POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

OCPP Protocol in the Smart Charging Era: Formal Verification of Security-Related Use Cases through ProVerif Analysis.

Supervisors

Prof. Riccardo SISTO

Dott. Simone BUSSA

Candidate

Rebecca FALCO

2023/2024

Abstract

With the growing adoption of electric vehicles and the definition of the Agenda 2030 goals, the demand for efficient charging infrastructure has significantly increased. The need for secure and reliable communication between Charging Stations (CS) and Charging Stations Management System (CSMS) is crucial for the digitalisation and management of the charging process. From the study of the landscape of protocols proposed for this purpose, the Open Charge Point Protocol (OCPP) has emerged as the de facto standard for communication between charging stations and management systems. Accordingly, the thesis work proceeded examining the evolution and key characteristics of the OCPP protocol, developed by the Open Charge Alliance (OCA) to enable interoperability and smart features in EV charging networks. As the electric vehicle infrastructure expands, so do concerns over the security and trustworthiness of communication systems: vulnerabilities in the charging infrastructure could lead to potential cyberattacks. The security of charging transactions data exchange is, therefore, of paramount importance. This study focused on OCPP's use cases most relevant to security, with emphasis on direct manipulation of security parameters. The Formal Verification of these use cases is carried out through the modelling of data exchange, authentication security profiles, and cryptographic algorithms, using the ProVerif formal prover to analyse and validate the security properties such as Confidentiality, Integrity and Authentication of the Charging Station (CS) to the Charging Station Management System (CSMS) communication protocol. This thesis provides a detailed examination of the use cases and their associated security measures, ultimately confirming that the OCPP protocol ensures secure communication in the studied scenarios, contributing to the goal of a safe and sustainable e-mobility ecosystem.

Grazie
Ai miei professori per avermi ispirata
Alle mie fantastiche amiche
Ai miei genitori
A Diego

A tutti quelli che hanno rispettato i miei silenzi
e festeggiato i miei successi

Table of Contents

List of Figures

Acronyms

ACEA European Automobile Manufacturers' Association

AES Advanced Encryption Standard

BEV Battery Electric Vehicle

CA Certification Authority

CHO Clearing House Operator

CIA Confidentiality, Integrity, Availability

CS Charging Station

CSMS Charging Station Management System

CSO Charging Station Operator

DDoS Distributed Denial of Service

DH Diffie–Hellman

DSO Distribution System Operator

DUT Device Under Test

ECDHE Elliptic-curve Diffie–Hellman

ECDSA Elliptic Curve Digital Signature Algorithm

EMS Energy Management Systems

eMSP e-Mobility Service Provider

EU European Union

EV	Electric Vehicle
EVSE	EV supply equipment
GCM	Galois Counter Mode
GDPR	General Data Protection Regulation
HTTP	Hyper Text Transfer Protocol
IoT	Internet of Things
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
NSP	Network Service Provider
OCA	Open Charge Alliance
OCHP	Open Clearing House Protocol
OCPI	Open Charge Point Interface
OCPP	Open Charge Point Protocol
OOB	Out Of Band
OSCP	Open Smart Charging Protocol
PHEV	Plug-in Hybrid Electric Vehicle
PICS	Protocol Implementation Conformance Statement
PRF	Pseudorandom Function
REST	Representational state transfer
RFID	Radio frequency identification
SHA	Secure Hash Algorithm
TLS	Transport Layer Security
V2G	Vehicle to Grid

Chapter 1

Introduction

1.1 Introduction

In light of the ambitious Agenda 2030 and European Green Deal objectives, the growing adoption of electric vehicles is creating a market need for effective communication in the Electric Vehicle charging process. The Open Charge Alliance is responsible for the development of OCPP, an open protocol that, within the landscape of available protocols for this purpose, facilitates interoperability and seamless data exchange between the entities involved in the smart charging ecosystem. In recent years, the OCPP has become the de facto standard for communication between Charging Stations and Charging Station Management Systems. The increasing evolution of smart connectivity also introduces potential targets for cyberattacks. The first challenge in cybersecurity is preventing possible security threatening events, and one effective way to address this challenge is performing Formal Verification. Given this introduction to the entities and needs involved, this study focuses on OCPP's sections most relevant to security, performing formal verification carried out using the ProVerif formal prover. The scope of the verification is to analyse and validate the security properties such as confidentiality, integrity and authentication. This thesis provides a detailed examination of the use cases, their associated security measures and the verification result interpretation.

1.2 Structure of the document

This section contains a summary of the content of each charper of this thesis, to better explain the work done.

1. **Introduction**
   Brief introduction to the chapters and the objective of this thesis.

2. **State of the art of Electric Charging**
   Introduction to the state of the art of electric charging scenario and the digitisation process. It includes the state of the art of communication protocols between the entities involved in the charging process and an in detail explaination of the concepts of openness and interoperability of open protocols, and comparing these characteristics to proprietary solutions.

3. **The OCPP Protocol**
   Introduction to OCPP: milestones and values underlying its continuous evolution, presentation of the main features and its specification architecture, reasons for its spreading global adoption.

4. **The ProVerif Tool**
   The concept of formal verification with a description of the ProVerif tool principles and usage for security properties analysis. Presentation of possible and desirable results, with a brief introduction to the concept of undecidability.

5. **ProVerif model of OCPP's Use Cases**
   Summary of the protocol requirements for the selected use cases: the model implementation in ProVerif input language is provided, along with the detailed descriptions of the entities involved, the messages exchanged, the communication and cryptographic functions, and the overall decisions taken for the implementation.

6. **Results of the analysis**
   Analysis of properties required for the use cases with detailed description of the ProVerif verification model. Description of the queries, interpretation of the results and analisys of correspondence to expectations.

7. **Conclusions and future work**
   Summary of the results, possible further evolution and suggestions for future work.

1.3 Objective of the thesis

Given the state of the art of the Electric Vehicles global adoption, a preliminary stage and first objective of this thesis has been the examination of the smart charging protocols and a study of the main charateristics of the most relevant ones. A comprehensive study and detailed report on the OCPP protocol evolution, defining characteristics and architecture outlines how the objectives of the analysis have been selected. For the scope of this thesis, the use cases, which are section of the protocol with a specific purpose, that have been selected are the most critical in terms of security, with less secure requirements and direct manipulation of security parameters. This has required a comprehensive understanding of the OCPP architectural design, its associated specifications, and the algorithms that support its communication and security protocols. The successive phase, and main objective, of this thesis work has been to perform Formal Verification against the ProVerif formal prover. This stage has required to model the use cases, the security profiles and the algorithms of the cipher suite required by OCPP specification. The model of each selected protocol section has then been analysed through the definition of a set of property queries. The final stage of this process has been the interpretation of the results, with the final aim to provide a reliable interpretation of the ProVerif formal verification outcome about the analysed security properties.

Chapter 2

State of the art of Electric Charging

2.1 Introduction to Electric Vehicles Charging scenario

2.1.1 Evolution and future

In recent years, theAgenda 2030 and the European Green Deal [9] have highlighted the importance of promoting low-emission vehicles in the transport sector. In fact, road transport accounts for the highest percentage of emissions from transport and is therefore an important part of the package of strategic initiatives aimed at putting the EU on the path to a green transition. To achieve these goals, EU Member States must take concrete measures to reduce emissions and decarbonise the economy. Among the initiatives included in the Green Deal, the 55 % Ready package aims to translate the Green Deal's ambition to achieve climate neutrality by 2050 into legislation. In the series of proposals to align EU legislation with its climate change objectives, we find new rules on CO2 emissions from cars and vans, with progressive EU-wide emission reduction targets for 2030 and beyond, and an end to the sale of light-duty internal combustion engine vehicles by 2035.

The response of the car market to these changes is reflected in the number of new registrations of vehicles with reduced environmental impact recorded by Acea, the European Automobile Manufacturers' Association, which shows a clear market direction in the percentage of fuel type of new cars registered [Figure 2.1]. As the market for battery-powered vehicles grows, so does the market for charging stations: there has been a significant increase in the number of electric vehicle charging points in both residential and public areas. There is a clear correlation [Figure 2.2] between the availability of public charging points and the sales of battery electric vehicles: buyers want more EV chargers to overcome range anxiety and protect their investment in an electric vehicles it is essential that the necessary infrastructure is in place to facilitate the widespread adoption of electric vehicles.

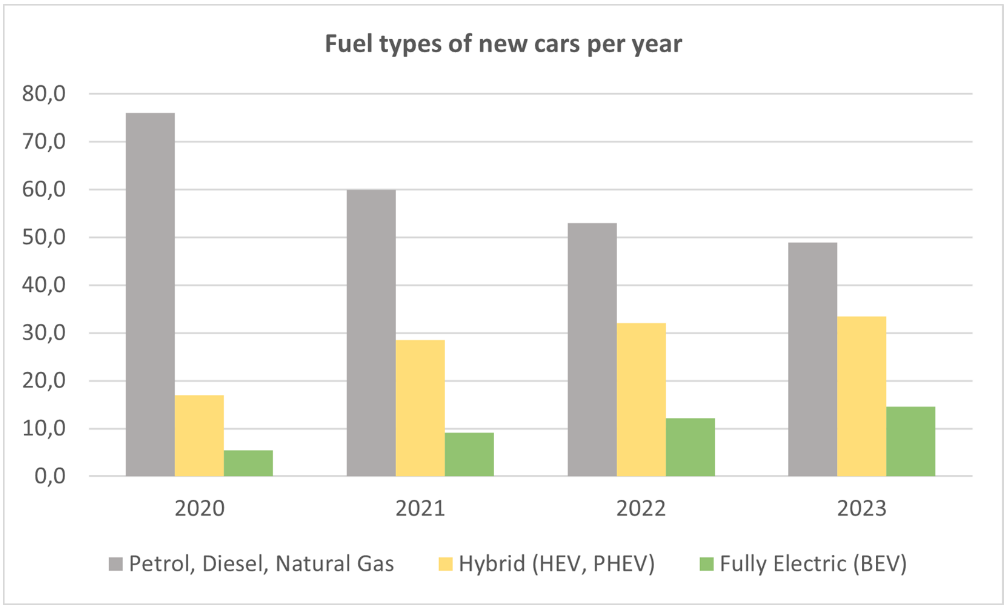


Figure 2.1: Fuel Types Of New Cars Per Year, from ACEA report Q3 2024

The European Commission has estimated that 3.5 million charging points must be installed by 2030 to accommodate the anticipated 30 million electric vehicles on European roads [10]. However, the ACEA has provided a higher estimate of 8.8 million. The deployment of publicly accessible charging infrastructure for light electric vehicles is uneven across the Union, so cooperation across the e-mobility ecosystem is key to rapidly scaling up Europe's transition to electric vehicles. In view of the above, the ACEA and the electric vehicle charging industry ChargeUp Europe have signed a joint declaration to strengthen their cooperation and dialogue [19]. The agreement for the deployment of publicly accessible recharging infrastructure sets out minimum targets for Member States at the national level, based on the total number of registered electric vehicles in each Member State. These targets should be set using a uniform methodology that accounts for technological advancements and should integrate distance-based targets for the trans-European transport network.

The effects of digitalization and need of trustworthiness Digital technologies have become an integral part of vehicle development, from controlling critical driving functions (e.g., airbags, brakes, and steering) to delivering convenience features. EVs are not different, relying primarily on the battery pack and its

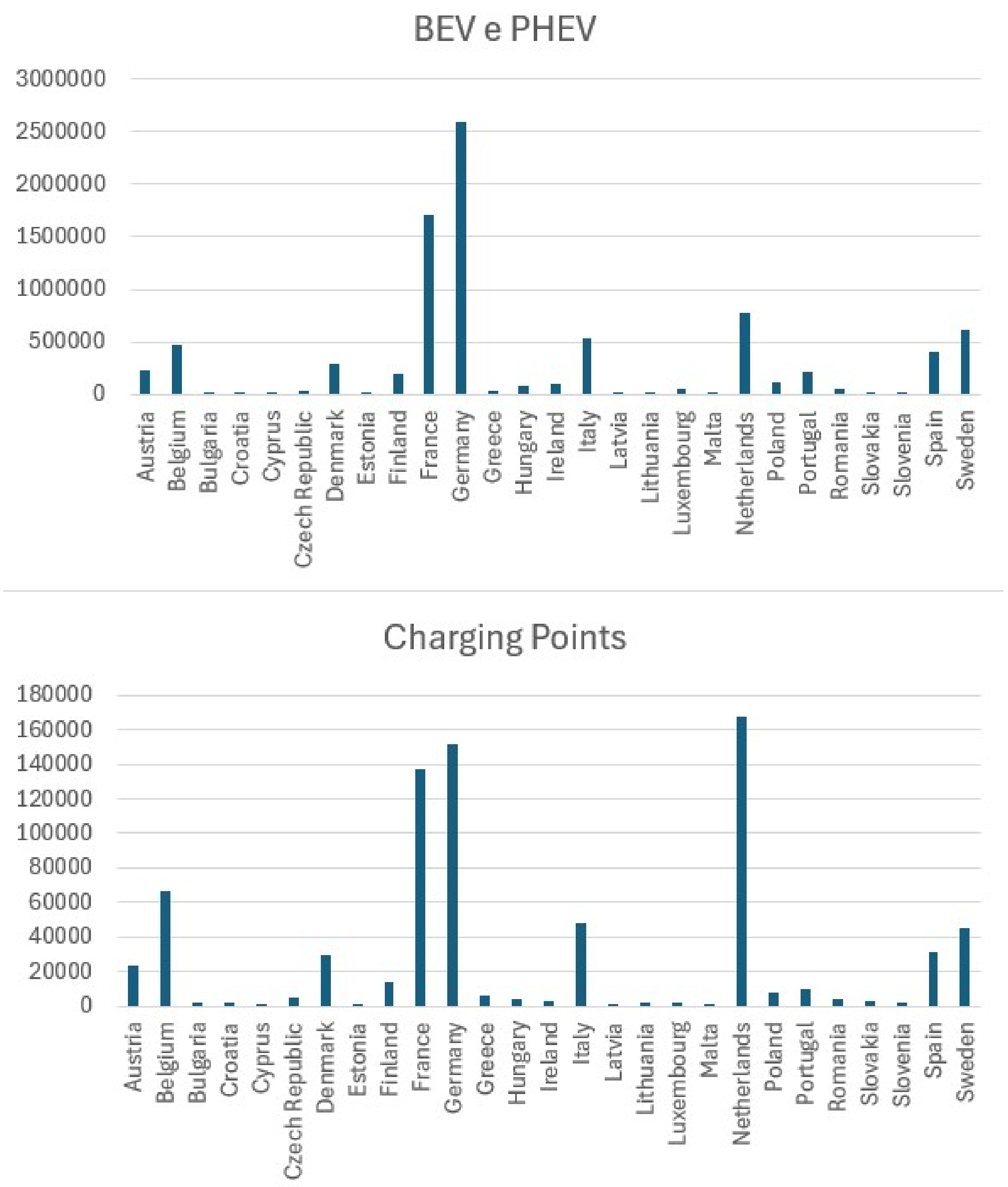


Figure 2.2: Charging Points per Country, from ACEA report Q1 2024

charging systems. It is evident that the electrification process is having a relevant impact on the automotive industry. This transformation is also affecting digital infrastructure. In the utility sector, for example, the need of electricity to power the existing vehicles has significant implications for the way distribution grids are managed. There is also potential for benefits from the grid management, allowing EVs to feed their stored energy back to the electricity network, or through charge point operators aggregating their charging networks, for flexible demand.

In the long term, digitalisation will be essential for integrating decentralised forms of renewable energy sources into the electricity grid, reducing the dependence on fossil fuels. To achieve this integration, a cyber response focused on the energy sector is required. This should include improving cyber resilience and developing and implementing a specific European cyber security maturity framework for the energy sector: a public-private partnership to increase resilience of the supply chain of the energy industry.

One of the key factors driving the successful transition to electric vehicles is the readiness to embrace innovation and confidence in technology. By improving the efficiency of EVs, the transportation industry can become more environmentally friendly, efficient, and adaptable. However, user data security and privacy concerns are increasing, as more and more information is collected, including personal and financial data, car-related data, and charging logs reporting location coordinates and driving habits.

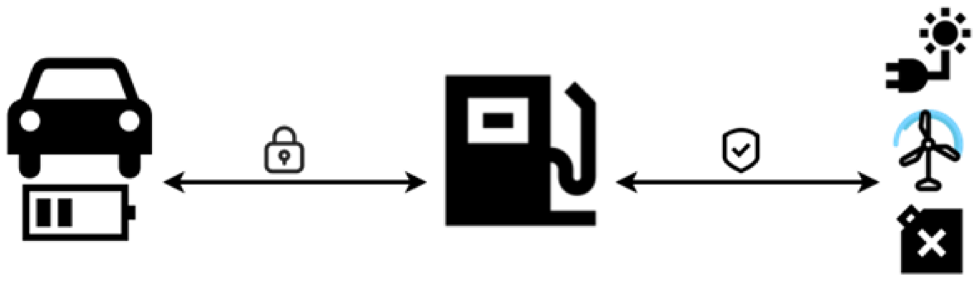


Figure 2.3: Safety and Availability of the chain

With this increase in EV use and adoption comes an increase in cybersecurity risks for EV Supply Equipment and the wider EV charging infrastructure ecosystem. First and foremost, the connection and supply system must be secure and available [Figure 2.3], otherwise malicious actors could access user data, disrupt charging or even cause a supply chain blackout. Collaborative projects will lead to innovative solutions that, as technologies evolve and standards are harmonised across existing and emerging technologies, will help ensure that the future of EV charging is secure. Key strategies to improve system reliability and security include testing emerging EV charging technologies for cybersecurity vulnerabilities, coordinating cybersecurity risk management strategies and approaches among EV stakeholders, improving secure communications within the EV charging infrastructure, and evaluating and coordinating EVSE cybersecurity standards.

2.1.2 Securing the Infrastructure

Smart and V2G chargers connect EVs to the power grid using a charging device that includes a data connection to exchange information and control commands between various entities in the EV ecosystem. Implementing a secure system is therefore crucial to enable both consumers and grid operators to trust smart charging and V2G. This connectivity introduces a potential target for cybersecurity attacks, with examples of threats including unauthorized access to information (e.g., banking details), tampering (e.g., energy used), and denial of service (e.g., unavailability of the charger).

To address these risks, the cybersecurity measures can be divided into prevention, detection, and recovery. These categories can be further expanded to include preventive measures and planning, security audits and verification, incident detection, response to cyber incidents, and system resilience. Preventive measures encompass risk assessment, planning, and the implementation of security protocols. Security audits involve a thorough review of an organization's defenses and form a crucial part of a comprehensive risk management strategy, including policy verification, attack simulations, and activity monitoring. Incident response entails the readiness to detect and recover from cyber incidents efficiently, adapting to the ever-changing landscape of digital threats.

Identifying cybersecurity risks

The importance of data privacy requirements that should be applied to connected smart charging systems is highlighted by new government regulations and guidance, including the General Data Protection Regulation, a European Union regulation that governs how companies and other organisations process personal data and provides guidance on the Internet of Things and cybersecurity principles for connected and automated vehicles. The GDPR specifically, define and protects fundamental rights of the users have to be trasparently informed on the use and it's rights, to access, rectify or erase their data, to restrict or object processing, also related to automated decision making and profiling, and others.

Taking into account the state of the art at the time of the data processing, the controller and the processor must ensure a level of security appropriate to the risk, guaranteeing the aforementioned properties of confidentiality, integrity, availability and resilience of the processing systems and services. According to the GDPR, this requires designing information processing with appropriate security measures, including regular testing, assessment and evaluation of their effectiveness, to ensure the security of processing.

As emphasised by NIST, a framework that stand out as the most highly regarded, in the guideline about incident response [Figure 2.4] the first challenge in cybersecurity is preventing possible security threatening events. The Identify function is perhaps the most crucial as it requires assessing the cybersecurity risk faced by the organisation.

What are the procedures for identifying such vulnerabilities? There are several methods of security assessment, which involve evaluating the overall security of a system. One fundamental method is vulnerability assessment, which aims to provide a list of security issues by predicting and classifying vulnerabilities. This is achieved by scanning the system to find weak configurations using analysis tools which can detect known vulnerabilities, such as errors or outdated software.

IDENTIFY
PROTECT
DETECT
RESPOND
RECOVER

Figure 2.4: NIST guideline main steps

Security assessment methods are divided into static and dynamic approaches, where dynamic is based on testing the running system, while static is the analysis of code, protocols and general settings. The dynamic ones include penetration testing, the execution of simulated attacks on the system under test. Static ones include static code analysis and formal verification: the static code analysis is about examining the code without executing it, looking for potential bugs, to identify known patterns or logic errors and improve the overall quality of the code during development.

Finally the one that this thesis aims to apply as part of the prevention measures: Formal Verification [further details at chapter 4]. This method focuses on the validation and analysis of system and protocols implementation, to ensure that they comply with security standards and do not contain any vulnerabilities. In other words, formal verification helps to identify and address potential risks before they can be exploited by cyber-attacks.

2.2 State of the Art of EV Charging

2.2.1 EV Charging Scenario

To provide a seamless and efficient electric charging experience for EV users, a network of specialised actors works together [Figure 2.5]. The first actor is the owner of the electric vehicle, who has the role of registering with an e-Mobility Service Provider (eMSP) and a Network Service Provider (NSP). They provide charging authorisation, manage charging point reservations and support the billing process. The Charging Station (CS), which is the physical infrastructure where the EV plugs in to recharge, operates under the control of the Charging Station Management System (CSMS), which has the role not only of monitoring the overall stability of the infrastructure, but also of communicating directly with the EV charger and monitoring its usage and health status.

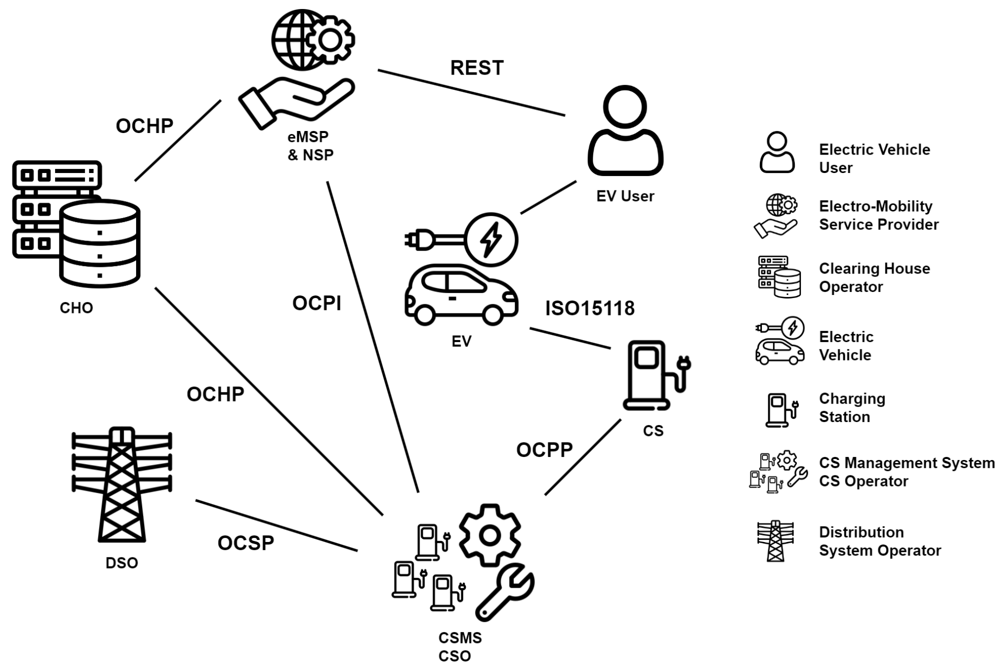


Figure 2.5: EV Charging Scenario

The charging stations are managed by the Charging Station Operator (CSO), who runs them commercially and ensures that they remain operational. Ensuring that electricity is delivered efficiently to the charging stations is a key part of managing the energy grid. This includes also integrating renewable energy sources when available and balancing the load on the grid during peak charging times: both of these are under the responsibility of the Distribution System Operator (DSO). The financial transactions and data exchange between the different service providers are managed by the Clearing House Operator (CHO), whose role includes enabling owners to charge their vehicles across different networks, facilitating the billing process.

Potential targets and desired properties Just as any other connected device is vulnerable to cyber-attack, there is an array of vulnerabilities in the communicating components of the charging station infrastructure. Adversary control can compromise the safety and security of the basic functionality of the devices, therefore, it is critical to protect vital information against attacks.

All operations between the different actors of the charging station are performed via a digital connection, making them potential attack vectors for the entire energy network. In this ecosystem, there are at least three attack surfaces related to charging. The first is the EV charging infrastructure, where charging fraud could occur through vehicle impersonation. The second is the electricity grid, which could be the target of DDoS attacks against charging networks, aimed at disrupting the ability to charge EVs at scale. Finally, there is the charging infrastructure to the fleet: charging stations could potentially attack multiple vehicles at the same time. However, all the infrastructure components and communication channels of EV charging could be potential targets of the cyberattacks.

In order to verify that each device and protocol can be considered sufficiently secure, the concept of security must be broken down into several categories of desired security properties to be verified, such as the well-known CIA triad: confidentiality, integrity and availability. In essence, confidentiality ensures that information is only readable by the intended recipients and is protected from unauthorised third parties, usually through encryption. This prevents information and sensitive data from being leaked. An adversary might want to modify exchanged messages to change operations: Integrity is the property ensuring that a message has not been tampered with or manipulated, otherwise it would be detected. Availability ensures that the services data or information provided by a server is accessible and usable upon demand by an authorized client within an expected time. Availability i.e. it measures the ability of an attacker to disrupt or prevent access to services or data.

furthermore, authentication and authorisation are the properties associated with controlling access to resources and enforcing policies. Authentication is about ensuring that a particular identified party is not being impersonated by someone else, thereby providing access control, while authorisation is the determination of a user's level of access and then granting access based on that level, following the principle of least privilege. Finally, the property of non-repudiation is about being able to prove that certain actions were taken by a user or entity at a certain point in time, so that they cannot be denied.

2.2.2 EV Charging Protocols

Due to the fact that smart EV charging is relatively new, different protocols exist for each connection between participants, and new protocols and extensions are constantly being developed: the protocol landscape for this infrastructure is therefore in continuous evolution.

Front End and Back End Protocols The main division of protocols in direct communication with electric vehicles is between front-end and back-end protocols [Table 2.1]. Front-end protocols such as REST and ISO 15118 interact directly with the owner to enable communication with the charging infrastructure. The

Representational State Transfer (REST) protocol is used for communication, data exchange and user interaction between the EV User and the service provider's platform (eMSP, NSP), typically through a web or mobile application. ISO 15118 is an international standard proposed by ISO/IEC 15118 Joint Working Group, defines a communication interface and plug-and-charge functionality, including to dynamically exchange information based on which a proper charging schedule can be negotiated, preventing the grid from overload.

Focusing on back-end protocols Open Charge Point Protocol [15], Open Charge Point Interface [11], Open Clearing House Protocol [13] and Open Smart Charging Protocol [16], which manage the operations and the interaction with the CSMS and the network management, making the whole system work behind the scenes. The back end protocols selected for this thesis are open. The OCPP is used for communication between the smart CS and the back-end systems of the CSMS. When the charging station is turned on, the software confirms its identity and OCPP manages the transaction messages from the start of charging to the end. The OCPI manages communication between charge station operators and the eMobility service providers facilitating roaming for EV drivers across different charge points. The OCHP is used to connect the service provider to the back-end networks of another service provider to verify charging transactions at third party chargers. Finally, key role in interoperability is played by the OSCP, as it allows EV users to charge their vehicles at different charging stations operated by various CSOs.

2.3 Why Open Protocols?

2.3.1 Openness and Interoperability

Among the various existing protocols, the backend protocols presented in section 2.2.2 (OCPP, OCPI, OCHP, OCSP) have been chosen to be quoted in this thesis because, as their acronym denotes, they are open. Main traits of open protocols are openess and interoperability: they allow an integrated infrastructure of compatible entities, allowing seamless data exchange. They also offer freedom of choice and competition, which imply cost savings and innovation.
Openess is an overarching concept: in the context of open protocols, it refers to the accessibility of knowledge and its collaborative nature of the development and implementation. By being freely available to use, modify and redistribute, an open standard implies a strong emphasis on collaboration in reviewing the code and contributing to its evolution. Openness promotes a cooperative environment where anyone with the appropriate expertise can participate in developing and improving the protocol to adapt it to the changing needs of the industry. Making a protocol accessible and transparent not only encourages innovation, but also builds trust among users and stakeholders in the electric vehicle charging infrastructure, which is

ISO 15118	International Organization for Standardization	Proposed international standard for the definition of a communication interface to manage communication between the electric vehicle and the charging station, including plug-and-charge functionality.
REST	Rapresentational State Transfer	Facilitates direct communication between the EV user and the charging infrastructure.
OCPP	Open Charge Point Protocol	Application protocol for communication between EV charging stations and the backend management systems. It handles charging operations and transactions.
OCPI	Open Charge Pont Interface	Manages communication between charging station operators and eMobility service providers, providing real-time station information and enabling roaming.
OCHP	Open Clearing House Protocol	Connects service providers to other networks to verify transactions on third-party chargers, enabling boundless charging across charging station networks.
OSCP	Open Charge Smart Protocol	Connects to grid operators to manage electricity grid balance: communicates physical net capacity from the DSO (or site owner) to the back-office of the charge spot operator.

Table 2.1: Protocols in EV Charging Scenario

a key factor in the success and widespread adoption of the protocol. Open technical standards are developed and maintained by a standards organisation and made available to the public on a royalty-free basis. In the context of EV charging, the OCPP, OCPI, OCHP and OCSP protocols are managed by different organisations: the Open Charge Alliance (OCA), an established global industry alliance of EV charging hardware and software vendors responsible for the maintenance of OCPP and OSCP, the EVRoaming Foundation, a non-profit community of contributors to OCPI, the innovation centres ElaadN and SmartLab GmbH, which are responsible for the implementation of OCHP.

Interoperability warrants that the interface between the interacting entities is compatible, and ensures seamless flow of information. It therefore guarantees that the charging system would function as intended without necessarily having to replace equipment or undertake significant programming to re-establish the compatibility of the interface between the interacting entities. An interoperability relevant aspect is enabling freedom of choice: it would avoid situations where controllability of charge points could be disabled if a third party operator implementing a proprietary protocol is changed. OCPP, OCPI, OCHP, OCSP: these protocols collectively enhance the interoperability of EV charging infrastructure, making it easier for users to access and use charging stations regardless of the provider or network. Some of these protocols play a more direct role: OCPP promotes interoperability across different manufacturers and operators, OCPI ensures seamless user experience across different networks, OCHP enhances network interoperability. This is crucial for the widespread adoption of electric vehicles and the development of a robust, user-friendly charging ecosystem.

2.3.2 Adopting Open over Proprietary communication protocols

As highlighted in Section 2.3.1, the main properties of Open protocols are Openess and Interoperability: analysing their implications, a chain of improvements and benefits emerges for companies that adopt open protocols [Figure 2.6].

Starting with the basic need of a company, the property of openness saves costs by allowing collaborative development and innovation of the bulk of grid integration protocols, allowing resources to be focused on services offered to customers. Interoperability saves costs by enabling integration with existing systems, reducing the need to develop or purchase expensive proprietary solutions.

Key to this is freedom of choice, which helps prevent vendor lock-in, avoids fragmented billing infrastructure and minimises wasted investment in stranded assets. The compatibility of entities and data exchange enables more efficient operations through standardised protocols.

By removing competition from the shared groundwork by collaborating on the development of open protocols, companies can instead compete to provide better services to customers, leading to a general acceleration in innovation and the development of optimised and customised products. In addition to enabling the seamless integration of new technologies and systems, scalability is also achieved through the concept of standardisation. Openness encourages the adoption of standardised protocols, simplifying the addition of new components or services without requiring extensive rework, while Interoperability ensures that these components can communicate and cooperate effectively, regardless of the vendor or technology. Overall, the benefits of standardisation are to reduce the complexity and cost of scaling operations, from which it depends the businesses growt and a

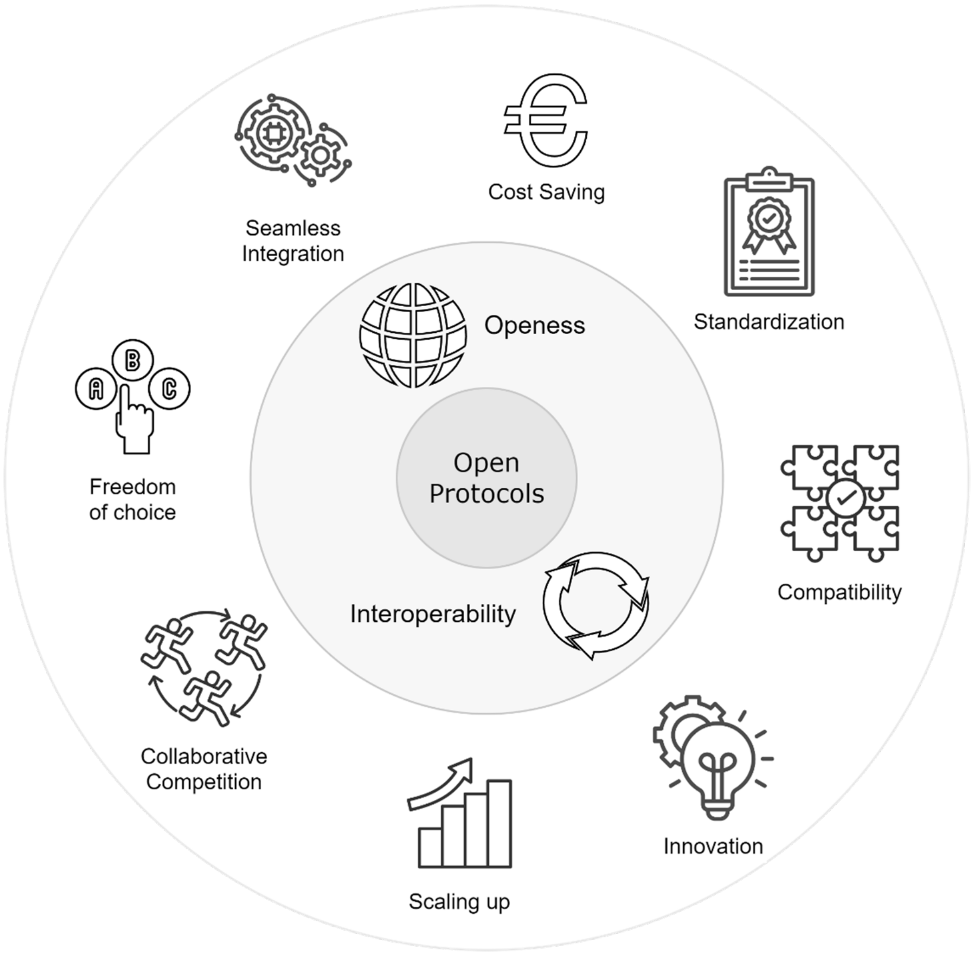


Figure 2.6: Benefits of Open Protocols

more efficiently adaptation to the market demands evolutions.

Chapter 3

The OCPP Protocol

3.1 The Open Charge Point Protocol

Focus of this thesis is the Open Charge Point Protocol, developed by Open Charge Alliance, a non-profit industry alliance focused on open standards and improving the development of sustainable charging. OCPP is an open standard communication protocol that enables the exchange of messages between Charging Stations (CS) and central Charging Station Management Systems (CSMS). Thanks to its open source nature and robust feature set, OCPP has been widely adopted worldwide and is considered the de facto standard for EV charging communication.

3.1.1 Evolution and Diffusion

The necessity to procure charging stations from different vendors led this emerging market to call for the development of a common standard for information exchange. In the absence of a standard at this early stage of industry development, the Dutch network operators took the initiative to found the non-profit ElaadNL organisation. ElaadNL then started to write the OCPP.

In 2009, ElaadNL, an innovation centre specialising in smart and sustainable charging for electric vehicles, proudly announced the launch of the OCPP. The protocol has been made freely available from the start, in order to provide support to the growing industry and to facilitate the process of adoption: in 2010 OCPP version 1.2 was released to the public. In 2015 the successor OCPP 1.6 versions really made a difference in the e-mobility world as in 2023 is still the most adopted OCPP version worldwide. The Open Charge Alliance has been established in 2014. ElaadNL transferred the governance of OCPP to the OCA, which is currently responsible for the development, compliance testing, certification, and promotion of the protocol. OCPP 2.0 has been the first version published by the OCA in

2018. Several improvements has been made when the latest OCPP version 2.0.1 launched in 2020.

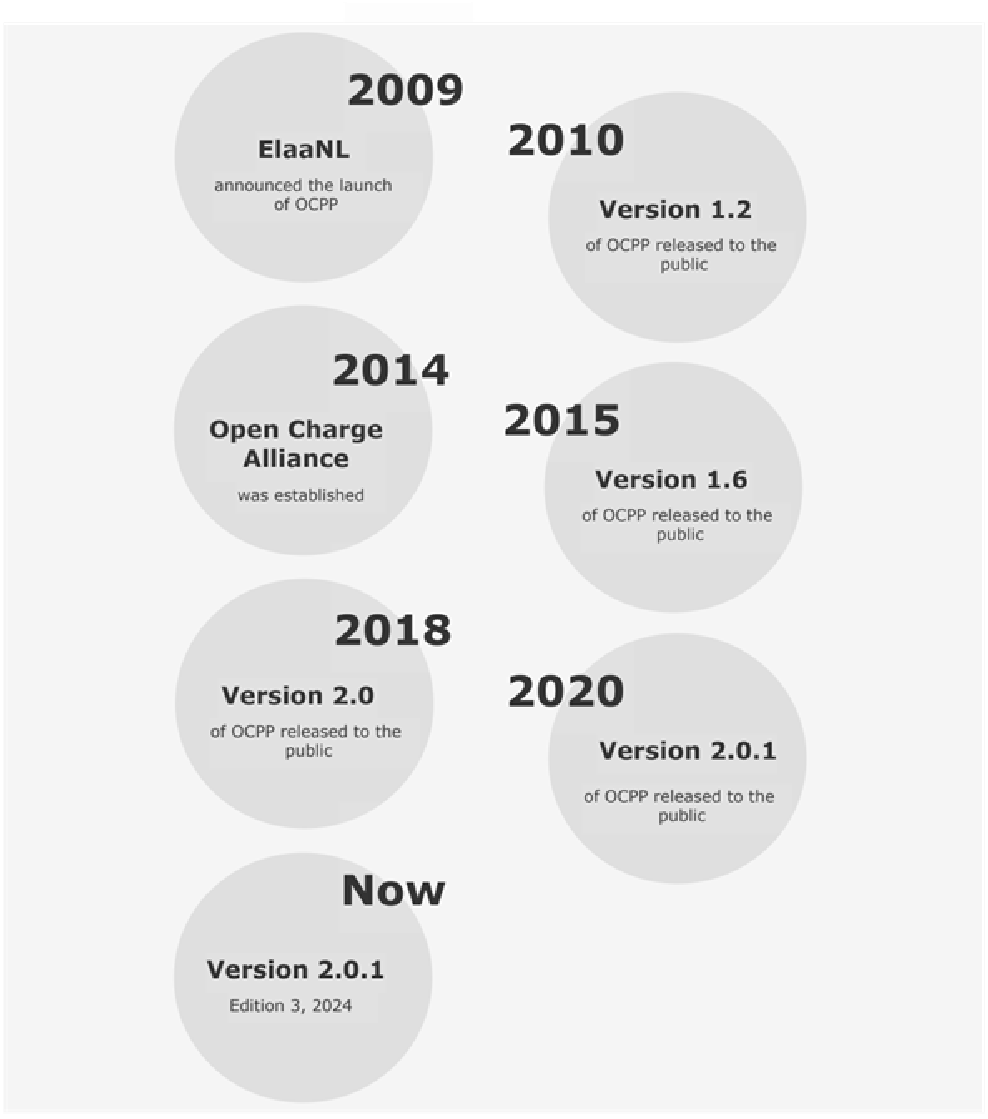

2009
ElaaNL
announced the launch of OCPP
2010
Version 1.2
of OCPP released to the public
2014
Open Charge Alliance
was established
2015
Version 1.6
of OCPP released to the public
2018
Version 2.0
of OCPP released to the public
2020
Version 2.0.1
of OCPP released to the public
Now
Version 2.0.1
Edition 3, 2024

Figure 3.1: OCPP Evolution Milestones

3.1.2 Version 2.0.1

After the release of OCPP 2.0, a number of issues were identified that could not be resolved through the issuance of errata to the specification text alone, as was the case with OCPP 1.6: these issues required changes to the machine-readable schema definition files, which were not backwards compatible. To avoid confusion in the market and potential interoperability issues, OCA opted to call this new version OCPP 2.0.1. Introduced in 2020, OCPP 2.0.1 is gradually replacing OCPP 1.6 as the new industry standard for communication between charging stations (CS) and management systems (CSMS).

The OCPP 2.0.1 version offers a number of additional features compared to the previous version, OCPP 1.6. The new advanced features include the Device Management, a long awaited feature especially welcomed by CSOs who manage a network of charging stations from different vendors. The new version also offers an improved configuration, inventory, error and state reporting, and a customizable monitoring system. These features should assist CSOs in reducing the costs associated with operating a charging station network. Traffic Management in OCPP 2.0.1 offers improved and integrated smart charging, visualization and messaging support. The number of charging stations and transactions that the CSMS is required to manage is increasing. The enhanced transaction management, which permits more effective tariff administration and cost reporting, has improvements for better handling of large amounts of transactions like the concept of one message for all transaction related functionalities. About data reduction, the introduction of JSON over Websockets in OCPP 1.6 resulted in a significant reduction of mobile data; with OCPP 2.01 also the support for WebSocket Compression is introduced, which reduces the amount of data even more. Load balancing facilitates the protocol's support for Energy Management Systems (EMS), it renders the protocol more suitable for large-scale charging networks. The introduction of security improvements hardens the robustness of the protocol against cyber attacks: the protocol incorporates secure firmware updates and more sophisticated error handling, facilitating the diagnosis and resolution of issues. It defines three levels of Security profiles [section 5.4] for authentication and communication between the charging station and CSMS, utilising TLS [7] encryption, certificate-based authentication and HTTP Basic Authentication[17], with implementation of Key management for Client-Side certificates.

3.2 Spreading adoption of the OCPP Protocol

OCPP is used across the world: the protocol has been downloaded since 2015 to more than 70.000 individual IP addresses (from OCA website, Q3 2024), inspiring and helping developers in all regions to develop EV charging infrastructure. As OCPP is an open-source standard, developers are not required to join the Open Charge Alliance in order to implement it. Those wishing to become more involved in the development of OCPP have the option of becoming members of the OCA. There are over 350 companies in 42 countries participating in the OCA [Figure 3.2], with more than a quarter of them having obtained OCPP certification [subsection 3.2.2] demonstrating their compliance with the standards for secure and interoperable electric vehicle charging systems.

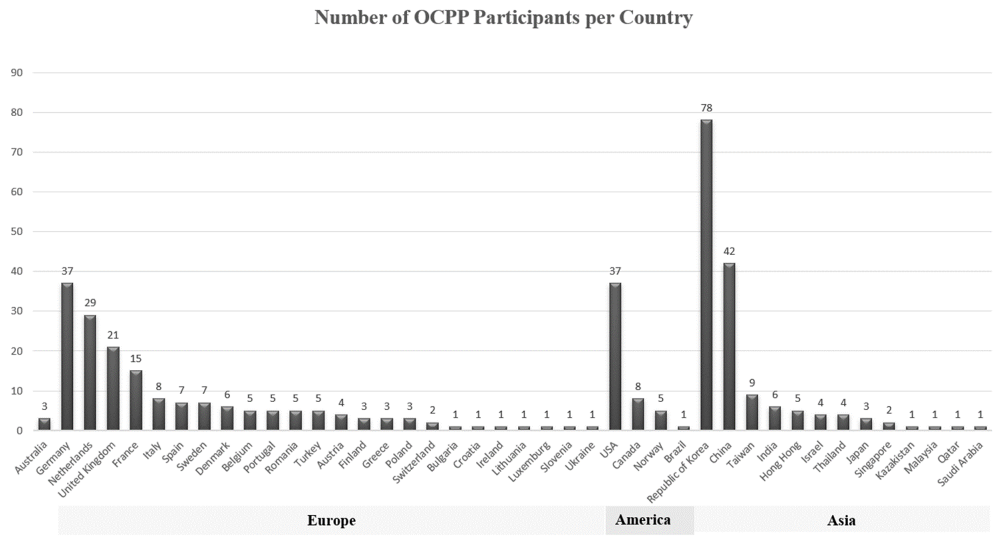


Figure 3.2: OCA Participants around the Globe, from OCA website, august 2024

3.2.1 Why OCPP?

Many key charasteristics of OCPP facilitate innovation and cost-effectiveness, thereby encouraging the adoption of the Protocol in the long term.

- **Openess**. OCPP is an open standard with no cost or licensing barriers, designed to enable different brands and models of EV charging stations to communicate with a central system, regardless of the manufacturer This enables a wide range of participants to engage in the growing industry and to benefit from the openness and interoperability features [Section 2.3].
- **Seamless Charging Experience** One of the main advantages of OCPP is Interoperability, it provides a consistent charging experience for EV owners, thereby eliminating the technical complexities and confusion that may arise from the utilisation of multiple standards and chargers.
- **Reliability for the User** EV users can expect consistency in timing and performance, permitting them to plan their charging schedule better. Ultimately, such reliability and consistency may facilitate greater adoption of electric vehicles.

- **Regulatory Compliance** OCPP offers built-in compliance with the General Data Protection Regulation (GDPR) and other emerging data security laws. This eases the burden of demonstrating compliance for those involved in the production and distribution of EVs.
- **Modular Scalability** Modular scalability represents a significant advantage of the OCPP protocol, as it enables the charging infrastructure to adapt and expand to new technologies as they emerge. This mitigates the concern among owners and operators of EV charging points that their equipment may become obsolete, facilitating the adoption of emerging technologies by the EV industry.Moreover, OCPP facilitates the integration of new charging stations or networks to existing infrastructure, reducing associated costs and encouraging the expansion of stations.
- **Management Efficacy and Monitoring** Charge point operators can leverage OCPP to remotely monitor and manage charging stations this simplifies maintenance and accelerate the resolution of issues, leading to better operational efficiency. Charge point owners and operators can also maximize their revenue by using OCPP to manage the usage of chargers efficiently.

3.2.2 Certification Program

The Open Charge Alliance offers a certification programme, which is designed to assess and verify the competence of individuals and organisations in the field of open charging infrastructure. The principal objective of the OCPP certification process is to verify compliance with the OCPP standard on the part of both vendors and purchasers of OCPP implementations. An OCPP certificate provides assurance of interoperability and flexibility.

Certification Profiles and Procedure The certification procedure is applicable to the certification profiles published by the OCA. A charging station may apply for certification for a variety of profiles, reflecting the diversity of charging stations (home chargers, fast chargers). However, the number of profiles is limited to ensure clarity within the industry. Currently, the available profiles for certification are the Core profile, which is mandatory for all implementations, and the Advanced Security Profile. In order to obtain OCPP certification, a number of tests must be successfully completed. Test of Conformity: the device under test (DUT) is evaluated against the OCPP Compliance Testing Tool. The tool has built-in validations that should not fail during certification tests: these are performed to verify whether the DUT has implemented the OCPP specification correctly. Performance Measurement: the performance parameters are stated by the vendor in the Protocol Implementation Conformance Statement (PICS) and are verified by the test lab.

3.3 OCPP Architecture

This section defines the function of the OCPP protocol, operative between a charging station (CS) and the charging station management system (CSMS) in an electric vehicle charging infrastructure, in the form of functional blocks and use cases, as well as the related test cases and certification profiles.

3.3.1 Key concepts

The charging process [Figure 3.3] for electric vehicles is greatly enhanced by the Open Charge Point Protocol (OCPP), which enables bidirectional communication between EV Charging Stations and the charging Charging Station Management System (CSMS). This communication allows the CSMS to monitor and control the charging station while also receiving valuable data such as charging status, session details, and energy consumption. The CSMS, in turn, may relay relevant information to the vehicle owner's app, ensuring they stay informed throughout the charging process. The process of charging begins with the user reserving a charging

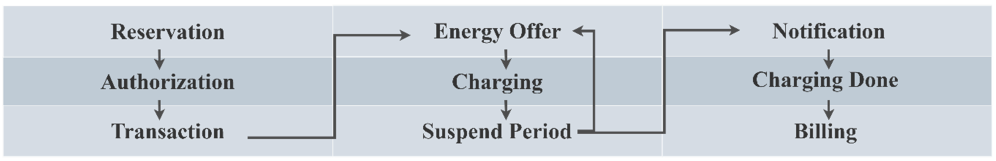


Figure 3.3: Charging Process

slot, ensuring that the availability of a station at the designated time. Before the charging session can start, the user's identity and payment method are verified through the authorization process. Once authorized, a transaction is initiated, allowing the charging to proceed. Upon the commencement of the charging process, the charging station begins to supply energy to the vehicle, thereby initiating the transfer of energy to the electric vehicle's battery. During this phase, data belonging to the charging session is exchanged continuously between the CS and the CSMS, facilitated by OCPP. If necessary, the charging session can be paused or suspended, providing flexible control over the process. Once the session concludes, the user is notified that the charging is complete. This marks the end of the energy transfer to the vehicle, and the billing process is then initiated, charging the user for the energy consumed. By enabling the seamless exchange of critical data between the charging station and the central management system, OCPP allows product managers to create optimised EV charging experiences, ensuring that every aspect of the process is smooth and efficient.

Structure and Components OCPP 2.0.1 [15] specification [Figure 3.4] is divided into functional blocks, each of which defines a role that OCPP plays in the communication between the CSMS and the CS. Each functional block is composed of a list of use cases that represent a specific scenario. In order to obtain certification, a set of test cases must be validated, representing a specific feature required by the profile. Each test case can be related to one or more use cases.

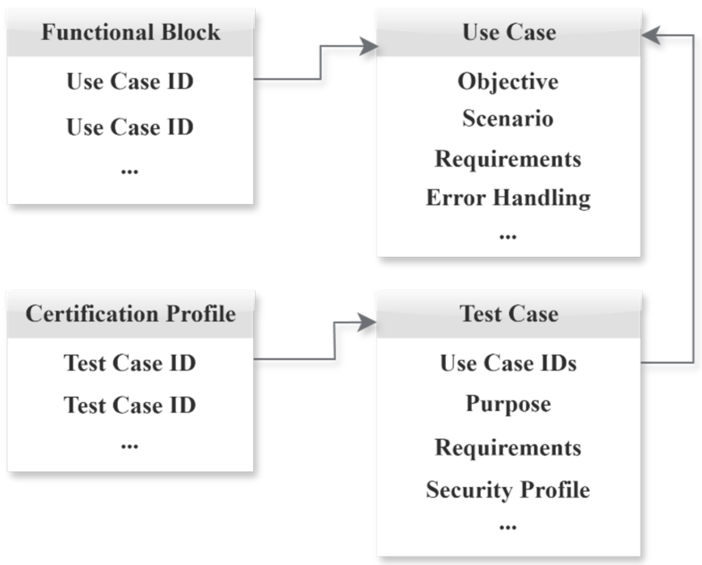


Figure 3.4: Relations in OCPP Specification

3.3.2 Functional Blocks

Each OCPP'S functional blocks [Figure 3.5] defines a role in the communication between the CSMS and the CS:

Service Management Tracking and recording the transactions of charging sessions functionality enables the generation of billing records, which are created at the charging station and transmitted to the CSMS. The system is designed to communicate the status of the charging station to the CSMS, including whether it is available or out of order. This enables operators to inform users and manage operations accordingly. The system also monitors and reports detailed metering data to the CSMS during a charging session. This data is essential for billing and energy management purposes.

Advanced control Provisioning manages the initial setup and configuration of Charging Stations, including their connection to a CSMS and retrieval of configuration data, ensuring effective communication. The optimisation of charging is enabled by Smart Charging, which serves to balance grid load, integrate renewable energy sources, and enable the CSMS to control energy consumption at the station or EV level through the utilisation of various smart charging configurations. The Remote Control functionality allows the CSMS to manage Charging Stations remotely, including starting/stopping sessions, unlocking connectors, and triggering messages for status updates or troubleshooting, while Data Transfer functionality facilitates the exchange of additional data between the CS and CSMS, enabling flexibility and custom functionalities in communication.

Access Control The authorization process determines whether a user is permitted to commence or terminate a charging session. This is typically achieved through the use of RFID cards or mobile applications, which then communicate the authorization status to the CSMS. By maintaining a locally stored list on the CS, users can continue charging even when they are disconnected from the CSMS.

Smart Features The tariff and cost functionality provides the EV driver with cost-related information at various stages of the charging process, including before, during and after the charging session. This information is displayed on the charging station screen and includes detailed tariff plans and the total costs incurred. Display Message enables the CSMS to send text messages or notifications to the Charging Station's screen, such as user instructions, alerts, or personalized messages. Finally, the Reservation block allows users to reserve a Charging Station or connector in advance, ensuring its availability when needed, which is crucial for trip planning with limited charging stations.

Security Management Diagnostic block provides a suite of tools for the diagnosis and troubleshooting of issues with the charging station, facilitating smooth operation through remote diagnostics and information retrieval. In contrast, the Security block is responsible for ensuring secure communication between the CSMS and the Charging Station. This is achieved through the implementation of encryption, authentication, and data integrity measures, which serve to reinforce the overall security framework as defined by OCPP. The Firmware Management feature enables the CSMS to remotely update the Charging Station's firmware, ensuring its operation with the latest software version and security patches, with the CSMS

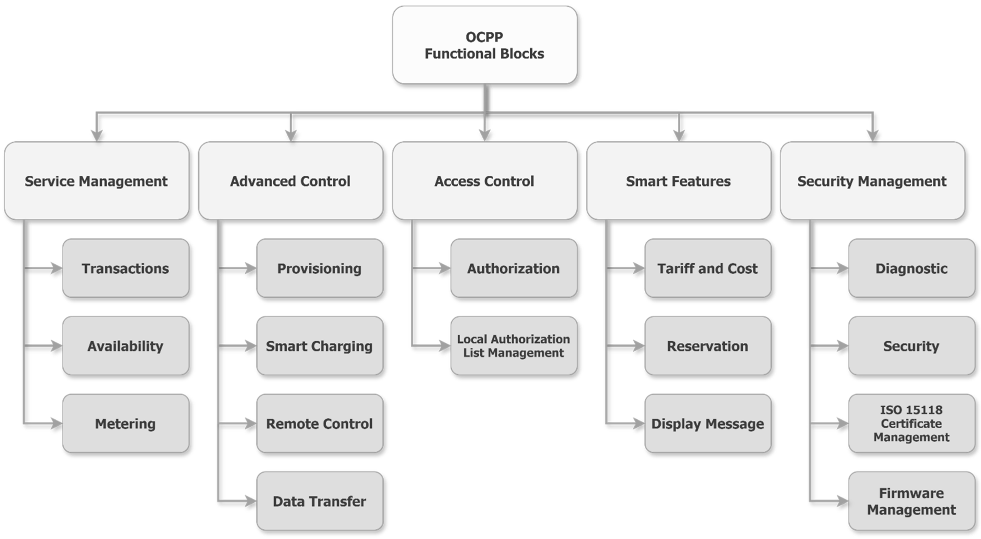


Figure 3.5: OCPP Functional Blocks Categories

informed of each update step. Finally, the ISO 15118 Certificate Management has the role of managing digital certificates for secure communication, including Plug & Charge, enabling certificate-based authentication and authorisation at the Charging Station. Display Message enables the CSMS to send text messages or notifications to the Charging Station's screen, such as user instructions, alerts, or personalized messages. Finally, the Reservation block allows users to reserve a Charging Station or connector in advance, ensuring its availability when needed, which is crucial for trip planning with limited charging stations.

3.3.3 Use Cases, Test Cases, Certification Profiles

Use Cases A use case delineates a section, a specific action or response from one party or the other, associated with the functional block in question. Each use case is identified in the specification with a descriptive name, an ID and the functional block to which it belongs. It defines the objective and description of the messages and data types required for the use case, as well as the prerequisites for its applicability and the postcondition desired. Each use case is subject to a list of specific requirements, including error handling and other remarks.

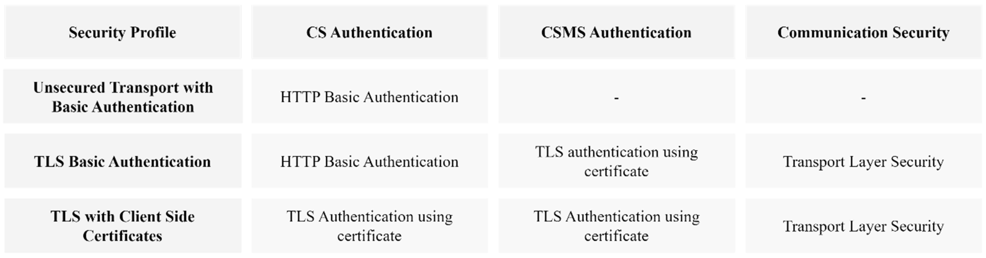
Test cases In the context of OCPP, a test case is defined as a sequence of messages that are used for the purpose of testing a particular use case. The objective is to test a specific feature, which represents a particular functionality, using one or more test cases. Test cases are associated with a certification profile, which constitutes a component of the OCPP certification programme. A test case may be either mandatory to implement a certification profile or conditional, i.e. subject to a certain condition, and mandatory in the case the condition is verified. Test cases are identified by a descriptive name, test case ID, and use case ID(s). They are related to a specific test scenario for both CS and CSMS, which specifies the requisite security profile (subsection 3.3.4). They are followed by a list of requirements, prerequisites, and post-scenario validations.

Certification profiles The OCPP certification is structured around certification profiles, which define a set of use cases that can be certified through the Open Charge Alliance. Each certification profile includes a list of test cases and an overview of the required controller components for certification testing. The OCTT test tool determines which test cases need to be executed for a charging station or CSMS based on the features within each certification profile. The OCPP protocol has been designed to accommodate a wide range of charging stations, and given the varying capabilities of these stations, it is not practical to require every vendor to certify the entire OCPP functionality, as only a subset may be necessary for specific applications. Full OCPP certification covers all profiles, but vendors can choose to certify only the necessary profiles for their needs. The OCPP Core profile must always be present. It contains the basic OCPP functionality. In addition to this, optional profiles such as Advanced Security can be included in the certification.

3.3.4 Security Objectives and Security Profiles

OCPP security has been designed to achieve several key objectives. Most important there is the imposition of a secure communication channel between the CSMS and the and the charging station, ensuring the integrity and confidentiality of transmitted messages through the implementation of robust cryptographic techniques. A fundamental aspect is the support of a secure firmware update process, which allows the Charging Station to verify the source and ensures non-repudiation and integrity. Great importance is given to the logging of security events, in order to monitor the security of the smart charging system.

Security Profiles As outlined in the security functional block [section 3.3.2] in OCPP the security approach relies on standard web technologies, leveraging TLS and public key cryptography with X.509 certificates: application layer security measures are not included. Given that the CSMS acts as the server, OCPP reccomends that different users or role-based access control to be implemented to ensure the appropriate level of access control on the CSMS. OCPP 2.0.1 is capable of supporting three distinct security profiles, as illustrated in [Figure 3.6]. These profiles are distinguished by the specific authentication measures that are applied to each component and the communication channel security level; only one security profile can be set at a time. Each test case, as outlined in [section 3.3.1], specifies the requisite security profile for the parties involved. For the scope



Security Profile	CS Authentication	CSMS Authentication	Communication Security
Unsecured Transport with Basic Authentication	HTTP Basic Authentication	-	-
TLS Basic Authentication	HTTP Basic Authentication	TLS authentication using certificate	Transport Layer Security
TLS with Client Side Certificates	TLS Authentication using certificate	TLS Authentication using certificate	Transport Layer Security

Figure 3.6: OCPP Security Profiles Authentication Requirements

of this thesis, the Security Profile 1, Unsecured Trasport with Basic Authentication, is not taken in consideration because it should only be used in trusted networks, as the Unsecured Transport with Basic Authentication Profile does not include authentication for the CSMS, or measures to set up a secure communication channel. This means that a security analysis of the protocol when this profile is set would be irrelevant as the protocol relies on a trusted network out of the scope of the protocol itself. In the TLS with Basic Authentication, Security Profile 2, the communication channel is secured using Transport Layer Security. The CSMS, which acts as the server, authenticates itself using a TLS[7] server-side certificate, while the charging stations authenticate themselves using HTTP Basic Authentication[17], i.e. by means of username and password credentials. In the TLS with Client Side Certificates profile, the communication channel is secured using Transport Layer Security. Both the Charging Station and CSMS authenticate themselves using certificates. In the TLS with Client-Side Certificates profile, the Security Profile 3, the communication channel is secured using TLS, and both the Charging Station and CSMS authenticate themselves using certificates. It is requisite that both Security Profile 2 and Security Profile 3 are supported by TLS and CSMS, and that they support a minimum set of ciphersuites:

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384

It is recommended that the TLS_ECDHE option be selected; therefore, for the scope of this thesis TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 ciphersuite will be the chosen option[7][4].

Chapter 4

The ProVerif Tool

4.1 Formal Verification

Formal verification is the process of ensuring that a formal model of a system satisfies some specified formal properties and check wheter the design of the protocol may make the system exposed to malicious interferences: analysing a protocol by manual review can often not be enough to identify potential flaws in the design. The goal is to verify that the system under test behaves according to the specification and satisfies the desired properties as intended. The formal verification process generally consists of a deep understanding of the protocol's under test specification and requirements, followed by the creation of a model based on it. This defined model must then be translated into the input language of the model checker, which will perform the verification through the automated process. The final step is a detailed review of the outcomes and an evaluation of whether it is necessary to recommend protocol changes based on the results of the verification.

4.1.1 Theorem Proving and Model Checking

Formal verification can be divided into two categories of approaches: Theorem Proving and Model Checking. While both approaches can be used to verify that a formal model satisfies some formal properties, the strategy for performing such verification is slightly different. Theorem provers base the analysis on the mathematical correctness of statements, the process involves formalizing assertions using logic and deduction. This process involves the breaking down the theorems into sub-objectives and applying the logical rules to produce a step-by-step demonstration. This method provides high assurance of the correctness of a system but is typically reliable on human intervention for verify the correctness of complex algorithms. The second approach, model checking, consists of automating the exploration of all possible executions of the system to check whether it is possible to reach a state where the formal requirements are not met. Model checkers tend to be more immediate to use, they are designed to verify properties focused on specific protoocl sections, making them suitable for identifying design errors and security violations. An inevitable consequence of the state exploration approach is that the range of system states it can handle is necessarily limited, since the number of possible states grows exponentially with the size of the system. This means that this technique is particularly effective for automated verification, but is limited by the complexity and size of the model.

4.2 The ProVerif Tool

Developed by Bruno Blanchet, and reported to have shown the fastest performance when compared with other tools [6], ProVerif [1] [2] [3] is a state-of-the-art automatic cryptographic protocol verifier.

4.2.1 Proverif abstractions

Based on Dolev-Yao formal modelling [8], it assumes that attackers have full access to the communication channel: they can intercept, modify, send and replay the data exchanged by the entities. However, they cannot break the cryptographic functions or have direct access to the private data without performing an attack. As a theorem prover, ProVerif analyses the symbolic behaviour of protocols rather than actually executing them. The internal representation of the protocol is based on Horn clauses[18], which are used to express logical rules that define how a protocol should behave. The security properties to be proved are translated into derivability queries, which determine whether the desired security properties hold by resolving these clauses. The horn clause representation introduces abstractions which are key to verifying an unbounded number of sessions of protocols. The abstraction is still precise because it preserves relationships between the messages and data exchanged but the termination outcome is not generally guaranteed. The reason of this is that if the outcome reports that the protocol satisfies some property, then the property is actually satisfied. Viceversa, when ProVerif cannot prove a property, it tries to reconstruct an attack, i.e. an execution trace of the protocol that falsifies the desired property: the derivation may correspond to an attack, but it may also correspond to false attacks, due to the horn clause abstractions.

4.2.2 Verifying Protocols Properties with ProVerif

The ProVerif tool takes as input a description of the protocol based on the pi calculus, which is a mathematical formalism used to describe and analyse a system's communication and interaction between concurrent processes through message exchange. To model and verify security protocols, ProVerif uses an extension of the pi calculus with cryptography (such as encryption or decryption of messages). A ProVerif model of a protocol written in the tool's input language can be divided into three parts. The first part is the declaration of types, free names, constants and the set of constructors and destructors representing the cryptographic operations and other functions required for the execution of the processes. This is followed by the definition of main process and sub-processes macros, which are reusable definitions of processes particularly useful for easing development. The final step is the definition of the queries, which represent the properties to be verified in the protocol. ProVerif is capable of proving several important security properties for cryptographic protocols by means of queries definition: these include secrecy, authentication and observational equivalence. Secrecy is about ensuring that an adversary cannot access confidential information, while authentication guarantees that communication is between the intended parties. Observational equivalence means proving the equivalence between processes that differ only in certain values. ProVerif can also prove strong secrecy, meaning that an adversary cannot detect changes in the value of a secret. In general, the properties used to prove all of the above are correspondence and injective correspondence. In addition, these properties can ensure that the actions performed by the protocol happen in a particular order.

For each of the queries provided into the input file to prove a security property, Proverif can display three possible reports. When the property is proved and there are no attacks, the result of the query is true; viceversa if the result is false, it means that the prover has discovered an attack against the protocol that falsifies the desired security property. Along with the queries resulting in a false output, ProVerif will provide attack derivations and attack traces, which report te property breach that the tool has identified in the protocol's security. The attack derivation explains the actions the attacker must take in terms of its internal logic to break the security property, while the attack trace corresponds to an executable trace of the process under consideration. Given that ProVerif attempts to prove that a state in which a property is violated is unreachable, verifying protocols for an unbounded number of sessions is an undecidable problem as is does not scale, so another outcome is possible: "cannot be proved". This result means that ProVerif could not prove that the query is true, but also could not find an attack that proves that the query is false.

Chapter 5

The ProVerif model of OCPP's Use Cases

5.1 A01 - Update Charging Station Password for HTTP Basic Authentication

This section presents the ProVerif model implementation of the OCPP protocol section responsible for updating the password of the credentials used by the Charging Station to perform authentication. The presentation is divided into a summary of the role of the use case that implements the mentioned function [subsection 5.1.1], followed by the description of the environment setup [subsection 5.1.2], the schema of the messages exchanged [subsection 5.1.3] and a detailed report of the functions and messages implemented [subsection 5.1.4].

5.1.1 Description Use Case A01

This Use Case A01- is part of the Security Functional Block [subsection 3.3.2]. The Objective of the message exchange between the CSMS and the CS is to define how to use the Basic Authentication security profile and update the credentials at the request of the CSMS, which transmits a new value for the CS Password. The implementation of Security Profile 2 [subsection 5.4.1] is necessary for this use case to establish an authenticated communication channel and a shared session symmetric key via the key exchange algorithm ECDHE [4][subsection 5.3.4]. The Charging Station authenticate itself using HTTP Basic Authentication [17] by means of the installed credentials, and the CSMS authenticates to the CS using a TLS server certificate [5]. The communication between Charging Station and CSMS is secured using TLS [3]. For the full code, please refer to Appendix A.

5.1.2 Setup of the environment

The cryptographic functions setup is defined by the Security Profile 2 and the chosen ciphersuite[subsection 5.3.1]. The model of the function for encryption, key exchange, signature and others are further explained at [section 5.3]. For this use case three parties partecipate to the communication. The main processesprocesses are the Charging Station and the Charging Station Management System, the entities which establish connection through OCPP Protocol and then implement this Use Case. There is also the Certification Authority process, which is responsible for supplying signed certificates.

CA, Certification Authority

```
let pCA(km_CA:keymat, pk_CSMS:pkey) =
out  ( c, penc(signCA( pk_CSMS, sk(km_CA)), pk_CSMS))  ;
0.
```

The Certification Authority process is responsible for signing and supplying the signed certificate to the CSMS, which is used to authenticate with a server certificate. The CA process represents a root CA that is trusted by both the CS and the CSMS: it produces the certificates applying a digital signature [subsection 5.3.3] to the Certificate, represented by the public key of the CSMS. Then sends it encrypted by means of asymmetric Encryption [subsection 5.3.2] with the certificated public key, to prove that the receiver is in possession of the correspondent secret key.

Installation

```
free UsernameCS: bitstring.
free PasswordCS: bitstring [private].
```

One aspect of this implementation in Proverif is the configuration of the protocol actors and the associated environment. The initial element is the implementation of OCPP Use Case A00-Installation: unique credentials are used to authenticate the CS to the CSMS. These credentials must be installed OOB during the manufacturing or installation process, both on the CS by the manufacturer and on the CSMS through the CSO. In the context of this thesis, these credentials are defined as private global variables, UsernameCS and PasswordCS, with the password is declared private.

Public channel

```
free c: channel.
```

Declaration of the public communication channel for traffic transport.

CS, CSMS, CA key material

```
process
new km_CSMS: keymat;
new km_CA:keymat;

let pk_CSMS = pk(km_CSMS) in out(c, pk_CSMS);
let pk_CA= pk(km_CA)in out(c, pk_CA);
((!pCSMS(km_CSMS, pk_CA))) | (!pCS( pk_CA) | (!pCA(km_CA, pk_CSMS)))
```

Key material for CSMS and CA are input for the correspondent process. Both CSMS and the CS trust the CA and its' public key pkCA. Certification Authority also knows CSMS publik key to be signed to generate the correspondent certificate.

5.1.3 A01 - Schema and Messages

Schema of A01-Update Charging Station Password for HTTTP Basic Authentication in the Proverif model of this thesis. To set a new Charging Stations basic

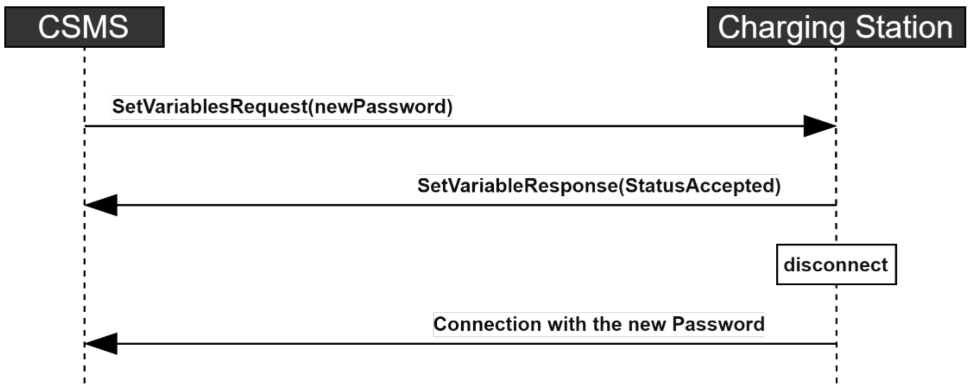

CSMS
Charging Station
SetVariablesRequest(newPassword)
SetVariableResponse(StatusAccepted)
disconnect
Connection with the new Password

Figure 5.1: Sequence Diagram: Update Charging Station Password for HTTP Basic Authentication

authorization password via OCPP, the CSMS generates a new unique password NewPassword for the CS, then sends a SetVariablesRequest message with the BasicAuthPassword Configuration Variable with ComponentName: SecurityCtrlr, VariableName: BasicAuthPassword and value: NewPassword to the Charging Station. The Charging Station responds with SetVariablesResponse and StatusAccepted, disconnects its current connection and connects to the CSMS with the new password. The CSMS on the reception of StatusAccepted assumes that the authorization key change was successful. If the Charging Station responds to the SetVariablesRequest with a SetVariablesResponse with a status other than StatusAccepted, the Charging Station will keep using the old credentials.

5.1.4 Message exchange implementation

In this section the implementation of each of the exchanged messages is reported in further details.

SetVariablesRequest

```
new NewPassword: bitstring;
let m = SetVariablesRequest( SecurityCtrl, BasicAuthPassword,
   NewPassword ) in
let i = next(i) in
let enc_m = encdh( (i, m), symk) in
out (c, enc_m);
```

The CSMS generates a new password NewPassword and encapsulates it in a message by means of the SetVariablesRequest function, along with SecurityCtrl and BasicAuthPassword, which are modeled as global costants in this implementation. The resulting m bitstring message is encrypted by encdh using the symmetric key symk to provide confidentiality, togheder with i, the message identifier, which has been incremented to the next element of the serie next(i) with next function and avoids the message to be replaied. The encrypted message is then signed with the CSMS secret key sk(kmCSMS) to provide authentication and non-repudiation. Further explanation of the functions at [section 5.3].

```
in (c, m:bitstring);
let xi = next(xi) in
let ( = xi, msg:bitstring) = decdh (m, symk) in
let (sc: bitstring, bap: bitstring, xnewPassword: bitstring) =
    SetVariablesRequestRet(msg) in
if(sc <> SecurityCtrl || bap <> BasicAuthPassword ) then 0
else
```

The CS receives the SetVariablesRequest, check the sign with the function checksign, which must return a positive result for the CSMS public key xpkCSMS to which it authenticated. The CS checks the correctness of the message identifier xi, which must correspond to the next element of the serie next(i), and gets the original message by means of getmess and decrypt function. Extraction of the values is performed by SetVariablesRequestRet function, that reverses the SetVariablesRequest's incapsulation of the values in the message. If the values coincide to the expected, the process proceeds. Further explanation of the functions at [section 5.3].

SetVariablesResponse

```
let xi = next(xi) in
let m = encdh((xi,SetVariablesResponse(StatusAccepted)), symk) in
out(c, m);

```

The CS encapsulates in a SetVariablesResponse message the StatusAccepted value, to confirm the NewPassword received in SetVariablesRequest from the CSMS was accepted. The message is paired to the next message identifier next(xi) and encrypted with function encdh and the symmetric key symk.

```
in(c, m: bitstring);
let i=next(i) in
let (=i, msg:bitstring) = (decdh( m, symk )) in
let msg= SerVariableResponseRet(msg) in
if( msg <> StatusAccepted ) then 0 else
```

The CSMS receives the message, decrypts it by means of the decdh decrypting function using the symmetric key symk, checks the message identifier to coincide to next(i) and retrives the value of the message using SetVariablesResponseRet function. If the received response is StatusAccepted, CSMS proceeds with the protocol.

Connection with the new password For both CS and CSMS the connection with the new password NewPassword has been modeled for this thesis following the schema specification of the Security Profile 2 [subsection 5.4.1]. Starting from ChangeChiperSpec, Client performs authentication at the reconnection using the updated set of CS credentials, the only accepted by the CSMS Server once the SetVariablesRequest received a StatusAccepted response from the CS. For the full code [Appendix A].

5.2 A05 – Upgrade Charging Station Security Profile

This section presents the ProVerif model implementation of the OCPP protocol section responsible for upgrading the Charging Station security profile. The presentation is divided into a summary of the role of the use case that implements the mentioned function, followed by the description of the environment setup, the schema of the messages exchanged and a detailed report of the functions and messages implemented.

5.2.1 Description of Use Case A05

The use case A05-Upgrade Charging Station Security Profile has the scope to increase the security of the OCPP connection between the CSMS and a Charging Station. In this work, the model implements the upgrade for the Charging Station from Security Profile 2 [Section 5.4.1] to Security Profile 3 [Section 5.4.2]. Communication between the Charging Station and the CSMS is secured using TLS [7]. At the request of the charging station operator, and provided that the requirements of security profile 3 are met, the charging station is required to change from authentication via HTTP Basic Authentication [17] to TLS authentication using certificates[5]. The CSMS authenticates in both security profiles with TLS authentication with certificates. For further information on security profile implementation, please refer to [section 5.4]. The complete code can be found in [Appendix B].

5.2.2 Setup of the environment

The cryptographic functions setup is defined by the security profile and the selected ciphersuite. In this scenario, two profiles are involved. For both profiles, the same ciphersuite has been selected and implemented, as outlined in subsection 5.3.1. Further details on the model of the function for encryption, key exchange, signature and other processes can be found in section 5.3. In this use case, four parties are involved in the communication process. The main processes are the Charging Station and the Charging Station Management System. These entities establish a connection through the OCPP Protocol and implement this use case. The Certification Authority (CA) process is responsible for supplying signed certificates for both the CS and the CSMS. Finally, the Charging Station Operator (CSO) process checks that the prerequisites for the upgrade are met and signals to the CSMS to perform the upgrade of the security profile.

CSO, Charging Station Operator

```
let pCSO(km_CSO:keymat, pk_CSMS:pkey) =
let sec_profile = SecProfile2 in
out ( c , sign((ChangeNetworkConfig, upgrade(sec_profile)), sk(
    km_CSO)));
0.
```

The CSO role is to oversee and enhance the security of the OCPP connection between CSMS and a charging station. In accordance with the configuration of this use case, which commences with security profile 2, secprofile, the CSO transmits a ChangeNetworkConfig message to the CSMS, indicating the necessity for an upgrade to the subsequent security profile. In this instance, the upgraded profile is security profile 3, represented by the value upgrade(secprofile).

CA, Certification Authority

```
let pCA(km_CA:keymat, pk_CSMS:pkey, pk_CS:pkey) =
out ( c, penc(signCA( pk_CSMS, sk(km_CA)), pk_CSMS) );
in (c, =ack);
out ( c, penc(signCA( pk_CS, sk(km_CA)), pk_CS) );
in (c, =ack);
0.
```

Before migrating to an upgraded security profile, OCPP 2.0.1 must fulfil the prerequisites, which include the installation of certificates or the configuration of passwords. In order to upgrade to the security profile 3, the prerequisites are the installation of authentication certificates for both the CSMS server and the CS client. The Certification Authority is modelled to provide such certificates for the public keys pkCS and pkCSMS of the CS and CSMS processes, which are signed and encrypted in accordance with the specifications outlined in [subsection 5.3.2].

Installation

```
free UsernameCS:bitstring.
free PasswordCS:bitstring[private].
```

Part of this implementation in Proverif is the setup of the Protocol actors and environment. Initial element is the implementation OCPP Use Case A00 Installation: unique credentials are used to authenticate the CS to the CSMS. These credentials must be installed out of band (OOB) during the manufacturing or installation process, both on the CS by the manufacturer and on the CSMS through the CSO. In the context of this thesis, these credentials are defined as private global variables, UsernameCS and PasswordCS, where password is declared private.

Public channel

```
free c: channel.
```

Declaration of the public communication channel for traffic transport.

CS, CSMS, CA key material

```
process
 new km_CSMS: keymat;
 new km_CA:keymat;
 new km_CSO:keymat;
 new km_CS: keymat;

let pk_CSMS = pk(km_CSMS) in out(c, pk_CSMS);
let pk_CA=  pk(km_CA)in out(c, pk_CA);
let pk_CSO= pk(km_CSO)in out(c,pk_CSO);
let pk_CS = pk(km_CS) in out(c, pk_CS);

  ((!pCSMS(km_CSMS, pk_CA, pk_CSO)))
    | (!pCS(km_CS, pk_CA))
    | (!pCA(km_CA, pk_CSMS, pk_CS))
    | (!pCSO(km_CSO, pk_CSMS))
```

The key material for the Charging Station Management System , Certification Authority and Charging Station are input for the corresponding process. The CA is aware of the CSMS and CS public keys pkCSMS and pkCS, which it uses to sign and encrypt in order to generate the corresponding certificate. Both the CSMS and the CS repose trust in the CA and its public key pkCA to perform the requisite signing and provision of certificates. They also trust the received certificates, which have been signed with the aforementioned key, for the purpose of mutual authentication. The Charging Station Operator utilises its key material for the purpose of authenticating the message that is exchanged with the CSMS. The CSMS, trusts the public key pkCSO of the Charging Station Operator, obtained from the environment setup.

5.2.3 A05 - Schema and Messages

Schema of A05 Upgrade Charging Station Security Profile messages exchange implementation in the Proverif model of this thesis. Once the security profile prerequisites have been met by both the Charging Station and the Charging

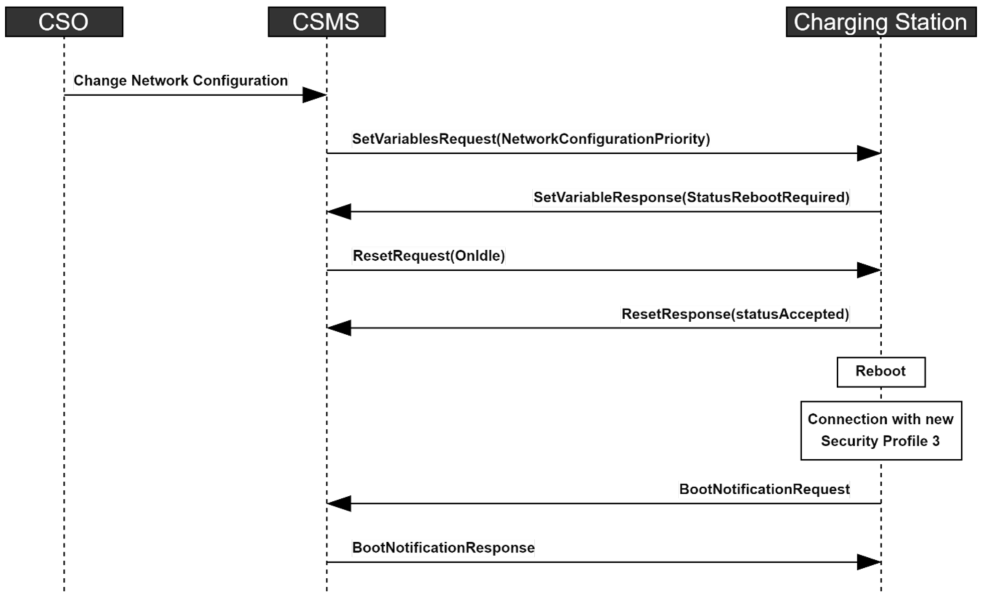

CSO
CSMS
Charging Station
Change Network Configuration
SetVariablesRequest(NetworkConfigurationPriority)
SetVariableResponse(StatusRebootRequired)
ResetRequest(OnIdle)
ResetResponse(statusAccepted)
Reboot
Connection with new Security Profile 3
BootNotificationRequest
BootNotificationResponse

Figure 5.2: Sequence Diagram: Upgrade Charging Station Security Profile

Station Management System, the Charging Station Operator signals to the CSMS Server to upgrade the Charging Station security profile to the next level. In this thesis, is implemented the upgrade from security profile 2 [subsection 5.4.1] to security profile 3 [subsection 5.4.2]. The CSMS sets a new value for the new, higher security profile 3 and requests that the charging station upgrade to the same one. The charging station responds in the affirmative and signals that a reboot is required: in this work, for completeness the messages regarding reboot require and reset request and response, which have the scope to let the transactions at CS terminate, are modeled even if transaction are omittes, as they are not part of this use case. The CSMS sends a ResetRequest(OnIdle), to which the CS responds, indicating that it accepts, and then the charging station reboots and connects via the new primary security profile 3.

5.2.4 Message exchange implementation

In this section the implementation of each of the exchanged messages is reported in further details.

Change Network Configuration

```
out  ( c  , sign((ChangeNetworkConfig, upgrade(sec_profile)), sk(
    km_CSO)));
```

The CSO sends the signal to the CSMS to request the upgrade to the next security profile 3, the value obtained by upgrading the security profile 2 value with the function upgrade as implemented in this work for details see [section 5.4].

```
in(c, msg: bitstring);
let msg=sdec(msg, sk(km_CSMS)) in
if (checksign(msg, pk_CSO)<>ok()) then 0
else
let (cnf: bitstring, xsec_profile_cso: profile)= (getmess(msg)) in
if( cnf <> ChangeNetworkConfig  || ( isupgraded(sec_profile,
    xsec_profile_cso) <> ok()) ) then 0
else
```

The CSMS receives the request, checks the CSO signature and decrypts the message, then checks that the profile to which the upgrade is required is a higher security profile, as downgrading is not part of the OCPP protocol.

SetVariablesRequest

```
let m = SetVariablesRequest( SecurityCtrl,
    NetworkConfigurationPriority, sec_profile ) in
let i=next(i) in
let enc_m = encdh((i,m), symk) in
out (c, enc_m);
```

The CSMS forwards the request to update the security profile to the CS Client, signed and protected with its own key material and the symmetric key symk.

```
in (c, m:bitstring);
let xi = next(xi) in
let (=xi, m:bitstring) = decdh (m, symk) in
let (sc: bitstring, ncp: bitstring, xsec_profile: profile)=
    SetVariablesRequestRet (m) in
if( sc <> SecurityCtrl  || ncp <> NetworkConfigurationPriority  ||  (
     isupgraded(sec_profile, xsec_profile) <> ok())) then 0
else
```

The Charging Station receives from the CSMS a SetVariablesRequest for Network-ConfigurationPriority containing a profile slot for a security profile value higher than the current value. For security reasons it is not allowed to revert to a lower

Security Profile using OCPP: for this reason the CS determines if the profile required represents an upgrade calling the is upgraded function on the received value against the currently set profile.

SetVariablesResponse

```
let xi = next(xi) in
let m=SetVariablesResponse(RebootRequired) in
let m = encdh((xi, m), symk) in
out(c, m);
```

The charging station responds to the CSMS with a SetVariablesResponse in response to a SetVariablesRequest. The attribute status RebootRequired, modelled as a global value for simplicity, indicates that a restart is required.

```
in(c, m: bitstring);
let i=next(i) in
let (=i, m: bitstring) = (decdh( m, symk )) in
let msg= SerVariableResponseRet(m) in
if( msg <> RebootRequired ) then 0 else
```

The CSMS receives the response and checks the received status.

ResetRequest

```
let i=next(i) in
let m = encdh((i,ResetRequest(OnIdle)), symk) in
out (c, m);
```

The CSMS responds to the CS with a ResetRequest in response to a SetVariablesResponse with status RebootRequired. The attribute status OnIdle, modelled as a global value for simplicity, indicates that a the reset is delayed until no more transaction are active.

```
in(c, m:bitstring);
let xi = next(xi) in
let (=xi, m:bitstring) = decdh(m, symk) in
let m= ResetRequestRet(m) in
if( m <> OnIdle) then 0
        else
```

The CS receives the ResetRequest and retreives the value by means of the ResetRequestRet function.

ResetResponse

```
let xi = next(xi) in
let m = encdh((xi,ResetResponse(StatusAccepted)), symk) in
out(c, m);
```

ResetResponse sent by the Charging Station to the CSMS in response to ResetRequest(OnIdle): the Charging Station responds with (StatusAccepted) where the status means the received command to delay reboot until transaction are terminated will be executed.

```
in(c, m: bitstring);
let i=next(i) in
let (=i, m:bitstring) = (decdh( m, symk )) in
let msg=ResetResponseRet(m) in
if( msg <> StatusAccepted ) then 0 else
```

The CSMS receives the Reset Response messages and checks if the status is declared accepted.

Connection with the new Security Profile 3 After the reboot, the Charging Station initiates the handshake to connect to the CSMS with the new upgraded security profile 3 TLS with Client side Certificates. Further details about this connection and messages exchanged are described in details at subsection 5.4.2.

BootNotificationRequest

```
let xi = next(xi) in
let m = encdh((xi, BootNotificationRequest(RemoteReset, Id_Cs)), symk
    ) in
out(c, m);
```

This message sent by the CS signals to the CSMS that the charging station has rebooted.

```
in(c, m: bitstring);
let i=next(i) in
let(=i, m:bitstring)=decdh(m,symk) in
```

```
let (pu:bitstring, xid_cs:bitstring)= BootNotificationRequestRet(m)
    in
if ( pu <> RemoteReset) then 0 else
```

CSMS at the receiving of the message checks the reason of the reboot.

BootNotificationResponse

```
let i=next(i) in
let m = encdh((i,BootNotificationResponse(RegistrationType, Interval)
    ), symk) in
out(c, m);
```

This message sent by the CSMS signals to the CS the Registration.

```
in(c, m: bitstring);
let xi = next(xi) in
let (=xi, m:bitstring)= decdh(m, symk) in
let ( rt: bitstring, int: bitstring) = BootNotificationResponseRet(m
    ) in
if(rt <> RegistrationType) then 0
else
```

CSMS at the receiving of the message checks the RegistrationType.

5.3 Cryptographic Elements of the Proverif Models

In this section are presented the cryptographic functions implemented in the OCPP Proverif Model Implemented for this thesis. They are common to both the Use Cases [section 5.1, section 5.2]

5.3.1 Chosen Ciphersuite

The choice of the Cipher Suite is for the implementation of the Use Cases in this thesis has been made between the four ciphersuites which in OCPP Specification are required to be supported as a minimum for the Security Profiles [section 5.4.1, section 5.4.2] required for the Use Cases application, and then selectiong the recommended one:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256.
For the scope of this thesis, for the model of the protocol sections in Proverif [Chapter 4], the relevant alghoritms required in this ciphersuite are: Protocol: Transport

Layer Security (TLS) [3] Key Exchange: Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) [4] Authentication: Elliptic Curve Digital Signature Algorithm (ECDSA) [4] TLS defines the security level of the communication, negotiated during the Handshake phase of the Security Profiles model [section 5.4]. Key Exchange defines the algorithm used to compute the symmetric shared session key for the CS and CSMS encrypted phase of the protocol section. Authentication defines how the communication parties should authenticate and check the authentication of the sent and received messages.

5.3.2 Asymmetric encryption

```
(* Asymmetric Encryption *)
type pkey.      (* public key *)
type skey.      (* private key *)
type keymat.    (* key material *)

fun pk(keymat): pkey.
fun sk(keymat): skey.
fun penc(bitstring, pkey): bitstring.
fun senc(bitstring, skey): bitstring.
reduc forall x:bitstring, y:keymat; sdec(penc(x,pk(y)),sk(y)) = x.
```

Types generated for the Asymmetric Encryption are the Public Key and Secret key types, and Key Material. Secret and Public key pair are generated from the key material by sk and pk functions. Each certified entity has its's own key material. The penc function encrypts a bitstring message by means of the public key, so that only the meant receiver can decrypt with the correspondent secret key, to ensure confidentiality.

5.3.3 Signature

```
(* Signature *)
type result.
fun ok():result.
fun sign(bitstring, skey): bitstring.
forall m: bitstring, y: keymat; getmess(sign(m,sk(y))) = m.
reduc forall m: bitstring, y: keymat; checksign(sign(m, sk(y)), pk(y)
   ) = ok().
```

For the non-repudiation property, a message sent by an entity is siged by the sign function with the secret key of the sender: checksign function is used by the receiver to check wheter the message was signed by the owner of the secret key corresponding to the public key owner, the result is ok() in case the check result is positive. Only after the check the receiver will recover the message with the getmes function.

5.3.4 ECDHE

```
type G.      (* diffie-hellman Curve parameter*)
type dhkey.
type sdhpar.
type pdhpar

const g: G.
fun dh(sdhpar, G): bitstring.
fun sdh(bitstring, sdhpar): dhkey.
equation forall a, b: sdhpar, g:G ; sdh(dh(b, g), a) = sdh(dh(a, g),
    b).
fun encdh(bitstring, dhkey): bitstring.
reduc forall m: bitstring, k: dhkey; decdh(encdh(m,k),k) = m.
```

Elliptic Curve Diffie Hellman Ephemeral : Given a curve parameter and an ephemeral key of type sdhpar, the secret Diffie Hellman parameter, the function dh computes a bitstring which represents the public parameter of the ECDHE exchange. An entity, with the function sdh, given the public parameter of the other entity and it's own ephemeral key, can compute the symmetric Diffie Hellman premaster secret, of type dhkey. The following equation guarantees the symmetric property of the computation. Using the PRF [section 5.3.5], the master secret of type dhkey will be computed from the pre master secret. By means of the encdh and decdh functions, a message can be crypted and decrypted with the same key, i.e. symmetric encryption. A relevant aspect is that this type of encryption definition in Proverif has an integrated check of integrity of the message decription: therefore, MAC is intrinsecally implemented.

5.3.5 Other functions

Other Function have been implemented to guarantee properties required for the protocol or to ensure the readability of the code.

Certificate Signature

```
fun signCA(pkey, skey): bitstring.
```

```
reduc forall m: pkey, km_RA: keymat; checksignCA(signCA(m, sk(km_RA))
    , pk(km_RA)) = ok().
reduc forall k: pkey, km_RA: keymat; getpKey(signCA (k, sk(km_RA)),
    pk(km_RA)) = k.
```

This function scope is to avoid the type conversion or incapsulation of the certified pkey. The function signCA represents the signature of a message of type pkey instead of bitstring. The checksignCA and getPKey are equivalent to checksign and getmess from Signature functions.

Message Identifier

```
(*Message counter*)
 type counter.
 free i_seed: counter.
fun next(counter):counter.
```

To avoid replay attacks, each message is identified by a value i in the CSMS, xi in the CS. The type is called counter, starts from a seed in the CSMS. The value is sent to the Client, then both Client and Server computes next(i) to be sent with sent messages and to be checked if sequential to the previous one in received messages.

Messages encapsulation functions examples

```
 fun HTTPGET( bitstring, bitstring, bitstring) : bitstring.
  reduc forall PD, AB, UP: bitstring; HTTPGETret(HTTPGET(PD, AB, UP))
    = (PD, AB, UP).

fun HTTP401( bitstring ): bitstring.
    reduc forall m:bitstring; HTTP401ret(HTTP401(m)) = m.

fun HTTP200 ( bitstring ): bitstring.
    reduc forall m: bitstring; HTTP200ret( HTTP200(m) ) = m .
```

For code readability, some message are encapsulated with functions with a representative name. The function itself is transparent to data, i.e. doesn't modify the input values. Return functions identified by the “ret” string return the original messages incapsulated.

PRF Pseudorandom master secres

```
fun masterSecret( dhkey, nonce, nonce):dhkey.
```

Given the public symmetric premaster secret Diffie Hellman parameter of the ECDHE key exchange and the two nonces identifiyng the session, this function computes the master secret.

Credential Management

```
fun  credentialString( bitstring , bitstring ): bitstring .
    reduc forall username, password: bitstring ; credentialStringRet
  (credentialString( username, password)) = (username , password).

```

These functions are used to construct the credential string given the username and password, or viceversa to retrieve them given the credential string.

5.4 Security Profiles

As outlined in [subsection 3.3.4], OCPP 2.0.1 [15] is capable of supporting three distinct security profiles, each of which defines the specific security measures that are employed by the entity in question and the related requirements. Generic requisites for the profiles are: Charging Station and CSMS utilise a single security profile at any given time and terminate the connection if the other entity attempts to establish a connection with a different profile. If the prerequisites are met, the security profile may be upgraded via the use case A05. Conversely, the OCPP specification does not permit the reduction of the security profile to a less secure level. The decision regarding the selected use case is based on the necessity to test the lower security profile. Security Profile 1, which is an unsecured transport with basic authentication, does not include any authentication to the CSMS or measures to establish a secure communication channel. It is therefore recommended that this profile should only be used in trusted networks, and that it is outside the scope of this work. Excluding from consideration for the aforementioned reasons the security profile 1, the choice was made to model to be verified against the ProVerif prove the use cases requiring security profile 2. The following section provides a detailed explanation of the schema and exchanged messages required to implement security profile 2 and security profile 3, which is related to the modelling of use case A05 - upgrade CS security Profile [section 5.2].

5.4.1 Security Profile 2 -TLS with Basic Authentication

In the TLS with Basic Authentication profile, the communication channel is secured through the use of Transport Layer Security (TLS). The Charging Station (CS) authenticates the CSMS via the TLS server certificate, whereas the authentication of the CS is performed using the HTTP Basic authentication method. The utilisation of TLS to secure communication within this profile will result in the transmission of the password in an encrypted form, thereby mitigating the potential risks associated with this authentication method. The client (CS) initiates the TLS

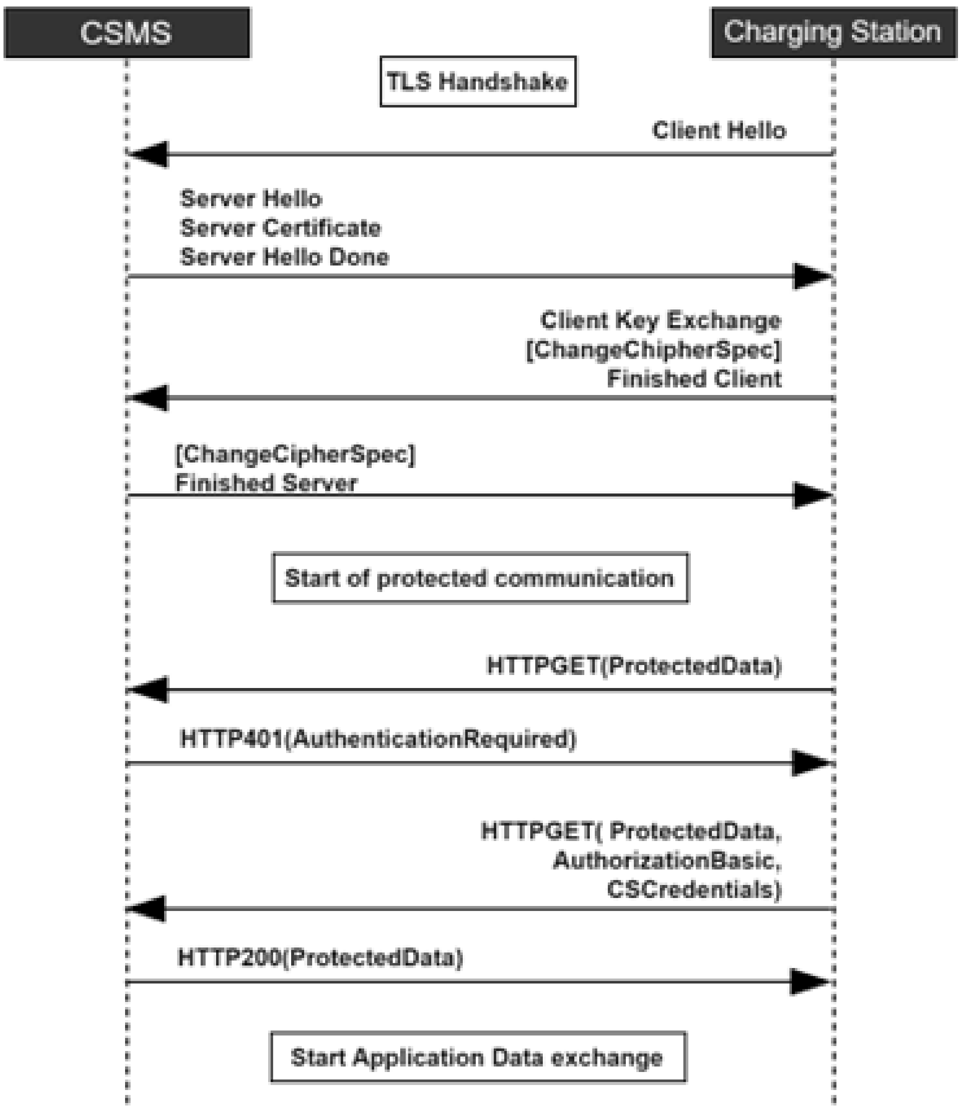


Figure 5.3: Sequence Diagram: TLS with Basic Authentication sequence diagram

handshake with the transmission of a Client Hello message, which serves to propose the session exchange parameters supported by the CS. In this context, security profile 2 is specified in this message. Subsequently, the client awaits the arrival of a

ServerHello message. In response to a ClientHello message, the server will transmit a Server Hello message when it has identified an acceptable set of algorithms. This exchange formalises the agreement upon the key exchange method, and the use of certificates for authentication. The server transmits a Server Certificate message, as delineated in the selected authentication scheme, and a ServerHelloDone message, which signals the conclusion of the ServerHello and associated messages.

The parameters for the established key exchange method are contained within the Client Key Exchange, which is immediately followed by a ChangeCipherSpec that signals the transition to encrypted communication. The Finished Client is the first message protected with the shared session key. It contains a summary of the exchanged messages thereby confirming the parameters agreed upon by the entities. The CSMS Server similarly transmits a ChangeCipherSpec and Finished Server with analogous meaning. Once a side has transmitted its Finished message and received and validated the Finished message from its peer, it may then begin to transmit and receive application data over the connection.

5.4.2 Security Profile 3 - TLS with Client Side Certificates

Similary to security profile 2 [subsection 5.3.1] in security profile 3 the communication channel is secured using Transport Layer Security (TLS) and the Charging Station authenticates the CSMS via the TLS server certificate. Differently, Charging Station authentication method is upgraded toTLS with client side certificate. The client (CS) initiates the TLS handshake with the transmission of a Client Hello message, the purpose of which is to propose the session exchange parameters supported by the CS. For this work, security profile 3 is declared in this message. Subsequently, the client awaits the arrival of a ServerHello message. In response to a ClientHello message, the server will transmit a Server Hello message when it has identified an acceptable set of algorithms. This message formalises the agreement upon the key exchange method, and the use of certificates for authentication. Subsequently, the server transmits a Certificate Server Request, thereby requiring the client to send its own certificate and the Server Certificate message, as delineated in the selected authentication scheme. The final message is the Server Hello Done message, which signals the conclusion of the ServerHello and associated messages.

In response to the certificate request, the CS Client transmits a Client Certificate message, as specified by security profile 3 for the CS authentication. The parameters for the established key exchange method are contained within the Client Key Exchange, which is immediately followed by a ChangeCipherSpec that signals the transition to encrypted communication. The Finished Client is the first message protected with the shared session key. It contains a summary of

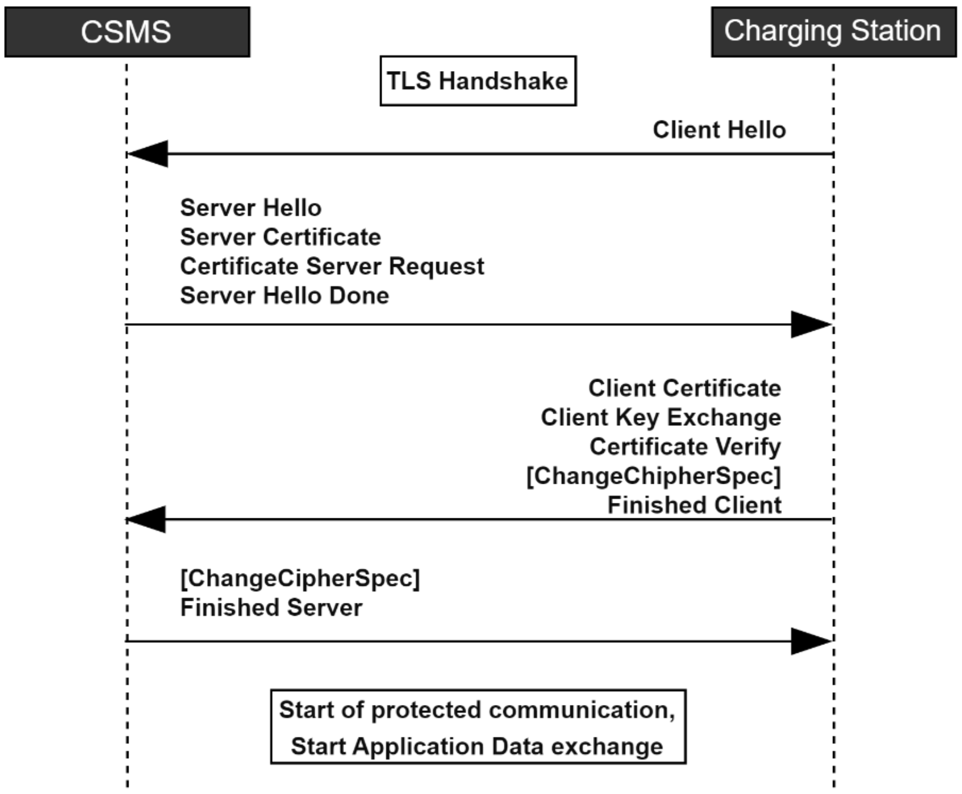


Figure 5.4: Sequence Diagram: TLS with Client Side Certificates

the exchanged messages to confirm the parameters agreed upon by the entities. The CSMS Server similarly transmits a ChangeCipherSpec and Finished Server with analogous significance. Once a side has transmitted its Finished message and received and validated the Finished message from its peer, it may then commence transmitting and receiving application data over the connection.

5.4.3 Messages

This subsection defines the message structures for the ProVerif model presented in this thesis. Despite the simplified structure of the message, the modelled values accurately reproduce and maintain the content and purpose of the original message definition taken from the OCPP 2.0.1 specification. The original message names have been maintained for clarity and readability of the modelled code.

Client Hello

```
out(c, (nonce_CS, proposed_suite, tls_version, sec_profile));
```

The initial message transmitted by the CS Client comprises the CS nonce, nonceCS, a randomly generated structure generated by the client, the version of the TLS protocol by which the client wishes to conduct the session, the proposed cipher suite and the security profile implemented. Thoses are modelled as a list of bit-string variables passed from the client to the server in the ClientHello message and contains the combinations of cryptographic algorithms supported by the client, simplified to a single choice in this work.

Server Hello

```
let chosen_suite = xproposed_suite in
new nonce_CSMS: nonce;
let m = (nonce_CSMS, chosen_suite) in
```

Server Hello Message is modeled to contain a nonce and a suite: this fields will contain a random generated Server nonce, indipendent from the client nonce, and one of the ciphersuites supported by the CS client, in this work modeled for simplicity as a single choice, and received with Client Hello.

Server Certificate

```
let m = ( CSMS_CAsigned_certificate ) in
out(c, m);
```

This message transmits the server's certificate chain to the client. In this thesis model, the public key of the CSMS server can be directly verified with a trusted root certification authority by the CS client. For further clarification, please refer to section 5.3.

Certificate Server Request

```
let m = ( CertificateServerRequest, CSMS_CAsigned_certificate ) in
out(c, m);
```

Sent alongside with the CSMS Server Certificate, the Certificare Server Request in the security profile 3 context has the role to require the CS Client to provide its

Certificate to perform authentication.

Server Hello Done

```
let m = sign(HelloDone, sk(km_CSMS)) in
out(c, m);
```

The message contains the constant "HelloDone" which signifies that the server has concluded its transmission of messages to facilitate the key exchange and that the client may now proceed with its designated phase of the key exchange.

Client Certificate

```
out(c, penc(CS_CAsigned_certificate, xpk_CSMS));
```

In accordance with the security profile 3 requirements of CS Authentication with Certificate, this message is transmitted subsequent to the Server's Certificate Server Request. It contains the CA authenticated certificate, which serves to certify the public key of the CS.

Client Key Exchange

```
out(c, penc( dhpar_CS, xpk_CSMS));
```

The transmission of key exchange parameters, better explained in subsection 5.3.3, allows each entity to agree on a shared key. As the chiphersuite chosen for this work requires, this message contains the parameters for Ephemeral Diffie Hellman key exchange. In summary, this message contains the client's Diffie Hellman public parameter, dhparCS, which has been computed from the CS secret ephemeral DH key and encrypted with the public key xpkCSMS of the CSMS server for confidentiality.

Chapter 6

Results of the analysis

In this chapter are reported all the analysis performed on the implemented use cases, with reference to the relative security profiles required. For each property, a description of the elements, events and queries is provided. For each verification, the results of the ProVerif analysis are reported, along with their interpretation.
About MAC and Integrity The use of MACs in conjunction with symmetric encryption, as mentioned in the ProVerif manual [1], is generally useless in ProVerif because the basic encryption is already authenticated. For the same reason, integrity checks are not modelled explicitly in this work because integrity is implemented and tested intrinsically with the decryption function and the values correspondence tests.

6.1 Results of A01 - Update Charging Station Password

This section presents the desired properties for the password update with use case A01: the model includes the connection of the CS and the CSMS with Security Profile 2 [subsection 5.3.1], and the implementation of Use Case A01 [subsection 5.1] where the CSMS requires to the CS to update the password of its credentials.This is followed by the CS reconnecting to the CSMS using the updated password. For each of the desired properties, a brief summary of its meaning is reported, how the verified elements are implemented in the model, the queries used for verification, the result reported in the ProVerif output and its interpretation.

6.1.1 Description Use Case A01

This Use Case A01- is part of the Security Functional Block [section 3.3.2]. The Objective of the message exchange between the CSMS and the CS is to define how to use the Basic Authentication security profile and update the credentials at the request of the CSMS, which transmits a new value for the CS Password. The implementation of Security Profile 2 [section 5.4.2] is necessary for this use case to establish an authenticated communication channel and a shared session symmetric key via the key exchange algorithm ECDHE. The Charging Station authenticate itself using HTTP Basic Authentication [17] by means of the installed credentials, and the CSMS authenticates to the CS using a TLS server certificate [5]. The communication between Charging Station and CSMS is secured using TLS [7]. For the full code, please refer to Appendix A.

6.1.2 Secrecy

Secrecy queries check whether the attacker can reconstruct the tested term or not. This can be done using two types of queries, attacker and secret. The terms tested for secrecy in this implementation of use case A01 [subsection 5.1] are the following:

- secretApplicationData: general definition of data exchanged by the entities,typically after the implemented use cases and security profiles authentications, which should stay confidential.
- PasswordCS: CS's password, installed OOB by manufacturer and CSO to CS and CSMS, for CS HTTP Basic authentication.
- eph_k_CS: CS's Ephemeral Key parameter for ECDHE key exchange
- eph_k_CSMS: CS's Ephemeral Key parameter for ECDHE key exchange.
- credentialsCS1: string containing CS's username and password, used to authenticate to the CSMS.
- NewPasswordCS: password created to the CSMS and sent to the CS to update the installed one.
- newCredentialsCS: string containing CS's username and the new password NewPasswordCS, used to authenticate to the CS to the CSMS after the password update.

With the attacker queries, ProVerif attempts to prove that a state in which the parameters are known to the attacker is unreachable, that is true when the names are not derivable by the attacker. This queries requires to be applied on private global free names.

```
query attacker(secretApplicationData).
query attacker (PasswordCS).
```

Query secret x provides an alternative way to test secrecy to query attacker(x), testing the secrecy of the term x, where x must correspond to a bound variable or name inside the considered processes. In this implementation, the secret queries used to test secrecy of aforementioned local names and variables are:

```
query secret eph_k_CS.
query secret eph_k_CSMS.
query secret credentialsCS1.
query secret NewPasswordCS.
query secret newCredentialsCS.
```

Results are reported below.

Interpretation of the results:

```
 Query not attacker(secretApplicationData[]) is true.
 Query not attacker(PasswordCS[]) is true.

 Query secret eph_k_CS is true.
 Query secret eph_k_CSMS is true.
 Query secret credentialsCS1 is true.
 Query secret NewPasswordCS is true.
 Query secret newCredentialsCS_1,newCredentialsCS is true.
```

ProVerif attempts to prove that a state in which a property is violated is unreachable.It follows that, given the query attacker(M), ProVerif internally attempts to show not attacker(M) and hence result not attacker(M) is true means that the secrecy of M is preserved by the protocol. The output of the queries are all positive results: true means that the tested property of secrecy holds for the term being tested. Note: The newCredentialCS term was tested in both the CS client process and the CSMS server process because two local names with the same identifier and meaning were defined in the implementation.

6.1.3 Observational Equivalence

ProVerif can prove some observational equivalences, including strong secrecy. Strong secrecy means that the attacker is unable to distinguish when the secret changes. In other words, the value of the secret should not affect the observable behavior of the protocol. This property has been tested for this work on the private global free names secretApplicationData and PasswordCS. The queries used for this scope in the implemented model are:

```
noninterf secretApplicationData.
noninterf PasswordCS.
```

Results are reported below.

Interpretation of the results:

```
Non-interference secretApplicationData is true.
Non-interference PasswordCS is true.
```

The results of the queries are all positive. The true outcome indicates that the tested property of strong secrecy is holds for both the tested terms. This means that an attacker is unable to distinguish when the tested secrets change. In particular, this concept is important when the secret consists of known values, as in the case of secretApplicationData, which could represent known messages of the protocol.

6.1.4 Authentication

In order to implement A01 Update Password, the CS Client is required to perform authentication using HTTP Basic Authentication. The CSMS will then authenticate with a certificate, as required by Security Profile 2. The ciphersuite selected for implementation includes ECDHE key exchange. The authentication is verified by means of correspondence injective assertions. The verification implementation and result are not included in this section but can be found in section 6.3. This was done for brevity, as the same security profile is implemented and verified for use case A05 Upgrade CS Security Profile, and the authentication verification is analogous. In this section of the document is reported a correspondence query as a sanity check on the ECDHE key exchange performed. The events are:

```
event dhparCSMS( pkey, bitstring, bitstring).
event dhparCS( pkey, bitstring, bitstring).
```

The query contains the computed symmetric key and the public parameters used to create it. It verifies whether the CS believes it has exchanged parameters and created a shared key with the intended CSMS server.

```
query x:pkey, z,w:bitstring ; inj-event(dhparCS(x,z,w)) ==> inj-event
    (dhparCSMS(x,z,w)).
```

Results are reported below.

Interpretation of the result: The results of the queries verify that an entity performs the ECDHE key exchange with security profile 2. At each instance of the event creation of the DH parameters at the CS Client corresponds the creation of the same parameters at the CSMS Server. This reflects the fact that CSMS is already authenticated.

```
Query inj-event(dhparCS(x_2,z,w)) ==> inj-event(dhparCSMS(x_2,z,w))
    is true.
```

The query output is positive, which is the expected outcome, given that the CSMS server is already authenticated.

6.1.5 Reachability of the Events

Proving reachability properties is ProVerif' s most basic capability. The processes are annotated with events, which mark important stages reached by the protocol but do not otherwise affect behavior. Relationships between event scan be specified as correspondence assertions. These are used in this model with the objective to be sanity checks of the implementation, as in case an event is unreachable, the result would be meaningless. In this section are reported the queries of event reachability for all the events defined to perform verification of the other properties presented in this section. Reachability queries of events used for Message Ordering verification:

```
                  (* Message Ordering events*)
query event(e00_ca()).
query event(e01_csms()).
query event(e01()).
query event(e02()).
query event(e03()).
query event(e04()).
query event(e05()).
query event(e06()).
query event(e07()).
query event(e08()).
query event(e09()).
```

Reachability queries of events used for Correspondence Session verification:

```
                  (*Correspondence Session events*)
```

```
query n,m:nonce; event(beginCSsession(n,m)).
query n,m:nonce; event(beginCSMSsession(n,m)).
query n,m:nonce; event(endCSsession(n,m)).
query n,m: nonce; event(endCSMSsession(n,m)).
```

Events created for the Authentication parameters correspondence verification:

```
                    (*Authentication parameters*)
query x:pkey, z,w:bitstring ; event(dhparCS(x,z,w)).
query x:pkey, z,w:bitstring ; event(dhparCSMS(x,z,w)).
```

Results are reported below.

Interpretation of the result:

```
Query not event(e00_ca) is false.
Query not event(e01_csms) is false.
Query not event(e01) is false.
Query not event(e02) is false.
Query not event(e03) is false.
Query not event(e04) is false.
Query not event(e05) is false.
Query not event(e06) is false.
Query not event(e07) is false.
Query not event(e08) is false.
Query not event(e09) is false.
Query not event(beginCSsession(n,m_15)) is false.
Query not event(beginCSMSsession(n,m_15)) is false.
Query not event(endCSsession(n,m_15)) is false.
Query not event(endCSMSsession(n,m_15)) is false.
Query not event(dhparCS(x_2,z,w)) is false.
Query not event(dhparCSMS(x_2,z,w)) is false.
```

The results of each query indicate that ProVerif has identified an attack trace against the desired security property. It should be noted that ProVerif is designed to demonstrate that a state in which a property is violated is unreachable. It follows that when query event (M) is supplied, ProVerif attempts to demonstrate that event (M) is not reachable. Therefore, the result is not event(M) is false when the event is reachable. In conclusion, all the events are reachable. The reachability verification of the events is crucial for interpreting the other correspondence queries. If an event A is not reachable, the query event A ¯ event B would lead to a positive result, which could lead to an erroneous conclusion that the property of correspondence holds. Conversely, the result depends on the fact that event A could never happen.

6.1.6 Session Correspondence

The correspondence ProVerif queries assert that if an event e has been executed, then event e1 has been previously executed. This is useful for verifying the injective correspondence of message exchange execution between the involved entities. The session correspondence has been modelled with events to verify the following facts:

```
event beginCSsession(nonce, nonce).
event beginCSMSsession(nonce, nonce).
event endCSsession(nonce, nonce).
event endCSMSsession(nonce, nonce).
```

The pair (nonceCS, nonceCSMS) identifies a specific session. Each of the event records a meaningful fact, in details:

- event (beginCSsession(nonceCS, xnonceCSMS)) which is used by the client to record the belief that she has accepted to run the protocol with that session identifier.

- event(beginCSMSsession(xnonceCS, nonceCSMS)), which is used to record the fact that the server considers he has accepted to run the protocol with that session identifier.

- event(endCSsession(nonceCS, xnonceCSMS)) which means the client has terminated a protocol run with that session identifier.

- event(endCSMSsession(xnonceCS, nonceCSMS)), which denotes the server's belief that he has terminated a protocol run with that session identifier.

Event are placed in the model as reported in schema [Figure 6.1], where for brevity are reported only the significant messages exchanged by the protocol for the objective of this property verification. To check that if the CS Client has terminated the session identified by the two nonce parameters, it follows that a CSMS Server has initiated a session with the same parameters, the following query has been modeled:

```
query n,m:nonce; inj-event(endCSsession(n,m)) ==> inj-event(
    beginCSMSsession(n,m)).
```

To check that if the CSMS Server has terminated the session identified by the two nonce parameters, it follows that CS Client has initiated a session with the same parameters, the following query has been modeled:

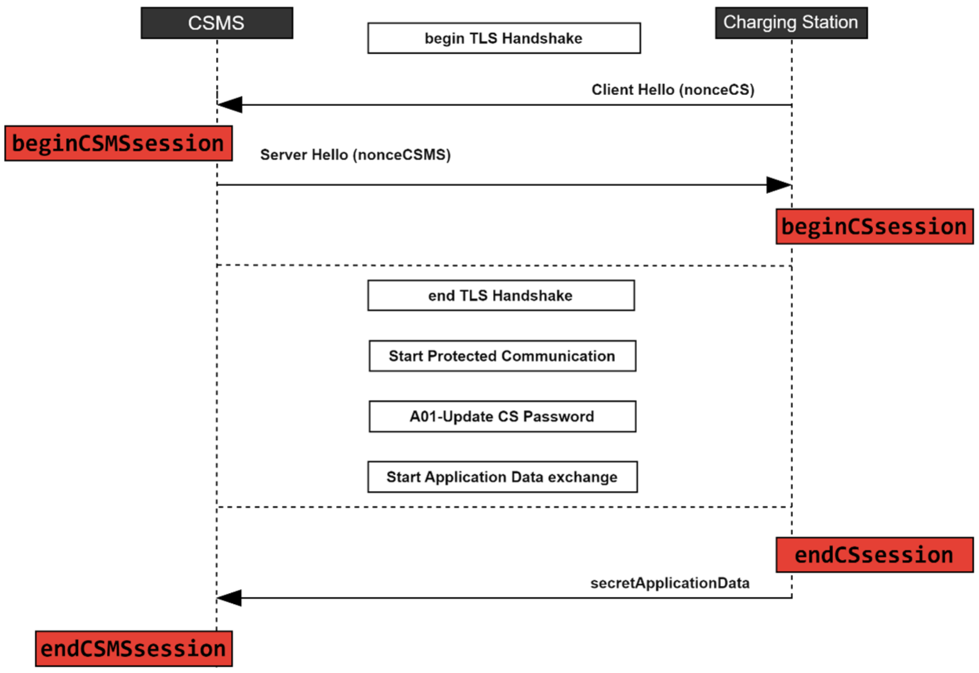


Figure 6.1: Sequence Diagram: Events for A01 session correspondence

```
query  n,m:nonce; inj-event(endCSMSsession(n,m)) ==> inj-event(
    beginCSsession(n,m)).
```

To check that if the CSMS Server has terminated the session identified by the two nonce parameters, it follows that CS Client has initiated a session with the same parameters, the following query has been modeled:

```
query  n,m:nonce; inj-event(endCSMSsession(n,m)) ==> inj-event(
    endCSsession(n,m)).
```

Results are reported below.

Interpretation of the result:

```
Query inj-event(endCSsession(n,m_15)) ==> inj-event(beginCSMSsession(
    n,m_15)) is true.

```

```
Query inj−event(endCSMSsession(n,m_15)) ==> inj−event(beginCSsession(
    n,m_15)) is true.

Query inj−event(endCSMSsession(n,m_15)) ==> inj−event(endCSsession(n,
    m_15)) is true.
```

The ProVerif output is true for all the aforementioned queries: this indicates that for each session, identified by the nonce value pair, terminated by the CS Client, the same session has been initiated and terminated at the CSMS Server, and vice versa. Consequently, it is not possible for an attacker to impersonate one of the parties in a session.

6.1.7 Message Ordering

A typical use of correspondences is to order all messages in a protocol. ProVerif can prove that each execution of event e01 is preceded by the execution of an instance of e00, and that, when event e02 is executed, each execution of that instance of e02 is preceded by the execution of an instance of e1. This can be implemented until the last message of the protocol has been received. The following events have been defined for this use:

```
event e00_ca().
event e01_csms().
```

These events have the objective to signal the CA providing the signed CSMS Authentication certificate at the beginning of the protocol and the CSMS to receive it.

```
event e01(). event e02(). event e03(). event e04().
event e05(). event e06(). event e07(). event e08(). event e09().
```

The objective of these events is to signal the reception of messages exchanged in the modelled section of the protocol. The placement of the events can be seen in the ProVerif implementation schema, as reported in figure 6.2. The defined queries, given the reachability of the events, tested in subsection 6.1.3.have the objective to test wheter the order of the events, i.e. the order of the messages exchanged, is respected.

```
query event (e01_csms()) ==> event (e00_ca()).
query event(e02()) ==> event(e01()).
query event(e03()) ==> event(e02()).
query event(e04()) ==> event(e03()).
query event(e05()) ==> event(e04()).
query event(e06()) ==> event(e05()).
query event(e08()) ==> event(e05()).
query event(e09()) ==> event(e08()).
```

Results are reported below.

Interpretation of the result:

```
Query event(e01_csms) ==> event(e00_ca) is true.

Query event(e02) ==> event(e01) is true.
Query event(e03) ==> event(e02) is true.
Query event(e04) ==> event(e03) is true.
Query event(e05) ==> event(e04) is true.
Query event(e06) ==> event(e05) is true.
Query event(e08) ==> event(e05) is true.
Query event(e09) ==> event(e08) is true.
```

The queries about the order of the events have a positive outcome. This implies that, given the fact that each event represents the receipt of a message, the reception of said message has been preceded by the sending of the correct message. Consequently, a message cannot be received from an entity that is not the expected recipient.

6.2 Results of A05 - Upgrade Charging Station Profile

In this section are presented the desired properties for the Security Profile Upgrade with use case A05: the model includes the connection of the CS and CSMS with Security Profile 2 [subsection 5.3.1], use case A05 implementation [subsection 5.2] where the CSMS requires to the CS to upgrade the security profile, and finally the reconnection of the CS to the CSMS using Security Profile 3. For each of the desired and tested properties, is reported a brief summary of its significate, how the verified elements are implemented in the model, the queries used for the verification, the result reported in the ProVerif output and its interpretation.

6.2.1 Secrecy

Secrecy queries verify wheter the attacker can reconstruct the tested term or not. This can be done by means of two types of queries, attacker and secret. The terms

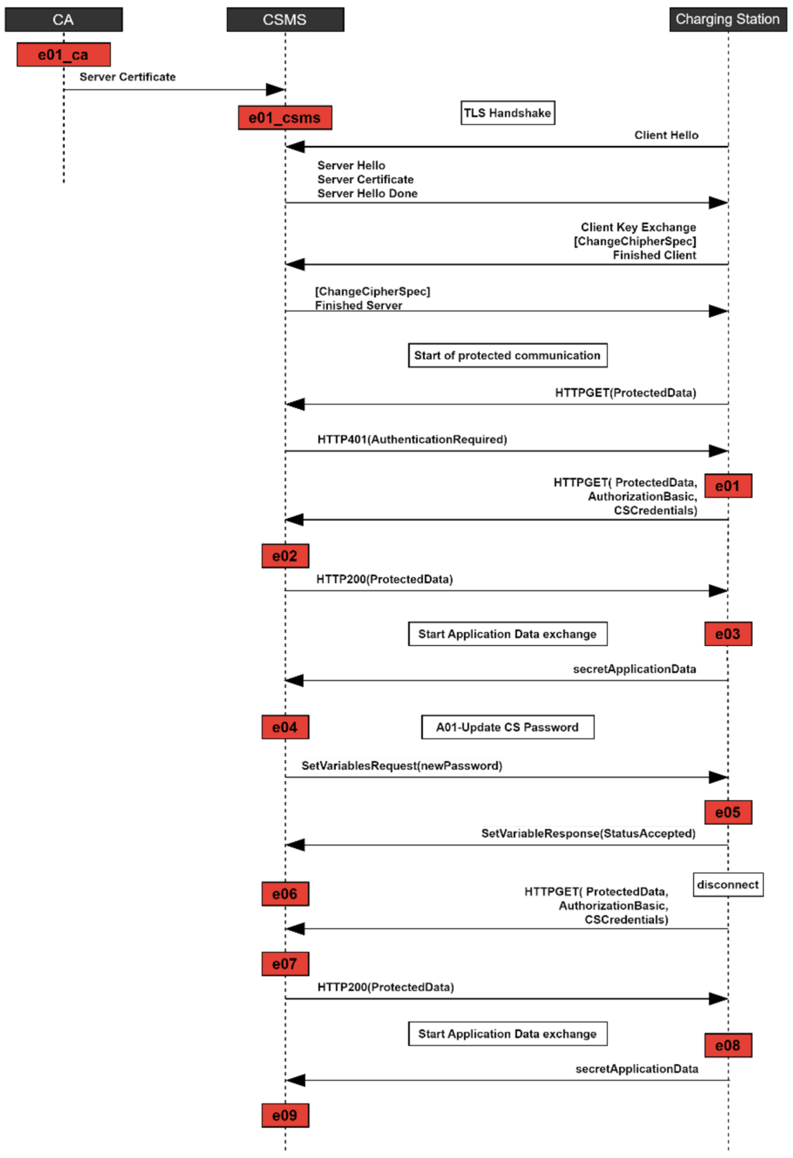


Figure 6.2: Sequence Diagram: Events for A01 message ordering

which are tested for secrecy in this implementation of the use case A05 [subsection 5.2] are the following:

- secretApplicationData: general definition for data exchanged by the entities, out of the scope of the implemented use case, which should remain confidential.
- PasswordCS: CS's password, installed OOB by manufacturer and CSO to CS

and CSMS, for CS HTTP Basic authentication

- eph_k_CS and eph_k_CS2 : CS's Ephemeral Key parameters for ECDHE key exchange in session 1 and 2.
- eph_k_CSMS and eph_k_CSMS2: CS's Ephemeral Key parameters for ECDHE key exchange in session 1 and 2.
- credentialStringCS: string containing CS's username and password, used to authenticate to the CSMS with Security profile 2 in session 1.

With the attacker queries, ProVerif attempts to prove that a state in which the parameters are known to the attacker is unreachable, that is true when the names are not derivable by the attacker. This queries requires to be applied on private global free names.

```
query attacker(secretApplicationData).
query attacker (PasswordCS).
```

Query secret x provides an alternative way to test secrecy to query attacker(x), testing the secrecy of the term x, where x must correspond to a bound variable or name inside the considered processes. In this implementation, the secret queries used to test secrecy of aforementioned local names and variables are:

```
query attacker(secretApplicationData).
query attacker (PasswordCS).

query secret eph_k_CS.
query secret eph_k_CSMS.

query secret eph_k_CS2.
query secret eph_k_CSMS2.
```

Results are reported below.

Interpretation of the results:

```
Query not attacker(secretApplicationData[]) is true.
Query not attacker(PasswordCS[]) is true.

Query secret eph_k_CS is true.
Query secret eph_k_CSMS is true.
Query secret eph_k_CS2 is true.
Query secret eph_k_CSMS2 is true.
```

ProVerif attempts to prove that a state in which a property is violated is unreachable. It follows that, given the query attacker(M), ProVerif internally attempts to show not attacker(M) and hence result not attacker(M) is true means that the secrecy of M is preserved by the protocol. The output of the queries are all positive results: true means that the tested property of secrecy holds for the term being tested.

6.2.2 Observational Equivalence

ProVerif can prove observational equivalences, including strong secrecy. Strong secrecy means that the attacker is unable to distinguish when the secret changes. In other words, the value of the secret should not affect the observable behavior of the protocol. This property has been tested for this work on the private global free names secretApplicationData and PasswordCS, whose role is explained in subsection 6.2.1. The queries used for this scope in the implemented model are:

```
noninterf secretApplicationData.
noninterf PasswordCS
```

Results are reported below.

Interpretation of the results:

```
Non-interference secretApplicationData is true.
Non-interference PasswordCS is true.
```

The output of the queries are all positive results: true means that the tested property of strong secrecy holds for both the terms. This means that an attacker is not able to distinguish when the those secrets change. This concept is important when the secret consists of known values such as in the case of secretApplicationData, that could represent known messages of the protocol.

6.2.3 Authentication

For this implementation of A05 Upgrade Security Profile, the CS Client is required to performs authentication using HTTP Basic Authentication and the CSMS authenticates with certificate, as required by Security Profile 2 in use. Then the security profile is upgraded by the CSO: CS and CSMS reconnects by means of the security profile 3 authentication requirements, which includes certificate authentication for both Client and Server. The Ciphersuite selected for the implementation requires a ECDHE key exchange. The mutual authentication is verified by means of correspondence injective assertions. The implementation of the verification of security profiles 2 and 3 and their result can be found in section 6.3. This choice was made for brevity, as the security profile 2 is also implemented and verified for the use case A01 Update CS Password, and the authentication verification is analogue. In this section of the document is reported a correspondence query as a sanity check on the ECDHE key exchange performed for both Session 1 (connection with security profile 2) and Session 2 (reconnection with security profile 3). The events are:

```
event dhparCSMS( pkey, bitstring, bitstring).
event dhparCS( pkey, bitstring, bitstring).

event dhparCS2(pkey, bitstring, bitstring).
event dhparCSMS2(pkey, bitstring, bitstring).
```

Which contain the computed symmetric key and the public parameters used to create it. The query reported below verifies if the CS belief to have exchanged parameters and created a shared key with the intended CSMS Server.

```
query x: pkey, z,w: bitstring ; inj-event(dhparCS(x, z, w)) ==> inj-
    event(dhparCSMS(x, z, w)).
query x: pkey, z,w: bitstring ; inj-event(dhparCS2(x, z, w)) ==> inj-
    event(dhparCSMS2(x, z, w)).
```

Results are reported below.

Interpretation of the results:

```
Query inj-event(dhparCS(x_2,z,w)) ==> inj-event(dhparCSMS(x_2,z,w))
    is true.
Query inj-event(dhparCS2(x_2,z,w)) ==> inj-event(dhparCSMS2(x_2,z,w))
    is true.
```

The results of the queries verify that an entity performs the ECDHE key exchange with security profile 2 and also with security profile 3 after the reconnection. At each instance of the event creation of the DH parameters at the CS Client corresponds the creation of the same parameters at the CSMS Server. This reflects the fact that CSMS is already authenticated.

6.2.4 Reachability of the Events

Proving reachability properties is ProVerif' s most basic capability: we annotate processes with events, which mark important stages reached by the protocol but do not otherwise affect behavior. Relationships between events may now be specified as correspondence assertions. These are used in this model with the objective to be sanity checks of the implementation, as in case an event is not reachable, other queries i.e. correspondence queries, would be meaningless. In this section are reported the queries of event reachability for all the events used to perform verification of the other properties presented in this Section for the Password Update implementation. Reachability queries of events used for Message Ordering verification:

```
(* Message Ordering events*)
query event (e05_cso()).
query event (e01_ca()).
query event (e02_ca()).
query event(e01()).
query event(e02()).
query event(e03()).
query event(e04()).
query event(e05()).
query event(e06()).
query event(e07()).
query event(e08()).
query event(e09()).
query event(e10()).
query event(e11()).
query event(e12()).
query event(e13()).
query event(e14()).
query event(e15()).
query event(e16()).
query event(e17()).
```

Reachability queries of events used for Correspondence Session verification:

```
                    (*Correspondence Session 1 events*)
query n,m: nonce ; event(beginCSsession(n, m)).
query n,m: nonce ; event(beginCSMSsession(n, m)).
query n,m: nonce ; event(endCSsession(n, m)).
query n,m: nonce ; event(endCSMSsession(n, m)).
                (*Correspondence Session 2 events*)
query n,m: nonce ; event(beginCSsession2(n, m)).
query n,m: nonce ; event(beginCSMSsession2(n, m)).
query n,m: nonce ; event(endCSsession2(n, m)).
query n,m: nonce ; event(endCSMSsession2(n, m)).
```

Events created for the Authentication parameters correspondence verification:

```
query x:pkey, z,w:bitstring ; event(dhparCS(x,z,w)).
query x:pkey, z,w:bitstring ; event(dhparCSMS(x,z,w)).

query x:pkey, z,w:bitstring ; event(dhparCS2(x,z,w)).
query x:pkey, z,w:bitstring ; event(dhparCSMS2(x,z,w)).
```

These events placement in the implemented model and usage and meaning can be seen in futher details at subsections 6.2.6, 6.2.5 and 6.2.3. Results are reported below.

Interpretation of the results:

```
Query not event(e01) is false.
Query not event(e02) is false.
Query not event(e03) is false.
Query not event(e04) is false.
Query not event(e05) is false.
Query not event(e06) is false.
Query not event(e07) is false.
Query not event(e08) is false.
Query not event(e09) is false
Query not event(e10) is false.
Query not event(e11) is false.
Query not event(e12) is false.
Query not event(e13) is false.
Query not event(e14) is false.
Query not event(e15) is false.
Query not event(e16) is false.
Query not event(e17) is false.

Query not event(beginCSsession(n,m_33)) is false.
Query not event(beginCSMSsession(n,m_33)) is false.
Query not event(endCSsession(n,m_33)) is false.
Query not event(endCSMSsession(n,m_33)) is false.

Query not event(beginCSsession2(n,m_33)) is false.
Query not event(beginCSMSsession2(n,m_33)) is false.
Query not event(endCSsession2(n,m_33)) is false.
Query not event(endCSMSsession2(n,m_33)) is false.
```

The results of each query indicate that ProVerif has identified an attack trace against the desired security property. It should be noted that ProVerif is designed to demonstrate that a state in which a property is violated is unreachable. It follows that when query event (M) is supplied, ProVerif attempts to demonstrate that event (M) is not reachable. Therefore, the result is not event(M) is false when the event is reachable. In conclusion, all the events are reachable. The reachability verification of the events is crucial for interpreting the other correspondence queries. If an event A is not reachable, the query 'event A corresponds to event B' would output a positive result, which could lead to an erroneous conclusion that the property of correspondence holds. Conversely, the result depends on the fact that event A could never happen.

6.2.5 Session Correspondence

Correspondence ProVerif queries asserts that if an event e has been executed, then event e1 has been previously executed. This is useful to verify the injective correspondence of message exchange execution between the involved entities. The session correspondence has been modeled with events to verify the following facts:

```
event beginCSsession(nonce, nonce).
event beginCSMSsession(nonce, nonce).
event endCSsession(nonce, nonce).
event endCSMSsession(nonce, nonce).

query n,m: nonce ; event(beginCSsession2(n, m)).
query n,m: nonce ; event(beginCSMSsession2(n, m)).
query n,m: nonce ; event(endCSsession2(n, m)).
query n,m: nonce ; event(endCSMSsession2(n, m)).
```

The pair (nonceCS, nonceCSMS) identifies a specific session. Each of the event records a meaningful fact, in details:

- event(beginCSsession(nonceCS, xnonceCSMS)) and event(beginCSsession2(nonceCS2, xnonceCSMS2)) which is used by the client to record the belief that she has accepted to run the protocol with that session identifier.

- event(beginCSMSsession(xnonceCS, nonceCSMS)) and event(beginCSMSsession2(xnonceCS2, nonceCSMS2)), which is used to record the fact that the server considers he has accepted to run the protocol with that session identifier.

- event(endCSsession(nonceCS, xnonceCSMS)) and event(endCSsession2(nonceCS2, xnonceCSMS2)) which means the client has terminated a protocol run with that session identifier.

- event(endCSMSsession(xnonceCS, nonceCSMS)) and event(endCSMSsession2(xnonceCS2, nonceCSMS2)) , which denotes the server's belief that he has terminated a protocol run with that session identifier.

Event are placed in the model as reported in schema [Figure 6.3], where for brevity are reported only the significant messages exchanged by the protocol for the objective of this property verification. To check that if the CS Client has terminated the session identified by the two nonce parameters, it follows that a CSMS Server has initiated a session with the same parameters, the following query has been modeled:

```
query  n,m:nonce; inj-event(endCSsession(n,m)) ==> inj-event(
    beginCSMSsession(n,m)).
query  n,m:nonce; inj-event(endCSsession2(n,m)) ==> inj-event(
    beginCSMSsession2(n,m)).
```

To check that if the CSMS Server has terminated the session identified by the two nonce parameters, it follows that CS Client has initiated a session with the same parameters, the following query has been modeled:

```
query  n,m:nonce; inj-event(endCSMSsession(n,m)) ==> inj-event(
    beginCSsession(n,m)).
query  n,m:nonce; inj-event(endCSMSsession2(n,m)) ==> inj-event(
    beginCSsession2(n,m)).
```

To check that if the CSMS Server has terminated the session identified by the two nonce parameters, it follows that CS Client has initiated a session with the same parameters, the following query has been modeled:

```
query  n,m:nonce; inj-event(endCSMSsession(n,m)) ==> inj-event(
    endCSsession(n,m)).
query  n,m:nonce; inj-event(endCSMSsession2(n,m)) ==> inj-event(
    endCSsession2(n,m)).
```

Results are reported below.

Interpretation of the results:

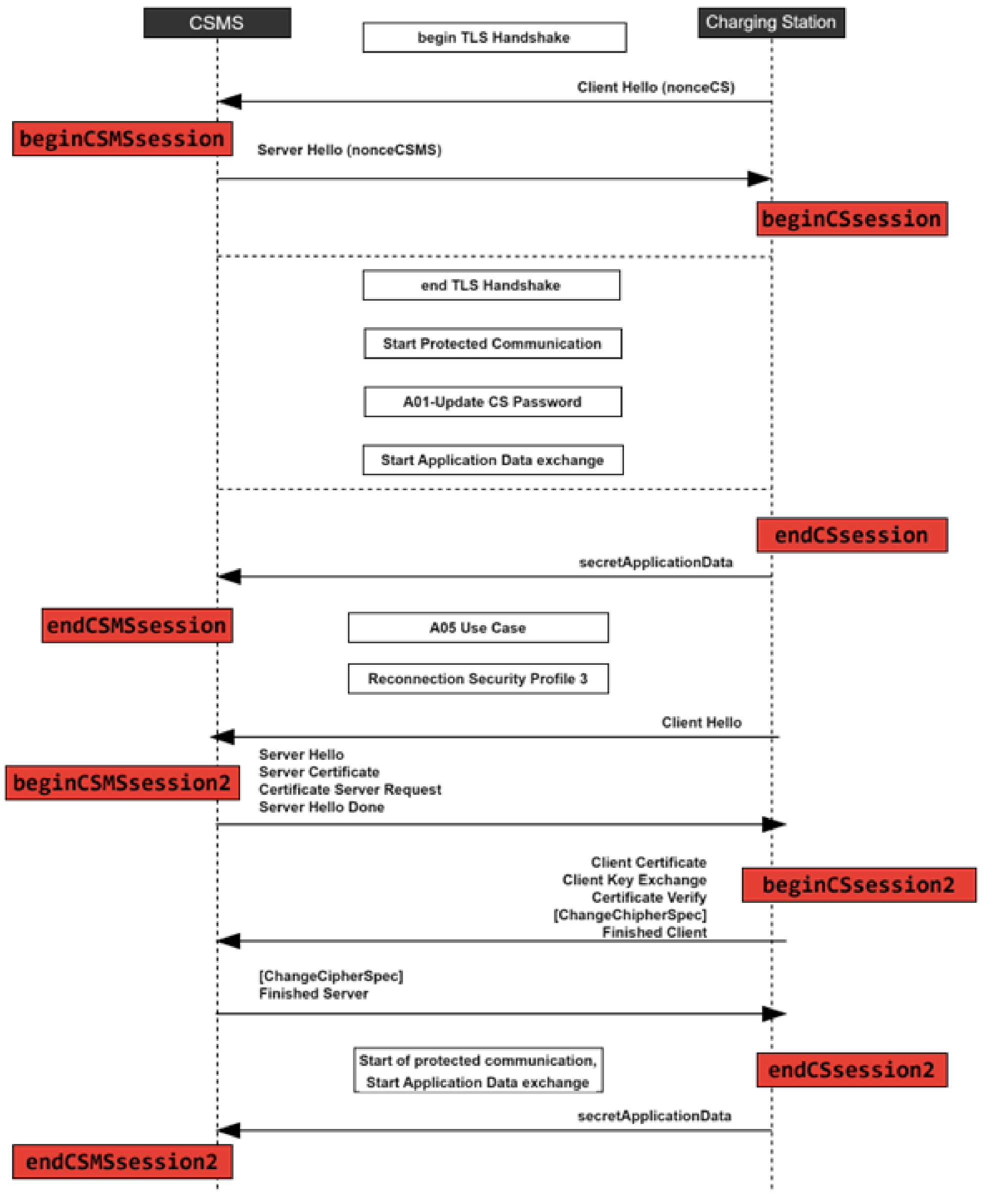


Figure 6.3: Sequence Diagram: Events for A05 session correspondence

```
Query inj−event(endCSsession(n,m_33)) ==> inj−event(beginCSMSsession(
    n,m_33)) is true.
Query inj−event(endCSMSsession(n,m_33)) ==> inj−event(beginCSsession(
    n,m_33)) is true.
Query inj−event(endCSMSsession(n,m_33)) ==> inj−event(endCSsession(n,
    m_33)) is true.
```

```

Query inj-event(endCSsession2(n,m_33)) ==> inj-event(
    beginCSMSsession2(n,m_33)) is true.
Query inj-event(endCSMSsession2(n,m_33)) ==> inj-event(
    beginCSsession2(n,m_33)) is true.
Query inj-event(endCSMSsession2(n,m_33)) ==> inj-event(endCSsession2(
    n,m_33)) is true.
```

The ProVerif output is true for all the aforementioned queries: this indicates that for each session, identified by the nonce value pair, terminated by the CS Client, the same session has been initiated and terminated at the CSMS Server, and vice versa. Consequently, it is not possible for an attacker to impersonate one of the parties in each of the session.

6.2.6 Message Ordering

A typical use of correspondences is to order all messages in a protocol. ProVerif can prove that each execution of event e01 is preceded by the execution of an instance of e00, and that, when event e02 is executed, each execution of that instance of e02 is preceded by the execution of an instance of e1. This can be implemented until the last message of the protocol has been received. The following events have been defined for this use:

The placement of the events can be seen in the ProVerif implementation schema, as reported in figure 6.4. The defined queries, given the reachability of the events tested in subsection 6.2.4, have the objective to verify wheter the order of the events, i.e. the order of the messages exchanged, is respected.

```
event e01_ca().
event e02_ca().
event e05_cso().
```

The first pair of events have the objective to signal the CA providing the signed CSMS Authentication certificate at the beginning of the protocol and the CSMS to receive it. The second is one of the pair used to test if the reception by the CSMS of the signal to upgrade the security profile, is preceded by the request of the CSO to perform it.

```
event e01(). event e02(). event e03(). event e04(). event e05().
    event e06().
event e07(). event e08(). event e09(). event e10(). event e11().
    event e12().
event e13(). event e14(). event e15(). event e16(). event e17().
```

The objective of these events is to signal the reception of messages exchanged in the modelled section of the protocol. The placement of the events can be seen in the ProVerif implementation schema, as reported in figure 6.4 and 6.5.

```
query event (e01()) ==> event (e01_ca()).
query event (e11()) ==> event (e02_ca()).

query event (e05()) ==> event (e05_cso()).

query event(e02()) ==> event(e01()).
query event(e03()) ==> event(e02()).
query event(e04()) ==> event(e03()).
query event(e05()) ==> event(e04()).
query event(e06()) ==> event(e05()).
query event(e07()) ==> event(e06()).
query event(e08()) ==> event(e07()).
query event(e09()) ==> event(e08()).
query event(e10()) ==> event(e09()).
query event(e11()) ==> event(e10()).
query event(e12()) ==> event(e11()).
query event(e13()) ==> event(e12()).
query event(e14()) ==> event(e13()).
query event(e15()) ==> event(e14()).
query event(e16()) ==> event(e15()).
query event(e17()) ==> event(e16()).
```

The defined queries, given the reachability of the events, tested in subsection 6.1.3.have the objective to test wheter the order of the events, i.e. the order of the messages exchanged, is respected.

Results are reported below.

Interpretation of the results:

```
Query event(e01) ==> event(e01_ca) is true.
Query event(e11) ==> event(e02_ca) is true.

Query event(e05) ==> event(e05_cso) is true.

Query event(e02) ==> event(e01) is true.
Query event(e03) ==> event(e02) is true.
Query event(e04) ==> event(e03) is true.
Query event(e05) ==> event(e04) is true.
Query event(e06) ==> event(e05) is true.
Query event(e07) ==> event(e06) is true.
```

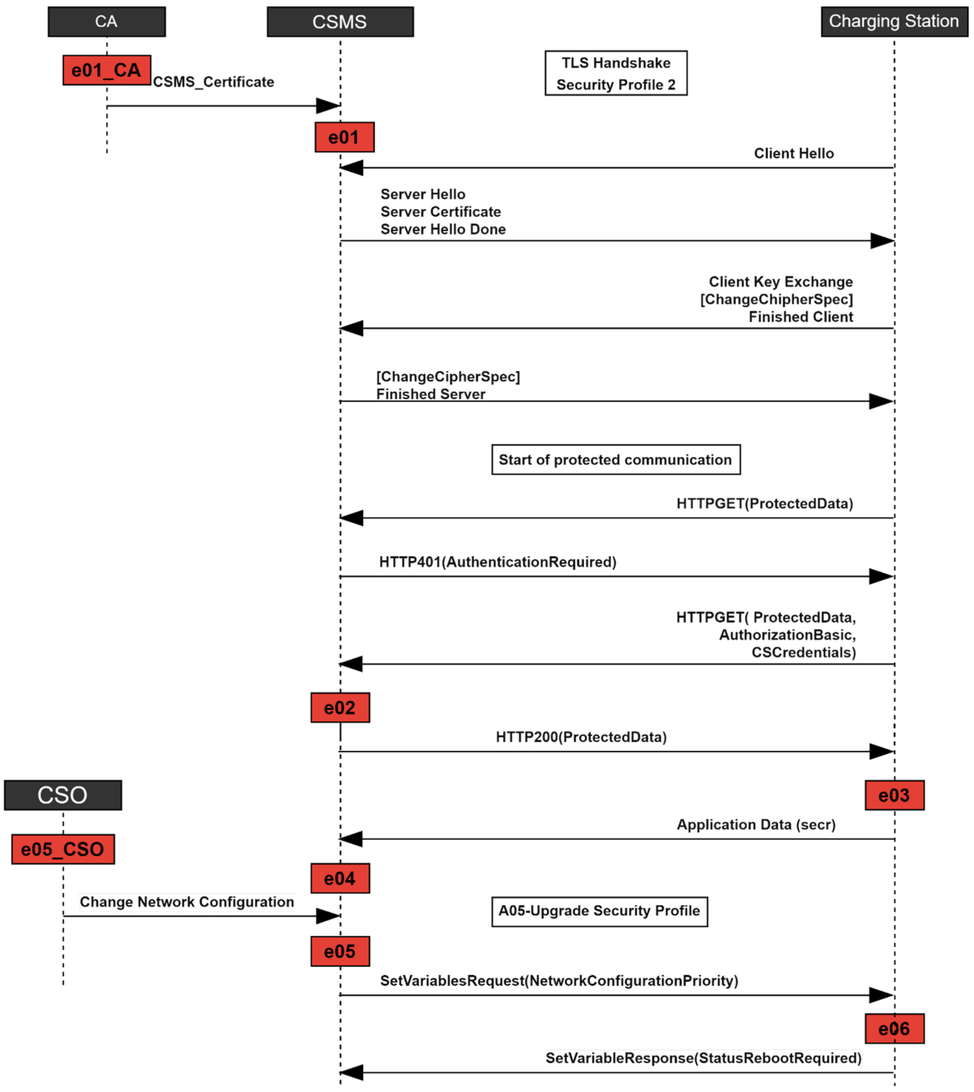

CA
CSMS
Charging Station
e01_CA
CSMS_Certificate
TLS Handshake Security Profile 2
e01
Client Hello
Server Hello
Server Certificate
Server Hello Done
Client Key Exchange
[ChangeChipherSpec]
Finished Client
[ChangeCipherSpec]
Finished Server
Start of protected communication
HTTPGET(ProtectedData)
HTTP401(AuthenticationRequired)
HTTPGET(ProtectedData, AuthorizationBasic, CSCredentials)
e02
HTTP200(ProtectedData)
CSO
e03
Application Data (secr)
e05_CSO
e04
Change Network Configuration
A05-Upgrade Security Profile
e05
SetVariablesRequest(NetworkConfigurationPriority)
e06
SetVariableResponse(StatusRebootRequired)

Figure 6.4: Sequence Diagram: Events for A05 message ordering 1

```
Query event(e08) ==> event(e07) is true.
Query event(e09) ==> event(e08) is true.
Query event(e10) ==> event(e09) is true.
Query event(e11) ==> event(e10) is true.
Query event(e12) ==> event(e11) is true.
Query event(e13) ==> event(e12) is true.
Query event(e14) ==> event(e13) is true.
Query event(e15) ==> event(e14) is true.
Query event(e16) ==> event(e15) is true.
Query event(e17) ==> event(e16) is true.
```

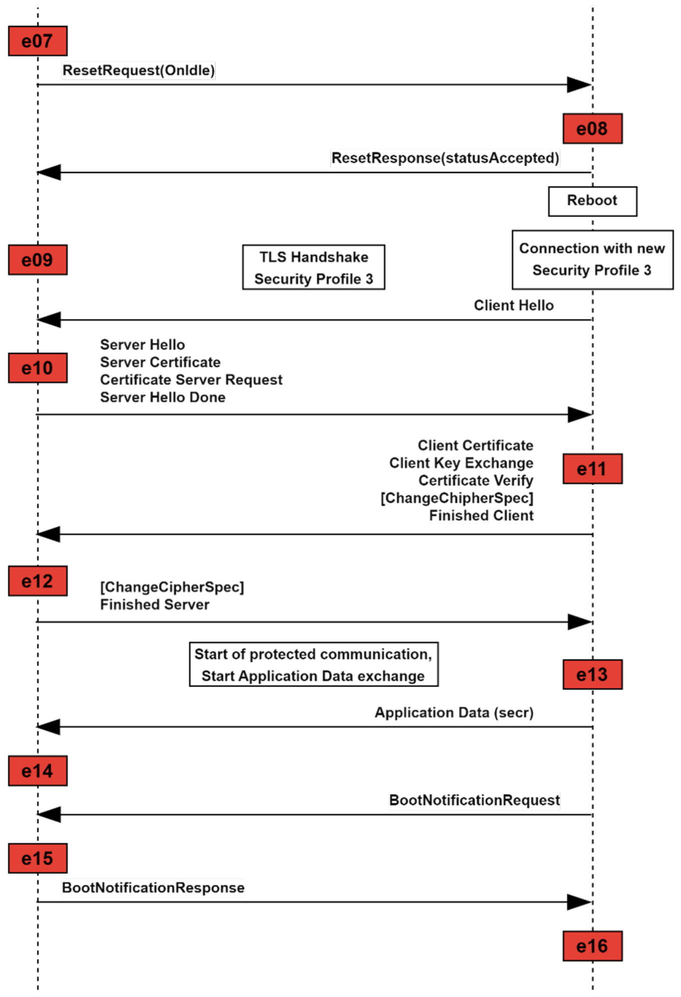


Figure 6.5: Sequence Diagram: Events for A05 message ordering 2

The queries about the order of the events have a positive outcome. This implies that, given the fact that each event represents the receipt of a message, the reception of said message has been preceded by the sending of the correct message. Consequently, a message cannot be received from an entity that is not the expected recipient.

6.3 Results Security Profiles Authentication

In this section is presented the verification of the Authentication property for the Security Profiles required in the model of the Use Cases analysed in this thesis. About the property of authentication, the protocol is intended to ensure that, if client A thinks she executes the protocol with server B, then she really does so, and vice versa. For each Security Profile are reported the declarations of events involved in the property verification, the schema of the events placement into the model, the queries and their meaning. Results of ProVerif verification and the relative interpretation are reported in subsection 6.3.

6.3.1 Security Profile 2

Security profile 2 [subsection 5.4.1] requires the Charging Station to perform authentication by means of HTTP Basic authentication, i.e. username and password credentials, and the CSMS to authenticate with TLS Certificate. To verify the authentication in this section of the protocol, some events are defined.

```
event acceptsCS (dhkey, pkey).
event acceptsCSMS(dhkey, bitstring).
event termCS(dhkey, bitstring).
event termCSMS (dhkey, pkey).
```

Each of this event, placed in the model as reported in Figure 6.6, records a meaningful fact, in details:

- event acceptsCS(symk, xpkCSMS), which is used by the client to record the belief that she has accepted to run the protocol with the server B and the supplied symmetric key.
- event acceptsCSMS(symk, xcredentialsCS), which is used to record the fact that the server considers he has accepted to run the protocol with a client, with the proposed key supplied as the first argument and the client's public key as the second.
- event termCS(symk, credentialsCS), which means the client believes she has terminated a protocol run using the symmetric key supplied as the first argument and the client's public key as the second.
- event termCSMS(symk, pkCSMS), which denotes the server's belief that he has terminated a protocol run with the CS client correspondent to the credentials received, with the symmetric key supplied as the first argument.

The first verification to execute as a sanity check is the reachability, otherwise the sequent corrispondence queries would be meaningless.

```
query s:dhkey, k:pkey; event(acceptsCS(s, k)).
query s:dhkey, k:pkey; event(termCSMS(s, k)).
query s:dhkey, c:bitstring; event(acceptsCSMS(s, c)).
query s:dhkey, c:bitstring; event(termCS(s, c)).
```

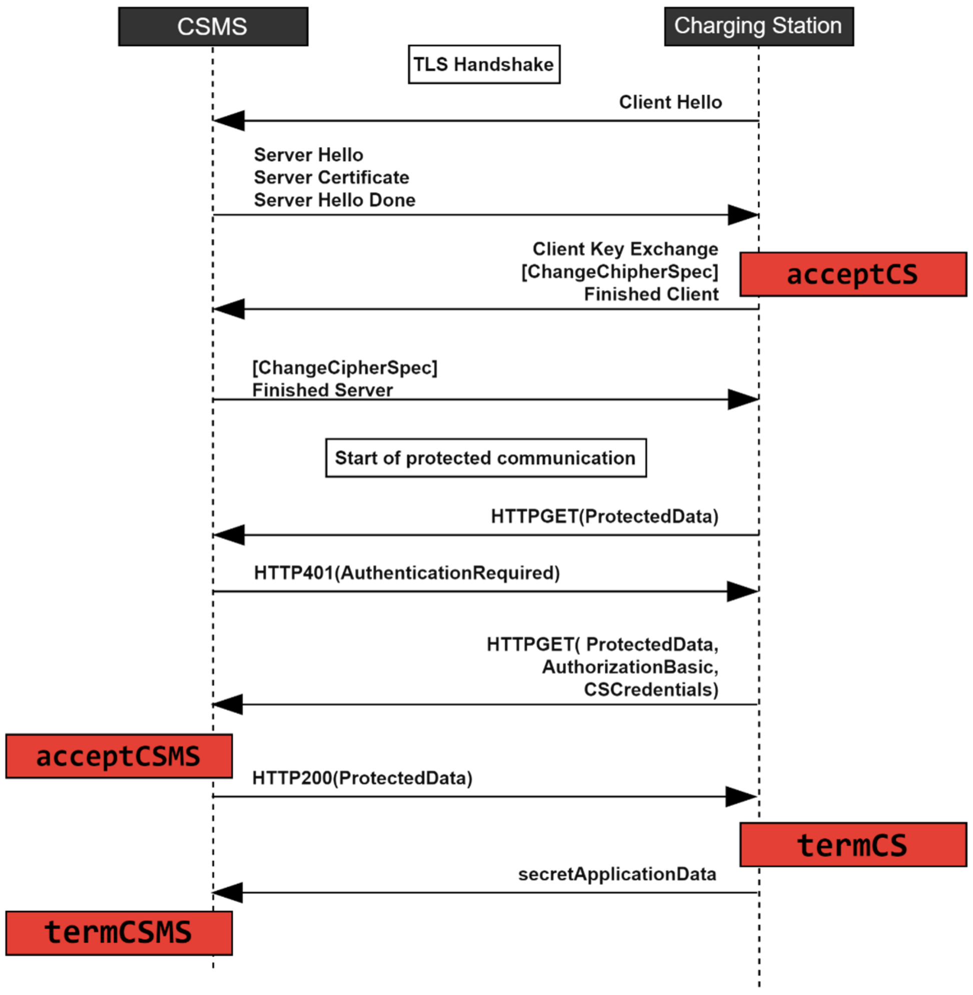


Figure 6.6: Sequence Diagram: Events Authentication Security Profile 2

It follows that verify the authentication of the CSMS, must hold the injective correspondence between termClient and acceptsServer events: if the client has terminated the protocol with the Server who owns the secret key correspondent to the public key, so the same CSMS Server process must have accepted the connection with the shared key.

```
query x : dhkey, k:pkey ; inj-event ( termCSMS( x, k ) ) ==> inj-
    event ( acceptsCS( x, k ) ) .
```

For verifying the authentication of the CS, must hold the injective correspondence between termClient and acceptsServer events: if the client has terminated the protocol with the Server who owns the secret key correspondent to the public key, so the same CSMS Server process must have accepted the connection with the shared key.

```
query x : dhkey, k:pkey ; inj-event ( termCS( x, k ) ) ==> inj-event
    ( acceptsCSMS( x, k ) ) .
```

Results are reported below.

Interpretation of the results:

```
Query not event(acceptsCS(s,k)) is false.
Query not event(termCSMS(s,k)) is false.
Query not event(acceptsCSMS(s,c_1)) is false.
Query not event(termCS(s,c_1)) is false.

Query inj-event(termCSMS(x_2,k)) ==> inj-event(acceptsCS(x_2,k)) is
    true.
Query inj-event(termCS(x_2,c_1)) ==> inj-event(acceptsCSMS(x_2,c_1))
    is true.
```

The results reflexts what was expected: False result for the reachability queries, which the prover translates in "not event(e())" means that exists a path which lead to the execution of the event, i.e. the event is reachable. A positive result for the sanity check. True result for the injective events queries: this means that for each event of process termination with the reported parameters of identification, the respective and only process initiated the exchange with the same parameters. A positive result for authentication verification.

6.3.2 Security Profile 3

Security profile 3 [subsection 5.4.2] requires both the Charging Station and the CSMS to authenticate with TLS Certificate. To verify the authentication in this section of the protocol, present for the reconnection after the Use Case A05 Update profile [section 5.2], some events are defined.

```
event acceptsCS2 (dhkey, pkey).
event acceptsCSMS2(dhkey, pkey).
event termCS2(dhkey, pkey).
event termCSMS2 (dhkey, pkey).
```

Each of this event, placed in the model as reported in Figure 6.7, records a meaningful fact, in details:

- event acceptsCS2(symk, xpkCSMS), which is used by the client to record the belief that she has accepted to run the protocol with the server B and the supplied symmetric key.
- event acceptsCSMS2(symk, xpkCSMS), which is used to record the fact that the server considers he has accepted to run the protocol with a client, with the proposed key supplied as the first argument and the client's public key as the second.
- event termCS2(symk, xpkCS), which means the client believes she has terminated a protocol run using the symmetric key supplied as the first argument and the client's public key as the second.
- event termCSMS2(symk, xpkCS), which denotes the server's belief that he has terminated a protocol run with the CS client correspondent to the public key received, with the symmetric key supplied as the first argument.

The first verification to execute, as a sanity check, is the reachability of the events, otherwise the sequent queries would be meaningless.

```
query s:dhkey, k:pkey; event(acceptsCS2(s, k)).
query s:dhkey, k:pkey; event(acceptsCSMS2(s, k)).

query s:dhkey, k:pkey; event(termCSMS2(s, k)).
query s:dhkey, k:pkey; event(termCS2(s, k)).
```

It follows that verify the authentication of the CSMS, must hold the injective correspondence between termClient and acceptsServer events: if the client has terminated the protocol with the Server who owns the secret key correspondent to the public key, so the same CSMS Server process must have accepted the connection with the shared key.

```
query x : dhkey, k:pkey ; inj-event ( termCSMS2( x, k ) ) ==> inj-
    event ( acceptsCS2( x, k ) ) .
```

For verifying the authentication of the CS, must hold the injective correspondence between termClient and acceptsServer events: if the client has terminated the protocol with the Server who owns the secret key correspondent to the public key, so the same CSMS Server process must have accepted the connection with the shared key.

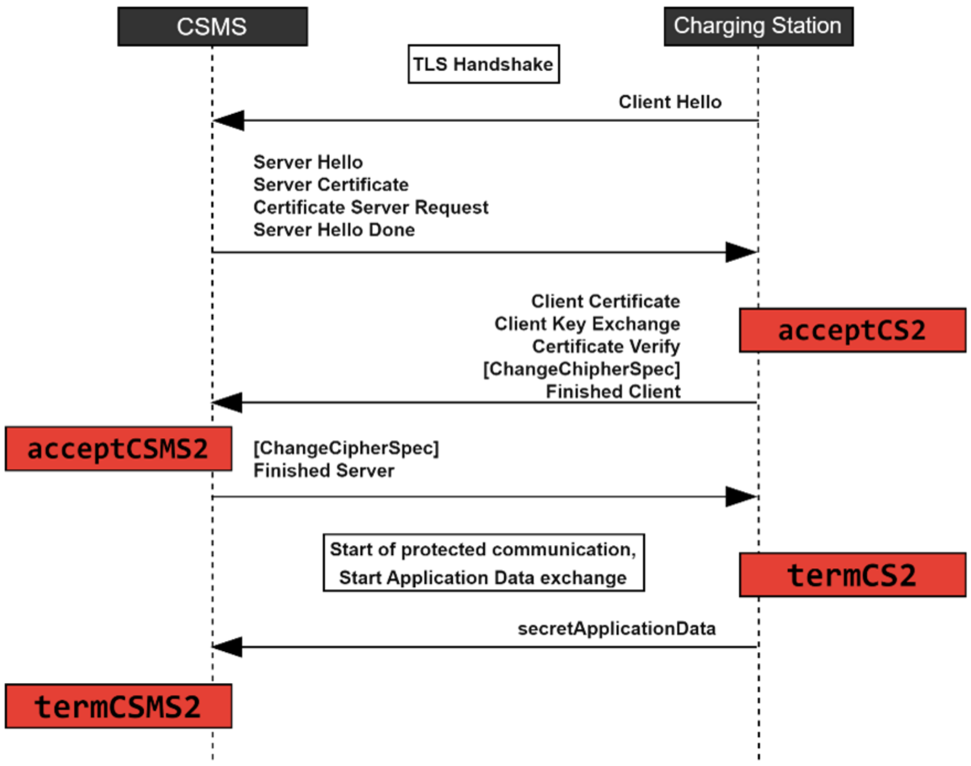


Figure 6.7: Sequence Diagram: Events Authentication Security Profile 3

```
query x : dhkey, k: pkey; inj-event ( termCS2( x , k ) ) ==> inj-
    event ( acceptsCSMS2( x, k ) ) .
```

Results are reported below.

Interpretation of the results:

```
Query not event(acceptsCS2(s,k)) is false.
Query not event(termCSMS2(s,k)) is false.
Query not event(acceptsCSMS2(s,k)) is false.
Query not event(termCS2(s,k)) is false.

Query inj-event(termCSMS2(x_3,k)) ==> inj-event(acceptsCS2(x_3,k)) is
    true.
Query inj-event(termCS2(x_3,k)) ==> inj-event(acceptsCSMS2(x_3,k)) is
    true.
```

The results for Security Profile 3 reflexts what desired: False result for the reachability queries means that exists a path which lead to the execution of the event, i.e. the event is reachable. A positive result for the sanity check. The true result for the injective events queries means that for each event of process termination with the reported parameters of identification, the respective and only process initiated the exchange with the same parameters. A positive result for authentication verification.

Chapter 7

Conclusions and Future Work

In summary, this thesis investigates the reasons for the widespread adoption of the OCPP protocol, which has led to its status as the de facto standard. A ProVerif model of the most relevant security-related section of the protocol has been developed, comprising use case messages, security profiles, a handshake for the authentication and cipher suite algorithms. The model has been tested against the ProVerif formal prover, verifying the security properties and providing an in-depth interpretation of the results for each modeled query. In conclusion, the ProVerif output demonstrates that the desired properties were verified and that the OCPP protocol ensures secure and trustworthy communication in the studied use cases.

For future work, this thesis aims to serve both as a starting point and as documentation. The verification of the selected use cases has demonstrated the absence of any attack presence, allowing the model to be analyzed using alternative verification tools. Alternatively, this work can also be regarded as a suitable point of departure for modelling and implementing other OCPP use cases. It includes a complete set of implementation choices, with each omission or assumption explained and justified, aiming to create a reliable ProVerif model of the selected use cases. This model may be expanded in the future to cover other sections of the protocol. This is particularly relevant given the ongoing evolution of the protocol, which may result in the identification of new security analysis targets for further investigation.

Bibliography

[1] Blanchet, Bruno & Smyth, Ben. (2011). ProVerif 1.85: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.

[2] Bruno Blanchet, Vincent Cheval ProVerif: Cryptographic protocol verifier in the formal model. `https://bblanche.gitlabpages.inria.fr/proverif/`

[3] Bruno Blanchet. (2016). Modeling and verifying security protocols with the applied pi calculus and ProVerif. Foundations and Trends in Privacy and Security 1, 1–2 (Oct. 2016), 1–135. `https://bblanche.gitlabpages.inria.fr/publications/BlanchetFnTPS16.pdf`

[4] Certicom Research, Standards for efficient cryptography, SEC 1: Elliptic Curve Cryptography, Version 2.0, May 21, 2009. `https://www.secg.org/sec1-v2.pdf`

[5] Cooper, D., et al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, Internet Engineering Task Force, Request for Comments 5280, May 2008. `http://www.ietf.org/rfc/rfc5280.txt`

[6] Cremers, C.J.F., Lafourcade, P., Nadeau, P. (2009). Comparing state spaces in automatic security protocol analysis. In: Cortier, V., Kirchner, C., Okada, M., Sakurada, H. (eds.) Formal to Practical Security. LNCS, vol. 5458, pp. 70–94. Springer, Heidelberg.

[7] Dierks, T. and Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2, Internet Engineering Task Force, Request for Comments 5246, August 2008. `http://www.ietf.org/rfc/rfc5246.txt`

[8] Dolev, D.; Yao, A. C. (1983). On the security of public key protocols. IEEE Transactions on Information Theory. `https://www.cs.huji.ac.il/~dolev/pubs/dolev-yao-ieee-01056650.pdf`

[9] European Commission, Directorate-General for Climate Action, Communication From The Commission To The European Parliament, The Council, The European Economic And Social Committee And The Committee Of The Regions. Stepping up Europe's 2030 climate ambition Investing in a climate-neutral future for the benefit of our people `https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:52020DC0562`

[10] European Commission – Speech by Commissioner Thierry Breton, Launch of the Route 35 platform. https://ec.europa.eu/commission/presscorner/detail/en/SPEECH_22_7785

[11] EVRoaming Foundation, Open Charge Point Interface 2.2.1-d2. https://evroaming.org/app/uploads/2024/04/OCPI-2.2.1-d2.pdf

[12] Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000. https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[13] Impressum, Datenschutz, Open Clearing House Protocol 1.4. https://www.ochp.eu/

[14] NIST, Computer Security Incident Handling Guide Recommendations of the National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf

[15] Open Charge Alliance, Open Charge Point Protocol 2.0.1. https://openchargealliance.org/protocols/open-charge-point-protocol/

[16] Open Charge Alliance, Open Smart Charging Protocol 2.0. https://openchargealliance.org/protocols/open-smart-charging-protocol/

[17] RFC 2617. HTTP Authentication: Basic and Digest Access Authentication. https://www.ietf.org/rfc/rfc2617.txt

[18] Sakharov, Alex. "Horn Clause." From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein. https://mathworld.wolfram.com/HornClause.html

[19] Sigrid de Vries, Lucie Mattera, From grids to vehicle charging experience: building a seamless e-mobility ecosystem. https://www.acea.auto/files/ACEA_ChargeUp_Europe_joint_declaration.pdf

Appendix A

Update Charging Station Password for HTTP Basic Authentication

```
(* VARIABLES AND TYPES *)

(*Public key cryptography*)
type pkey.      (* public key *)
type skey.      (* private key *)
type keymat.    (* key material *)

(*ECDHE*)
type G.     (* diffie-hellman Curve parameter*)
type dhkey. (* symmetric diffie-hellman key, symk*)
type sdhpar.(* secret diffie-hellman parameter, a,b  *)

(*Miscellanea*)
type result.    (* result of check signature *)
type nonce.
type suite.
type tlsversion.
type counter.
type profile.

(*Global variables*)
const SecProfile2: profile.
free c: channel.
free secretApplicationData: bitstring [private].

```

```
(* /VARIABLES AND TYPES *)

(* FUNCTIONS *)

(*Basic Functions*)

(* Pseudorandom TLS master secret computation *)
fun masterSecret( dhkey, nonce, nonce):dhkey.

(* Credentials management *)
free UsernameCS: bitstring.
free PasswordCS: bitstring [private].
fun credentialString( bitstring, bitstring): bitstring.
reduc forall username, password: bitstring ; credentialStringRet(
    credentialString( username, password)) = (username , password).

(*Message counter*)
free i_seed: counter.
fun next(counter):counter.

(* Asymmetric Encryption *)
fun pk(keymat): pkey.
fun sk(keymat): skey.
fun penc(bitstring, pkey): bitstring.
reduc forall x:bitstring, y:keymat; sdec(penc(x,pk(y)),sk(y)) = x.

(* Signature *)
fun ok():result.
fun sign(bitstring, skey): bitstring.
reduc forall m: bitstring, y: keymat; getmess(sign(m,sk(y))) = m.
reduc forall m: bitstring, y: keymat; checksign(sign(m, sk(y)), pk(y)
    ) = ok().

(* ECDHE Diffie-Hellman *)
(* const g: G.   DH Curve *)
fun dh(sdhpar, G): bitstring. (* A=a*G, B=b*G *)
fun sdh(bitstring, sdhpar): dhkey.    (* symk = B*a = A*b = a*b*G *)
equation forall a, b: sdhpar, g:G ; sdh(dh(b, g), a) = sdh(dh(a, g),
     b).
fun encdh(bitstring, dhkey): bitstring.
reduc forall m: bitstring, k: dhkey; decdh(encdh(m,k),k) = m.

(*Root CA  Certificate Signature*)
fun signCA(pkey, skey): bitstring.
reduc forall m: pkey, km_RA: keymat; checksignCA(signCA(m, sk(km_RA))
     , pk(km_RA)) = ok().
reduc forall k: pkey, km_RA: keymat; getpKey(signCA (k, sk(km_RA)),
    pk(km_RA)) = k.

```

```
(*/Basic Functions*)

(*Handshake messages*)

free Dumb:bitstring.
free ProtectedData: bitstring[private].
free AuthorizationBasic: bitstring.
free AuthenticationRequired: bitstring.
const HelloDone: bitstring.
const ServerFinished: bitstring.
const ClientFinished: bitstring.

fun HTTPGET( bitstring, bitstring, bitstring) : bitstring.
reduc forall PD, AB, UP: bitstring; HTTPGETret(HTTPGET(PD, AB, UP)) =
    (PD, AB, UP).

fun HTTP401( bitstring ): bitstring.
reduc forall m:bitstring; HTTP401ret(HTTP401(m)) = m.

fun HTTP200 ( bitstring ): bitstring.
reduc forall m: bitstring; HTTP200ret( HTTP200(m) ) = m .

(*/Handshake messages*)

(*A01 Messages - Update Password by CSMS Request*)

const SecurityCtrl: bitstring.
const BasicAuthPassword: bitstring.
const StatusAccepted: bitstring.

fun SetVariablesRequest(bitstring, bitstring, bitstring): bitstring.
reduc forall x, y, z: bitstring; SetVariablesRequestRet(
    SetVariablesRequest(x, y, z)) = (x, y, z).

fun SetVariablesResponse(bitstring):bitstring.
reduc forall x:bitstring; SerVariableResponseRet(SetVariablesResponse
    (x)) = x.

(*/A01 Messages - Update Password by CSMS Request*)

(* TESTS *)

(* Events *)

(* Test Secrecy *)

query attacker(secretApplicationData).
query attacker (PasswordCS).

```

```
(* Test Secret*)

query secret eph_k_CS.
query secret eph_k_CSMS.
query secret credentialsCS1.
query secret NewPasswordCS.
query secret newCredentialsCS.

(* Strong secrecy *)

noninterf secretApplicationData.
noninterf PasswordCS.

(* Session Correspondence OK *)

event beginCSsession(nonce, nonce).
event beginCSMSsession(nonce, nonce).
event endCSsession(nonce, nonce).
event endCSMSsession(nonce,nonce).

query n,m:nonce; event(beginCSsession(n,m)).
query n,m:nonce; event(beginCSMSsession(n,m)).
query n,m:nonce; event(endCSsession(n,m)).
query n,m: nonce; event(endCSMSsession(n,m)).

query  n,m:nonce; inj-event(endCSsession(n,m)) ==> inj-event(
    beginCSMSsession(n,m)).
query  n,m:nonce; inj-event(endCSMSsession(n,m)) ==> inj-event(
    beginCSsession(n,m)).

query  n,m:nonce; inj-event(endCSMSsession(n,m)) ==> inj-event(
    endCSsession(n,m)).

(* Test ECDHE parameters OK*)

event dhparCSMS( pkey, bitstring, bitstring).
event dhparCS( pkey, bitstring, bitstring).

query x:pkey, z,w:bitstring ; event(dhparCS(x,z,w)).
query x:pkey, z,w:bitstring ; event(dhparCSMS(x,z,w)).

query x:pkey, z,w:bitstring ; inj-event(dhparCS(x,z,w)) ==> inj-event
    (dhparCSMS(x,z,w)).

(* Test Authentication OK *)

event acceptsCS (dhkey, pkey).
event acceptsCSMS(dhkey, bitstring).
event termCSMS (dhkey, pkey).
event termCS(dhkey, bitstring).

query s:dhkey, k:pkey; event(acceptsCS(s, k)).
query s:dhkey, k:pkey; event(termCSMS(s, k)).
query s:dhkey, c:bitstring; event(acceptsCSMS(s, c)).
query s:dhkey, c:bitstring; event(termCS(s, c)).

query x : dhkey, k:pkey ; inj-event ( termCSMS( x, k ) ) ==> inj-
    event ( acceptsCS( x, k ) ) .
query x : dhkey, c: bitstring; inj-event ( termCS( x , c  ) ) ==> inj
    -event ( acceptsCSMS( x, c ) ) .
query x : dhkey, c: bitstring; inj-event ( termCS( x , c  ) ) ==> inj
    -event ( acceptsCSMS( x, c ) ) .

event acceptsCSMS2(dhkey, bitstring).
event termCSMS2 (dhkey, pkey).
event termCS2(dhkey, bitstring).

query s:dhkey, k:pkey; event(termCSMS2(s, k)).
query s:dhkey, c:bitstring; event(acceptsCSMS2(s, c)).
query s:dhkey, c:bitstring; event(termCS2(s, c)).

query x : dhkey, k:pkey ; inj-event ( termCSMS2( x, k ) ) ==> inj-
    event ( acceptsCS( x, k ) ) .
query x : dhkey, c: bitstring; inj-event ( termCS2( x , c  ) ) ==>
    inj-event ( acceptsCSMS2( x, c ) ) .

(* Test Message Order OK *)
(*CA-CSMS*)

event e00_ca(). event e01_csms().
query event(e00_ca()).
query event(e01_csms()).

query event (e01_csms()) ==> event (e00_ca()).
(*CS-CSMS (when both Authenticated) *)

event e01(). event e02(). event e03(). event e04().
event e05(). event e06(). event e07(). event e08(). event e09().

query event(e01()).
query event(e02()).
query event(e03()).
query event(e04()).
query event(e05()).
query event(e06()).
query event(e07()).
query event(e08()).
query event(e09()).

query event(e02()) ==> event(e01()).
query event(e03()) ==> event(e02()).
query event(e04()) ==> event(e03()).
query event(e05()) ==> event(e04()).
query event(e06()) ==> event(e05()).
query event(e08()) ==> event(e05()).
query event(e09()) ==> event(e08()).


(* /TESTS *)

(* PROCESSES *)

(* CSMS Server *)
let pCSMS(km_CSMS: keymat, pk_CA:pkey) =

let sec_profile = SecProfile2 in
let i=i_seed in

(* TLS with Basic Authentication *)



in(c, CSMS_CAsigned_certificate: bitstring);

let CSMS_CAsigned_certificate = sdec(CSMS_CAsigned_certificate,sk(
    km_CSMS)) in

if(checksignCA(CSMS_CAsigned_certificate, pk_CA) <> ok() ) then 0
else
event e01_csms();

in(c, (xnonce_CS: nonce,  xproposed_suite: suite, xtls_versio:
    bitstring, xsec_profile: profile, g:G));
if(xsec_profile <> sec_profile)then 0
else
let chosen_suite = xproposed_suite in
new nonce_CSMS: nonce;

event beginCSMSsession(nonce_CSMS, xnonce_CS);

let m1 = (nonce_CSMS, chosen_suite) in
out(c, m1 );

let m2 = ( CSMS_CAsigned_certificate ) in
out(c, m2);                          (* out Server Certificate *)

let m3 = HelloDone in
```

```
out(c, m3);                          (* out Hello Done *)

(* ECDHE, ClientKeyExchange *)

in(c, m: bitstring);

let xdhpar_CS = sdec(m, sk(km_CSMS)) in
new eph_k_CSMS: sdhpar;
let dhpar_CSMS = dh(eph_k_CSMS, g) in
let sympar = sdh(xdhpar_CS, eph_k_CSMS) in
let symk = masterSecret(sympar, xnonce_CS, nonce_CSMS) in

event dhparCSMS(pk(km_CSMS), dhpar_CSMS, xdhpar_CS);

out(c, sign( (xdhpar_CS, dhpar_CSMS), sk(km_CSMS)));

(* /ECDHE, ClientKeyExchange *)

in(c, m: bitstring);                 (* in ClientFinished *)

let  ( = xnonce_CS, =chosen_suite, =nonce_CSMS, =
    CSMS_CAsigned_certificate, =xsec_profile, xfinished:bitstring) =
    decdh( m, symk) in
if(xfinished<> ClientFinished) then 0
else
                                     (*out ServerFinished*)
out(c, encdh((xnonce_CS, xproposed_suite, nonce_CSMS,
    CSMS_CAsigned_certificate, sec_profile, i, ServerFinished), symk )
    );

(* ————————ChangeCipherSpec—————————— *)

in(c, m:bitstring);                  (* in HTTPGET (ProtectedData) *)
let i = next(i) in
let (=i, m:bitstring) = decdh((m),symk) in
let (pd:bitstring, ab:bitstring, up:bitstring) = HTTPGETret(m) in
if ( pd <> ProtectedData) then 0
else
let i=next(i) in
let msg = encdh( (i, AuthenticationRequired), symk) in

out(c, msg );                        (* out HTTP401 Authentication
    Required *)

in(c, m3: bitstring);

let i=next(i) in                     (* in HTTP GET /ProtectedData,
    Authorization Basic, Username/Password *)
let (=i, m1:bitstring) = decdh( m3, symk) in
let (pd: bitstring, ab: bitstring, up: bitstring) = HTTPGETret(m1) in
if( ( pd <> ProtectedData )  || ab <> AuthorizationBasic ) then 0
else
let (m1:bitstring, m2:bitstring)=credentialStringRet(up) in
if( m1 <> UsernameCS || m2<>PasswordCS ) then 0
else

event acceptsCSMS(symk, up);

event e02();
let i=next(i) in
let msg200 = encdh((i, ProtectedData), symk) in

out(c, msg200 );                       (* out HTTP200 Protected Data *)

in(c, x:bitstring);                     (* in secret *)

let i=next(i) in
let (=i, msg:bitstring)=decdh((x),symk) in
if ( msg <> secretApplicationData ) then 0
else
event e04();

event termCSMS (symk, pk(km_CSMS));

(* /TLS with Basic Authentication *)

(*A01*)
new NewPasswordCS: bitstring;
let newCredentialsCS=credentialString(UsernameCS,NewPasswordCS) in

let m = SetVariablesRequest( SecurityCtrl, BasicAuthPassword,
    NewPasswordCS ) in
let i = next(i) in

let enc_m = encdh( (i, m), symk) in

out (c, enc_m);                            (* out SetVariableRequest *)

in(c, m: bitstring);                       (* in SetVariableResponse *)
let i=next(i) in
let (=i, msg:bitstring) = (decdh( m, symk )) in
let msg= SerVariableResponseRet(msg) in
if( msg <> StatusAccepted ) then 0
else
event e06();

    (* ------ Reconnection with new Password (ChangeCipherSpec)
  -------- *)
```

```


in(c, m3: bitstring);
let i = next(i) in
let (=i, m: bitstring) = decdh( m3, symk) in    (*in HTTP GET /
    ProtectedData, Authorization Basic, Username/Password *)
let (pd: bitstring, ab: bitstring, up: bitstring) = HTTPGETret(m) in

if( ( pd <> ProtectedData )  || ( ab <> AuthorizationBasic ) || ( up
    <> newCredentialsCS )) then 0
else

event acceptsCSMS2(symk, newCredentialsCS);

event e07();
let i= next(i) in
let msg200 = encdh( (i, ProtectedData), symk) in

out(c, msg200 );                        (* out HTTP200 Protected Data *)

in(c, x:bitstring);                     (* in secret *)

let i=next(i) in
let (=i, msg:bitstring) = decdh(x,symk) in
if ( msg <> secretApplicationData ) then 0
else
event e09();
event termCSMS2 (symk, pk(km_CSMS));


event endCSMSsession(nonce_CSMS, xnonce_CS);


0.

(* /CSMS Server *)


(* CS Client *)

let pCS( pk_CA:pkey, pk_CSMS:pkey) =

let sec_profile = SecProfile2 in

(* TLS with Basic Authentication *)

new nonce_CS: nonce;
new proposed_suite: suite;
new tls_version: tlsversion;
new g: G;

out(c, (nonce_CS, proposed_suite, tls_version, sec_profile, g )); (*
    out Client Hello *)

in(c, (xnonce_CSMS: nonce, chosen_suite: suite)); (* in Server Hello
    *)

event beginCSsession(xnonce_CSMS, nonce_CS);

in(c, xserver_cert: bitstring);          (* in Server Certificate *)

in(c, HD:bitstring);                     (* in Hello Done *)

if ((checksignCA(xserver_cert, pk_CA) <> ok())) then 0
else
let xpk_CSMS = getpKey(xserver_cert, pk_CA) in

(*ECDHE, ClientKeyExchange*)

new eph_k_CS: sdhpar;
let dhpar_CS = dh(eph_k_CS, g) in

out(c, penc( dhpar_CS, xpk_CSMS));

in(c, y:bitstring);
if checksign( y, xpk_CSMS ) <> ok() then 0
else
let (=dhpar_CS,  xdhpar_CSMS: bitstring) = getmess(y) in
let sympar = sdh(xdhpar_CSMS , eph_k_CS) in
let symk= masterSecret(sympar, nonce_CS, xnonce_CSMS) in

event dhparCS(xpk_CSMS, xdhpar_CSMS, dhpar_CS);

event acceptsCS (symk, xpk_CSMS);

(*/ECDHE, ClientKeyExchange*)

out(c, encdh( (nonce_CS, chosen_suite, xnonce_CSMS, xserver_cert,
    sec_profile, ClientFinished), symk));
            (*out ClientFinished*)

in(c, m: bitstring);              (*in ServerFinished*)

let ( = nonce_CS, =chosen_suite, =xnonce_CSMS, =xserver_cert, =
    sec_profile, xi:counter, xfinished:bitstring) = decdh(m,symk) in

if( xfinished <> ServerFinished) then 0
else
```

```

(*——————ChangeCipherSpec————————————————*)

let msg = HTTPGET(ProtectedData, Dumb, Dumb) in
let xi = next(xi)in
let m = encdh( (xi, msg) , symk) in
out(c, m);                          (*out HTTP GET ProtectedData*)

in(c, m: bitstring);               (* in HTTP401 Authentication
    Required*)

let xi = next(xi) in
let (=xi, m1:bitstring)= decdh(m,symk) in

if( m1 <> AuthenticationRequired)  then 0
else

event e01();
let xi = next(xi) in
let credentialsCS1= credentialString(UsernameCS,PasswordCS) in
let m3 = encdh((xi,HTTPGET(ProtectedData, AuthorizationBasic,
    credentialsCS1 )), symk) in

out(c, m3);                        (*out  UsernamePasswordCS*)

in(c, m: bitstring);                (* in HTTP200 Protected Data*)

let xi = next(xi) in
let (=xi, m1:bitstring) = decdh(m, symk) in
if( m1 <> ProtectedData  )  then 0
else
event e03();

if(pk_CSMS <> xpk_CSMS) then 0 else

event termCS(symk, credentialsCS1);

let xi = next(xi) in
out(c, encdh((xi,secretApplicationData), symk)); (* out secret*)

(*/TLS with Basic Authentication*)

(*A01*)

in (c, m:bitstring);            (* in SetVariableRequest *)

let xi = next(xi) in
let ( = xi, msg:bitstring) = decdh (m, symk) in
let (sc: bitstring, bap: bitstring, xnewPassword: bitstring) =
    SetVariablesRequestRet(msg) in

if(sc <> SecurityCtrl || bap <> BasicAuthPassword ) then 0
else
event e05();

let newCredentialsCS=credentialString(UsernameCS,xnewPassword) in

let xi = next(xi) in
let m = encdh((xi,SetVariablesResponse(StatusAccepted)), symk) in
out(c, m);                         (* out SetVariableResponse *)

(*—————————————Disconnect——————————————————————*)

    (*———————Reconnection with new Password (ChangeCipherSpec)
    ——————————*)

let xi = next(xi) in
let m3 = encdh((xi, HTTPGET(ProtectedData, AuthorizationBasic,
    newCredentialsCS)), symk) in

out(c, m3);                        (* out HTTP GET /ProtectedData,
    Authorization Basic, UsernamePassword*)

in(c, m: bitstring);               (* in HTTP200 Protected Data*)
let xi=next(xi) in
let (=xi, m1:bitstring) = decdh(m, symk) in
if( m1 <> ProtectedData  )  then 0
else
event e08();

event termCS2(symk, newCredentialsCS);

let xi=next(xi) in

let m = encdh((xi,secretApplicationData), symk) in

event endCSsession(xnonce_CSMS, nonce_CS);

let xi = next(xi) in
out(c, (xi,m));                    (* out secret*)

    0.

(*/CS Client*)

(*CERTIFICATE AUTHORITY, trusted root CA*)
let pCA(km_CA:keymat, pk_CSMS:pkey) =

event e00_ca();
out  ( c, penc(signCA( pk_CSMS, sk(km_CA)), pk_CSMS)) ;
     0.

process
new km_CSMS: keymat;
new km_CA:keymat;

let pk_CSMS = pk(km_CSMS) in out(c, pk_CSMS);
let pk_CA=  pk(km_CA)in out(c, pk_CA);

((!pCSMS(km_CSMS, pk_CA))) | (!pCS(  pk_CA, pk_CSMS) | (!pCA(km_CA,
    pk_CSMS)))
```

Appendix B

Upgrade Charging Station Security Profile

```
(* Update CS Security Profile by CSMS request *)

(* VARIABLES AND TYPES *)

(*Public key cryptography*)
type pkey.       (* public key *)
type skey.       (* private key *)
type keymat.     (* key material *)

(*ECDHE*)
type G. (* diffie-hellman Curve parameter*)
type dhkey. (* symmetric diffie-hellman key, symk*)
type sdhpar.     (* secret diffie-hellman parameter, a,b  *)

(*Miscellanea*)
type result.
type nonce.
type suite.
type tlsversion.
type counter.
type profile.

(*Global variables*)
free c: channel.    (* public channel *)
free secretApplicationData: bitstring [private].    (* secr data *)
free ack: bitstring.

(* /VARIABLES AND TYPES *)

(* FUNCTIONS *)

(*Basic Functions*)

(* Pseudorandom TLS master secret computation *)
fun masterSecret( dhkey, nonce, nonce):dhkey.

(* Credentials management *)
free UsernameCS:bitstring.
free PasswordCS:bitstring[private].
fun credentialString( bitstring, bitstring): bitstring.
reduc forall username, password: bitstring ; credentialStringRet(
    credentialString( username, password)) = (username , password).

(*Message counter*)
free i_seed: counter.
fun next(counter):counter.

(* Asymmetric Encryption *)
fun pk(keymat): pkey.
fun sk(keymat): skey.
fun penc(bitstring, pkey): bitstring.
reduc forall x:bitstring, y:keymat; sdec(penc(x,pk(y)),sk(y)) = x.

(* Signature *)
fun ok():result.
fun sign(bitstring, skey): bitstring.
reduc forall m: bitstring, y: keymat; getmess(sign(m,sk(y))) = m.
reduc forall m: bitstring, y: keymat; checksign(sign(m, sk(y)), pk(y)
    ) = ok().

(* ECDHE Diffie-Hellman *)
(*  const g: G.  DH Curve *)

fun dh(sdhpar, G): bitstring. (* A=a*G, B=b*G  *)
fun sdh(bitstring, sdhpar): dhkey.    (* symk = B*a = A*b = a*b*G *)
equation forall a, b: sdhpar, g:G ; sdh(dh(b, g), a)  = sdh(dh(a, g),
     b).
fun encdh(bitstring, dhkey): bitstring.
reduc forall m: bitstring, k: dhkey; decdh(encdh(m,k),k) = m.

(*Root CA  Certificate Signature*)
fun signCA(pkey, skey): bitstring.
reduc forall m: pkey, km_RA: keymat; checksignCA(signCA(m, sk(km_RA))
    , pk(km_RA)) = ok().
reduc forall k: pkey, km_RA: keymat; getpKey(signCA (k, sk(km_RA)),
    pk(km_RA)) = k.

```

```
(*Upgrade Security profile check*)
const SecProfile2: profile.

fun upgrade(profile): profile.
reduc forall s:profile; isupgraded(s, upgrade(s)) = ok().

(*/Basic Functions*)
(*Handshake messages*)

free Dumb:bitstring.
free ProtectedData: bitstring[private].
free AuthorizationBasic: bitstring.
free AuthenticationRequired: bitstring.
const HelloDone: bitstring.
const ClientFinished: bitstring.
const ServerFinished: bitstring.

fun HTTPGET( bitstring, bitstring, bitstring) : bitstring.
reduc forall PD, AB, UP: bitstring; HTTPGETret(HTTPGET(PD, AB, UP)) =
     (PD, AB, UP).

fun HTTP401( bitstring ): bitstring.
reduc forall m:bitstring; HTTP401ret(HTTP401(m)) = m.

fun HTTP200 ( bitstring ): bitstring.
reduc forall m: bitstring; HTTP200ret( HTTP200(m) ) = m .

(*/Handshake messages*)

(* A05 Messages - Update Security Profile by CSMS Request *)
free StatusAccepted: bitstring.
free newProfile: bitstring.
free Interval: bitstring.
free Id_Cs:bitstring.

const SecurityCtrl: bitstring.
const NetworkConfigurationPriority: bitstring.
const ChangeNetworkConfig: bitstring.
const RebootRequired: bitstring.
const OnIdle: bitstring.
const PowerUp: bitstring.
const RegistrationType: bitstring.
const CertificateServerRequest: bitstring.

fun SetVariablesRequest(bitstring, bitstring, profile): bitstring.
reduc forall x, y: bitstring, z:profile; SetVariablesRequestRet(
    SetVariablesRequest(x, y, z)) = (x, y, z).

fun SetVariablesResponse(bitstring):bitstring.
```

```
reduc forall x: bitstring; SerVariableResponseRet(
    SetVariablesResponse(x)) = x.

fun ResetRequest(bitstring):bitstring.
reduc forall x: bitstring; ResetRequestRet(ResetRequest(x)) = x.

fun ResetResponse(bitstring):bitstring.
reduc forall x: bitstring; ResetResponseRet(ResetResponse(x)) = x.

fun BootNotificationRequest(bitstring, bitstring):bitstring.
reduc forall x, y: bitstring; BootNotificationRequestRet(
    BootNotificationRequest(x, y)) = (x, y).

fun BootNotificationResponse(bitstring, bitstring):bitstring.
reduc forall x, y: bitstring; BootNotificationResponseRet(
    BootNotificationResponse(x, y)) = (x, y).

(* /A05 Messages - Update Security Profile by CSMS Request *)

(*/FUNCTIONS*)

(* TESTS *)

(* Test Secrecy *)

query attacker(secretApplicationData).
query attacker (PasswordCS).

query secret eph_k_CS.
query secret eph_k_CSMS.

query secret eph_k_CS2.
query secret eph_k_CSMS2.
query secret symk2.

(* Strong secrecy *)

noninterf secretApplicationData.
noninterf PasswordCS.

(* Test Session  *)

(*Security Profile 2*)

event beginCSsession(nonce, nonce).
event beginCSMSsession(nonce, nonce).

event endCSsession(nonce, nonce).
event endCSMSsession(nonce, nonce).
```

```

query n,m: nonce ; event(beginCSsession(n, m)).
query n,m: nonce ; event(beginCSMSsession(n, m)).

query n,m: nonce ; event(endCSsession(n, m)).
query n,m: nonce ; event(endCSMSsession(n, m)).  (* *)

query n,m: nonce ; inj-event(endCSsession(n ,m)) ==> inj-event(
    beginCSMSsession(n, m)).
query n,m: nonce ; inj-event(endCSMSsession(n, m)) ==> inj-event(
    beginCSsession(n, m)).

query n,m: nonce ; inj-event(endCSMSsession(n, m)) ==> inj-event(
    endCSsession(n, m)).

(*Security Profile 3*)

event beginCSsession2(nonce, nonce).
event beginCSMSsession2(nonce, nonce).

event endCSsession2(nonce, nonce).
event endCSMSsession2(nonce, nonce).

query n,m: nonce ; event(beginCSsession2(n, m)).
query n,m: nonce ; event(beginCSMSsession2(n, m)).

query n,m: nonce ; event(endCSsession2(n, m)).
query n,m: nonce ; event(endCSMSsession2(n, m)). (*  *)

query n,m: nonce ; inj-event(endCSsession2(n ,m)) ==> inj-event(
    beginCSMSsession2(n, m)).
query n,m: nonce ; inj-event(endCSMSsession2(n, m)) ==> inj-event(
    beginCSsession2(n, m)).

query n,m: nonce ; inj-event(endCSMSsession2(n, m)) ==> inj-event(
    endCSsession2(n, m)).

(* Test ECDHE parameters correspondence *)

event dhparCSMS(pkey,bitstring, bitstring).
event dhparCS(pkey,bitstring, bitstring).

event dhparCSMS2(pkey, bitstring, bitstring).
event dhparCS2(pkey, bitstring, bitstring).

query x: pkey, z,w: bitstring ; event(dhparCS(x, z, w)). (*false*)
query x: pkey, z,w: bitstring ; event(dhparCSMS(x, z, w)). (*false*)
query x: pkey, z,w: bitstring ; inj-event(dhparCS(x, z, w)) ==> inj-
    event(dhparCSMS(x, z, w)). (*true*)
```

```

query x: pkey, z,w: bitstring ; event(dhparCS2(x, z, w)). (*false*)
query x: pkey, z,w: bitstring ; event(dhparCSMS2(x, z, w)). (*false*)
query x: pkey, z,w: bitstring ; inj-event(dhparCS2(x, z, w)) ==> inj-
    event(dhparCSMS2(x, z, w)). (*true*)

(*Test Authentication *)

(*Security Profile 2*)

event acceptsCS (dhkey, pkey).
event acceptsCSMS(dhkey, bitstring).
event termCSMS (dhkey, pkey).
event termCS(dhkey, bitstring).

query s:dhkey, k:pkey; event(acceptsCS(s, k)).
query s:dhkey, k:pkey; event(termCSMS(s, k)).
query s:dhkey, c:bitstring; event(acceptsCSMS(s, c)).
query s:dhkey, c:bitstring; event(termCS(s, c)).

query x : dhkey, k:pkey ; inj-event ( termCSMS( x, k ) ) ==> inj-
    event ( acceptsCS( x, k ) ) .
query x : dhkey, c: bitstring; inj-event ( termCS( x , c  ) ) ==> inj
    -event ( acceptsCSMS( x, c ) ) .
query x : dhkey, c: bitstring; inj-event ( termCS( x , c  ) ) ==> inj
    -event ( acceptsCSMS( x, c ) ) .

(*  Security Profile 3*)
event acceptsCS2 (dhkey, pkey).
event acceptsCSMS2(dhkey, pkey).
event termCSMS2 (dhkey, pkey).
event termCS2(dhkey, pkey).

query s:dhkey, k:pkey; event(acceptsCS2(s, k)).
query s:dhkey, k:pkey; event(termCSMS2(s, k)).
query s:dhkey, k:pkey; event(acceptsCSMS2(s, k)).
query s:dhkey, k:pkey; event(termCS2(s, k)).

query x : dhkey, k:pkey ; inj-event ( termCSMS2( x, k ) ) ==> inj-
    event ( acceptsCS2( x, k ) ) .
query x : dhkey, k: pkey; inj-event ( termCS2( x , k  ) ) ==> inj-
    event ( acceptsCSMS2( x, k ) ) .

(*   Test Message Order *)

(*CA-CSMS*)

event e01_ca().
event e02_ca().
```

```
query event (e01_ca()).
query event (e02_ca()).
query event (e01()) ==> event (e01_ca()).
query event (e11()) ==> event (e02_ca()).
    (* CSO-CSMS*)

event e05_cso().
query event (e05_cso()).
query event (e05()) ==> event (e05_cso()).

(* CS-CSMS (when both Authenticated) *)

event e01(). event e02(). event e03(). event e04(). event e05().
    event e06().
event e07(). event e08(). event e09(). event e10(). event e11().
    event e12().
event e13(). event e14(). event e15(). event e16(). event e17().

query event(e01()).
query event(e02()).
query event(e03()).
query event(e04()).
query event(e05()).
query event(e06()).
query event(e07()).
query event(e08()).
query event(e09()).
query event(e10()).
query event(e11()).
query event(e12()).
query event(e13()).
query event(e14()).
query event(e15()).
query event(e16()).
query event(e17()).

query event(e02()) ==> event(e01()).
query event(e03()) ==> event(e02()).
query event(e04()) ==> event(e03()).
query event(e05()) ==> event(e04()).
query event(e06()) ==> event(e05()).
query event(e07()) ==> event(e06()).
query event(e08()) ==> event(e07()).
query event(e09()) ==> event(e08()).
query event(e10()) ==> event(e09()).
query event(e11()) ==> event(e10()).
query event(e12()) ==> event(e11()).
query event(e13()) ==> event(e12()).
query event(e14()) ==> event(e13()).
query event(e15()) ==> event(e14()).
query event(e16()) ==> event(e15()).
query event(e17()) ==> event(e16()).

(* /TESTS *)

(* PROCESSES *)

(* CSMS Server *)
let pCSMS(km_CSMS: keymat, pk_CA:pkey, pk_CSO:pkey) =

let sec_profile = SecProfile2 in
let i=i_seed in

(* TLS with Basic Authentication *)

in(c, CSMS_CAsigned_certificate: bitstring); (*in CSMS certificate
    from CA*)
let CSMS_CAsigned_certificate = sdec(CSMS_CAsigned_certificate,sk(
    km_CSMS)) in

if( (checksignCA(CSMS_CAsigned_certificate, pk_CA) <> ok()) || (
    getpKey(CSMS_CAsigned_certificate, pk_CA)<> pk(km_CSMS)) ) then 0
else

out(c, ack);
event e01();
                                    (* in Client Hello *)
in(c, (xnonce_CS: nonce, xproposed_suite: suite, xtls_version:
    bitstring, xsec_profile: profile, g:G));

if(xsec_profile <> sec_profile)then 0
else

let chosen_suite = xproposed_suite in
new nonce_CSMS: nonce;

event beginCSMSsession(nonce_CSMS, xnonce_CS);
let m1 = (nonce_CSMS, chosen_suite) in
out(c, m1 );                        (* out Server Hello *)

let m2 = ( CSMS_CAsigned_certificate ) in
out(c, m2);                         (* out Server Certificate *)

let m3 = HelloDone in
out(c, m3);                         (* out Hello Done *)

(* ECDHE, ClientKeyExchange *)

```

```
in(c, m: bitstring);

let xdhpar_CS = sdec(m, sk(km_CSMS)) in
new eph_k_CSMS: sdhpar;
let dhpar_CSMS = dh(eph_k_CSMS, g) in
let sympar = sdh(xdhpar_CS, eph_k_CSMS) in
let symk = masterSecret(sympar, xnonce_CS, nonce_CSMS) in

event dhparCSMS(pk(km_CSMS), dhpar_CSMS, xdhpar_CS);

out(c, sign( (xdhpar_CS, dhpar_CSMS), sk(km_CSMS)));

(* /ECDHE, ClientKeyExchange *)

in(c, m: bitstring);            (* in ClientFinished *)
let ( = xnonce_CS, =chosen_suite, =nonce_CSMS, =
    CSMS_CAsigned_certificate, =xsec_profile, xfinished2:bitstring)=
    decdh(m, symk) in
if(xfinished2 <> ClientFinished) then 0
else
                                (*out ServerFinished*)
out(c, encdh((xnonce_CS, xproposed_suite, nonce_CSMS,
    CSMS_CAsigned_certificate, xsec_profile, i, ServerFinished), symk
    ));

(* ---------ChangeCipherSpec-------------- *)

in(c, m:bitstring);                (* in HTTPGET (ProtectedData) *)
let i = next(i) in
let (=i, m:bitstring) = decdh((m),symk) in
let (pd:bitstring, ab:bitstring, up:bitstring) = HTTPGETret(m) in
if ( pd <> ProtectedData) then 0
else
let i=next(i) in
let msg = encdh( (i, AuthenticationRequired), symk) in

out(c, msg );                      (* out HTTP401 Authentication
    Required *)

in(c, m3: bitstring);

let i=next(i) in                   (* in HTTP GET /ProtectedData,
    Authorization Basic, Username/Password *)
let (=i, m1:bitstring) = decdh( m3, symk) in
let (pd: bitstring, ab: bitstring, up: bitstring) = HTTPGETret(m1) in
if( ( pd <> ProtectedData ) || ab <> AuthorizationBasic ) then 0
else
let (m1:bitstring, m2:bitstring)=credentialStringRet(up) in
if( m1 <> UsernameCS || m2<>PasswordCS ) then 0
```

```
else

event acceptsCSMS(symk, up);

event e02();
let i=next(i) in
let msg200 =encdh((i, ProtectedData), symk) in

out(c, msg200 );                    (* out HTTP200 Protected Data *)

in(c, x:bitstring);                 (* in secr *)

let i=next(i) in
let (=i, msg:bitstring)=decdh((x),symk) in
if ( msg <> secretApplicationData ) then 0
else
event e04();
event endCSMSsession(nonce_CSMS, xnonce_CS);
event termCSMS (symk, pk(km_CSMS));

(* /TLS with Basic Authentication *)

(*A05*)

in(c, msg: bitstring);              (*in ChangeNetworkConfig from CSO
    *)
let msg=sdec(msg, sk(km_CSMS)) in
if (checksign(msg, pk_CSO)<>ok()) then 0
else
let (cnf: bitstring, xsec_profile_cso: profile)= (getmess(msg)) in
if( cnf <> ChangeNetworkConfig  || ( isupgraded(sec_profile,
    xsec_profile_cso) <> ok()) ) then 0
else
event e05();
let sec_profile=xsec_profile_cso in

let m = SetVariablesRequest( SecurityCtrl,
    NetworkConfigurationPriority, sec_profile ) in
let i=next(i) in
let enc_m = encdh((i,m), symk) in

out (c, enc_m);                     (* out SetVariableRequest *)

in(c, m: bitstring);                (* in SetVariableResponse *)
let i=next(i) in
let (=i, m: bitstring) = decdh( m, symk ) in
let msg= SerVariableResponseRet(m) in

if( msg <> RebootRequired ) then 0
```

```
else
event e07();

let i=next(i) in
let m = encdh((i,ResetRequest(OnIdle)), symk) in

out (c, m);                          (* out ResetRequest OnIdle *)

in(c, m: bitstring);                 (* in SetVariableResponse *)
let i=next(i) in
let (=i, m:bitstring) = (decdh( m, symk )) in
let msg=ResetResponseRet(m) in

if( msg <> StatusAccepted ) then 0
else
event e09();

(*−−−−−−−−−−−−CS Reboots −−−−−−−−−−−−*)

(*−−−−−−Connection with new Security Profile 3 −−−−−−−−*)

(* TLS with Client Side Certificates *)
                                  (* in Client Hello *)
in(c, (xpk_CSMS:pkey, xnonce_CS2: nonce, xproposed_suite2: suite,
    xtls_version2:bitstring, xsec_profile_cs:profile, g2:G));

if( (xpk_CSMS<> pk(km_CSMS)) || (xsec_profile_cs<> xsec_profile))
    then 0
else
event e10();
let chosen_suite2 = xproposed_suite2 in
 new nonce_CSMS2: nonce;

event beginCSMSsession2(nonce_CSMS2, xnonce_CS2);

let m1 = (nonce_CSMS2, chosen_suite2) in
out(c, m1 );                        (* out Server Hello *)

let m2 = ( CertificateServerRequest, CSMS_CAsigned_certificate ) in
out(c, m2);                         (* out Server Certificate *)

let m3 = sign(HelloDone, sk(km_CSMS)) in
out(c, m3);                         (* out Hello Done *)

in(c, client_cert:bitstring);   (*in Client Certificate from CS*)
let client_cert=sdec(client_cert, sk(km_CSMS)) in
if( checksignCA(client_cert, pk_CA) <> ok() ) then 0
else
let xpk_CS = getpKey(client_cert, pk_CA) in
```

```

(* ECDHE, ClientKeyExchange *)

in(c, m: bitstring);          (*in DH parameter sec_prof_3*)
if(checksign(m, xpk_CS)<>ok()) then 0
else
let (xpk_CSMS:pkey, xdhpar_CS2:bitstring) = sdec(getmess(m), sk(
    km_CSMS)) in
if(xpk_CSMS<> pk(km_CSMS)) then 0
else
new eph_k_CSMS2: sdhpar;
let dhpar_CSMS2 = dh(eph_k_CSMS2, g2) in
let sympar2 = sdh(xdhpar_CS2, eph_k_CSMS2) in
let symk2 = masterSecret(sympar2, xnonce_CS2, nonce_CSMS2) in

event dhparCSMS2(pk(km_CSMS), dhpar_CSMS2, xdhpar_CS2);

event acceptsCSMS2(symk2, xpk_CS);

out(c, sign( (xpk_CS, xdhpar_CS2, dhpar_CSMS2), sk(km_CSMS)));

(* /ECDHE, ClientKeyExchange *)

in(c, m: bitstring);    (* in ClientFinished *)

let( = xnonce_CS2, =chosen_suite2, =nonce_CSMS2, =
    CSMS_CAsigned_certificate, =sec_profile, xfinished2:bitstring)=
    decdh(m, symk2) in
if( xfinished2 <> ClientFinished) then 0
else
event e12();

                              (*out ServerFinished*)
out(c,  encdh((xnonce_CS2, xproposed_suite2, nonce_CSMS2,
    CSMS_CAsigned_certificate, sec_profile, i, ServerFinished), symk2
    ));

(* ————————ChangeCipherSpec———————— *)

in(c, x:bitstring);            (* in ApplicationData (secret ) *)

let i=next(i) in
let (=i, msg:bitstring)=decdh(x,symk2) in
if ( msg <> secretApplicationData ) then 0
else
event e14();

event endCSMSsession2(nonce_CSMS2, xnonce_CS2);

event termCSMS2 (symk2, pk(km_CSMS));

(* /TLS with Client Side Certificates *)

(*-----/Connection with new Security Profile 3 ---------------*)

in(c, m: bitstring);          (* in BootNotificationRequest *)
let i=next(i) in
let(=i, m:bitstring)=decdh(m,symk) in
let (pu:bitstring, xid_cs:bitstring)= BootNotificationRequestRet(m)
    in

if ( pu <> PowerUp ) then 0
else
event e15();
let i=next(i) in
let m = encdh((i,BootNotificationResponse(RegistrationType, Interval)
    ), symk) in

out(c, m);                     (* out BootNotificationResponse *)

(*/A05*)

in(c, x:bitstring);            (* in secret *)

let i=next(i) in
let (=i, msg:bitstring) = decdh(x,symk) in
if ( msg <> secretApplicationData ) then 0
else
event e17();
              0.

(* /CSMS Server *)


(* CS Client *)

let pCS( km_CS:keymat, pk_CA:pkey) =

let sec_profile = SecProfile2 in

(* TLS with Basic Authentication *)

new nonce_CS: nonce;
new proposed_suite: suite;
new tls_version: tlsversion;
new g:G;

```

```
out(c, (nonce_CS, proposed_suite, tls_version, sec_profile, g)); (*
    out Client Hello *)

in(c, (nonce_CSMS: nonce, chosen_suite: suite)); (* in Server Hello
    *)

event beginCSsession(nonce_CSMS, nonce_CS);

in(c, server_cert: bitstring);   (* in Server Certificate *)

in(c, HD: bitstring);            (* in Hello Done *)

if ((checksignCA(server_cert, pk_CA) <> ok())) then 0
else
let xpk_CSMS = getpKey(server_cert, pk_CA) in

(*ECDHE, ClientKeyExchange*)

new eph_k_CS: sdhpar;
let dhpar_CS = dh(eph_k_CS, g) in

out(c, penc( dhpar_CS, xpk_CSMS));      (*out DH parameter*)

in(c, y:bitstring);
if checksign( y, xpk_CSMS ) <> ok() then 0
else
let (=dhpar_CS, xdhpar_CSMS: bitstring) = getmess(y) in
let sympar = sdh(xdhpar_CSMS , eph_k_CS) in
let symk = masterSecret(sympar, nonce_CS, nonce_CSMS) in

event dhparCS(xpk_CSMS, xdhpar_CSMS, dhpar_CS);
event acceptsCS (symk, xpk_CSMS);

(*/ECDHE, ClientKeyExchange*)

out(c, encdh(( nonce_CS, chosen_suite, nonce_CSMS, server_cert,
    sec_profile, ClientFinished), symk)); (*out ClientFinished*)

in(c, m: bitstring);       (*in ServerFinished*)

let  ( = nonce_CS, =chosen_suite, =nonce_CSMS, =server_cert, =
    sec_profile, xi:counter, xfinished:bitstring) = decdh(m,symk) in

if( xfinished <> ServerFinished) then 0
else

(*----------ChangeCipherSpec--------------------*)

let msg = HTTPGET(ProtectedData, Dumb, Dumb) in
let xi = next(xi)in
let m = encdh( (xi, msg) , symk) in
out(c, m);                    (*out HTTP GET ProtectedData*)

in(c, m: bitstring);          (* in HTTP401 Authentication Required*)

let xi = next(xi) in
let (=xi, m1:bitstring)= decdh(m,symk) in

if( m1 <> AuthenticationRequired)  then 0
else
let xi = next(xi) in
let credentialStringCS = credentialString(UsernameCS,PasswordCS) in
let m3 = encdh((xi,HTTPGET(ProtectedData, AuthorizationBasic,
    credentialStringCS )), symk) in

 out(c, ( m3));          (*out HTTP GET /ProtectedData, Authorization
    Basic, UsernamePassword*)

in(c, m: bitstring);  (* in HTTP200 Protected Data*)

let xi = next(xi) in
let (=xi, m1:bitstring) = decdh(m, symk) in
if( m1 <> ProtectedData  )  then 0
else
event e03();
event endCSsession(nonce_CSMS, nonce_CS);
event termCS(symk, credentialStringCS);

let xi = next(xi) in
out(c, encdh((xi,secretApplicationData), symk));  (* out secret*)

(*/TLS with Basic Authentication*)

(*A05*)

in (c, m:bitstring);          (* in SetVariableRequest *)

let xi = next(xi) in
let (=xi, m:bitstring) = decdh (m, symk) in
let (sc: bitstring, ncp: bitstring, xsec_profile: profile)=
    SetVariablesRequestRet (m) in

if( sc <> SecurityCtrl  || ncp <> NetworkConfigurationPriority  || (
     isupgraded(sec_profile, xsec_profile) <> ok())) then 0
else
let xi = next(xi) in
let m=SetVariablesResponse(RebootRequired) in
let m = encdh((xi, m), symk) in

out(c, m);                    (* out SetVariableResponse *)

in(c, m:bitstring);           (* in ResetRequest OnIdle *)

let xi = next(xi) in
let (=xi, m:bitstring) = decdh(m, symk) in
let m= ResetRequestRet(m) in
if( m <> OnIdle) then 0
else
event e08();
let xi = next(xi) in
let m = encdh((xi,ResetResponse(StatusAccepted)), symk)  in
out(c, m);                    (* out ResetResponse StatusAccepted *)

(*--------------Reboot----------------*)

(*----------Connection with new Security Profile 3 ---------------*)

(* TLS with Client Side Certificates *)
new nonce_CS2: nonce;
new proposed_suite2: suite;
new tls_version2: tlsversion;
new g2:G;

out(c, (xpk_CSMS, nonce_CS2, proposed_suite2, tls_version2,
    xsec_profile, g2));   (* out Client Hello *)

in(c, (nonce_CSMS2: nonce, chosen_suite2: suite));    (* in Server
    Hello *)

event beginCSsession2(nonce_CSMS2, nonce_CS2);

in(c, (m:bitstring, server_cert: bitstring));      (* in
    CertificateServerRequest, Server Certificate *)
if ( (m<> CertificateServerRequest) || (checksignCA(server_cert,
    pk_CA) <> ok())) then 0
else
let xpk_CSMS2 = getpKey(server_cert, pk_CA) in
(* if(xpk_CSMS2 <> xpk_CSMS) then 0
else *)

in(c, CS_CAsigned_certificate: bitstring);   (*in CS Certificate from
    CA*)
let CS_CAsigned_certificate = sdec(CS_CAsigned_certificate,sk(km_CS))
     in
if( (checksignCA(CS_CAsigned_certificate, pk_CA) <> ok()) || (getpKey
    (CS_CAsigned_certificate, pk_CA)<> pk(km_CS)) ) then 0
```

```
else
out(c, ack);

in(c, HD2:bitstring);       (* in Server Hello Done *)

if(checksign(HD2, xpk_CSMS2)<> ok() || getmess(HD2) <> HelloDone)
    then 0
else
event e11();
out(c, penc(CS_CAsigned_certificate, xpk_CSMS2)); (*out Client
    Certificate*)

(*ECDHE, ClientKeyExchange*)

new eph_k_CS2: sdhpar;
let dhpar_CS2 = dh(eph_k_CS2, g2) in

out(c, sign(penc( (xpk_CSMS2, dhpar_CS2), xpk_CSMS2), sk(km_CS)));

in(c, y:bitstring);          (*in DH parameters sec prof 3*)
if (checksign( y, xpk_CSMS2 ) <> ok() ) then 0
else
let (=pk(km_CS), =dhpar_CS2,  xdhpar_CSMS2: bitstring) = getmess(y)
    in
let sympar2 = sdh(xdhpar_CSMS2 , eph_k_CS2) in
let symk2 = masterSecret(sympar2, nonce_CS2, nonce_CSMS2) in

event dhparCS2(xpk_CSMS2, xdhpar_CSMS2, dhpar_CS2);

event acceptsCS2 (symk2, xpk_CSMS2);

(*/ECDHE, ClientKeyExchange*)

out(c, encdh((nonce_CS2, chosen_suite2, nonce_CSMS2, server_cert,
    xsec_profile, ClientFinished), symk2)); (*out ClientFinished*)

in(c, m: bitstring);         (*in ServerFinished*)
let  ( = nonce_CS2, =chosen_suite2, =nonce_CSMS2, =server_cert, =
    xsec_profile, xi:counter, xfinished2:bitstring) = decdh(m,symk2)
    in

if( xfinished2 <> ServerFinished) then 0
else
event e13();

(*————————ChangeCipherSpec——————————*)

event endCSsession2(nonce_CSMS2, nonce_CS2);

```

```
let xi = next(xi) in
out(c,  encdh((xi,secretApplicationData), symk2));  (* out secret*)

(* /TLS with Client Side Certificates *)

(*--------/Connection with new Security Profile 3 ------------*)
let xi = next(xi) in
let m = encdh((xi, BootNotificationRequest(PowerUp, Id_Cs)), symk) in
      (* out BootNotificationRequest *)
out(c, m);

in(c, m: bitstring);        (* in BootNotificationResponse *)

let xi = next(xi) in
let (=xi, m:bitstring)= decdh(m, symk) in
let ( rt: bitstring, int: bitstring) =  BootNotificationResponseRet(m
    ) in
if(rt <> RegistrationType) then 0
else
 event e16();

(*/A05*)

event termCS2(symk2, pk(km_CS));
let xi=next(xi) in
let m = encdh((xi,secretApplicationData), symk) in

out(c, (m));                   (* out secret*)
0.

     (*/CS Client*)

(*CERTIFICATE AUTHORITY, trusted root CA*)
let pCA(km_CA:keymat, pk_CSMS:pkey, pk_CS:pkey) =

event e01_ca();

out  ( c, penc(signCA( pk_CSMS, sk(km_CA)), pk_CSMS) ); (* Server
    Certificate *)
in  (c, =ack);

event e02_ca();

out  ( c, penc(signCA( pk_CS, sk(km_CA)), pk_CS) );  (* Client
    Certificate *)
in  (c, =ack);
          0.

(*Charging Station Operator*)
```

```
let pCSO(km_CSO:keymat, pk_CSMS:pkey) =

let sec_profile = SecProfile2 in
event e05_cso();

out  ( c , sign((ChangeNetworkConfig, upgrade(sec_profile)), sk(
    km_CSO)));

0.

process
new km_CSMS: keymat;
new km_CA:keymat;
new km_CSO:keymat;
new km_CS: keymat;

let pk_CSMS = pk(km_CSMS) in out(c, pk_CSMS);
let pk_CA =  pk(km_CA)in out(c, pk_CA);
let pk_CSO = pk(km_CSO)in out(c, pk_CSO);
let pk_CS = pk(km_CS) in out(c, pk_CS);

((!pCSMS(km_CSMS, pk_CA, pk_CSO)))
| (!pCS(km_CS, pk_CA))
| (!pCA(km_CA, pk_CSMS, pk_CS))
| (!pCSO(km_CSO, pk_CSMS))
```