ITS, and outline how PKI can enable an authenticated exchange of information, the practical implementation in part two has hopefully been adequately motivated.

The ambition of the practical portion of this thesis is to explore how to introduce a PKI to an established C-ITS test network, and demonstrate how authentication can be achieved through the use of signed digital certificates. Three central communication flows in the PKI were identified. They are, in order of priority:

1. Authenticated CAM exchange between OBUs/RSUs.
2. Certificate exchange between ITS-S and Authorization Authority.
3. Authentication request/reply between ITS-S and EA.

The C-ITS PKI uses the communication flows to authenticate and authorize users. I will use the test network to explore how this design can be realized.

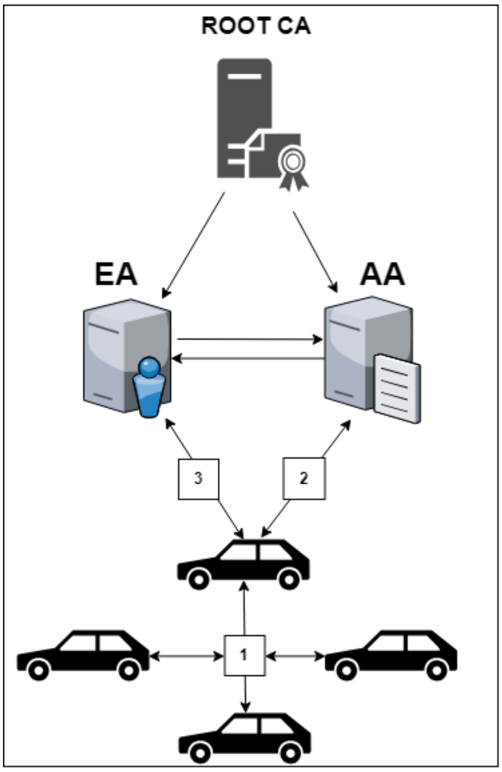


Figure 5.1: Communication flows.

5.2 Test Network

The test network initially consisted of two OBUs and one RSU from Kapsch[1]. These units were set up in the lab, and some time was spent testing and researching how to enable authentication in their communication. After some time it became clear that the Kapsch equipment did not have firmware supporting authentication through the use of certificates. Aventi decided to acquire new equipment, this time from Dynniq, so that the work could continue.

[1]The Kapsch Group is an international Road Telematics, Information Technology and Telecommunications Company.

After receiving the equipment from Dynniq, it was added to the existing network and made ready for further testing. In addition to the RSU and two OBUs already in place, one OBU and one RSU were added to the test network.

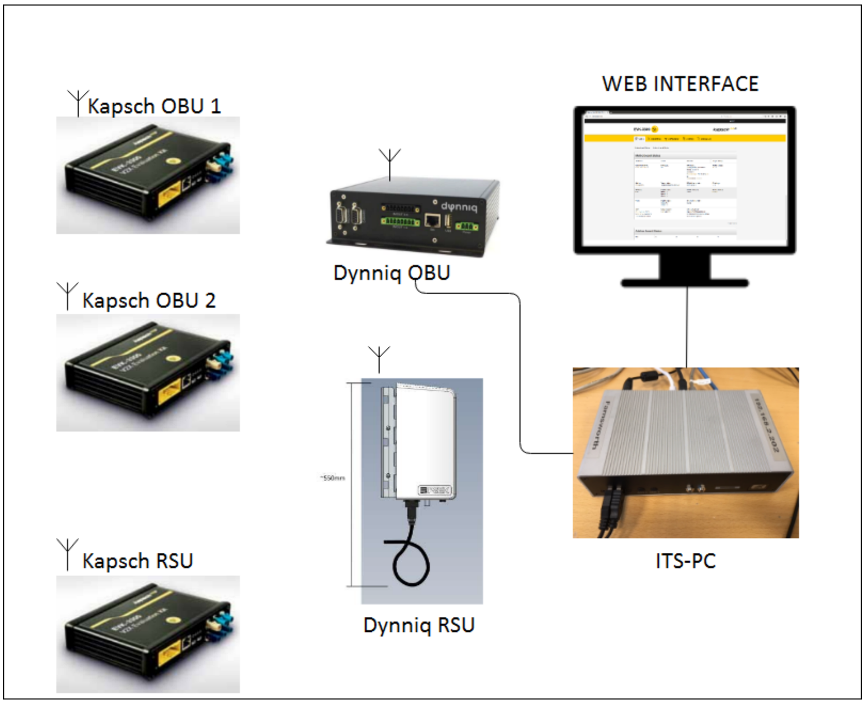


Figure 5.2: C-ITS test network.

Both the Kapsch and the Dynniq units arrived pre-configured, ready to transmit C-ITS messages. As background and documentation, Aventi provided an installation and configuration manual. After connecting the units to a power source, they were ready to transmit CAM packets. Before they would transmit, however, the GPS antennas had to have a signal. The antennas were placed in the window frame in the lab, to provide the best possible conditions for the GPS signal. The Kapsch units also provided a Bluetooth interface, which was used to connect the OBUs to a monitoring application. This application (fig 5.3) was used to monitor the configuration, and verify that the units were communicating through CAM messages.

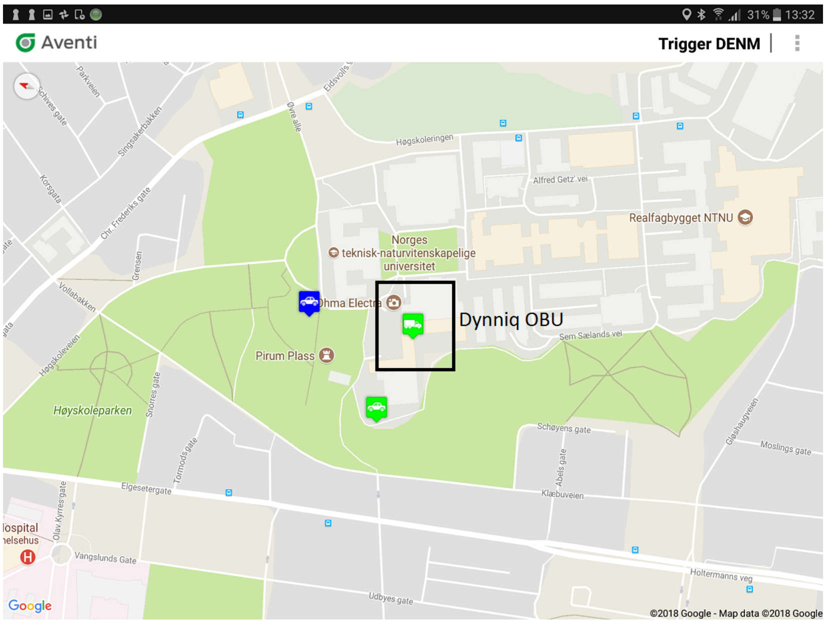


Figure 5.3: Kapsch link test application. Three OBUs are visible.

The Kapsch units are as previously mentioned not able to authenticate messages by using certificates. It was decided to keep them in the network, to have a basis for comparison for when enabling a secure mode of communication.

All the ITS-stations in the test network have a web interface, which can be reached by connecting the ITS-PC to the unit with a network cable. The Dynniq equipment was configured on a different subnet than the Kapsch units, making it necessary to change the IP-address of the ITS-PC, depending on which network it was connected to.

IP Network	
Unit Name	IP address
Kapsch OBU 1	192.168.2.74
Kapsch OBU 2	192.168.2.172
Kapsch RSU	192.168.2.54
Dynniq OBU	172.16.1.2
Dynniq RSU	172.16.1.4
ITS-PC (Kapsch)	192.168.2.202
ITS-PC (Dynniq)	172.16.1.6

Table 5.1: IP addresses used in test network.

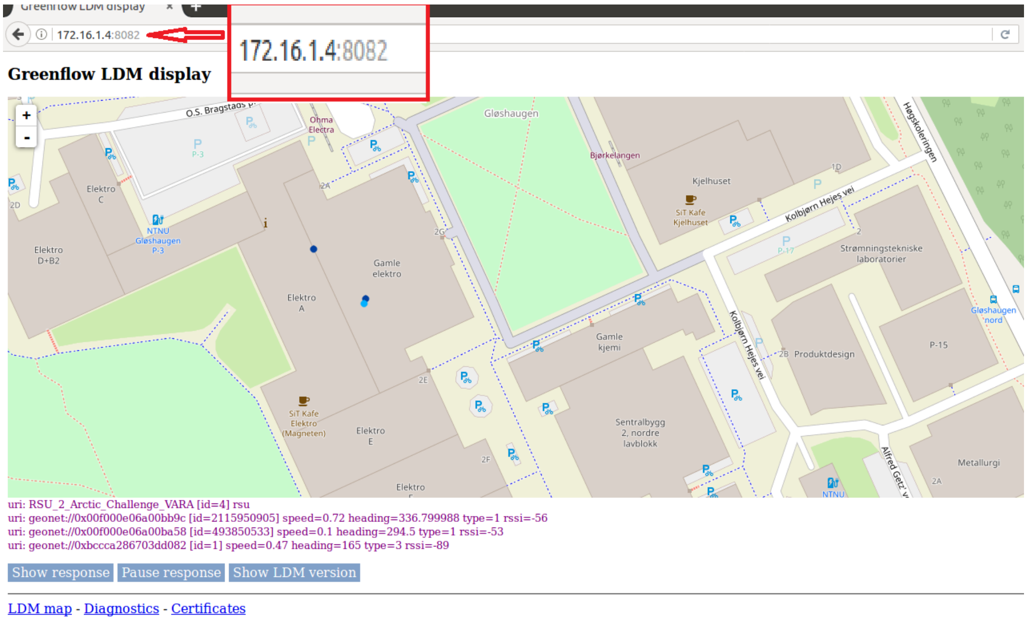
After pinging the units from the server to verify the connection, SSH was used to access the units and view the configuration.

- #ssh root@172.16.1.2
- #password: *********

The description of the configuration of the units was not very detailed from the manufacturer's side. Before any tests could be performed, a lot of time was spent trying to understand the file system, and how the configuration was built up. The root user has read and write permissions, and the text editor **vi** was used to edit configuration files. After the initial configuration of setting up the test network, no changes were made on the Kapsch units. All further work was performed on the Dynniq units.

5.3 Greenflow

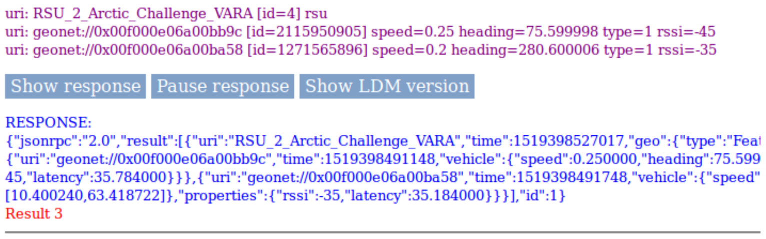
The Dynniq units come with a web interface with an application called Greenflow. It is useful for diagnostics, and provides a status overview of the system. To access the web interface, the address of the unit along with port number for the service was entered in a browser on the server. The Greenflow application from Dynniq is hosted at port 8082.



uri: RSU_2_Arctic_Challenge_VARA [id=4] rsu
uri: geonet://0x00f000e06a00bb9c [id=2115950905] speed=0.72 heading=336.799988 type=1 rssi=-56
uri: geonet://0x00f000e06a00ba58 [id=493850533] speed=0.1 heading=294.5 type=1 rssi=-53
uri: geonet://0xbccca286703dd082 [id=1] speed=0.47 heading=165 type=3 rssi=-89

Figure 5.4: Dynniq map display of ITS-units represented by blue dots in the Ngorongoro lab.

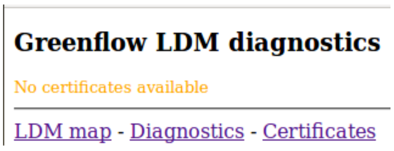
The application also provides a live feedback of data, listing the received units, and the associated parameters.



```
uri: RSU_2_Arctic_Challenge_VARA [id=4] rsu
uri: geonet://0x00f000e06a00bb9c [id=2115950905] speed=0.25 heading=75.599998 type=1 rssi=-45
uri: geonet://0x00f000e06a00ba58 [id=1271565896] speed=0.2 heading=280.600006 type=1 rssi=-35

RESPONSE:
{"jsonrpc":"2.0","result":[{"uri":"RSU_2_Arctic_Challenge_VARA","time":1519398527017,"geo":{"type":"Feat
{"uri":"geonet://0x00f000e06a00bb9c","time":1519398491148,"vehicle":{"speed":0.250000,"heading":75.599
45,"latency":35.784000}}},{"uri":"geonet://0x00f000e06a00ba58","time":1519398491748,"vehicle":{"speed"
[10.400240,63.418722]},"properties":{"rssi":-35,"latency":35.184000}}}],"id":1}
Result 3
```

Figure 5.5: Show response: Feedback

The Greenflow certificate overview displays the available certificates on the device. The devices does not come with any certificates pre-installed.



Greenflow LDM diagnostics

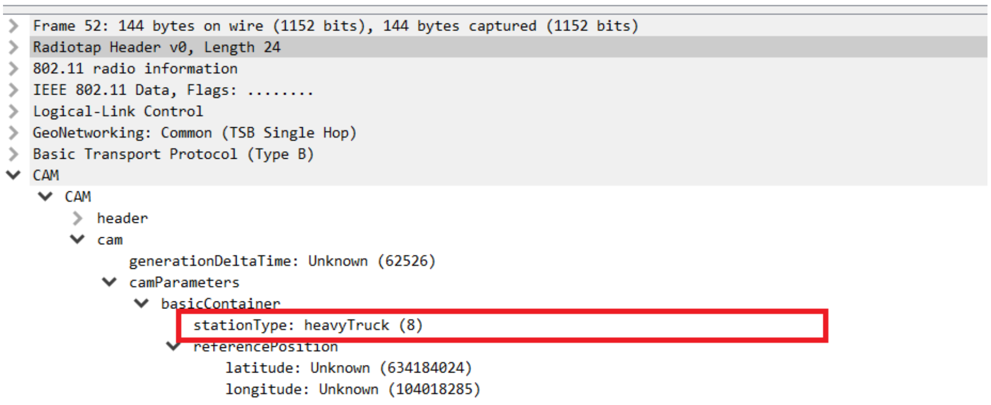
No certificates available

LDM map - Diagnostics - Certificates

Figure 5.6: Certificate overview.

5.4 Packet analysis

The Kapsch units log the C-ITS messages in pcap (packet capture) files, available through the Kapsch web interface. Before being able to view the CAM and DENM packets in Wireshark, it was necessary to install a packet dissector provided by Aventi.



```
> Frame 52: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits)
> Radiotap Header v0, Length 24
> 802.11 radio information
> IEEE 802.11 Data, Flags: ........
> Logical-Link Control
> GeoNetworking: Common (TSB Single Hop)
> Basic Transport Protocol (Type B)
v CAM
    v CAM
        > header
        v cam
            generationDeltaTime: Unknown (62526)
            v camParameters
                v basicContainer
                    stationType: heavyTruck (8)
                  v referencePosition
                        latitude: Unknown (634184024)
                        longitude: Unknown (104018285)
```

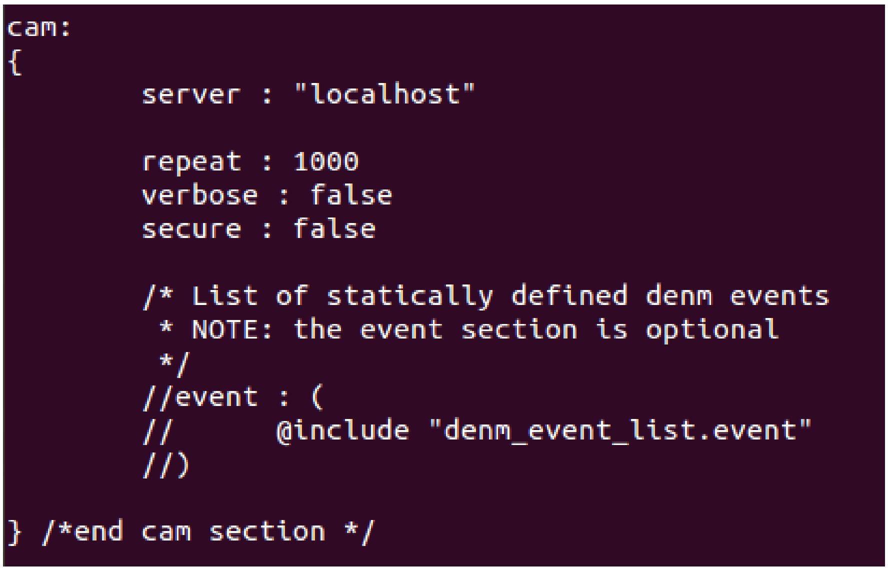
Figure 5.7: CAM packet broadcast from the Dynniq OBU. The OBU is configured as a heavy truck.

After verifying that all ITS-units were transmitting CAMs in un-authenticated mode and that all applications and web interfaces were functioning correctly, the work to implement authentication began.

Chapter 6 Authenticated CAM exchange

The primary objective of the practical work is to implement a PKI to the test network, enabling an authenticated exchange of CAM messages. Test scenarios were formulated to determine the progress of the PKI implementation. The purpose of these tests is to evaluate the results objectively and to ensure that the process is thoroughly documented.

In the manuals describing the configuration of the Dynniq OBU, there is a section about the file **its-facilities.conf**, and some of the variables in that file. The full path is **/etc/its-facilities.conf**. The file contains a section regarding CAM messages:



```
cam:
{
        server : "localhost"

        repeat : 1000
        verbose : false
        secure : false

        /* List of statically defined denm events
         * NOTE: the event section is optional
         */
        //event : (
        //      @include "denm_event_list.event"
        //)

} /*end cam section */
```

Figure 6.1: CAM section in its-facilities.conf.

One of the variables in this section is called **secure**, and this indicates whether or not the CAM service is in secure mode (using certificates).This variable has two states, true or false, with false as default.

6.1 Test 0: Enable Secure Mode

Before creating any certificate hierarchy, I wanted to see the effect of changing the value from false to true. The test network was put in the operational state, with both the Kapsch Link Test App and the Dynniq Greenflow App displaying the OBU units.

Status before changes made: with the variable set to false, the Kapsch units are shown in the Dynniq overview, and the Dynniq units are shown in the Kapsch display. Packets originating from the Dynniq OBU are present in the packet capture files.

Changes made: In the Dynniq OBU, the secure variable is set to true, and the configuration file saved under the same name.

Status after changes made: the immediate effect is that the Dynniq RSU loses contact with the Dynniq OBU. The OBU in secure mode is not visible on any platform.

This indicates that the **its-facilities.conf** file affects the CAM messages. The initial theory after this observation was that the secure mode was activated, but the packets dropped by the RSU because of the lack of certificates. This theory was later discarded after analyzing the packet capture files, where no CAM packets from the Dynniq OBU were present. The conclusion was that with no keys or certificates present, the OBU would not send any CAMs in secure mode. This was later supported by a Dynniq engineer. The next step was to create a simple certificate hierarchy, deploy it on the devices, and repeat the test.

6.2 Test 1: Secure mode using a certificate hierarchy (OpenSSL)

To further test the secure parameter in the Dynniq configuration, a simple certificate hierarchy was created using OpenSSL. OpenSSL is an open-source general-purpose cryptography library, which can be used to create cryptographic keys and certificates. The objective is to observe any changes in the messages when a simple PKI is in place, and the secure mode is enabled. The hierarchy consisted of a self-signed root CA, and two device certificates, signed by the root CA.

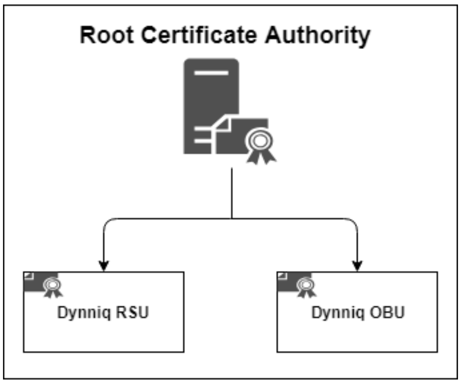


Figure 6.2: Simple PKI hierarchy.

The certificates were deployed on the OBU and RSU, respectively. The keys and certificates were placed in the associated folders:

- etc/its-security/keys
- etc/its-security/certs

The OBUs/RSUs contain a Hardware Security Module (HSM) where the keys will be stored to protect against tampering. This must be unlocked by the manufacturer (Dynniq), and was not prioritized for this thesis. After deploying the certificates and rebooting the system, the secure mode for CAM was enabled on the units. The status in the certificate listing was unchanged (fig 5.6), indicating that the system did not register the addition of the certificates. The RSU did not register any incoming CAM messages from the OBU, and no CAM packets originating from the OBU were present in the pcap file. The conclusion was that the system is not able to utilize the OpenSSL certificates, and therefore the status is the same as for the initial test, where no certificates were present. The results of this test were negative, and the next step was to inquire the Dynniq engineers about the certificate formats used by the system.

Double Certificate Bug

At this point, I was informed that some of the older OBU/RSU software versions required two pairs of certificates/keys before functioning correctly. The test using OpenSSL was repeated with second pairs deployed, with no change in the results. All following test involving certificates were performed with double sets of certificates/keys to mitigate this bug.

A detailed description of how the keys and certificates were generated using OpenSSL is available in Appendix A.

6.3 Test 2: Secure mode using a certificate hierarcy (ITSSEC)

The certificates generated with OpenSSL did not register with the devices, and I was not able to make progress with that approach. After seeking help from the Dynniq engineers through Aventi, I received a program called itssec. Itssec is a Linux command line tool which can generate keys and certificates used in the ITS PKI. To use itssec, four files must be present:

– itssec (executable file)

– libclientapi.so

– libitssecurity.so

– libjsonrpc.so.

Itssec can be used by placing the files in an empty directory, and issuing the command:
"export LD_LIBRARY_PATH=.".
Trying to run the executable at this stage yielded this response:
./itssec: error while loading shared libraries: libcrypto.so.1.0.0: cannot open shared object file: No such file or directory
This issue was resolved by downloading the package libssl1.0.0, which contains libcrypto.so.1.0.0.
https://packages.debian.org/jessie-backports/amd64/libssl1.0.0/download
The downloaded package was placed in the same folder as the itssec files and installed:

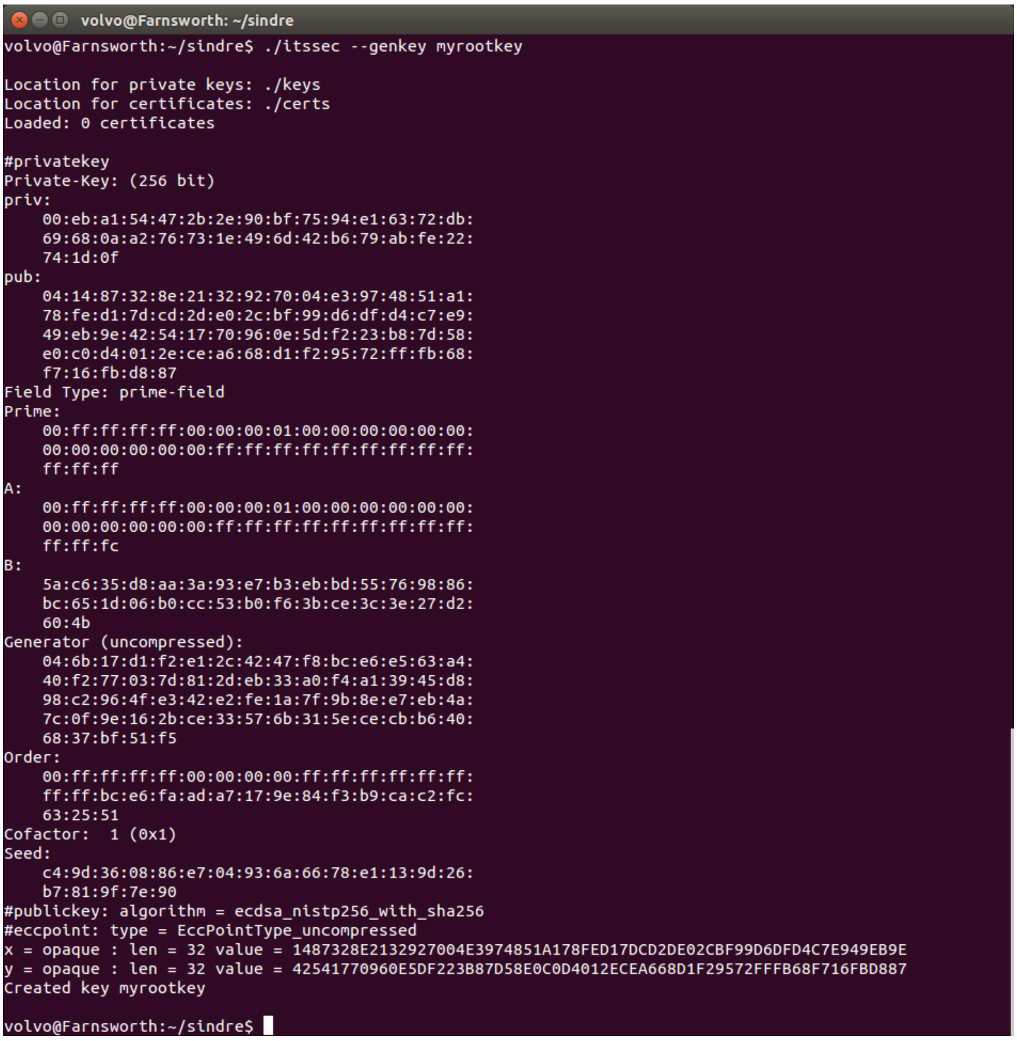
```
dpkg -i /path/to/downloaded-deb-file.deb
```

At this stage the itssec tool was executable. To be able to use the secure mode, where the messages will use authentication as specified by ETSI, a PKI needs to be available. In its simplest form, it can consist of a self-signed root CA, and unit level certificates (RSU/OBU). The next section describes how the itssec tool was used to generate a self-signed root CA, and two root-signed device certificates (RSU/OBU).

Creating keys and certificates

First, we create folders for the keys and certificates: **mkdir -p keys certs**

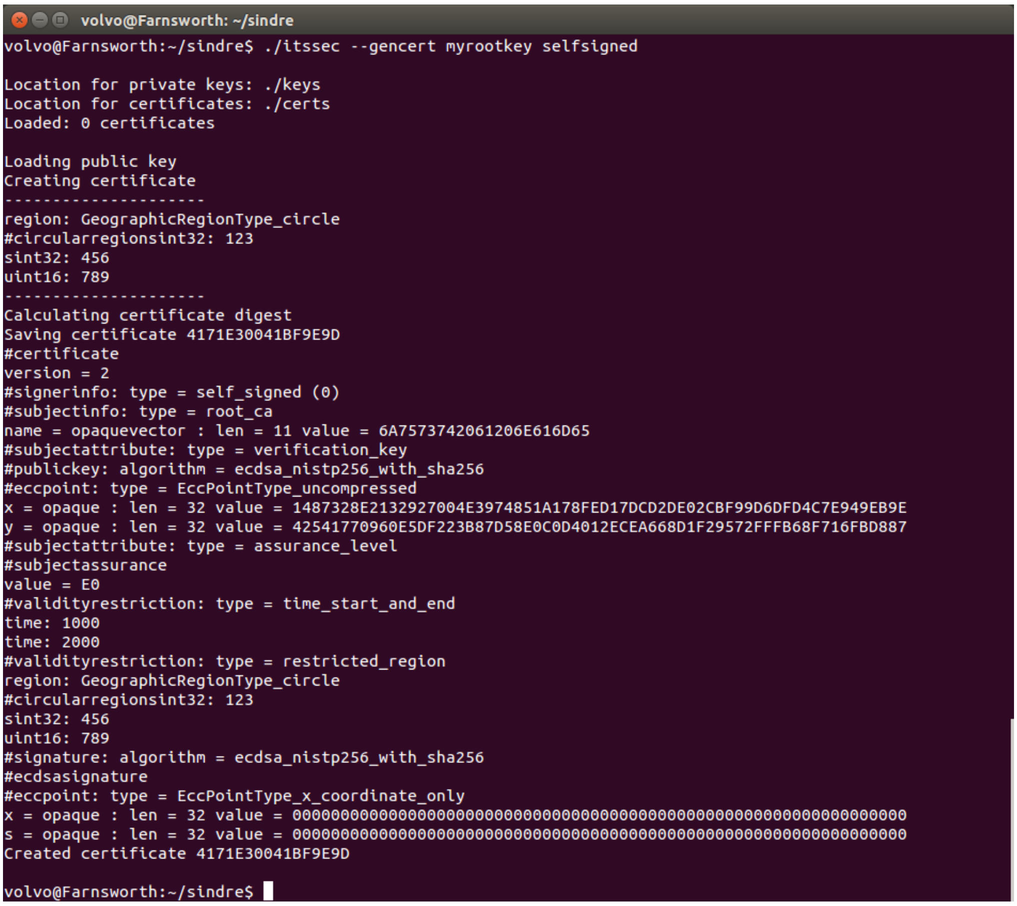
Creating a root key pair: **itssec –genkey myrootkey**



```
volvo@Farnsworth: ~/sindre
volvo@Farnsworth:~/sindre$ ./itssec --genkey myrootkey

Location for private keys: ./keys
Location for certificates: ./certs
Loaded: 0 certificates

#privatekey
Private-Key: (256 bit)
priv:
    00:eb:a1:54:47:2b:2e:90:bf:75:94:e1:63:72:db:
    69:68:0a:a2:76:73:1e:49:6d:42:b6:79:ab:fe:22:
    74:1d:0f
pub:
    04:14:87:32:8e:21:32:92:70:04:e3:97:48:51:a1:
    78:fe:d1:7d:cd:2d:e0:2c:bf:99:d6:df:d4:c7:e9:
    49:eb:9e:42:54:17:70:96:0e:5d:f2:23:b8:7d:58:
    e0:c0:d4:01:2e:ce:a6:68:d1:f2:95:72:ff:fb:68:
    f7:16:fb:d8:87
Field Type: prime-field
Prime:
    00:ff:ff:ff:ff:00:00:00:01:00:00:00:00:00:00:
    00:00:00:00:00:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:ff
A:
    00:ff:ff:ff:ff:00:00:00:01:00:00:00:00:00:00:
    00:00:00:00:00:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:fc
B:
    5a:c6:35:d8:aa:3a:93:e7:b3:eb:bd:55:76:98:86:
    bc:65:1d:06:b0:cc:53:b0:f6:3b:ce:3c:3e:27:d2:
    60:4b
Generator (uncompressed):
    04:6b:17:d1:f2:e1:2c:42:47:f8:bc:e6:e5:63:a4:
    40:f2:77:03:7d:81:2d:eb:33:a0:f4:a1:39:45:d8:
    98:c2:96:4f:e3:42:e2:fe:1a:7f:9b:8e:e7:eb:4a:
    7c:0f:9e:16:2b:ce:33:57:6b:31:5e:ce:cb:b6:40:
    68:37:bf:51:f5
Order:
    00:ff:ff:ff:ff:00:00:00:00:ff:ff:ff:ff:ff:ff:
    ff:ff:bc:e6:fa:ad:a7:17:9e:84:f3:b9:ca:c2:fc:
    63:25:51
Cofactor:  1 (0x1)
Seed:
    c4:9d:36:08:86:e7:04:93:6a:66:78:e1:13:9d:26:
    b7:81:9f:7e:90
#publickey: algorithm = ecdsa_nistp256_with_sha256
#eccpoint: type = EccPointType_uncompressed
x = opaque : len = 32 value = 1487328E2132927004E3974851A178FED17DCD2DE02CBF99D6DFD4C7E949EB9E
y = opaque : len = 32 value = 42541770960E5DF223B87D58E0C0D4012ECEA668D1F29572FFFB68F716FBD887
Created key myrootkey

volvo@Farnsworth:~/sindre$ █
```

Figure 6.3: Itssec generation of root key.

This command creates two keys, one public (.pub) and one private (.prv), and places them in the **keys** folder. Next, we create a self-signed certificate using the root key: **itssec –gencert myrootkey selfsigned**

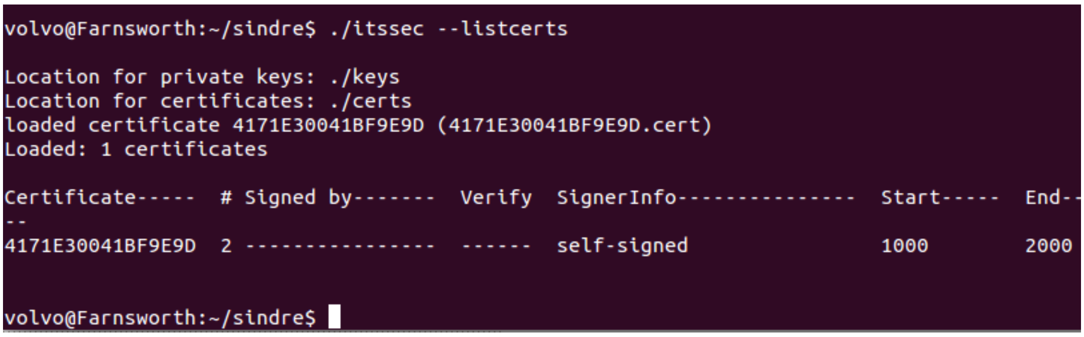


```
volvo@Farnsworth:~/sindre$ ./itssec --gencert myrootkey selfsigned

Location for private keys: ./keys
Location for certificates: ./certs
Loaded: 0 certificates

Loading public key
Creating certificate
---------------------
region: GeographicRegionType_circle
#circularregionsint32: 123
sint32: 456
uint16: 789
---------------------
Calculating certificate digest
Saving certificate 4171E30041BF9E9D
#certificate
version = 2
#signerinfo: type = self_signed (0)
#subjectinfo: type = root_ca
name = opaquevector : len = 11 value = 6A7573742061206E616D65
#subjectattribute: type = verification_key
#publickey: algorithm = ecdsa_nistp256_with_sha256
#eccpoint: type = EccPointType_uncompressed
x = opaque : len = 32 value = 1487328E2132927004E3974851A178FED17DCD2DE02CBF99D6DFD4C7E949EB9E
y = opaque : len = 32 value = 42541770960E5DF223B87D58E0C0D4012ECEA668D1F29572FFFB68F716FBD887
#subjectattribute: type = assurance_level
#subjectassurance
value = E0
#validityrestriction: type = time_start_and_end
time: 1000
time: 2000
#validityrestriction: type = restricted_region
region: GeographicRegionType_circle
#circularregionsint32: 123
sint32: 456
uint16: 789
#signature: algorithm = ecdsa_nistp256_with_sha256
#ecdsasignature
#eccpoint: type = EccPointType_x_coordinate_only
x = opaque : len = 32 value = 0000000000000000000000000000000000000000000000000000000000000000
s = opaque : len = 32 value = 0000000000000000000000000000000000000000000000000000000000000000
Created certificate 4171E30041BF9E9D

volvo@Farnsworth:~/sindre$
```

Figure 6.4: Itssec generation of root certificate.

The public key is now part of the root certificate. The private key must be kept secret at all times. For the tool to be able to “sign” subsequently generated certificates, it is necessary to give the private root key the same name as the root certificate.

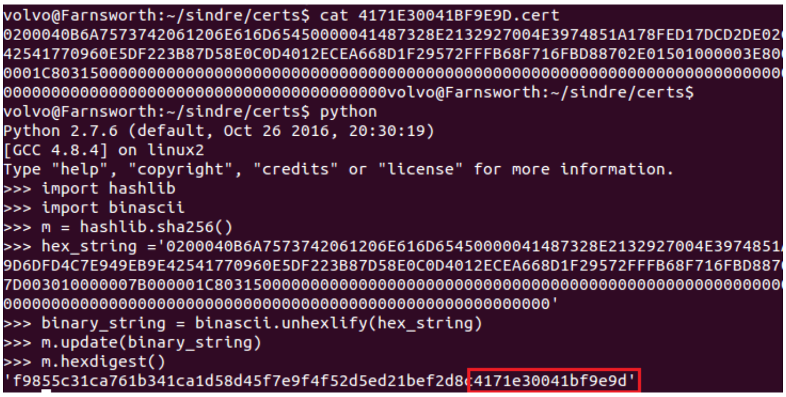
We can use the following command to view the certificates:
itssec –listcerts



```
volvo@Farnsworth:~/sindre$ ./itssec --listcerts

Location for private keys: ./keys
Location for certificates: ./certs
loaded certificate 4171E30041BF9E9D (4171E30041BF9E9D.cert)
Loaded: 1 certificates

Certificate-----  # Signed by-------  Verify  SignerInfo--------------  Start-----  End--
--
4171E30041BF9E9D  2 ----------------  ------  self-signed               1000        2000


volvo@Farnsworth:~/sindre$ 
```

Figure 6.5: Listing of available certificates.

The name of the certificate is the last eight bytes of a SHA-256 operation over the certificate. In fig 6.6 Python is used to verify the name by creating a SHA-256 digest using the root certificate as input. The last 8 bytes marked in red matches the certificate name from fig 6.5.



```
volvo@Farnsworth:~/sindre/certs$ cat 4171E30041BF9E9D.cert
0200040B6A7573742061206E616D65450000041487328E2132927004E3974851A178FED17DCD2DE02
42541770960E5DF223B87D58E0C0D4012ECEA668D1F29572FFFB68F716FBD88702E01501000003E80
0001C80315000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000volvo@Farnsworth:~/sindre/certs$
volvo@Farnsworth:~/sindre/certs$ python
Python 2.7.6 (default, Oct 26 2016, 20:30:19)
[GCC 4.8.4] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> import binascii
>>> m = hashlib.sha256()
>>> hex_string ='0200040B6A7573742061206E616D65450000041487328E2132927004E3974851
9D6DFD4C7E949EB9E42541770960E5DF223B87D58E0C0D4012ECEA668D1F29572FFFB68F716FBD887
7D003010000007B000001C80315000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000'
>>> binary_string = binascii.unhexlify(hex_string)
>>> m.update(binary_string)
>>> m.hexdigest()
'f9855c31ca761b341ca1d58d45f7e9f4f52d5ed21bef2d8c4171e30041bf9e9d'
```

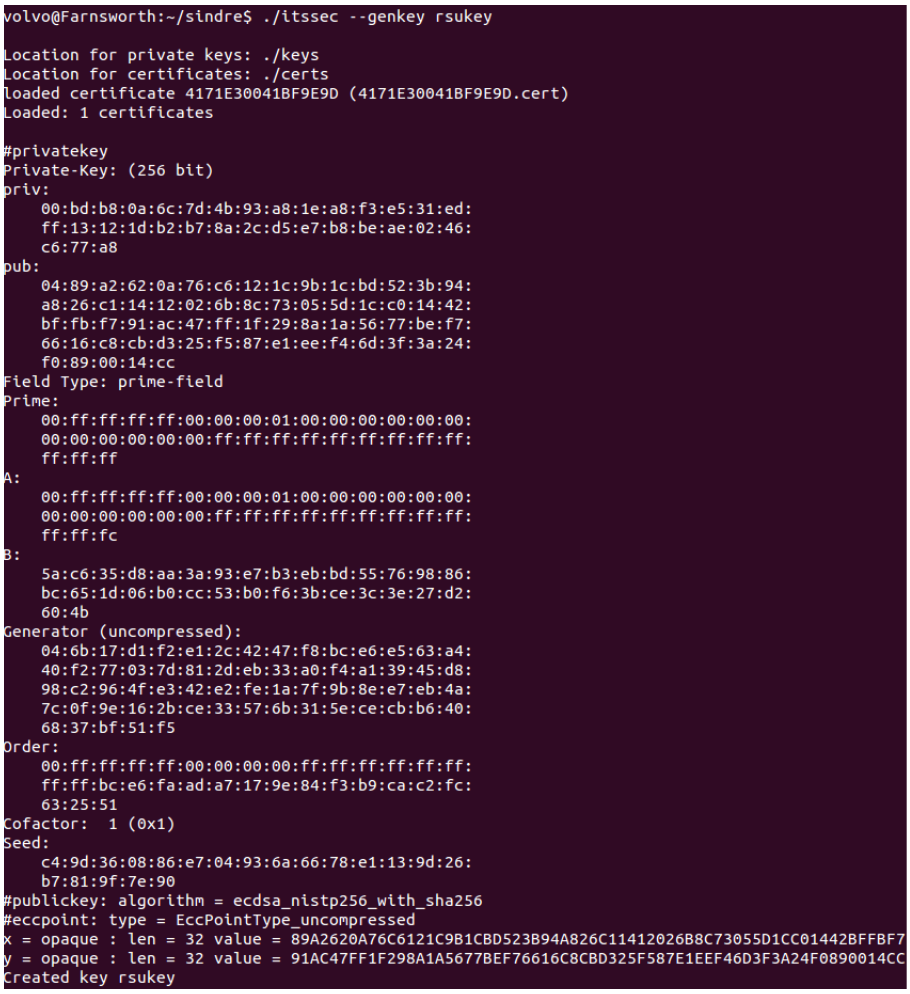
Figure 6.6: Verification of hex digest.

The file name of a private key must start with the hash of the certificate in capital letters. The following commands create a symbolic link from the private key, with the same name as the certificate.

cd keys/
ln -s myrootkey.prv 4171E30041BF9E9D.prv

This operation was repeated on the devices for the private keys of the RSU/OBU. In this test PKI we skip the intermediate CA and create certificates for the devices using the root CA.

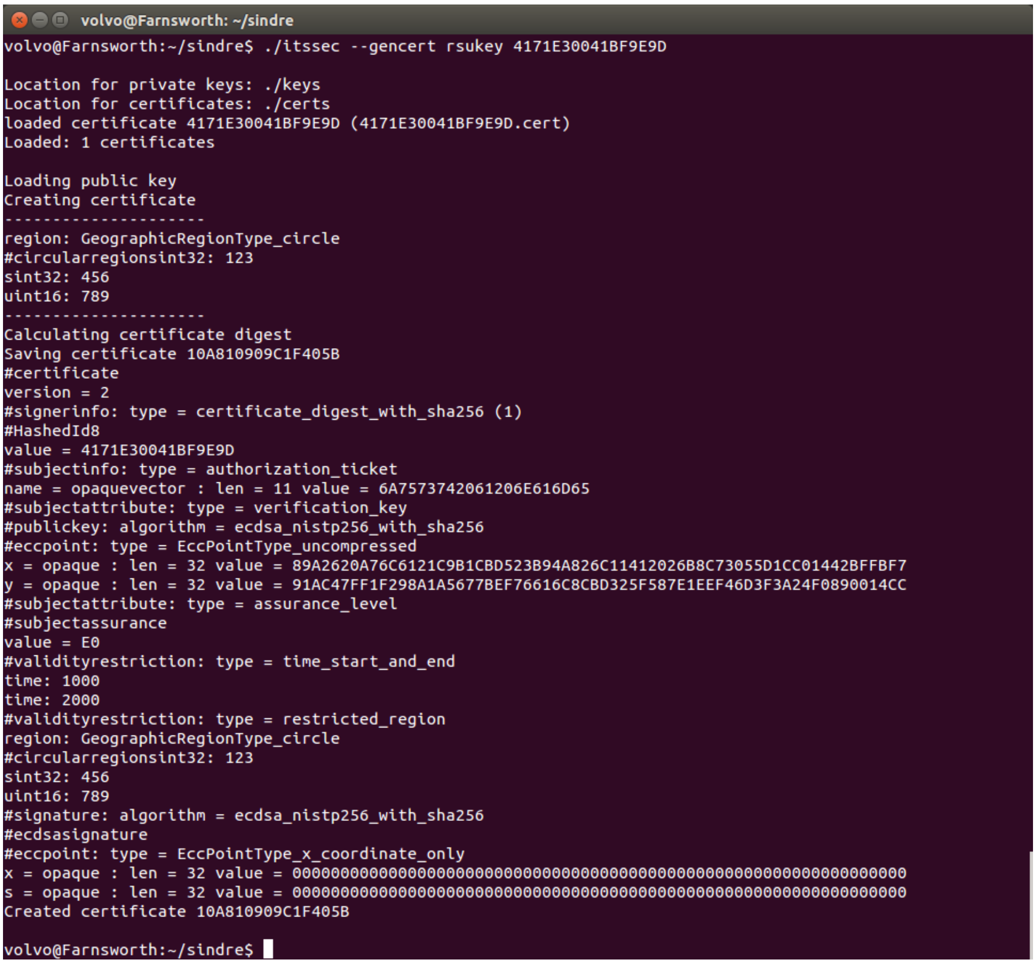
Generating a key pair for the RSU:
itssec –genkey rsukey



```
volvo@Farnsworth:~/sindre$ ./itssec --genkey rsukey

Location for private keys: ./keys
Location for certificates: ./certs
loaded certificate 4171E30041BF9E9D (4171E30041BF9E9D.cert)
Loaded: 1 certificates

#privatekey
Private-Key: (256 bit)
priv:
    00:bd:b8:0a:6c:7d:4b:93:a8:1e:a8:f3:e5:31:ed:
    ff:13:12:1d:b2:b7:8a:2c:d5:e7:b8:be:ae:02:46:
    c6:77:a8
pub:
    04:89:a2:62:0a:76:c6:12:1c:9b:1c:bd:52:3b:94:
    a8:26:c1:14:12:02:6b:8c:73:05:5d:1c:c0:14:42:
    bf:fb:f7:91:ac:47:ff:1f:29:8a:1a:56:77:be:f7:
    66:16:c8:cb:d3:25:f5:87:e1:ee:f4:6d:3f:3a:24:
    f0:89:00:14:cc
Field Type: prime-field
Prime:
    00:ff:ff:ff:ff:00:00:00:01:00:00:00:00:00:00:
    00:00:00:00:00:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:ff
A:
    00:ff:ff:ff:ff:00:00:00:01:00:00:00:00:00:00:
    00:00:00:00:00:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:fc
B:
    5a:c6:35:d8:aa:3a:93:e7:b3:eb:bd:55:76:98:86:
    bc:65:1d:06:b0:cc:53:b0:f6:3b:ce:3c:3e:27:d2:
    60:4b
Generator (uncompressed):
    04:6b:17:d1:f2:e1:2c:42:47:f8:bc:e6:e5:63:a4:
    40:f2:77:03:7d:81:2d:eb:33:a0:f4:a1:39:45:d8:
    98:c2:96:4f:e3:42:e2:fe:1a:7f:9b:8e:e7:eb:4a:
    7c:0f:9e:16:2b:ce:33:57:6b:31:5e:ce:cb:b6:40:
    68:37:bf:51:f5
Order:
    00:ff:ff:ff:ff:00:00:00:00:ff:ff:ff:ff:ff:ff:
    ff:ff:bc:e6:fa:ad:a7:17:9e:84:f3:b9:ca:c2:fc:
    63:25:51
Cofactor:  1 (0x1)
Seed:
    c4:9d:36:08:86:e7:04:93:6a:66:78:e1:13:9d:26:
    b7:81:9f:7e:90
#publickey: algorithm = ecdsa_nistp256_with_sha256
#eccpoint: type = EccPointType_uncompressed
x = opaque : len = 32 value = 89A2620A76C6121C9B1CBD523B94A826C11412026B8C73055D1CC01442BFFBF7
y = opaque : len = 32 value = 91AC47FF1F298A1A5677BEF76616C8CBD325F587E1EEF46D3F3A24F0890014CC
Created key rsukey
```

Figure 6.7: Generation of RSU key.

Now we can create a certificate based on the self-signed root certificate that we created before.
itssec –gencert rsukey 4171E30041BF9E9D

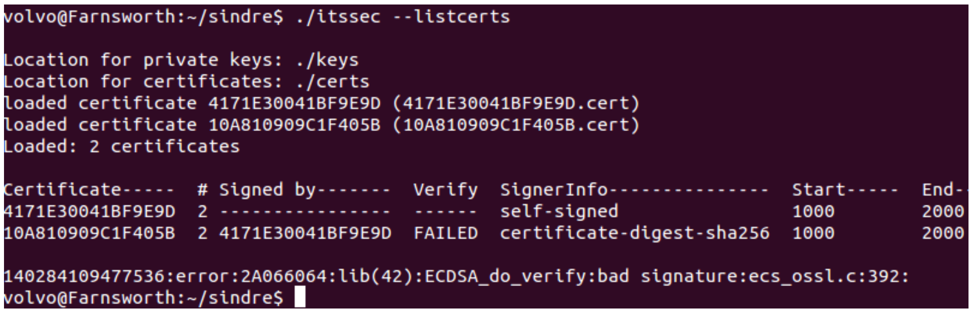


volvo@Farnsworth: ~/sindre

```
volvo@Farnsworth:~/sindre$ ./itssec --gencert rsukey 4171E30041BF9E9D

Location for private keys: ./keys
Location for certificates: ./certs
loaded certificate 4171E30041BF9E9D (4171E30041BF9E9D.cert)
Loaded: 1 certificates

Loading public key
Creating certificate
---------------------
region: GeographicRegionType_circle
#circularregionsint32: 123
sint32: 456
uint16: 789
---------------------
Calculating certificate digest
Saving certificate 10A810909C1F405B
#certificate
version = 2
#signerinfo: type = certificate_digest_with_sha256 (1)
#HashedId8
value = 4171E30041BF9E9D
#subjectinfo: type = authorization_ticket
name = opaquevector : len = 11 value = 6A7573742061206E616D65
#subjectattribute: type = verification_key
#publickey: algorithm = ecdsa_nistp256_with_sha256
#eccpoint: type = EccPointType_uncompressed
x = opaque : len = 32 value = 89A2620A76C6121C9B1CBD523B94A826C11412026B8C73055D1CC01442BFFBF7
y = opaque : len = 32 value = 91AC47FF1F298A1A5677BEF76616C8CBD325F587E1EEF46D3F3A24F0890014CC
#subjectattribute: type = assurance_level
#subjectassurance
value = E0
#validityrestriction: type = time_start_and_end
time: 1000
time: 2000
#validityrestriction: type = restricted_region
region: GeographicRegionType_circle
#circularregionsint32: 123
sint32: 456
uint16: 789
#signature: algorithm = ecdsa_nistp256_with_sha256
#ecdsasignature
#eccpoint: type = EccPointType_x_coordinate_only
x = opaque : len = 32 value = 0000000000000000000000000000000000000000000000000000000000000000
s = opaque : len = 32 value = 0000000000000000000000000000000000000000000000000000000000000000
Created certificate 10A810909C1F405B

volvo@Farnsworth:~/sindre$
```

Figure 6.8: Generation of RSU certificate.

This will produce a certificate that needs to be signed by the Root-CA. The listcerts command shows us the status of the certificates:
itssec –listcerts

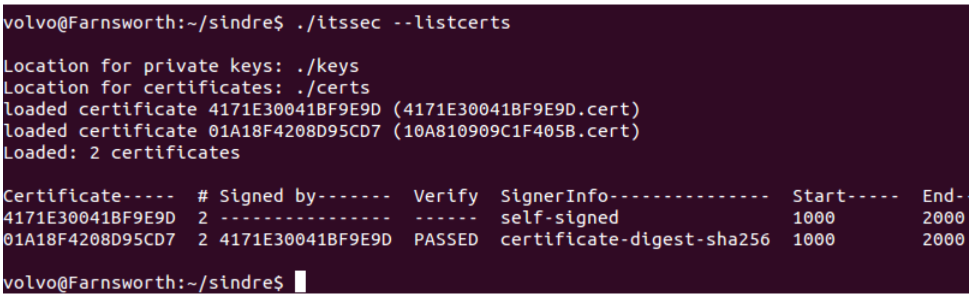


```
volvo@Farnsworth:~/sindre$ ./itssec --listcerts

Location for private keys: ./keys
Location for certificates: ./certs
loaded certificate 4171E30041BF9E9D (4171E30041BF9E9D.cert)
loaded certificate 10A810909C1F405B (10A810909C1F405B.cert)
Loaded: 2 certificates

Certificate----- # Signed by------- Verify  SignerInfo--------------- Start----- End-
4171E30041BF9E9D 2 ---------------- ------  self-signed               1000       2000
10A810909C1F405B 2 4171E30041BF9E9D FAILED  certificate-digest-sha256 1000       2000

140284109477536:error:2A066064:lib(42):ECDSA_do_verify:bad signature:ecs_ossl.c:392:
volvo@Farnsworth:~/sindre$
```

Figure 6.9: The RSU certificate is not signed.

We see that a new certificate (10A810909C1F405B) is in the list. That is the one that needs to be signed by the root CA.
itssec –signcert 10A810909C1F405B

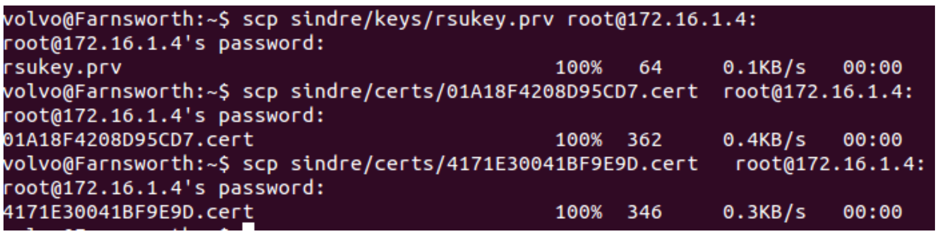
itssec –listcerts



```
volvo@Farnsworth:~/sindre$ ./itssec --listcerts

Location for private keys: ./keys
Location for certificates: ./certs
loaded certificate 4171E30041BF9E9D (4171E30041BF9E9D.cert)
loaded certificate 01A18F4208D95CD7 (10A810909C1F405B.cert)
Loaded: 2 certificates

Certificate----- # Signed by------- Verify  SignerInfo--------------- Start----- End-
4171E30041BF9E9D 2 ---------------- ------  self-signed               1000       2000
01A18F4208D95CD7 2 4171E30041BF9E9D PASSED  certificate-digest-sha256 1000       2000

volvo@Farnsworth:~/sindre$
```

Figure 6.10: The RSU certificate is now signed by root.

We see the change in status, and the verification of the signature on the RSU certificate. Signing the certificate changes the hash value, and the name of the certificate and the private key must be changed to reflect the new value. The same process is repeated to create an OBU certificate. The keys and certificates were generated on the ITS-PC and deployed to the devices using secure copy (SCP).

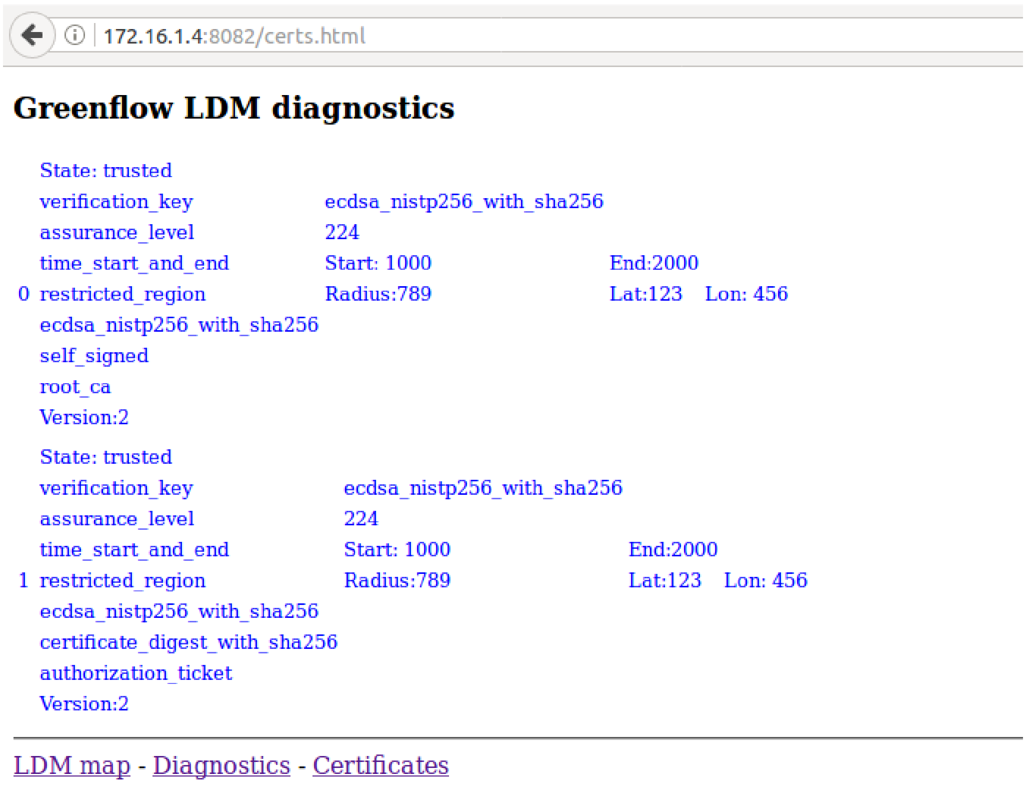


```
volvo@Farnsworth:~$ scp sindre/keys/rsukey.prv root@172.16.1.4:
root@172.16.1.4's password:
rsukey.prv                                    100%   64     0.1KB/s   00:00
volvo@Farnsworth:~$ scp sindre/certs/01A18F4208D95CD7.cert  root@172.16.1.4:
root@172.16.1.4's password:
01A18F4208D95CD7.cert                         100%  362     0.4KB/s   00:00
volvo@Farnsworth:~$ scp sindre/certs/4171E30041BF9E9D.cert   root@172.16.1.4:
root@172.16.1.4's password:
4171E30041BF9E9D.cert                         100%  346     0.3KB/s   00:00
```

Figure 6.11: Using secure copy to deploy the keys and certificates to the devices.

The root certificate, the device certificate, and the private key (named after the corresponding certificate) of the device must be deployed. The public key of the device is included in the device certificate. The files were placed in the associated directories:
/etc/its-security/keys and **/etc/its-security/certs** on the RSU and OBU.

The itssec-generated certificates and keys were deployed in the associated directories, and the system rebooted. When accessing the certificate overview in the Greenflow application, the imported certificates are available.


172.16.1.4:8082/certs.html

Greenflow LDM diagnostics

```
  State: trusted
  verification_key          ecdsa_nistp256_with_sha256
  assurance_level           224
  time_start_and_end        Start: 1000         End:2000
0 restricted_region         Radius:789          Lat:123   Lon: 456
  ecdsa_nistp256_with_sha256
  self_signed
  root_ca
  Version:2

  State: trusted
  verification_key          ecdsa_nistp256_with_sha256
  assurance_level           224
  time_start_and_end        Start: 1000         End:2000
1 restricted_region         Radius:789          Lat:123   Lon: 456
  ecdsa_nistp256_with_sha256
  certificate_digest_with_sha256
  authorization_ticket
  Version:2
```

LDM map - Diagnostics - Certificates

Figure 6.12: Greenflow certificate overview.

There are now one self-signed root certificate and two root-signed authorization tickets (RSU certificates) on the RSU. All the parameters are default values. ECDSA with SHA-256, as according to the specifications, is used to generate the digital signature.

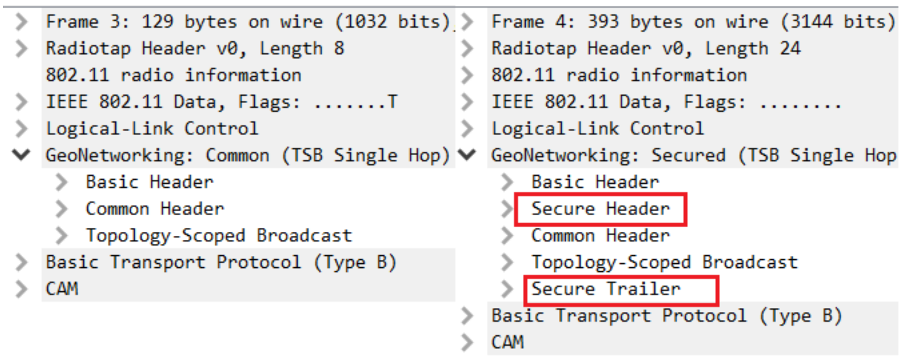
After verifying that the certificates were loaded, the secure mode was enabled in the CAM section of the configuration file. The results were positive, as the RSU received messages from the OBU in trusted mode. In the feedback module in Greenflow the parameters of the OBU (uri: 0xbcccd2a7ed3cde2) confirmed the status:

```
{"uri":"geonet://0xbcccd2a7ed3cde21",
"time":1519592632230,
"security":{"key":"D3A7ED3CDE218A7B",
"state":"trusted"}
```

D3A7ED3CDE218A7B is the name of one of the keys loaded on the Dynniq OBU. Using the interface on the OBU, we can view the status of the RSU CAM messages:

```
    {"uri":"geonet://0xbcccc61de13a3509",
    "time":1526030952779,
    "security":{"key":"C71DE13A3509CAA9",
    "state":"trusted"}
```

The change was also visible in the pcap files analyzed in Wireshark. The CAM messages from the OBU now contained a secure header and trailer, as referenced in fig 4.10.


```
> Frame 3: 129 bytes on wire (1032 bits)
> Radiotap Header v0, Length 8
  802.11 radio information
> IEEE 802.11 Data, Flags: .......T
> Logical-Link Control
v GeoNetworking: Common (TSB Single Hop)
    > Basic Header
    > Common Header
    > Topology-Scoped Broadcast
> Basic Transport Protocol (Type B)
> CAM
```

```
> Frame 4: 393 bytes on wire (3144 bits)
> Radiotap Header v0, Length 24
> 802.11 radio information
> IEEE 802.11 Data, Flags: ........
> Logical-Link Control
v GeoNetworking: Secured (TSB Single Hop
    > Basic Header
    > Secure Header
    > Common Header
    > Topology-Scoped Broadcast
    > Secure Trailer
> Basic Transport Protocol (Type B)
> CAM
```

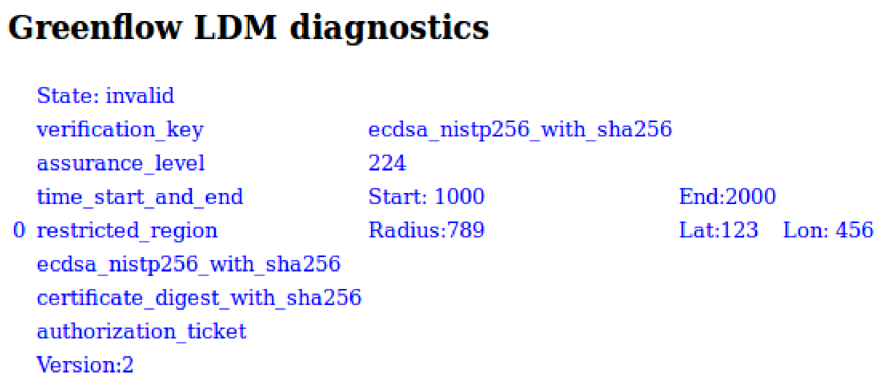
Figure 6.13: Wireshark unsecure CAM message (left) and secure CAM message (right).

A figure showing the complete content of the secure header and trailer is available in Appendix B, figure B.1.

The CAM messages sent from the Dynniq OBU are marked as "trusted" in the RSU, while the CAM messages from the Kapsch units are still displayed without any certificates. In the Kapsch Link Test app, only the units transmitting unsecured messages are displayed. This is because of the Kapsch equipment's lack of support for authentication. In conclusion, the test results were positive, as the units were able to communicate with each other in secure mode, using the certificates for authentication.

6.4 Test 3: Verification of the authentication

The objective of this test is to verify that the devices verify the signature of the certificates. By using unsigned certificates on the OBU, and signed certificates on the RSU, we can observe how the system reacts to invalid credentials.
Two certificate/key pairs were generated, but not signed by the root CA. The unsigned pairs were deployed on the OBU. From the certificate status in Greenflow, we can view the loaded certificates.


Greenflow LDM diagnostics

```
  State: invalid
  verification_key              ecdsa_nistp256_with_sha256
  assurance_level               224
  time_start_and_end            Start: 1000            End:2000
0 restricted_region             Radius:789             Lat:123   Lon: 456
  ecdsa_nistp256_with_sha256
  certificate_digest_with_sha256
  authorization_ticket
  Version:2
```

Figure 6.14: Un-signed certificate: State: invalid.

From the feedback module in the RSU Greenflow we can see status of the security parameter:
security": "key":"9B8CB3DFADF75C05","state":"invalid"
The unsigned certificates are marked as "invalid", indicating that the signatures are verified by the devices.

6.5 Test 4: Intermediate CA

The objective is to test the secure mode when the PKI has a chain of trust through an intermediate CA. The RSU remains signed by the root CA, while the OBU now is signed by the intermediate CA.

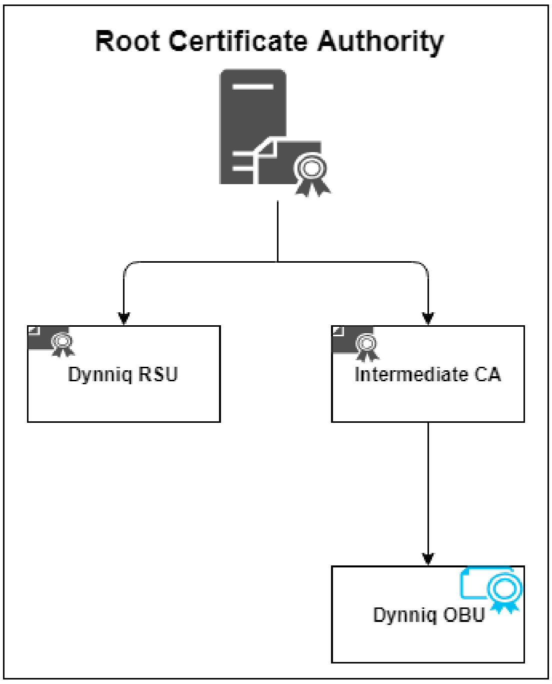


Figure 6.15: Signed certificate hierarchy.

The certificates and keys are deployed on the devices. For the first part, the intermediate certificate is only deployed on the OBU. When enabling the secure mode, the messages from the OBU are marked as "invalid", even though they are signed by the Intermediate CA. This status indicates that since the RSU does not have access to the Intermediate CA's information, it can not authenticate the messages sent from the OBU.
For the second part, the intermediate certificate is deployed on both the RSU and the OBU. The messages are now marked as "trusted". The RSU can validate the signature on the OBU messages against the intermediate certificate. The test indicates that devices can authenticate messages signed by a different entity in the trust hierarchy, as long as they have access to the certificates used in the chain of trust.

Chapter 7
Authorization Authority

7.1 Authorization Authority

The secure mode of the units have been tested and verified, and the units can exchange C-ITS messages using authentication. The objective now changes to adding features to the infrastructure, imitating the framework described in fig 4.5, within the confines of this thesis. The next step is to explore communication flow 2, between the ITS-S and the AA. During the tests of the secure CAM exchange, the certificates have been created on the ITS-PC (5.2) and manually copied to the devices. In a real-world scenario, this process would happen without the operator of the vehicle being involved. One plausible scenario is that the system initiates resupply of authentication tickets after a set amount of time, initiated by either the ITS-S or the AA.

From fig 4.15 we can see that it is the Authorization Authority which handles the distribution of ATs used by the vehicle. The communication flow between the ITS-S and the AA should provide authenticated users access to new ATs, to be used in the C-ITS network. Two possible approaches have been considered in this thesis:

– An Apache HTTP server.

– An MQTT Broker/Client architecture.

The two protocols were chosen because they were assessed to be viable candidates, and because they have contrasting characteristics. Both protocols are used in large-scale systems today (Internet/IoT), providing large amounts of documentation and experiences from previous implementations. In the following sections, the process of implementing and testing the solutions are described.

7.2 Apache HTTP Server

The Authorization Authority can distribute the ATs by hosting them on an HTTP server and allowing authenticated users to download them when necessary. This test will emulate a scenario where an ITS-S requires a new set of ATs, and contacts the AA for a resupply. The objective is to identify strengths and weaknesses of this approach. An Apache HTTP server was configured and made accessible on the local network to test this approach.

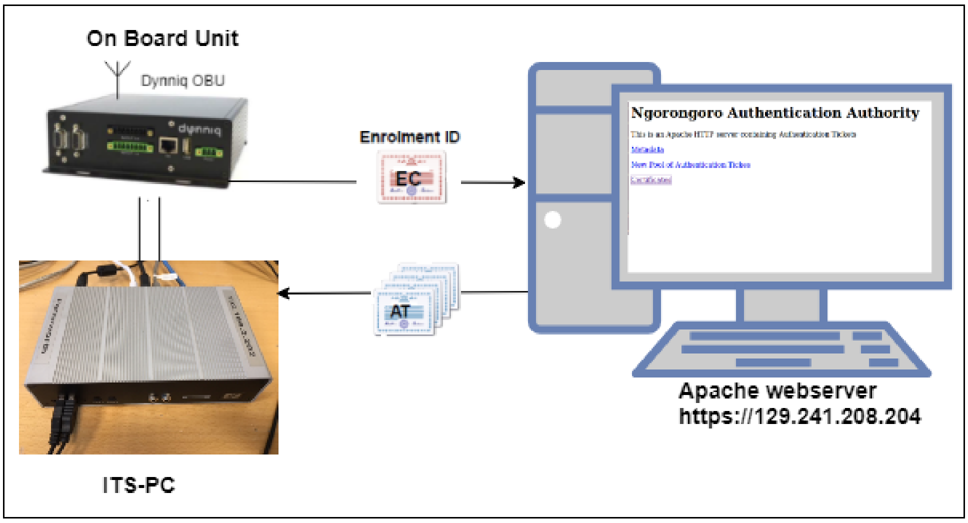

On Board Unit
Dynniq OBU
Enrolment ID
EC
AT
Ngorongoro Authentication Authority
This is an Apache HTTP server containing Authentication Tickets
Metadata
New Pool of Authentication Tickets
Certificates
Apache webserver
https://129.241.208.204
ITS-PC

Figure 7.1: Authentication Authority Apache Setup.

The OBU/RSU have only one Ethernet interface, used to connect them to the ITS-PC. The ITS-PC has two network interface cards, eth1 with internet access, and eth0 connected to the OBU. To allow the OBU to communicate with the server, the ITS-PC was configured to share its internet access with the OBU.

- **ITS-PC eth1**: 129.241.208.204
- **ITS-PC eth0**: 172.16.1.6
- **OBU eth0**: 172.16.1.2

The OBU was configured to use the ITS-PC as a gateway:

```
route add default gw 172.16.1.6
```

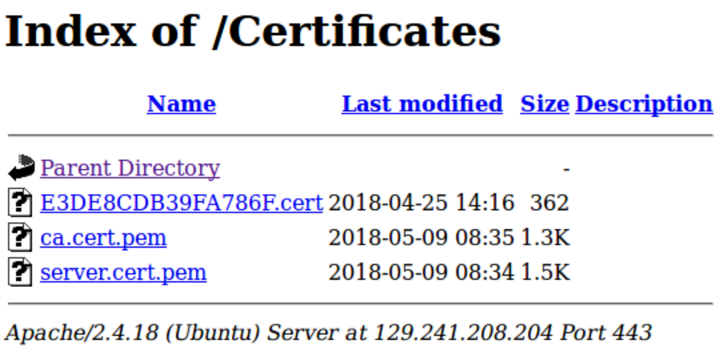
The ITS-PC was configured to act as a NAT router for the OBU:

```
modprobe iptable_nat
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
iptables -A FORWARD -i eth0 -j ACCEPT
```

At this stage, the OBU was able to ping the Apache server.

Setting up the Server

The server was configured on a dedicated Linux machine using the Apache HTTP Server. The default configuration was used, making it available on the local network. A simple web page was created, hosting certificates for the ITS-S to download:



Index of /Certificates

Name	Last modified	Size	Description
Parent Directory		-	
E3DE8CDB39FA786F.cert	2018-04-25 14:16	362	
ca.cert.pem	2018-05-09 08:35	1.3K	
server.cert.pem	2018-05-09 08:34	1.5K	

Apache/2.4.18 (Ubuntu) Server at 129.241.208.204 Port 443

Figure 7.2: Certificates hosted on the AA.

SSL was enabled to provide server-side authentication and the use of HTTPS (HTTP-Secure). OpenSSL was used to generate the certificate hierarchy. When configuring an Apache server, it is not enough to enable SSL in order to provide security [35]. However, a full implementation of the recommended security measures was not prioritized for this test. The Apache server can provide user authentication and access control, which can be included if it is decided to move forward with this approach.

Using Python, a simple script was created on the OBU to download the new certificates:

```
##########################################################################

import urllib, urllib2, urlparse, ssl

url = 'https://129.241.208.204/Certificates/E3DE8CDB39FA786F
    .cert'

split = urlparse.urlsplit(url)

filename = split.path.split('/')[-1]

print "Downlading␣new␣Authorization␣Ticket:" + filename
context = ssl._create_unverified_context()
# The context is created for testing purposes. Not safe
    using unverified in real life.
response = urllib2.urlopen('https://129.241.208.204/
    Certificates/E3DE8CDB39FA786F.cert', context = context)

cert = response.read()

f = open('/home/root/sindre/certs/'+ filename, 'w+')
#saving the cert and keeping the filename
f.write(cert)

f.close()
##########################################################################
```

In this design, the communication flow is initiated by the ITS-S, requesting the resources hosted on the server. The targeted certificate is downloaded and placed in the specified directory on the OBU. The script is very basic and is meant as an illustration of how ATs can be distributed to the OBUs.

An alternative to the python script is to use **wget**, an open-source tool for downloading files from web-servers.

```
wget -r -np -R 'index.html*' -P home/sindre/certs/

'https://129.241.208.204/Certificates/'
```

The **wget** command downloads the files present in the 'Certificates'-directory on the server and places them in the 'cert'-directory on the OBU, ignoring all 'index.html'-files.

A recurring challenge when working with the HTTP-server was the lack of support for downloading all files in a specific directory. Python supports modules which makes the process easier, but the overall impression was that of putting a square peg in a round hole.

7.3 MQTT Server using TLS

Another solution for realizing certificate distribution is to use the MQTT protocol. MQTT (Message Queuing Telemetry Transport) is a lightweight protocol designed to use a minimum of network bandwidth and device resources, making it popular in the world of IoT (Internet of Things) [36]. The protocol uses a broker/client architecture, where clients can publish and subscribe messages to and from a broker. HiveMQ [37] uses these definitions:

- **Client**: An MQTT client is any device from a microcontroller up to a full-fledged server, that has an MQTT library running and is connecting to an MQTT broker over any kind of network.
- **Broker**: The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients.

The publish/subscribe mechanism is useful in scenarios where you want to push messages to multiple clients in an effective way. A C-ITS scenario where this is relevant is in the case of Certificate Revocation Lists. The protocol is also scalable, as new clients can subscribe to an existing broker, without the need for altering the broker's configuration. The broker uses topics to sort and distribute messages. Topics are hierarchical strings consisting of multiple levels separated by a forward slash, e.g., **"Certs/test_network/OBU/OBU1"**. In this case, a connected client subscribing to the sub-topic "OB1" would receive messages published to that topic.

In the C-ITS test network, the OBUs and RSUs act as subscribing clients, with the AA functioning both as a broker and publisher of files (ATs). The idea is that the AA creates the ATs, and then uses a Mosquitto client to publish the files to the broker. The broker uses topics to sort and distribute the ATs to the subscribers.

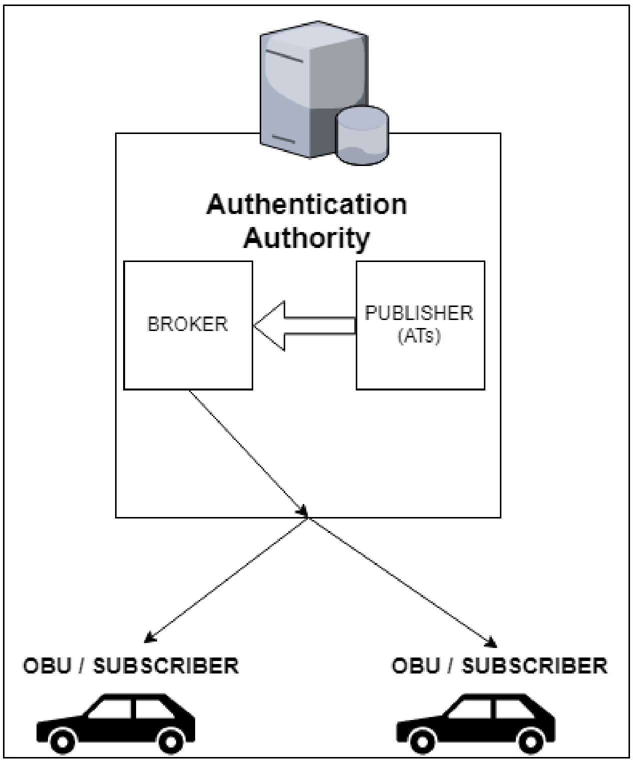


Figure 7.3: MQTT design in the test network.

MQTT does not provide security in itself, but security protocols in the underlying layers can provide encryption. MQTT is based on TCP/IP, and supports the use of TLS (Transport Layer Security). For authentication of broker and clients, digital certificates are used. The added overhead this imposes is a necessary trade-off for confidentiality.

MQTT configuration

Up until this point no new software has been installed on the OBU/RSU. The tests have been conducted by changing configuration and using tools already present on the units. When trying to install the necessary MQTT software on the OBU, I was faced with some challenges. No packet manager was available, and many commands usually found on a Linux system were unavailable. After struggling with the implementation for some time, it was decided to use the ITS-PC (running a more familiar version of Linux) to simulate the OBU. I perceived it as more pertinent to complete a test of MQTT as a certificate distribution protocol for the AA, rather than working on Linux distribution specific problems. My reasoning was that if the solution proved viable, more resources could be allocated to implementing it on the OBU.

For the tests involving MQTT an open source MQTT broker called Mosquitto was used. Mosquitto also provides clients, **mosquitto__sub** and **mosquitto__pub**, making the implementation user friendly. To install the broker and clients, these commands were issued on the ITS-PC:

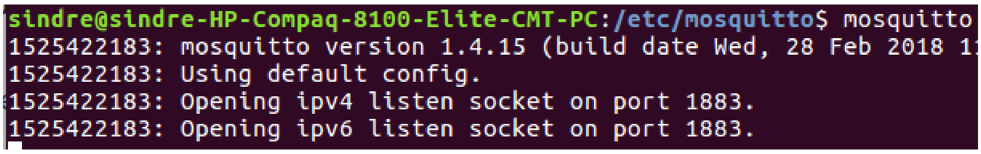
– sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa

– sudo apt-get update

– sudo apt-get install mosquitto

– sudo apt-get install mosquitto-clients

The Mosquitto broker can be initiated with the default configuration by typing "mosquitto" in a terminal window.



```
sindre@sindre-HP-Compaq-8100-Elite-CMT-PC:/etc/mosquitto$ mosquitto
1525422183: mosquitto version 1.4.15 (build date Wed, 28 Feb 2018 1
1525422183: Using default config.
1525422183: Opening ipv4 listen socket on port 1883.
1525422183: Opening ipv6 listen socket on port 1883.
```

Figure 7.4: Mosquitto MQTT broker running on port 1883.

The default Mosquitto configuration will run the broker on port 1883 with no security. Any client can connect to the broker and publish or subscribe to topics. To introduce authentication to the broker the following main steps were followed:

1. Create a CA key pair
2. Create a CA certificate and use the CA key from step 1 to sign it.
3. Create a broker key pair (no password)
4. Create a broker certificate request using the key from step 3.
5. Use the CA certificate to sign the broker certificate from step 4.
6. Copy the CA certificate to the client.
7. Edit the Mosquitto configuration file to enable authentication and reference the certificates created in previous steps.

OpenSSL was used to create the keys and certificates, following the procedure described in Appendix A. The default listening port was changed from 1883 to 8883 (Secure MQTT), and the use of certificates was enabled. The following lines in the configuration file were uncommented, and the certificates and keys were placed in the directories they reference:

– capath /etc/mosquitto/ca_certificates
– keyfile /etc/mosquitto/certs/server.key.pem
– certfile /etc/mosquitto/certs/server.cert.pem

The installation was performed on the AA and the ITS-PC connected to the OBU. After verifying that the default settings were functioning, client scripts allowing file transfer through the broker were created.

The client publisher script uses an MQTT client to connect to a broker hosted at 129.241.208.204 (AA), and publish the content of the directory "pub" to the topic "Cert/Obu". The payload of the publish message is in this example two certificates hosted on the AA.

MQTT publisher script

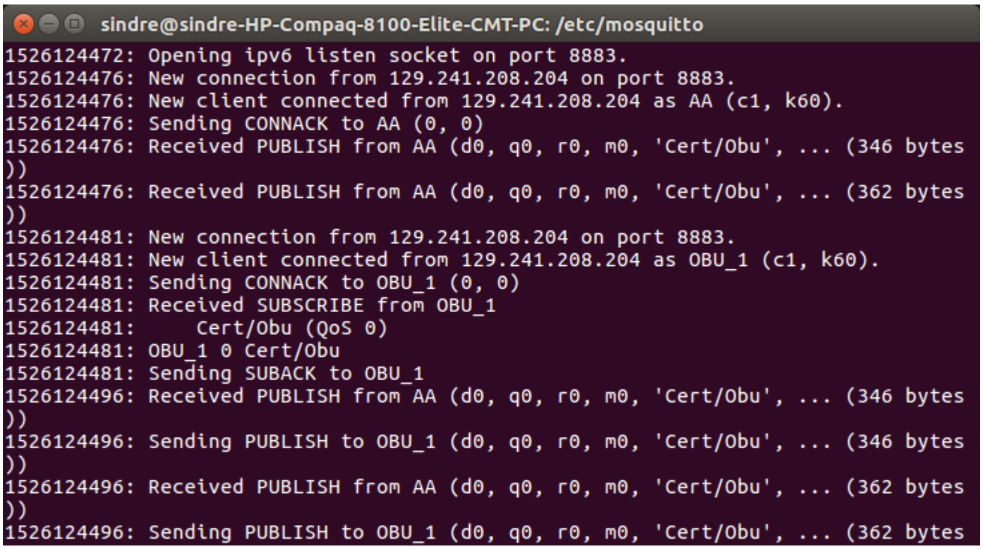
```
############################################################################
import paho.mqtt.client as mqtt
import time
import os

broker_address="129.241.208.204"    #Broker on AA
client = mqtt.Client("AA")          # Start MQTT Client
client.tls_set("/home/sindre/ca_new/certs/ca.cert.pem")
                    #Used for  Server authentication
client.tls_insecure_set(True)
#Allows connection if name on cert does not match signature
    (ONLY USED FOR TESTING)

client.connect(broker_address, 8883, 60)# Connect to broker

client.loop_start() # initial start before loop

while True:
    for file in os.listdir('/home/sindre/pub'): #Publish all
        files in directory
        current_file = os.path.join('/home/sindre/pub', file
            )       #Specify a file to process
        if os.path.isfile(current_file):
                #print current_file For testing
                data = open(current_file, 'rb') #Open file r
                    = read b = binary
                #print data.read() For testing
                certificate = data.read()
                byteArray = bytes(certificate) #Convert to
                    byte string
                client.publish(topic="Cert/Obu", payload=
                    byteArray ,qos=0) # publish to topic

    time.sleep(20)      # Wait
############################################################################
```

The subscriber client connects to the same broker, and subscribes to the topic "Cert/Obu". The payloads are published as byte arrays of the certificates, and the file names are not transferred. For the OBU to be able to use the published certificate, the name must be equal to the last eight bytes of a SHA-256 operation over the certificate. The subscriber-script therefore takes the received payload and performs the operation, and appends the ".cert" extension. The file is then saved with the correct name in a designated directory.

MQTT subscriber script

```
#########################################################################

import paho.mqtt.client as mqtt
import time
import hashlib
import binascii

def on_message(mosq, obj, msg):
#Function to retrieve file when received
          hex = msg.payload
          m = hashlib.sha256() #The name on the cert must be
               the last 8 bytes of a SHA256-digest of the
               payload
          binary = binascii.unhexlify(hex)
          m.update(binary)
          hash = m.hexdigest()
          certname_lower = hash[-16:] #Seperate the last 8
               bytes
          certname = certname_lower.upper()+'.cert' #Make the
               string uppercase and append .cert
     #print certname
          with open('/home/sindre/Documents/'+certname, 'wb')
               as fd: #Path and filename for the file
                    fd.write(msg.payload)



#port = 8883
client = mqtt.Client("OBU_1")
    #Start MQTT Client, ID = OBU_1
client.tls_set("/home/sindre/ca_new/certs/ca.cert.pem","/
    home/sindre/ca_new/client_sub.cert.pem","/home/sindre/
    ca_new/private/client_sub.key.pem")
#client.tls_set("/home/sindre/ca_new/certs/ca.cert.pem")
                              #used for server auth.
client.tls_insecure_set(True) #used for testing, allowing
    insecure connection (mismatch with name on cert)
client.connect("129.241.208.204", 8883, 60)
#Connect to server, port , keepalive
client.subscribe("Cert/Obu",0) #Subscribe to topic

client.on_message = on_message #This calls the function

while True: #Loop and wait for next file
    client.loop(20)

##################################################################
```

The message exchange can be viewed by starting the broker with the -verbose option:



```
sindre@sindre-HP-Compaq-8100-Elite-CMT-PC: /etc/mosquitto
1526124472: Opening ipv6 listen socket on port 8883.
1526124476: New connection from 129.241.208.204 on port 8883.
1526124476: New client connected from 129.241.208.204 as AA (c1, k60).
1526124476: Sending CONNACK to AA (0, 0)
1526124476: Received PUBLISH from AA (d0, q0, r0, m0, 'Cert/Obu', ... (346 bytes
))
1526124476: Received PUBLISH from AA (d0, q0, r0, m0, 'Cert/Obu', ... (362 bytes
))
1526124481: New connection from 129.241.208.204 on port 8883.
1526124481: New client connected from 129.241.208.204 as OBU_1 (c1, k60).
1526124481: Sending CONNACK to OBU_1 (0, 0)
1526124481: Received SUBSCRIBE from OBU_1
1526124481:     Cert/Obu (QoS 0)
1526124481: OBU_1 0 Cert/Obu
1526124481: Sending SUBACK to OBU_1
1526124496: Received PUBLISH from AA (d0, q0, r0, m0, 'Cert/Obu', ... (346 bytes
))
1526124496: Sending PUBLISH to OBU_1 (d0, q0, r0, m0, 'Cert/Obu', ... (346 bytes
))
1526124496: Received PUBLISH from AA (d0, q0, r0, m0, 'Cert/Obu', ... (362 bytes
))
1526124496: Sending PUBLISH to OBU_1 (d0, q0, r0, m0, 'Cert/Obu', ... (362 bytes
```

Figure 7.5: MQTT Mosquitto verbose output.

1. The publisher (AA) connects to the broker on secure port 8883.
2. The subscriber (OBU_1) connects on 8883.
3. The broker receives a PUBLISH method from AA.
4. The broker sends a PUBLISH method to OBU_1, containing the certificate.
5. The certificate has now been copied from the publisher to the subscriber using MQTT secured with TLS.

In this test scenario, two certificates were placed in the "pub" - directory on the AA. The publishing client then pushes the content of the directory to the broker, which distributed the certificates to the specified topic. The ITS-PC subscribes to the topic and downloads the certificates. The ITS-PC simulates the OBU, but the scenario still illustrates how MQTT can be used in the C-ITS PKI. The MQTT protocol's lightweight design and publish/subscribe mechanisms makes it a viable alternative for use in the certificate distribution. Combined with TLS and server/client authentication it provides a flexible, scalable and secure way of distributing ATs to users. The task of administrating the topics and ensuring that the ITS-S receives new ATs in a timely matter will be very comprehensive, but I believe that would be true for all possible solutions.

Chapter 8
Discussion & Conclusion

8.1 Discussion

The security system outlined in this thesis aims to provide authentication and message integrity through the use of a Public Key Infrastructure. It is not hard to imagine a future where all vehicles have access to the internet. Adopting the security technology that has been tried and tested on the world wide web for the C-ITS security enables the designers to use the blueprint from previous implementations. The characteristics of C-ITS and vehicular ad-hoc networks requires the security system to be fast and flexible, as vehicles communicating at high velocities may have a small window for communication. The overhead from the security header and trailer must be as low as possible. This concern has been reflected in the choice not to include confidentiality for the CAM/DENM messages, and in the choice of cryptographic algorithms (ECC). The introduction of PKI in the network adds overhead, as we can see in the frame-size in fig 6.13, but this is considered a necessary trade-off for authentication and message integrity.

The use of PKI is also known to many users from experiences using online banking, shopping and so forth, making the trust process easier to demonstrate. The security system has been developed with a heavy focus on ensuring the privacy of the users. With the use of ATs and the separation of the entities handling user information, it is plausible that tracking individual users through the C-ITS system will prove more difficult than using already existing methods. There are several unknown factors regarding the administration of this system, such as how the CA hierarchy will be established, and if the system will become mandatory in the future. However, the PKI implementation in the test network shows that the security system is capable of facilitating authenticated communication. The results from packet analysis in Wireshark are consistent with the design presented in the literature, and the units can read and verify the certificates. Having this in place is a step in the right direction towards realizing a full-scale PKI for C-ITS.

The manual generation and transfer of certificates is not a sustainable approach, and solutions for the distribution of ATs through the Authorization Authority were also explored. The use of pseudonymity for users through Authentication Tickets presents some challenges, due to the need to resupply users with certificates on a regular basis. Users must be allowed to connect and receive fresh ATs independent of their location, putting high demands on the flexibility of the system. The scale of the system is also an essential factor, as the solution must be able to handle vast numbers of requests from the end-users. This was hard to account for under the testing, as only two Dynniq units were available for that phase. Solutions using HTTP and MQTT were demonstrated, illustrating some of the strengths and weaknesses of the protocols.

Of the two solutions, the use of the MQTT protocol was the most interesting one, as the publisher/subscriber mechanism opens up for a flexible and scalable design. The publisher initiates the connection, and through the use of topics messages can quickly be broadcast to the network. The lightweight design of MQTT means less overhead, and security can be implemented using TLS. On the other hand, I could not find an example of the use of MQTT for file transfer in other large-scale operations. The MQTT broker publishes payloads without a file name, which creates the need for post-processing at the receiving unit. This is not required when using HTTP. The use of HTTP or other file server technology for file transfers can be found in a plethora of systems, and the designs seem transferable to the C-ITS PKI. However, the 1-1 characteristic of the HTTP protocol makes it expensive to broadcast messages to all the units in the network, and the headers and rules in HTTP make it a heavier protocol with more overhead. The request-respond mechanism is initiated by the client (ITS-S), unlike in MQTT, where the publisher (AA) would initiate an eventual update. The tests performed provided a proof of concept for both protocols, but given the restraints of the network, the results from those tests were not unambiguous enough to draw a conclusion on which approach to use going forward. However, the framework is in place for a larger and more realistic test scenario which can further clarify the position.

8.2 Conclusion

In this thesis the use of PKI in C-ITS have been explored, using the policy documents and standards from the EU, as well as a test network for a practical demonstration. The theoretical review shows that C-ITS with a public key infrastructure can through the use of digital signatures and certificates achieve the level of security required by the EU. Secure messaging was implemented in the test network, and the units were able to sign and verify CAMs as described in the policy documents. The secure messages contain a digital signature on the certificate, allowing the receiver to verify the identity of the sender, based on the trust relationship with the entity that signed the certificate. The PKI was expanded to include an Authorization Authority, used to demonstrate both an MQTT-broker and an HTTP-server as solutions for the administration and distribution of user certificates. Both methods are viable options in a small-scale network, but MQTT showed more promise regarding scalability. My recommendation would be to conduct further testing in a more realistic scenario with more ITS-units available to provide a better comparison. The work done in this thesis can form a starting point for a more extensive test. No matter which solution is chosen for the AA, the administration and distribution of the certificates will be a crucial and challenging task for the security system in C-ITS. Aventi has expressed an interest in testing the solutions demonstrated in this thesis in the pilot-project "E8 Borealis" on European route E8 in northern Norway [38].

8.3 Future work

Time and resources did not allow me to implement a solution for the Enrolment Authority for this thesis. The EA is responsible for authenticating the ITS-S to the network using an identifier provided by the vendor or manufacturer. When verified, the ITS-S should receive a token from the EA, which it can use to request services from the AA. It would be interesting to explore the use of the Kerberos authentication protocol for this purpose. Its use of tickets for mutual authentication could prove to be a useful framework in the C-ITS PKI. Certificate Revocation Lists were also on the short-list for features to be implemented. I was unable to come up with a viable solution for the implementation of CRLs, partly because I did not have access to a software development kit (SDK), making any change in how the units handle incoming messages very challenging. The implementation of the features described in previous sections took longer than expected, making it necessary to adjust the ambitions for this PKI implementation. However, with message authentication in place along with a framework for certificate distribution, the door is open for further expansion of the PKI and the creation of applications using secure CAMs as input. The technology used in this thesis will be used during upcoming tests in the "E8 Borealis" project, and it will be interesting to follow the development of C-ITS in the future.

References

[1] ETSI, “Intelligent transport systems (its); communications architecture,” Link: ETSI EN 302 665, note = Figure 1: Scenario illustration.

[2] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 5th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010.

[3] M. TechNet, “Digital certificates,” Link: Microsoft TechNet Digital Certificates, accessed 06.02.2018.

[4] M. Tech-Net, “Certification authority trust model,” Link: Microsoft TechNet Trust Model, accessed 06.02.2018.

[5] European Commission, “Certificate policy for deployment and operation of european cooperative intelligent transport systems (c-its),” Link: Certificate Policy, note = Accessed: 2017-08-27.

[6] *Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 5: Transport Protocols; Sub-part 1: Basic Transport Protocol*, Link : ETSI EN 302 636-5-1, address = 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, institution = European Telecommunications Standards Institute, Standard, May 2017.

[7] E. B. Hamida, H. Noura, and W. Znaidi, “Security of cooperative intelligent transport systems: Standards, threats analysis and cryptographic countermeasures,” *Electronics*, vol. 4, no. 3, pp. 380–423, 2015. [Online]. Available: http://www.mdpi.com/2079-9292/4/3/380

[8] J. Ďurech, M. Franeková, P. Lüley, and E. Bubeníková, “Safety aspects of pki architecture within c-its and their modelling,” in *2016 ELEKTRO*, May 2016, pp. 400–405.

[9] *NIST Recommendation for Key Management, Part 1: General*, Standard.

[10] Car 2 Car Communication Consortium, “Position paper regarding personal data protection aspects in c-its,” accessed: 2017-10-27.

[11] ETSC, “Briefing: 5th eu road safety action programme 2020-2030,” Link: ETSC Briefing, accessed 16.04.2018.

[12] E. T. S. Council, “C-its briefing,” Link: Briefing Cooperative Intelligent Transport Systems (C-ITS), note = Accessed 06.02.2018.

[13] European Commission, “A eu strategy on cooperative, connected and automated mobility,” Link: EU Memo on strategy, accessed: 2017-08-21.

[14] C. Consortium, “C2c consortium,” Link: C2C Homepage, accessed 06.02.2018.

[15] T. A. S. Foundation, “Apache security tips,” Link: Apache Security Tips, accessed 02.05.2018.

[16] M. Khodaei and P. Papadimitratos, “The key to intelligent transportation: Identity and credential management in vehicular communication systems,” *IEEE Vehicular Technology Magazine*, vol. 10, no. 4, pp. 63–69, Dec 2015.

[17] D.-G. for Mobility and T. (EU), “C-its platform final report phase 2,” Link: C-ITS Platform Final Report Phase 2 , note = Accessed 06.02.2018.

[18] S. W. Cadzow, *Security and Privacy for ITS and C-ITS*. Cham: Springer International Publishing, 2015, pp. 283–306. [Online]. Available: https://doi.org/10.1007/978-3-319-15497-8_10

[19] W. Mathworld, “One-way function,” http://mathworld.wolfram.com/One-WayFunction.htmlLink: One-Way Function, accessed 06.02.2018.

[20] J. A. Buchmann, *Cryptographic Hash Functions*. New York, NY: Springer New York, 2004, pp. 235–248. [Online]. Available: https://doi.org/10.1007/978-1-4419-9003-7_11

[21] W. Mathworld, “Trapdoor one-way function,” Link: Trapdoor One-Way Function, accessed 06.02.2018.

[22] C. A. Boyd, “Lecture 11 digital signatures and certificates,” tTM4135 Information Security NTNU Spring 2017.

[23] IBM, “x.509 certificate revocation,” Link: IBM Knowledge Center Certificate Revokation, accessed 15.03.2018.

[24] M. W. D. Center, “Public key infrastructure,” Link: Public Key Infrastructure, note = Accessed: 2017-10-27.

[25] *INTELLIGENT TRANSPORT SYSTEMS (ITS); SECURITY; TRUST AND PRIVACY MANAGEMENT*, Link : ETSI - TS 102 941 , address = 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, institution = European Telecommunications Standards Institute, Standard, Jan. 2012.

[26] IEEE, “Ieee std 802.11p-2010,” Link: IEEE 802.11p, accessed 18.05.2018.

[27] W. Sun, H. Zhang, C. Pan, and J. Yang, “Analytical study of the ieee 802.11p edca mechanism,” in *2013 IEEE Intelligent Vehicles Symposium (IV)*, June 2013, pp. 1428–1433.

[28] *Intelligent Transport Systems (ITS); Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band*, Link : ETSI EN 302 663 , institution = European Telecommunications Standards Institute, Standard.

[29] *Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 3: Network architecture*, Link : ETSI TS 102 636-3, address = 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, institution = European Telecommunications Standards Institute, Standard, march 2010.

[30] *Intelligent Transport Systems (ITS); Security; Security header and certificate formats*, Link:ETSI TS 103 097 V1.3.1 , address = 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, institution = European Telecommunications Standards Institute, Standard, Oct. 2017.

[31] *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service*, Link : ETSI EN 302 637-2, address = 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, institution = European Telecommunications Standards Institute, Standard, nov 2014.

[32] ETSI, "Intelligent transport systems (its); security; threat, vulnerability and risk analysis (tvra)," Link: ETSI TVRA, note = Accessed 13.03.2018.

[33] A. Al Imem, "Comparison and Evaluation of Digital Signature Schemes Employed in NDN Network," vol. 5, 08 2015.

[34] EUGDPR, "Gdpr," Link: GDPR, accessed 05.03.2018.

[35] N. B. Thomas, "U.s. dot advances deployment of connected vehicle technology to prevent hundreds of thousands of crashes," Link: U.S. DOT Advances Deployment Of Connected Vehicle Technology To Prevent Hundreds Of Thousands Of Crashes, accessed 09.05.2018.

[36] MQTT.org, "Mqtt faq," Link: MQTT FAQ, accessed 02.05.2018.

[37] HiveMQ, "Mqtt essentials part 3: Client, broker and connection establishment," Link: MQTT Essentials Part 3: Client, Broker and Connection Establishment, accessed 02.05.2018.

[38] S. vegvesen, "E8 borealis project," Link: E8 Borealis Homepage, accessed 18.05.2018.

[39] J. Nguyen, "Openssl certificate authority," OpenSSL Certificate Authority, accessed 17.04.2018.

Appendix A
OpenSSL Certificate Hierarchy

OpenSSL is an open source general-purpose cryptography library, which can be used to create cryptograhic keys and certificates. It comes pre-configured on the ITS-units, and can be accessed from the command line. The basic PKI used for the initial testing consisted of a self-signed root CA and a root-signed intermediate CA. The self-signed root CA was created and stored on the ITS-PC through OpenSSL. The first step was to create the directory structure to hold the files needed.

- **mkdir root/ca**
- **cd root/ca**
- **mkdir certs crl private newcerts**
- **touch index.txt**
- **echo 1000 > serial**

To use OpenSSL to create certificates and sign, a configuration file must be used for each signing entity. That means that both the root CA and the intermediate CA must have its dedicated openssl.cnf file. For this test, a guide on how to create your own CA hierarchy was used. The guide can be found at [39].

Creating the root CA

The root key will be used to sign the root certificate and the intermediate CA. The holder of the private key can issue trusted certificates, so it is paramount that it is kept secret. The following command was used to create the root key:

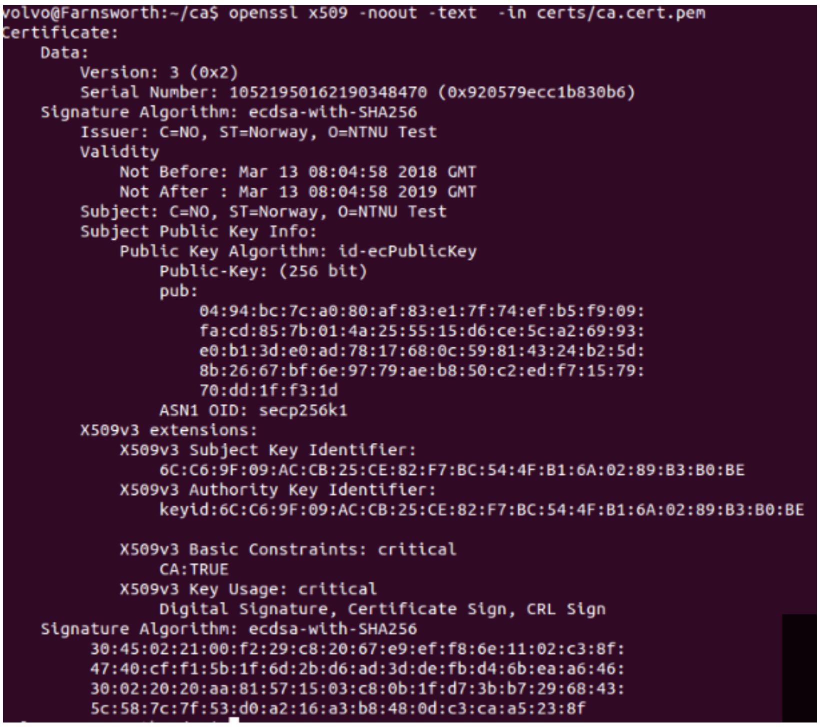
root/ca/private$ **openssl ecparam -genkey -name secp256k1 -noout -out ca.key.pem**

The output is the **ca.key.pem**, stored in the /private directory. Now that we have the key, we can use it to create a root certificate:

#cd /root/ca
#openssl req -config openssl.cnf -key private/ca.key.pem -new -x509 -days 7300 -sha256 -extensions v3_ca -out certs/ca.cert.pem

The certificate is then verified:

openssl x509 -noout -text -in certs/ca.cert.pem



```
volvo@Farnsworth:~/ca$ openssl x509 -noout -text  -in certs/ca.cert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 10521950162190348470 (0x920579ecc1b830b6)
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=NO, ST=Norway, O=NTNU Test
        Validity
            Not Before: Mar 13 08:04:58 2018 GMT
            Not After : Mar 13 08:04:58 2019 GMT
        Subject: C=NO, ST=Norway, O=NTNU Test
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:94:bc:7c:a0:80:af:83:e1:7f:74:ef:b5:f9:09:
                    fa:cd:85:7b:01:4a:25:55:15:d6:ce:5c:a2:69:93:
                    e0:b1:3d:e0:ad:78:17:68:0c:59:81:43:24:b2:5d:
                    8b:26:67:bf:6e:97:79:ae:b8:50:c2:ed:f7:15:79:
                    70:dd:1f:f3:1d
                ASN1 OID: secp256k1
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                6C:C6:9F:09:AC:CB:25:CE:82:F7:BC:54:4F:B1:6A:02:89:B3:B0:BE
            X509v3 Authority Key Identifier:
                keyid:6C:C6:9F:09:AC:CB:25:CE:82:F7:BC:54:4F:B1:6A:02:89:B3:B0:BE

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: ecdsa-with-SHA256
         30:45:02:21:00:f2:29:c8:20:67:e9:ef:f8:6e:11:02:c3:8f:
         47:40:cf:f1:5b:1f:6d:2b:d6:ad:3d:de:fb:d4:6b:ea:a6:46:
         30:02:20:20:aa:81:57:15:03:c8:0b:1f:d7:3b:b7:29:68:43:
         5c:58:7c:7f:53:d0:a2:16:a3:b8:48:0d:c3:ca:a5:23:8f
```

Figure A.1: Verification of the root CA certificate.

Creating the intermediate CA

The next step was to create a key/certificate pair for the intermediate CA. .

Directory structure for the intermediate CA:

```
# mkdir root/ca/intermediate

# cd /root/ca/intermediate

# mkdir certs crl csr newcerts private

# chmod 700 private

# touch index.txt

# echo 1000 > serial
```

The openssl.cnf file for the intermediate CA was placed in the /intermediate directory. The private key was generated using the same command as the root key:
root/ca/intermediate/private$ **openssl ecparam -genkey -name secp256k1 -noout -out intermediate.key.pem**
The intermediate certificate is not self-signed, therefore we first create a certificate signing request (CSR), which can be signed by the root CA.

```
# cd /root/ca
# openssl req -config intermediate/openssl.cnf -new -sha256 -key intermediate/private/intermediate.key.pem -out intermediate/csr/intermediate.csr.pem
```

The output is a CSR, stored in the /intermediate/csr directory. We then sign the signing request with the root CA key:

```
# cd /root/ca

# openssl ca -config openssl.cnf -extensions v3_intermediate_ca -days 3650 -notext -md sha256 -in intermediate/csr/intermediate.csr.pem -out intermediate/certs/intermediate.cert.pem
```

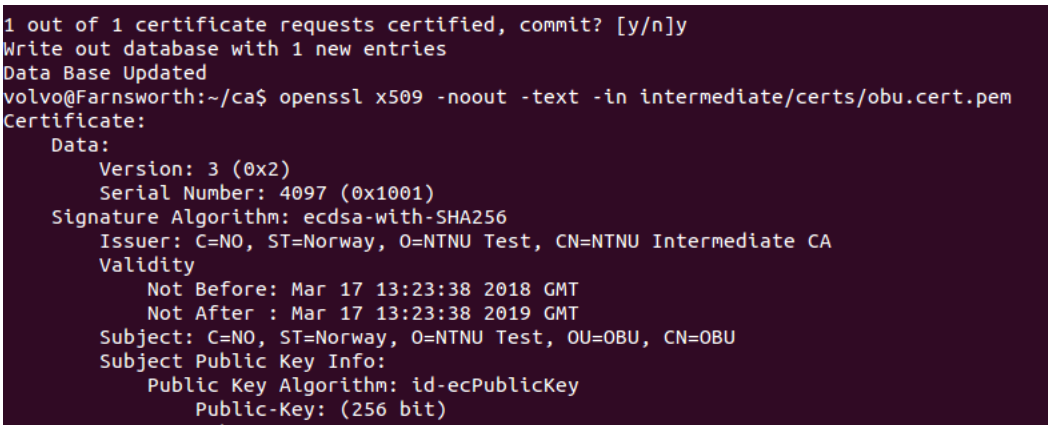
Verify that the details of the intermediate certificate is OK:

```
openssl verify -CAfile certs/ca.cert.pem intermediate/certs/intermediate.cert.pem
```

Result: **intermediate.cert.pem: OK**

Creating & signing the device certificates

The next step is to create a certificate that can be used by the OBU. First we create a private key to be used for signing. Then we use that key to generate a certificate signing request. The intermediate CA signs the CSR, thus creating the OBU certificate. The process was repeated for the RSU.



```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
volvo@Farnsworth:~/ca$ openssl x509 -noout -text -in intermediate/certs/obu.cert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4097 (0x1001)
    Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=NO, ST=Norway, O=NTNU Test, CN=NTNU Intermediate CA
        Validity
            Not Before: Mar 17 13:23:38 2018 GMT
            Not After : Mar 17 13:23:38 2019 GMT
        Subject: C=NO, ST=Norway, O=NTNU Test, OU=OBU, CN=OBU
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
```

Figure A.2: Partial view of the OBU certificate. The intermediate CA is listed as the issuer.

The certificates and keys were deployed to the units using scp.
On the OBU /etc/its-security/certs:

- ca.cert.pem
- intermediate.cert.pem
- obu1.cert.pem
- obu2.cert.pem

On the OBU /etc/its-security/keys:

- obu1.key.pem
- obu2.key.pem

On the RSU /etc/its-security/certs:

– ca.cert.pem
– intermediate.cert.pem
– rsu1.cert.pem
– rsu2.cert.pem

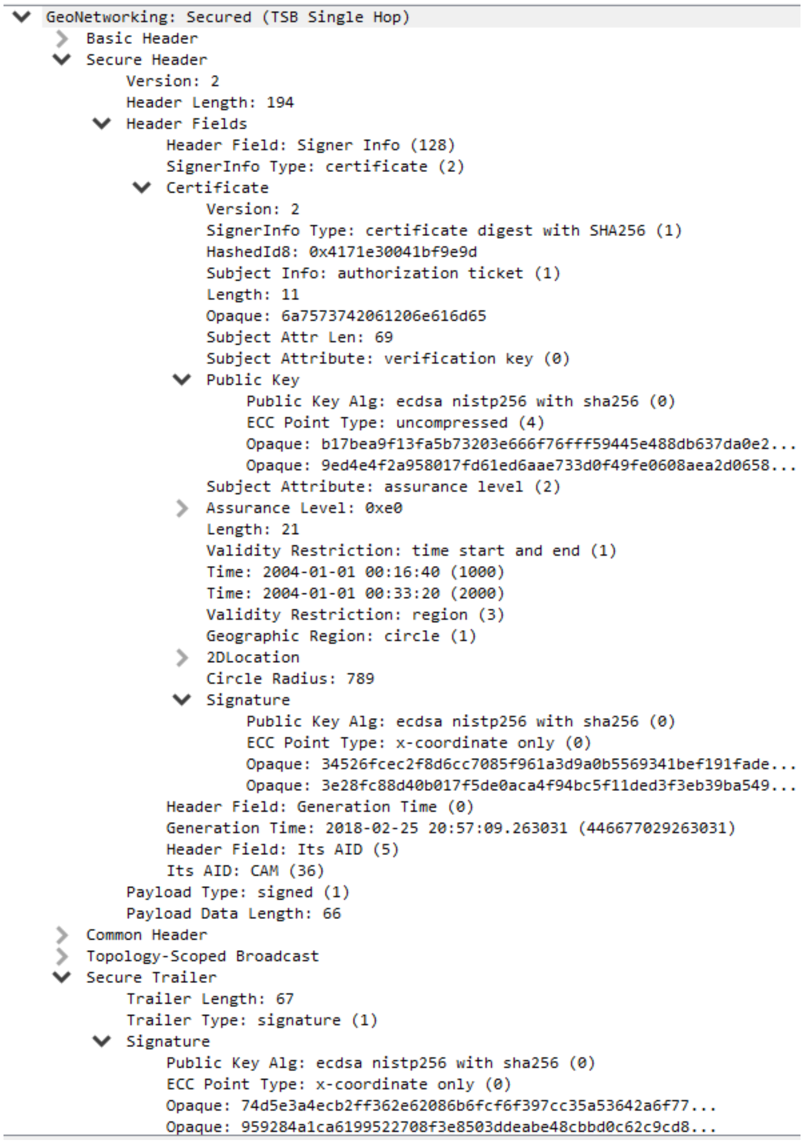
On the RSU /etc/its-security/keys:

– rsu1.key.pem
– rsu2.key.pem

The Greenflow application was used to inspect the certificate status on the devices. Neither the RSU nor OBU registered the certificates. The secure mode was enabled, but with negative results. The OBU and RSU were not able to communicate in secure mode using certificates created by OpenSSL.

Appendix B
Wireshark CAM packet

GeoNetwork header in secure mode



```
GeoNetworking: Secured (TSB Single Hop)
    Basic Header
    Secure Header
        Version: 2
        Header Length: 194
        Header Fields
            Header Field: Signer Info (128)
            SignerInfo Type: certificate (2)
            Certificate
                Version: 2
                SignerInfo Type: certificate digest with SHA256 (1)
                HashedId8: 0x4171e30041bf9e9d
                Subject Info: authorization ticket (1)
                Length: 11
                Opaque: 6a7573742061206e616d65
                Subject Attr Len: 69
                Subject Attribute: verification key (0)
                Public Key
                    Public Key Alg: ecdsa nistp256 with sha256 (0)
                    ECC Point Type: uncompressed (4)
                    Opaque: b17bea9f13fa5b73203e666f76fff59445e488db637da0e2...
                    Opaque: 9ed4e4f2a958017fd61ed6aae733d0f49fe0608aea2d0658...
                Subject Attribute: assurance level (2)
                Assurance Level: 0xe0
                Length: 21
                Validity Restriction: time start and end (1)
                Time: 2004-01-01 00:16:40 (1000)
                Time: 2004-01-01 00:33:20 (2000)
                Validity Restriction: region (3)
                Geographic Region: circle (1)
                2DLocation
                Circle Radius: 789
                Signature
                    Public Key Alg: ecdsa nistp256 with sha256 (0)
                    ECC Point Type: x-coordinate only (0)
                    Opaque: 34526fcec2f8d6cc7085f961a3d9a0b5569341bef191fade...
                    Opaque: 3e28fc88d40b017f5de0aca4f94bc5f11ded3f3eb39ba549...
            Header Field: Generation Time (0)
            Generation Time: 2018-02-25 20:57:09.263031 (446677029263031)
            Header Field: Its AID (5)
            Its AID: CAM (36)
        Payload Type: signed (1)
        Payload Data Length: 66
    Common Header
    Topology-Scoped Broadcast
    Secure Trailer
        Trailer Length: 67
        Trailer Type: signature (1)
        Signature
            Public Key Alg: ecdsa nistp256 with sha256 (0)
            ECC Point Type: x-coordinate only (0)
            Opaque: 74d5e3a4ecb2ff362e62086b6fcf6f397cc35a53642a6f77...
            Opaque: 959284a1ca6199522708f3e8503ddeabe48cbbd0c62c9cd8...
```

Figure B.1: The expanded headers in a secure CAM packet from Wireshark.

Appendix C

AppendixC

The configuration below contains the sections that were edited during the work with the Mosquitto broker.

```
# =================================================================
# Default listener
# =================================================================

# IP address/hostname to bind the default listener to. If not
# given, the default listener will not be bound to a specific
# address and so will be accessible to all network interfaces.
# bind_address ip-address/host name
#bind_address

# Port to use for the default listener.
#port 1883
port 8883
# The maximum number of client connections to allow. This is
# a per listener setting.
# Default is -1, which means unlimited connections.
# Note that other process limits mean that unlimited connections
# are not really possible. Typically the default maximum number of
# connections possible is around 1024.
#max_connections -1

# Choose the protocol to use when listening.
# This can be either mqtt or websockets.
# Websockets support is currently disabled by default at compile time.
# Certificate based TLS may be used with websockets, except that
# only the cafile, certfile, keyfile and ciphers options are supported.
#protocol mqtt

# When a listener is using the websockets protocol, it is possible to serve
# http data as well. Set http_dir to a directory which contains the files you
# wish to serve. If this option is not specified, then no normal http
# connections will be possible.
#http_dir
```

```
# Set use_username_as_clientid to true to replace the clientid that a client
# connected with with its username. This allows authentication to be tied to
# the clientid, which means that it is possible to prevent one client
# disconnecting another by using the same clientid.
# If a client connects with no username it will be disconnected as not
# authorised when this option is set to true.
# Do not use in conjunction with clientid_prefixes.
# See also use_identity_as_username.
#use_username_as_clientid

# -----------------------------------------------------------------
# Certificate based SSL/TLS support
# -----------------------------------------------------------------
# The following options can be used to enable SSL/TLS support for
# this listener. Note that the recommended port for MQTT over TLS
# is 8883, but this must be set manually.
#
# See also the mosquitto-tls man page.

# At least one of cafile or capath must be defined. They both
# define methods of accessing the PEM encoded Certificate
# Authority certificates that have signed your server certificate
# and that you wish to trust.
# cafile defines the path to a file containing the CA certificates.
# capath defines a directory that will be searched for files
# containing the CA certificates. For capath to work correctly, the
# certificate files must have ".crt" as the file ending and you must run
# "c_rehash <path to capath>" each time you add/remove a certificate.
#cafile
capath /etc/mosquitto/ca_certificates

# Path to the PEM encoded server certificate.
certfile /etc/mosquitto/certs/server.cert.pem

# Path to the PEM encoded keyfile.
keyfile /etc/mosquitto/certs/server.key.pem

# This option defines the version of the TLS protocol to use for this listener.
# The default value allows v1.2, v1.1 and v1.0, if they are all supported by
# the version of openssl that the broker was compiled against. For openssl >=
# 1.0.1 the valid values are tlsv1.2 tlsv1.1 and tlsv1. For openssl < 1.0.1 the
# valid values are tlsv1.
#tls_version

# By default a TLS enabled listener will operate in a similar fashion to a
# https enabled web server, in that the server has a certificate signed by a CA
# and the client will verify that it is a trusted certificate. The overall aim
# is encryption of the network traffic. By setting require_certificate to true,
# the client must provide a valid certificate in order for the network
# connection to proceed. This allows access to the broker to be controlled
# outside of the mechanisms provided by MQTT.
 require_certificate true

# If require_certificate is true, you may set use_identity_as_username to true
# to use the CN value from the client certificate as a username. If this is
# true, the password_file option will not be used for this listener.
#use_identity_as_username true

# If you have require_certificate set to true, you can create a certificate
# revocation list file to revoke access to particular client certificates. If
# you have done this, use crlfile to point to the PEM encoded revocation file.
#crlfile

# If you wish to control which encryption ciphers are used, use the ciphers
# option. The list of available ciphers can be obtained using the "openssl
# ciphers" command and should be provided in the same format as the output of
# that command.
# If unset defaults to DEFAULT:!aNULL:!eNULL:!LOW:!EXPORT:!SSLv2:@STRENGTH
#ciphers DEFAULT:!aNULL:!eNULL:!LOW:!EXPORT:!SSLv2:@STRENGTH
```