HAW HAMBURG

BACHELOR THESIS
Laurin J. Bär

DNSSEC im Fahrzeug: Zertifikatsmanagement für automobile Service-Architekturen in einer Umgebung heterogener Zulieferer

FAKULTÄT INFORMATIK UND DIGITALE GESELLSCHAFT

Faculty of Computer Science and Digital Society

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG

Hamburg University of Applied Sciences

Laurin J. Bär

DNSSEC im Fahrzeug: Zertifikatsmanagement für automobile Service-Architekturen in einer Umgebung heterogener Zulieferer

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
der Fakultät Informatik und digitale Gesellschaft
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Schmidt
Zweitgutachter: Prof. Dr. Franz-Josef Korf

Eingereicht am: 23. September 2026

Laurin J. Bär

Thema der Arbeit

DNSSEC im Fahrzeug: Zertifikatsmanagement für automobile Service-Architekturen in einer Umgebung heterogener Zulieferer

Stichwörter

DNSSEC, DANE, DANCE, Zertifikatsmanagement

Kurzzusammenfassung

Um Sicherheit in modernen Fahrzeugen sicher zu stellen, müssen sich die im Fahrzeug befindlichen Services gegenseitig authentifizieren können. Gegenwärtige Lösungen setzen häufig darauf, Schlüssel und Zertifikate statisch mit der Software gebündelt auszuliefern. Dies erschwert das Erneuern der Credentials. Solomon et al. haben ein Konzept zum agilen Credential-Management vorgestellt, welches DNSSEC und DANE verwendet um Zertifikate leichter verteilen zu können. Der Fahrzeughersteller betreibt hierfür ein DNS-Verzeichnis, in welchem die Zertifikate der Services vorliegen. Diese Arbeit implementiert eine Docker-basierte Emulation unterschiedlicher Szenarien und kommt zu dem Ergebnis, dass DNSSEC und DANE für das agile Zertifikatsmanagement in Fahrzeugen geeignet sind, da sie die Credentials und Zertifikate korrekt verteilen ohne eine unverhältnismäßig große Last auf den Software- oder Fahrzeughersteller zu setzen.

Laurin J. Bär

Title of Thesis

DNSSEC in vehicles: Certificate management for automotive service architectures in an environment of heterogeneous suppliers

Keywords

DNSSEC, DANE, DANCE, Certificate management

Abstract

To maintain safety and security in modern vehicles, the services installed in the vehicle must be able to authenticate one another. Current solutions often rely on delivering keys and certificates statically bundled with the software. This makes it difficult to renew credentials. Solomon et al. presented a concept for agile credential management that uses DNSSEC and DANE to facilitate the distribution of certificates. To enable this, the vehicle manufacturer operates a DNS directory in which the certificates for the services are stored. This work implements a Docker-based emulation of various scenarios and concludes that DNSSEC and DANE are suitable for agile certificate management in vehicles, as they distribute credentials and certificates correctly without placing a disproportionately heavy workload on the software supplier or vehicle manufacturer.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Ausgangslage und Motivation

Die Autoindustrie befindet sich im Wandel. Die Anzahl der digitalen Komponenten steigert sich, resultierend daraus, dass bestehende Komponenten digitalisiert wurden und neue Sensorik integriert wird, z.B. Abstandssensor, Außentemperatursensor oder Ultraschallsensoren. Diese vielen neuen Komponenten müssen auch untereinander kommunizieren. Außerdem braucht ihre Software Updates, die ggfs. über das Internet heruntergeladen werden können. Des weiteren sind Multimediasysteme normal geworden, auf denen zusätzliche Applikationen installiert werden können.

Diese zunehmende Vernetzung erhöht die Angriffsfläche signifikant. Angreifer können Komponenten über das interne Netzwerk und über das externe Internet attackieren. Ein Beispiel für diese Risiken haben Charlie Miller und Chris Valasek 2015 am Jeep Cherokee gezeigt [16]. Durch Password Guessing konnten sie über das WLAN-Netwerk auf das Multimediasystem zuzugreifen. Später gelang ihnen dasselbe über das Mobilfunknetz mit Hilfe von IP Scanning. Zu diesem Zeitpunkt inwar es für die Angreifer bereits möglich das Radio mitsamt Lautstärke zu kontrollieren und GPS Daten auszulesen. Die Forscher haben im Anschluss die Software einer Komponente überschrieben, indem sie über das Multimediasystem ihre eigene Software als Update geladen haben. Nun konnten sie Signale über den CAN Bus schicken und so unter anderem Türen, Scheibenwischer, Klimaanlage, Motor und Bremssystem kontrollieren. Dieser Angriff war während der Fahrt über das Internet möglich.

Dieser Angriff zeigt die drei Grundpfeiler der Cybersicherheit: Vertraulichkeit, Integrität, Verfügbarkeit. Man spricht von der CIA Triade (Confidentiality, Integrity, Availability) [9]. Die Angreifer konnten auf Daten zugreifen, die nicht für sie bestimmt waren (GPS Daten, IP-Addresen), sie konnten gefälschte Updates in das System laden und sie konnten die Komponenten des Fahrzeuges ausschalten. Wenn diese drei Ziele erreicht werden

sollen, so muss abgesichert werden, wer auf eine Ressource zugreifen darf (Autorisierung) und dass der Teilnehmer tatsächlich der ist, für den er sich ausgibt (Authentifizierung). Hierfür eignen sich asymetrische kryptographische Schlüsselpaare. Mit ihnen kann man sowohl Daten verschlüsseln und sie so vor unberechtigten Lesern schützen, als auch Daten signieren und so ihre Echtheit bestätigen. Bisher war es oft üblich die Schlüssel statisch in den Komponenten einzuarbeiten [15] [18]. Dies ist zwar einfach umzusetzen und zu verwalten, jedoch hat das mehrere Nachteile:

1. Die Schlüssel gelten oft für alle Komponenten mit dieser Software, das heißt alle Fahrzeuge, die diese Software nutzen, verwenden den selben Schlüssel. Wird nun der Schlüssel von einem Fahrzeug bekannt, macht das alle angreifbar.

2. Es ist schwieriger die Schlüssel zu erneuern.

3. Nicht nur die Komponente selbst braucht ihren Schlüssel. Andere benötigen den öffentlichen Schlüssel, um die Komponente zu verifizieren.

Wenn Schlüsselpaare aktualisierbar sein sollen, muss auch jeder Service, mit dem potentiell kommuniziert wird, aktualisiert werden, um den neuen Schlüssel zu erhalten. Das würde den Austausch eines einzigen Schlüsselpaares technisch sehr aufwendig gestalten.

Um dieses Problem zu lösen, haben Salomon et al. [20] ein Konzept vorgestellt, welches DNSSEC benutzt um Schlüsselpaare zu verifizieren und deren Lebenszyklus zu verwalten. Dafür wird bei einem Update vom Softwarehersteller ein Schlüsselpaar erstellt, hiermit ein Zertifikat signiert und dieses dann an den Fahrzeughersteller geschickt. Dieser verwaltet ein DNS-Verzeichnis, in dem für jedes Fahrzeug eine eigene Zone vorhanden ist. Mit dem Zertifikat wird nun ein TLSA Eintrag erstellt und in der Zone gespeichert. Diese wird nun mit dem Zone Signing Key (ZSK) des Fahrzeugherstellers signiert, um eine Chain-Of-Trust zu erreichen. Der private Schlüssel wird im sicheren Element der aktualisierten Komponente gespeichert. Möchte nun eine andere Komponente mit der aktualisierten Komponente kommunizieren, so kann diese über den DNS-Baum den Schlüssel verifizieren und die Chain-Of-Trust nachvollziehen.

Inhalt dieser Arbeit ist, eine Implementierung basierend auf diesem Design zu erstellen. Dazu gehört eine Analyse der Stakeholder und deren Anforderungen, ein Design für einzelne Komponenten und Interaktionen, die tatsächliche Emulation und schließlich die Evaluation. Anschließend folgt ein Ausblick, welche Veränderungen und Verbesserungen möglich wären.

1.2 Aufbau der Arbeit

Kapitel 2 betrachtet DNSSEC und DANE, die Probleme, die diese lösen und wie sie dies erreichen. Weiter folgt eine kurze Zusammenfassung von SOME/IP und schließlich eine Erklärung von Firmware-Over-The-Air-Updates.

Kapitel 3 geht auf die gegenwärtige Situation der Zertifikatsverwaltung innerhalb von Fahrzeugen und die Arbeiten von Mueller et al. und Salomon et al. ein.

Kapitel 4 beinhaltet eine Analyse der Akteure und eine Beschreibungen der Komponenten und der Abläufe inklusive Fehlerszenarien.

Kapitel 5 befasst sich mit der Implementierung des Systems. Dabei wird auf die technischen Entscheidungen eingegangen und es werden die Versuchsabläufe beschrieben.

Kapitel 6 ist eine Analyse der Ergebnisse und betrachtet die Installationszeiten und die Aktualisierungszeiten unter verschiedenen Gesichtspunkten.

Kapitel 7 fasst diese Arbeit zusammen und gibt einen Ausblick auf Sicherheitsrisiken und weitere Forschung.

2 Grundlagen

2.1 Domain Name System Security Extensions (DNSSEC)

Das Domain-Name-System (DNS) ist anfällig für verschiedene Angriffe. Dazu zählen das Verschweigen existierender Einträge (Authenticated Denial of Domain Names), maliziöse Antworten (Cache Poisoning) von vertrauenswürdigen Servern (Betrayal By Trusted Server) oder anderen Angreifern im selben Netwerk, beispielweise durch ID Guessing [3].

Eine Lösung für diese Probleme bietet DNSSEC, eine Reihe von Erweiterungen für das DNS Protokoll. Mit DNSSEC ist es möglich Authentizität und Integrität eines DNS Eintrags zu überprüfen. Dafür fügt DNSSEC jeder Zone mehrere asymmetrische Schlüssel hinzu. DNS Einträge werden zu RRSETs zusammengefasst und erhalten einen RRSIG Eintrag. Diese beinhalten eine Signatur des RRSETS. Signiert sind sie mit dem privaten Zone-Signing-Key (ZSK). Der öffentliche Teil des ZSK liegt in der selben Zone vor und ist signiert von dem Key-Signing-Key (KSK). Ein Hash des öffentlichen KSK liegt in der Elternzone. Einzelne DNS Einträge können so über ihren RRSIG überprüft werden, welche ihrerseits über den ZSK überprüft werden können, welcher über den KSK überprüft werden kann, welcher über die Elternzone überprüfen werden kann. Der Eintrag in der Elternzone ist ebenfalls mit DNSSEC gesichert, so dass sich eine Chain-of-trust ergibt, welche bis zu der Root-Zone reicht [10]. Dies erschwert es für Angreifer, maliziöse Antworten einzuschleusen, da diese von dem ZSK signiert sein müssen. Auch ein Authenticated Denial of Domain Names ist nicht ohne den ZSK möglich, da die ebenfalls aus DNSSEC stammenden NSEC Einträge eine Sequenz aus validen Namen bildet und es somit nachgewiesen werden kann, dass ein Eintrag nicht existiert [2].

2.2 DNS-based Authentication of Named Entities (DANE)

DANE TLSA ist eine Protokoll [11], mit dem Zertifikate über DNS verteilt und verifiziert werden können um eine TLS Verbindung aufzubauen. Grund für die Entstehung war es, dass Certificate Authorities (CA) Zertifikate von jeder Domain signieren können, was Potential für Missbrauch bietet. Ein Angriff auf eine CA könnte dazu führen, dass Zertifikate für jede Domain ausgestellt werden könnten, womit Authentifizierung umgangen werden würde. DANE bindet die Zertifikate an das DNS, wodurch eine zusätzliche Verifikation entsteht [7]. Dafür wird ein Resource Record namens TLSA genutzt. Dieser besteht aus

einem Port

einem Protokoll (TCP oder UDP)

der Domain des Eintrags

einer Time-To-Live (TTL)

der Certificate Usage. Dies ist eine Zahl zwischen 0 und 3. Eine 0 bedeutet, dass das Zertifikat während des Verifizierens zu der angegebenen Certificate Authority (CA) auflöst. 1 bedeutet, dass das Zertifikat nach X.509 valide ist. 2 bedeutet, dass das Zertifkat von einer privaten CA signiert wurde und 3 bedeutet, dass das Zertifikat nicht verifiziert werden kann oder muss.

dem Selector Field. 0 bedeutet, dass das gesamte Zertifikat genutzt wird. Bei einer 1 wird nur der Öffentliche Schlüssel des Zertifikats genutzt.

dem Matching Type. Dieser gibt an, ob und mit welchem Algorithmus gehasht wird. Bei 0 wird nicht gehasht, bei 1 wird Sha256 genutzt und bei 2 Sha512.

Um die Authentizität der Zertifikate sicherzustellen, müssen diese über DNSSEC geschützt sein. Deshalb ist DNSSEC zwingend notwendig für DANE [7].

2.3 Scalable service-Oriented MiddlewarE over IP (SOME/IP)

Scalable service-Oriented MiddlewarE over IP (SOME/IP) ist ein Protokoll, das von Autosar entwickelt wurde. Es wird für die Client-Server-Kommunikation zwischen ECUs genutzt und ermöglicht unter anderem Publish-Subscribe, Remote-Procedure-Calls und Service-Discovery. SOME/IP verwendet sowohl TCP als auch UDP [5]. Für die Service-Discovery sendet der Server eine **offer** Nachricht. Diese kann zyklisch gesendet werden oder als Reaktion auf eine **find** Nachricht. Die Service-Discovery besitzt keine Authentifikation der Services oder Clients [6].

2.4 Firmware Over The Air Update (FOTA)

Autosar hat ein Konzept für ein einheitliches Firmware-Over-The-Air-Update vorgestellt. Dieses beinhaltet drei Komponenten: das FOTA-Backend, den FOTA-Master und das FOTA-Target. Der FOTA-Master befindet sich im Fahrzeug und orchestriert das Update. Er lädt das Update, das sogenannte FOTA-Image von dem FOTA-Backend herunter. Dieser Prozess wird als Download bezeichnet. Das FOTA-Image wird in einzelnen Chunks geladen und der Download kann unterbrochen und über mehrere Fahrten hinweg fortgesetzt werden. Ist dieser abgeschlossen, beginnt der FOTA-Master mit dem Install. Dazu wird das FOTA-Image auf das FOTA-Target geladen. Beim FOTA-Target handelt es sich um die ECU, die das Update erhalten soll. Nachdem das FOTA-Image auf das FOTA-Target geladen wurde, wird es verifiziert. Erfolgt der Install ohne Fehler, so kann die ECU bei ihrem nächsten Bootvorgang in die neue Software laden. Dies wird Activation genannt. Die Activation kann nur dann stattfinden, wenn sich das Fahrzeug in einem sicheren Zustand befindet, d.h. wenn es steht, der Motor aus ist und die Handbremse angezogen ist. Wird während der Verifikation oder der Activation ein Fehler festgestellt, so wird ein Rollback eingeleitet. Dazu müssen zusätzlich jene ECUs, die eine Abhängigkeit von der aktualisierten ECU haben, ebenfalls zurückgerollt werden. Voraussetzung für einen Rollback ist, dass die Vorgängersoftware sich noch auf der ECU befindet [4].

3 Dienstauthentifikation im Auto

Vehicle Software Lifecycle Management beschreibt die Verwaltung von Software, die auf den Electrical Control Units (ECUs) innerhalb eines Fahrzeugs läuft. Dazu zählen das Bereitstellen, das Aktualisieren und das Löschen. Das Vehicle Software Lifecycle Management ändert sich, weil Software nun agiler entwickelt wird, z.B. mit Continuous delivery, und es dadurch möglich ist Features nachzureichen oder bereitzustellen, Sicherheitslücken zu schließen und Fehler zu beheben. Dies ist möglich durch Firmware-Over-The-Air-Updates, mit denen keine physische Verbindung oder Fahrt zu einer Werkstatt notwendig ist. ECUs ersetzen Komponenten, die früher mechanisch waren. Dabei ergab sich in den letzten Jahren der Trend, dass mehrere ECUs zu einer zentralisierten ECU zusammengelegt wurden [12], sofern es mit Sicherheit und den Echtzeitanforderungen kompatibel ist.

Um die Sicherheit des Fahrzeugs zu gewährleisten gibt es verschiedene Standards, wie z.B. ISO SAE 21434 [14], welche Anforderungen an den Fahrzeughersteller stellen, ohne eine spezielle Implementierung einzufordern. Schlüssel sowie Zertifikate sind oft statisch und werden zusammen mit der Software installiert [15] [18]. Das ist ein Problem, da es die Erneuerung von Credentials deutlich erschwert. Es wird empfohlen Schlüssel nicht länger als zwei Jahre zu verwenden [17]. Wird ein Schlüssel bekannt müssen nicht nur der Schlüssel selbst, sondern auch alle Zertifikate, die auf diesem Schlüsselpaar basieren gewechselt werden. Ohne eine einheitliche Schnittstelle ist der Austausch von Credentials erschwert.

3.1 SOME/IP Service Discovery mit DNSSEC gestützter Authentifikation

Diese Arbeit basiert auf den Arbeiten von Mueller et al. [8] und Salomon et al. [19] [20]. Diese schlagen eine Erweiterung für das von Autosar vorgestellte SOME/IP vor. Die

Diensterkennung von SOME/IP besitzt keine Authentifikation, weshalb Mueller et al. ein Konzept vorgestellt haben, welches eine DNS basierte Authentifikation und Zertifikatsverwaltung hinzufügt. DANE soll genutzt werden, um Zertifikate für Services verifizierbar zu machen und SVCB Einträge, um die Diensterkennung zu validieren.

Der Fahrzeughersteller soll ein DNS Verzeichnis verwalten, in welchem jedes Fahrzeug eine Zone erhält. In diesen Zonen soll jeder Service einen oder mehrere TLSA und SVCB Einträge haben. In den SVCB Einträgen stehen Informationen zu den Services, z.B. Adresse, Port und Protokoll. Mit diesen Informationen soll ein Client überprüfen können, ob ein in der Diensterkennung angebotener Service gültig ist. TLSA Einträge beinhalten ein Zertifikat mit einem öffentlichen Schlüssel, wobei der private Schlüssel sich in einem gesicherten Element in der ECU des Services befindet. Wenn ein Service einen anderen Service verifizieren möchte, so kann er über DNS den TLSA Eintrag anfordern und über den öffentlichen Schlüssel die Identität des Service bestätigen.

Desweiteren könnte der Fahrzeughersteller Teile des DNS Verzeichnisses an Drittanbieter übertragen, z.B. für Infotainment, da dieses keine Kritikalität hat.

In ihrer Arbeit „Building Automotive Security on Internet Standards“ [20] zählen Salomon et al. einige Voraussetzungen für Authentifizierung in Fahrzeugen auf. Dazu zählen, unter anderen, die Möglichkeit ohne Verbindung zum Internet zu agieren, hohe Skalierbarkeit, Verteilbarkeit, Performance und Anpassbarkeit. Zertifikatsverwaltung über DANE bietet einige Vorteile. Zum Beispiel ist die Authentizität und Integrität der TLSA Einträge durch DNSSEC gesichert, DNS ermöglicht schnelle und frequente Updates und es handelt sich bei DANE und DNSSEC um viel genutzte und dementsprechend erprobte Protokolle, die genug Flexibilität für das sich entwickelnde Feld mitbringen. DNS bietet hohe Skalierbarkeit und ein hierarchisches System, in welchem man nach Fahrzeug, Service und Version trennen könnte. Einträge können lokal gecached werden, um bis zum Ablauf ihrer Lebenszeit offline genutzt zu werden.

Salomon et al. erweitern das Design von Mueller et al. um Clientauthentifizeirung und eine Zertifikatsverwaltung. Wenn der Softwarehersteller ein Update für einen Service bereitstellen will, so erstellt er nicht nur das Programm als Binärdatei, sondern auch ein Schlüsselpaar, welches für jedes Fahrzeug individuell ist, und ein Zertifikat für das entsprechende Schlüsselpaar. Das Zertifikat wird an den Fahrzeughersteller gesendet, welcher daraus einen TLSA Eintrag erzeugt und diesen im DNS Verzeichnis speichert. Die Binärdatei und das Schlüsselpaar werden über ein Over-The-Air Update in das Fahrzeug geladen.

Weiter analysieren Salomon et al. die Sicherheit der mit DNSSEC gesicherten SOME/IP Service-Discovery mit dem STRIDE Model. Dafür betrachten sie folgende Angriffsszenarien: Identitätsbetrug (Spoofing identity), Manipulation von Nachrichten (Tampering with messages), Abstreiten getätigter Handlungen (Repudiation of actions), Veröffentlichung von Geheimnissen (Information disclosure), Denial-of-service und Ausweitung von Berechtigungen (Elevation of privilege). Salomon et al. kommen zu dem Ergebnis, dass das augmentierte SOME/IP Gegenmaßnahmen für die obigen Angriffe aufweist, jedoch keinen Schutz bietet vor Supply-Chain-Attacks, gestohlenen Schlüsseln und Denial-Of-Service durch eine Überladung während der Startphase des Fahrzeugs.

Da das Ziel des Systems das Ausliefern von Schlüsselpaaren und Zertifikaten ist und nicht das direkte Etablieren einer TLS-Verbindung, ist es nicht an TLS gebunden und kann als Basis für andere Protokolle genutzt werden. Das ist wichtig, da TLS möglicherweise nicht den Anforderungen des Fahrzeugherstellers entspricht, da es z.B kein Multicast unterstützt [13].

Diese Arbeit wird zusätzlich zu dem Design von Salomon et al. eine veränderte Variante des Systems betrachten, in welcher die ECU das Schlüsselpaar erstellt. Dies sorgt für zusätzliche Sicherheit, da der private Schlüssel nicht die ECU verlässt und nicht jedes sichere Element die Funktion unterstützt, externe private Schlüssel zu speichern [1].

4 Design

4.1 Stakeholder und Anwendungsfälle

Um die Anforderungen an das System zu formulieren, müssen zuerst die Stakeholder und deren Einsatzszenarien und Sicherheitsbedenken analysiert werden. Die zentralen Stakeholder des Systems sind der Nutzer, der Softwarehersteller, die Werkstatt und der Fahrzeughersteller.

Der Nutzer, oder auch Endnutzer, kann der Fahrer des Fahrzeugs sein, derjenige der das Fahrzeug in die Werkstatt gibt oder derjenige der Software im Infotainment System installiert, aktualisiert oder entfernt. Dem Nutzer müssen Sicherheit, das heißt sowohl Security als auch Safety, gegeben sein. Das Fahrzeug soll nicht durch abgelaufene Zertifikate blockiert werden, besonders nicht während der Fahrt. Auch ist zu bedenken, dass nicht immer eine Verbindung zum Internet besteht und das Laden von Zertifikaten demnach nicht immer möglich ist. Das korrekte Laden und Löschen von Zertifikaten ist folglich von höchster Priorität für den Nutzer.

Der Softwarehersteller ist für die Bereitstellung von Software verantwortlich. Dabei kann es sich beispielsweise um Treiber, um Infotainment Apps oder um Updates für bestehende Software handeln. Zusätzlich bietet er eine Schnittstelle, über die Zertifikate hochgeladen werden können. Diese leitet er über eine gesicherte Verbindung an den Fahrzeughersteller weiter. Der Softwarehersteller hat die Anforderungen, dass Software sicher zur Verfügung gestellt werden kann und dass durch das Hochladen von Zertifikaten keine unnötig große Last entsteht, beispielsweise durch zu häufige Retries.

Der Fahrzeughersteller stellt das Fahrzeug aus und betreibt den DNS-Baum, welcher die Zertifikate beinhaltet. Dies tut er für alle Fahrzeuge, welche er produziert hat. Folglich ist ihm wichtig wie das System, das Inhalt dieser Arbeit ist, skaliert. Deshalb zählt für ihn eine niedrige Installationszeit. Dies ist die Zeit, die nötig ist, ein Zertifikat zu erstellen und in den DNS-Baum einzupflegen. Daraus ergibt sich die Zeit, die benötigt wird, um ein

neues Fahrzeug fahrfähig zu machen. Außerdem möchte er möglichst wenig Speicherplatz pro Zertifikat und folglich pro Fahrzeug aufwenden, weil auch das mit der Anzahl der verkauften Fahrzeuge skaliert.

Die Werkstatt führt im Auftrag des Nutzers Reparaturen am Fahrzeug durch, oder baut neue Komponenten ein bzw. tauscht Komponenten aus. Diese können vorinstallierte Software haben. Beim Ein- und Ausbauen von Komponenten müssen Zertifikate erstellt oder gelöscht werden. Deshalb ist für die Werkstatt wichtig, dass dies korrekt geschieht und keine alten Zertifikate zurückbleiben oder neue fehlen.

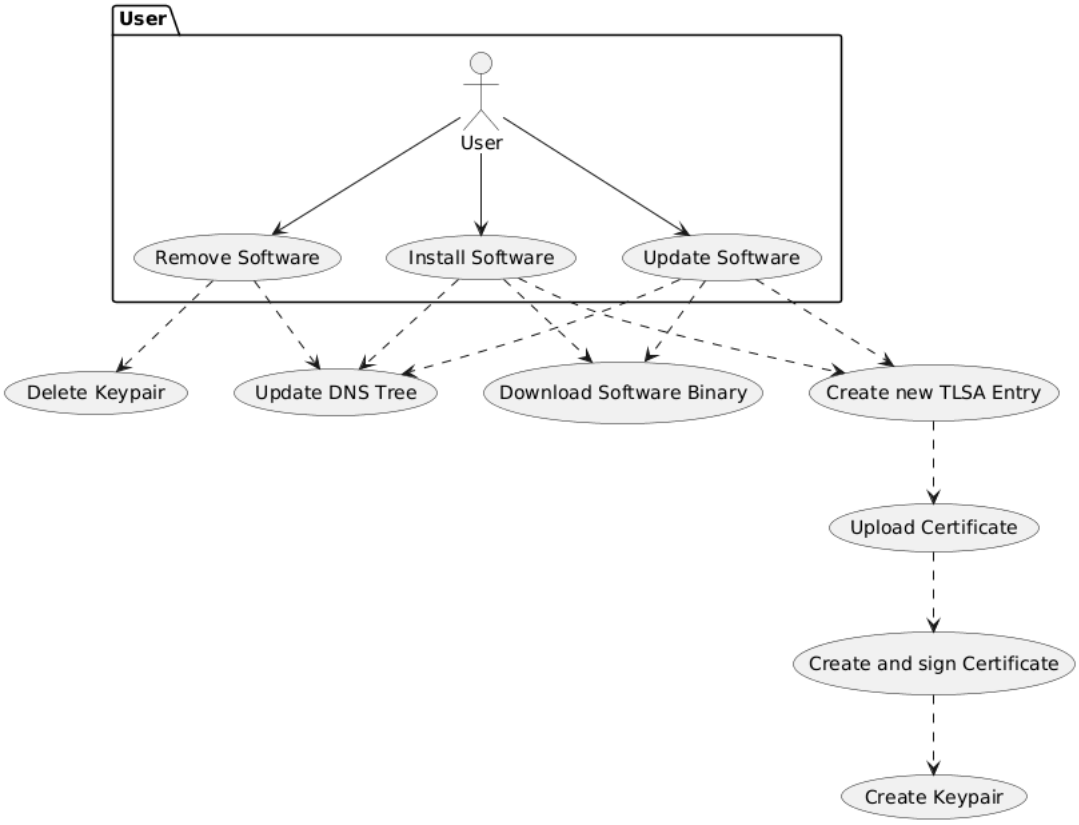

Abbildung 4.1: Anwendungsfälle eines Anwenders

Usecase 1.1

Beschreibung Der Nutzer installiert eine neue Software, z.B. eine Infotainment App.

Akteur Nutzer

Ziel Neue Funktionalität freischalten.

Stakeholder Nutzer, Softwarehersteller, Fahrzeughersteller

Vorbedingung Der Nutzer hat Zugriff auf die Softwareverwaltung.

Nachbedingung Die vom Nutzer angegebene App wurde hinzugefügt. Die Software hat Zugriff auf das erstellte Schlüsselpaar. Der neue TLSA Eintrag befindet sich im DNS Verzeichnis.

Ablauf

- Der Nutzer initialisiert die Installation neuer Software über die Softwareverwaltung.
- Die Software wird heruntergeladen.
- Es wird ein neues Schlüsselpaar erstellt.
- Es wird ein neues Zertifikat erstellt und mit dem privaten Schlüssel signiert.
- Das Zertifikat wird beim Softwarehersteller hochgeladen, welcher es weiter beim Fahrzeughersteller hochlädt.
- Aus dem Zertifikat wird ein TLSA Eintrag erstellt und in das DNS Verzeichnis eingepflegt.
- Die Software ist nun funktionsfähig, da sie über den privaten Schlüssel verfügt und das Zertifikat im DNS Verzeichnis eingetragen ist.

Usecase 1.2

Beschreibung Der Nutzer aktualisiert eine Software.

Akteur Nutzer

Ziel Die Software soll auf die neueste Version aktualisiert werden.

Stakeholder Nutzer, Softwarehersteller, Fahrzeughersteller

Vorbedingung Der Nutzer hat Zugriff auf die Softwareverwaltung. Es steht ein Update zur Verfügung.

Nachbedingung Die Software ist auf der aktuellsten Version. Die Software hat Zugriff auf das erstellte Schlüsselpaar. Der neue TLSA Eintrag befindet sich im DNS Verzeichnis.

Ablauf

- Der Nutzer initialisiert die Aktualisierung neuer Software über die Softwareverwaltung.
- Das Software Update wird heruntergeladen.
- Es wird ein neues Schlüsselpaar erstellt.
- Es wird ein neues Zertifikat erstellt und mit dem privaten Schlüssel signiert.
- Das Zertifikat wird beim Softwarehersteller hochgeladen, welcher es weiter beim Fahrzeughersteller hochlädt.
- Aus dem Zertifikat wird ein TLSA Eintrag erstellt und in das DNS Verzeichnis eingepflegt.
- Die aktualisierte Software ist nun funktionsfähig, da sie über den privaten Schlüssel verfügt und das Zertifikat im DNS Verzeichnis eingetragen ist.

Usecase 1.3

Beschreibung Der Nutzer entfernt eine Software

Akteur Nutzer

Ziel Es soll eine spezielle Software aus dem Fahrzeug gelöscht werden.

Stakeholder Nutzer, Softwarehersteller, Fahrzeughersteller

Vorbedingung Der Nutzer hat Zugriff auf die Softwareverwaltung.

Nachbedingung Die Software wurde deinstalliert. Das entsprechende Zertifikat, der TLSA Eintrag und das Schlüsselpaar wurden gelöscht.

Ablauf

- Der Nutzer initialisiert die Deinstallation der Software.
- Die Software selbst wird entfernt.
- Schlüssel und Zertifikat werden im Fahrzeug gelöscht.
- Der Softwarehersteller wird aufgefordert den TLSA Eintrag der Software zu löschen.
- Der Softwarehersteller gibt die Anfrage auf Löschung an den Fahrzeughersteller weiter.

- Der Fahrzeughersteller entfernt den TLSA Eintrag aus dem DNS Verzeichnis.

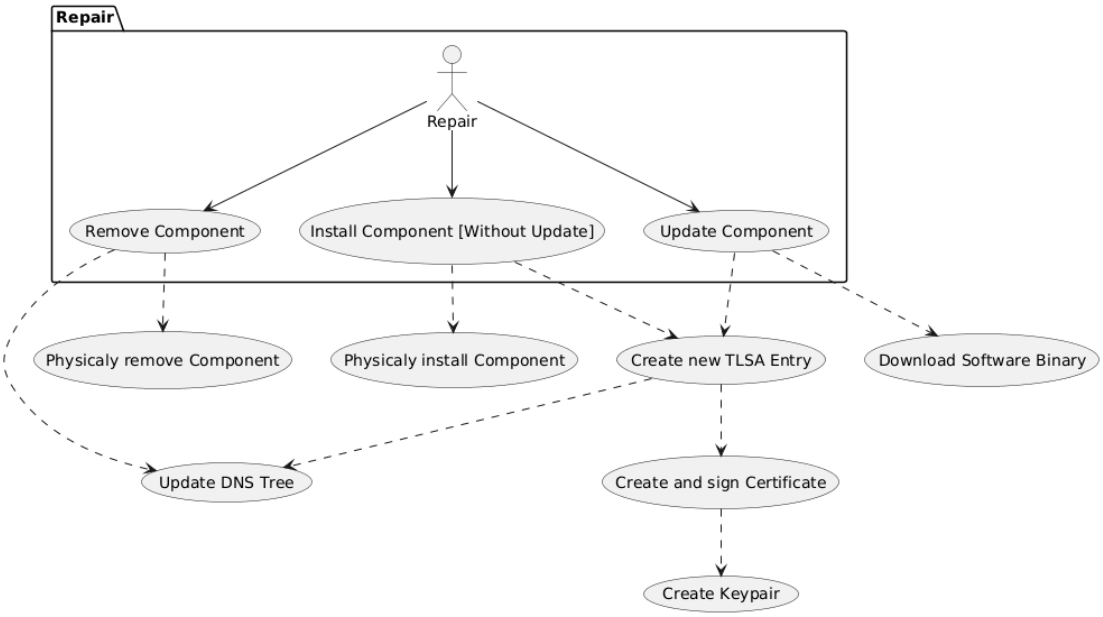

Abbildung 4.2: Anwendungsfälle einer Werkstatt

Usecase 2.1

Beschreibung Service hinzufügen

Akteur Werkstatt

Ziel Es soll eine Komponente eingebaut werden. Diese kann bereits vorinstallierte Software beinhalten.

Stakeholder Werkstatt, Nutzer, Softwarehersteller, Fahrzeughersteller

Vorbedingung Die Komponente ist physisch verfügbar und die Werkstatt hat Zugriff auf das Fahrzeugsystem.

Nachbedingung Die Komponente ist eingebaut und die Software darauf kann korrekt mit anderen Komponenten kommunizieren.

Ablauf

- Die Werkstatt baut die Komponente in das Fahrzeug ein.
- Die Software wird in der Softwareverwaltung des Fahrzeugs registriert.
- Es wird ein neues Schlüsselpaar erstellt.

- Es wird ein neues Zertifikat erstellt und mit dem privaten Schlüssel signiert.
- Das Zertifikat wird beim Softwarehersteller hochgeladen, welcher es weiter beim Fahrzeughersteller hochlädt.
- Aus dem Zertifikat wird ein TLSA Eintrag erstellt und in das DNS Verzeichnis eingepflegt.
- Die aktualisierte Software ist nun funktionsfähig, da sie über den privaten Schlüssel verfügt und das Zertifikat im DNS Verzeichnis eingetragen ist.

Usecase 2.2

Beschreibung Service entfernen

Akteur Werkstatt

Ziel Eine Komponente soll aus dem Auto ausgebaut werden.

Stakeholder Werkstatt, Nutzer, Softwarehersteller, Fahrzeughersteller

Vorbedingung Die Komponente ist im Fahrzeug vorhanden und im System integriert und die Werkstatt hat Zugriff auf das Fahrzeugsystem.

Nachbedingung Die Komponente wurde entfernt. Alle damit verbundenen Zertifikate, TLSA Einträge und Schlüsselpaare wurden entfernt.

Ablauf
- Die Werkstatt initialisiert die Deinstallation der Software.
- Die Software selbst wird entfernt.
- Schlüssel und Zertifikat werden im Fahrzeug gelöscht.
- Der Softwarehersteller wird aufgefordert den TLSA Eintrag der Software zu löschen.
- Der Softwarehersteller gibt die Anfrage auf Löschung an den Fahrzeughersteller weiter.
- Der Fahrzeughersteller entfernt den TLSA Eintrag aus dem DNS Verzeichnis.
- Die Werkstatt baut die Komponente aus das Fahrzeug aus.

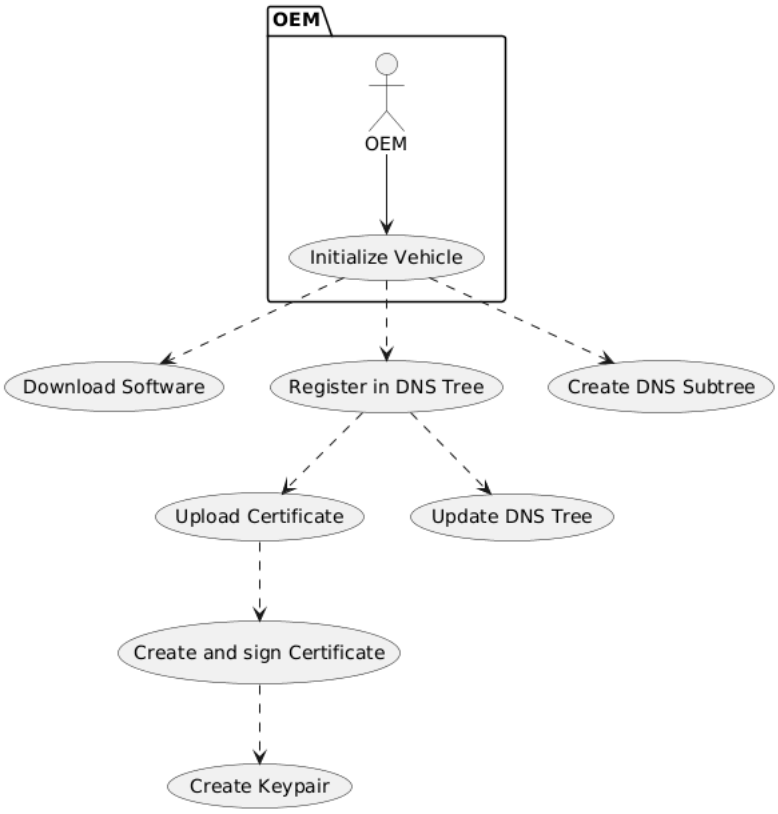

Abbildung 4.3: Anwendungsfälle eines Fahrzeugherstellers

Usecase 3.1

Beschreibung Ein neues Fahrzeug wird in Betrieb genommen.

Akteur Fahrzeughersteller

Ziel Das neue Fahrzeug ist fahrfähig.

Stakeholder Fahrzeughersteller, Softwarehersteller, (Nutzer)

Vorbedingung Das Fahrzeug ist fertig montiert. Es existiert eine Liste mit der benötigten Software im Fahrzeug.

Nachbedingung Alle Software ist auf der neuesten Version und besitzt ein Zertifikat, Schlüsselpaar und TLSA Eintrag im DNS Verzeichnis.

Ablauf

- Der Fahrzeughersteller lädt die initiale Konfiguration der Software in die Softwareverwaltung.

- Für jede Software im Fahrzeug wird nacheinander der Prozess zur Installation initiiert. Für jede Software wird jeder folgende Schritt angewandt.
- Es wird ein Schlüsselpaar erstellt.
- Es wird ein Zertifikat erstellt und mit dem privaten Schlüssel signiert.
- Das Zertifikat wird beim Softwarehersteller hochgeladen. Dieser sendet es weiter an den Fahrzeughersteller.
- Der Fahrzeughersteller erstellt aus dem Zertifikat ein TLSA Eintrag und lädt diesen in das DNS-Verzeichhnis.

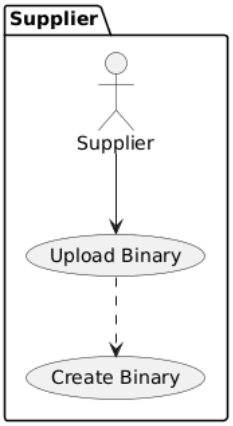

Abbildung 4.4: Anwendungsfälle eines Softwareherstellers

Usecase 4.1

Beschreibung Der Softwarehersteller stellt ein Update bereit.

Akteur Softwarehersteller

Ziel Die Software im Fahrzeug wird aktualisiert.

Stakeholder Softwarehersteller, Nutzer, Fahrzeughersteller

Vorbedingung Die Software muss fertig und vom Hersteller signiert sein (Verifikation der Binärdatei wird in dieser Arbeit nicht weiter betrachtet.).

Nachbedingung Für alle Fahrzeuge mit entsprechender Software gilt: Die Software ist auf der aktuellsten Version. Die Software hat Zugriff auf das erstellte Schlüsselpaar. Der neue TLSA Eintrag befindet sich im DNS Verzeichnis.

Ablauf

- Der Softwarehersteller stellt die Binärdatei für das Update über einen entsprechenden Server bereit.
- Die Fahrzeuge mit der entsprechenden Software erhalten entweder eine Push Benachrichtigung über das Update oder pollen nach Updates.
- Die Fahrzeuge laden das Software Update herunter.
- Jedes Fahrzeug erstellt ein neues Schlüsselpaar.
- Jedes Fahrzeug erstellt ein Zertifikat und signiert es mit seinem Schlüssel.
- Jedes Fahrzeug lädt sein Zertifikat beim Softwarehersteller hoch, dieser reicht dieses an den Fahrzeughersteller weiter.
- Aus den Zertifikaten werden TLSA Einträge erstellt und diese in das DNS Verzeichnis eingepflegt.
- In jedem Fahrzeug ist die Software nun auf der aktuellen Version und ist mit neuem Schlüsselpaar ausgestattet.

4.2 Komponenten

Das System lässt sich in drei Hauptkomponenten gliedern: Das Fahrzeug, der Softwarehersteller (Supplier) und der Fahrzeughersteller (OEM). Diese unterteilen sich ihrerseits weiter. Das Fahrzeug beinhaltet mehrere ECUs, einen lokalen DNS Resolver und einen Service, der alle Updates orchestriert. Der Softwarehersteller erlaubt das Herunterladen von Updates (via OTA) und das Hochladen von Zertifikaten (via HTTPS). Der OEM betreibt einen Zertifikatsserver (via HTTPS), und einen autoritativen DNS-Server, welcher über DDNS aktualisiert werden kann.

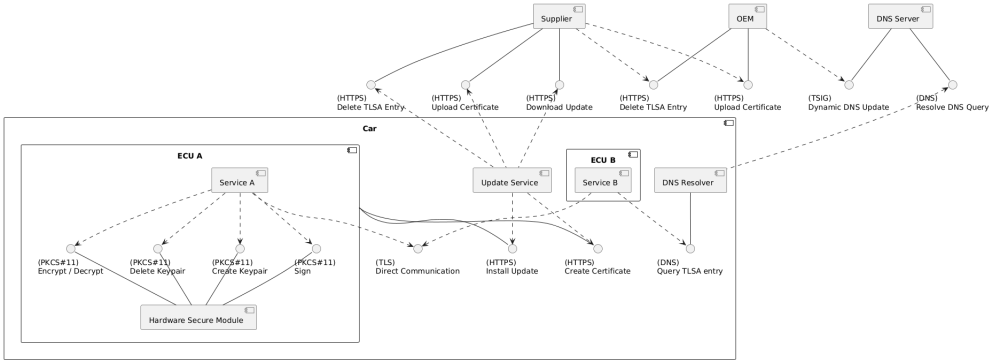

Abbildung 4.5: Komponenten und Interfaces

Update Service Der Update Service orchestriert das Update. Er ist der aktive Teil, der die anderen Komponenten über deren Schnittstellen steuert. Der Update Service ist der OTA Master. Demnach kann er vom Supplier (OTA Backend) Updates herunterladen und auf die ECU (OTA Target) Updates installieren. Zusätzlich kann er die ECU auffordern ein neues Schlüsselpaar mit passendem Zertifikat zu erstellen, welches er dann beim Supplier hochladen kann. Er kann auch alte TLSA Einträge vom Supplier und alte Schlüsselpaare von der ECU löschen lassen.

ECU A Die ECU A gibt die Möglichkeit auf ihr neue Software zu installieren, neue Credentials zu erstellen oder alte zu löschen.

Hardware Secure Module Das HSM von ECU A erlaubt das Erstellen und Löschen von Schlüsselpaaren und mit ihnen das Verschlüsseln, Entschlüsseln und Signieren.

Supplier OTA Server & HTTPS Server Der Supplier ermöglicht es dem Update Service Updates herunter zu laden, sowie das Hochladen von Zertifikaten und das Löschen alter TLSA Einträge. Um dies zu erreichen nutzt er die entsprechenden Schnittstellen des OEMs.

OEM HTTPS Server Der HTTPS Server des OEMS steht zwischen Supplier und autoritativem DNS-Server. Wir gehen davon aus, dass Supplier und OEM eine gesicherte Verbindung haben. Die Kommunikation ist demnach verschlüsselt und die Teilnehmer sind authentifiziert. Der Supplier kann nur die Einträge am DNS-Baum ändern, zu denen er berechtigt ist. Die Updates selbst erfolgen über DDNS.

Authoritive DNS-Server Der autoritative DNS-Server hält alle TLSA Enträge und die passenden RRSIGs. Für Updates bietet er eine DDNS Schnittstelle.

Recursive DNS-Resolver Der Recursive DNS-Resolver fordert die TLSA Einträge von autoritativen DNS-Server an und speichert diese in seinem Cache. Zusätzlich prüft er die DNSSEC Signaturen.

4.2.1 Interaktionen und Fehlerszenarien

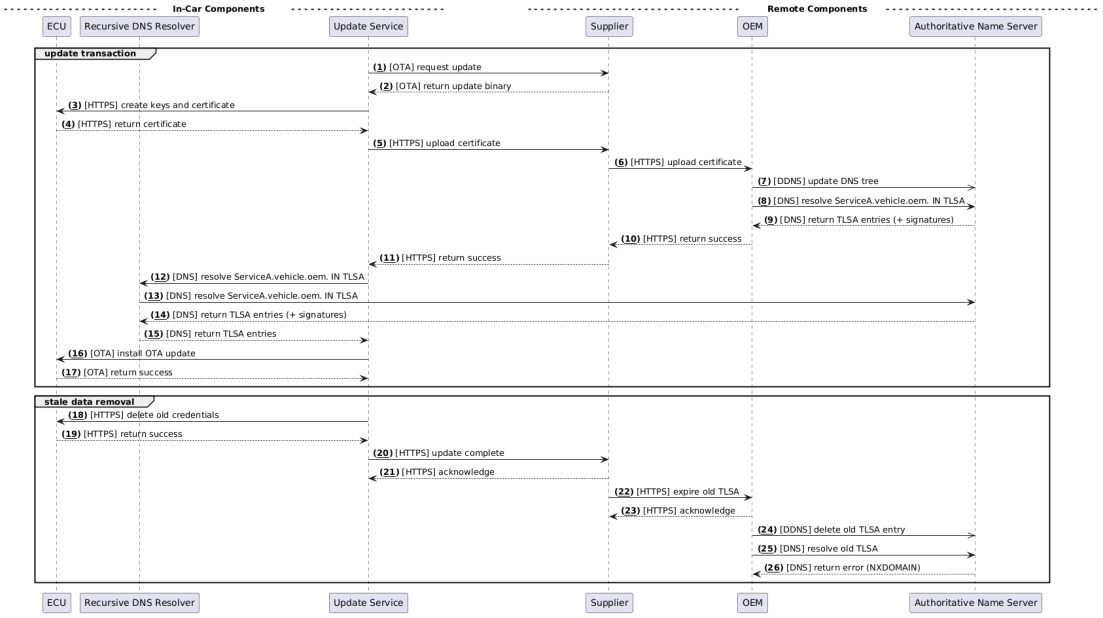

Abbildung 4.6: Ablauf ohne Fehler

1 - 2 Der Update Service (OTA Master) lädt das Update vom Supplier (OTA Backend) herunter

3 - 4 Die ECU wird vom Update Service aufgefordert ein neues Schlüsselpaar und Zertifikat zu erstellen. Das Zertifikat wird an den Update Service zurückgeschickt.

5 Das Zertifikat wird von Update Service beim Supplier hochgeladen. Die Antwort erhält der Update Service erst, wenn Schritt IV, V und VI abgeschlossen sind.

6 Der Supplier gibt das Zertifikat an den OEM weiter.

7 Der OEM erstellt einen TLSA Eintrag aus dem Zertifikat und aktualisiert mit diesem den DNS-Baum über DDNS.

8 - 9 Der OEM fragt den neu erstellten TLSA vom autoritativen DNS-Server ab. Ist der Eintrag noch nicht im DNS-Baum geladen, wird der Schritt per Polling wiederholt, bis der Eintrag erfolgreich zurückgegeben wird.

10 Die HTTPS Antwort auf 6

11 Die HTTPS Antwort auf 5

12 Der Update Service initiiert die Abfrage des neuen TLSA Eintrag vom DNS Resolver.

13-14 Der DNS Resolver fragt bei dem DNS Server nach den passenden Einträgen und erhält eine Antwort.

15 Der DNS Resolver gibt die TLSA Einträge an den Update Service weiter.

16 - 17 Der Update Service (OTA Master) installiert das Update auf der ECU (OTA Target).

18 - 19 Der Update Service lässt die ECU die Credentials der alten Version löschen.

20 - 21 Der Update Service lässt den alten TLSA Eintrag, vom Supplier, aus dem DNS Baum löschen.

22 - 23 Der Supplier leitet die Anfrage auf Löschung an den OEM weiter.

24 - 26 Der OEM löscht den alten TLSA Eintrag aus dem autoritativen DNS-Server. Bei Fehlern können Schritt 18 bis 26 beliebig wiederholt werden, da diese nicht kritisch für die Lauffähigkeit des Systems sind.

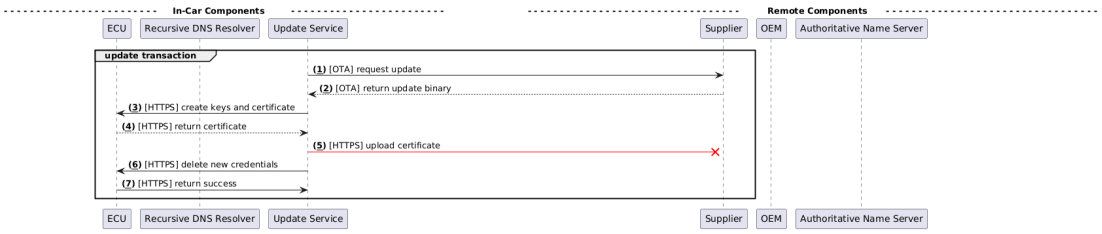

Abbildung 4.7: Fehler bei der Kommunikation mit dem Supplier

1 - 4 wie oben beschrieben

5 - 7 Der Supplier ist nicht erreichbar. Nach mehreren Retries wird das Update abgebrochen. Dafür werden die neu erstellten Credentials gelöscht.

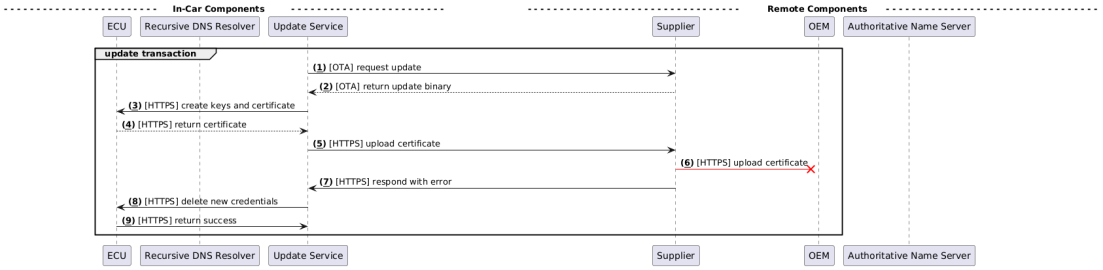

Abbildung 4.8: Fehler bei der Kommunikation mit dem OEM

1 - 5 wie oben beschrieben

6 Der Supplier kann den OEM auch nach mehreren Retries nicht erreichen. Der Supplier gibt eine Fehlermeldung zurück.

7 - 9 Das Update wird abgebrochen und die neuen Credentials gelöscht.

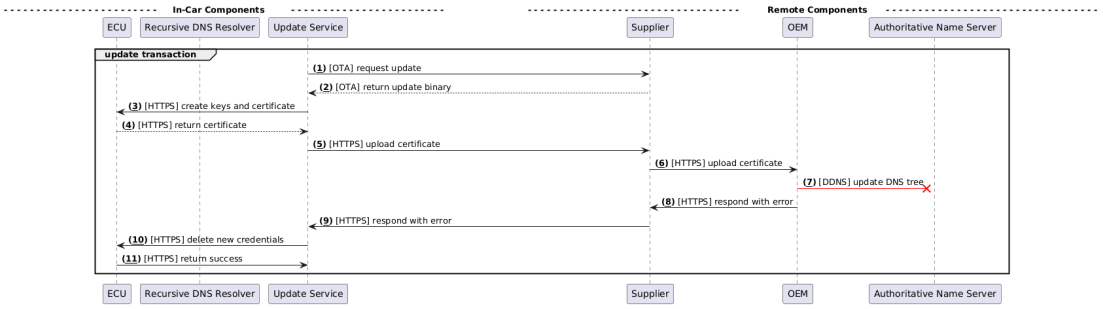

Abbildung 4.9: Fehler beim Updaten des autoritativen DNS-Servers

1 - 6 wie oben beschrieben

7 Der autoritative DNS Server kann wegen eines unbestimmten Grundes nicht geupdatet werden. Der OEM gibt einen Fehler an den Supplier und dieser an den Update Service zurück.

8 - 11 Das Update wird abgebrochen und die neuen Credentials gelöscht.

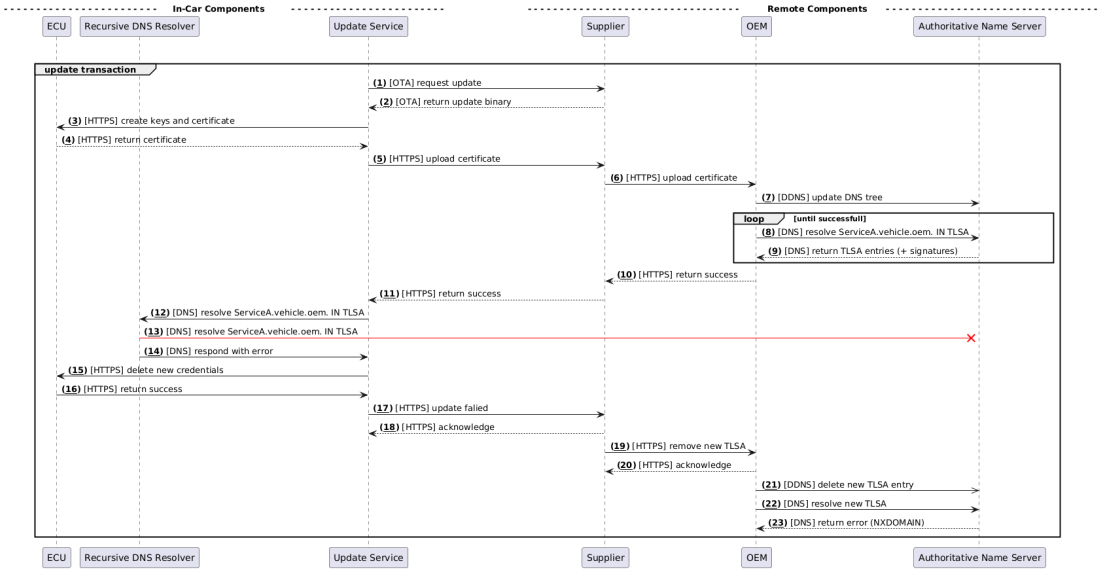

Abbildung 4.10: Fehler beim Erreichen des autoritativen DNS-Servers

1 - 12 wie oben beschrieben

13 Der Recursive Resolver kann den autoritativen DNS-Server auch nach mehreren Retries nicht erreichen.

14 - 16 Das Update wird abgebrochen und die neuen Credentials gelöscht.

17 - 23 Der Update Prozess meldet das fehlgeschlagene Update beim Supplier und gibt den Auftrag den neuen TLSA Eintrag zu löschen.

4.3 Alternatives Szenario - Softwarehersteller generiert Schlüsselpaar

Zusätzlich zu dem vorher beschriebenen Design, soll es noch ein weiteres geben. Bei diesem wird das Schlüsselpaar nicht auf der ECU, sondern vom Supplier erstellt. Diese Variante stimmt mit der Version von Solomon et al überein [20]. Zum Beginn des Updates ruft der Update Service eine entsprechende Route des Suppliers auf. Dieser erstellt anschließend ein Schlüsselpaar und ein Zertifikat, welches er an den OEM leitet. Schließlich stellt er dem Update Services das Schlüsselpaar und das Zertifikat zur Verfügung. Der Update Services validiert das Zertifkat über DNS und übergibt die Credentials an die ECU, welche diese, soweit möglich, in ihrem Sicheren Element speichert. Das Entfernen veralteter Credentials bleibt zwischen beiden Szenarien gleich.

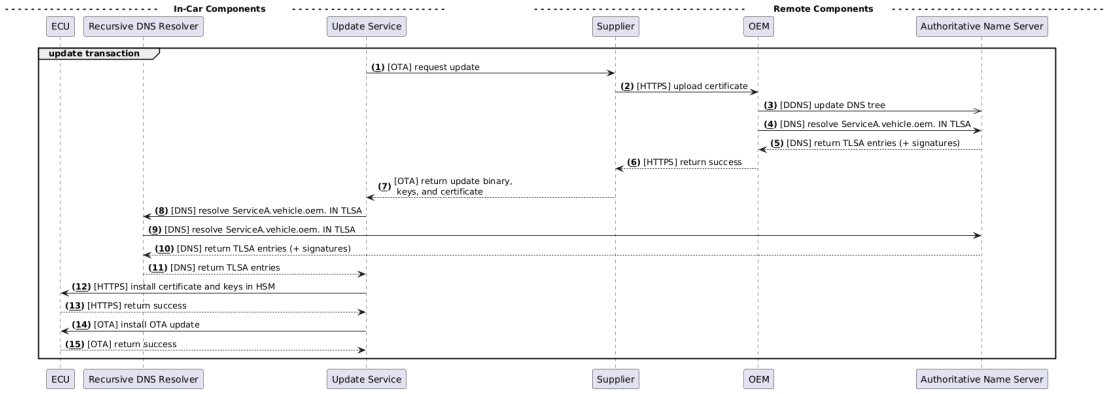

Abbildung 4.11: Ablauf ohne Fehler im alternativen Szenario

1 Der Update Service signalisiert dem Supplier, dass der Update Service ein Softwareupdate durchführen möchte.

2 Der Supplier erstellt ein Schlüsselpaar und ein Zertifikat. Das Zertifikat wird an den OEM gesendet.

3 Der OEM erstellt einen TLSA Eintrag aus dem Zertifikat und aktualisiert mit diesem den DNS-Baum über DDNS.

4 - 5 Der OEM fragt den neu erstellten TLSA vom autoritativen DNS-Server ab. Ist der Eintrag noch nicht im DNS-Baum geladen, wird der Schritt per Polling wiederholt, bis der Eintrag erfolgreich zurückgegeben wird.

6 Die HTTPS Antwort auf 2

7 Der Supplier stellt dem Update Service das Softwareupdate, sowie Schlüsselpaar und Zertifikat zur Verfügung.

8 Der Update Service initiiert die Abfrage des neuen TLSA Eintrag vom DNS Resolver.

9 - 10 Der DNS Resolver fragt bei dem DNS Server nach den passenden Einträgen und erhält eine Antwort.

11 Der DNS Resolver gibt die TLSA Einträge an den Update Service weiter.

12 - 13 Der Update Service lädt die Credentials in die ECU .

14 - 15 Der Update Service installiert das Update auf der ECU.

5 Implementierung

Die Implementierung soll alle Komponenten und deren Interaktionen emulieren. Dazu gehört auch, dass die Komponenten unterschiedliche Protokolle (HTTPS, DNS, etc.) nutzen können, dass sich die Komponenten in getrennten Netzwerken (In-Fahrzeug, Internet) befinden und dass die Komponenten getrennte Dateisysteme haben. Deswegen habe ich für die Implementierung Docker Compose gewählt. Dies bietet zusätzlich die Möglichkeit, Umgebungsvariablen zu setzen, sodass unterschiedliche Szenarien ohne Änderungen im Code selbst ausgeführt werden können, beispielsweise die Anzahl der ECUs oder die Anzahl der Updates, die durchzuführen sind. Die einzelnen Komponenten sollen als Container laufen und dazu brauchen sie Docker Images. Für jede Komponente müssen wir vorher folgende Bedingungen herausarbeiten:

1. Braucht die Komponente spezielle Software, z.B. um mit Zertifikaten oder kryptographischen Schlüsseln zu arbeiten, oder um ein spezielles Protokoll zu benutzen?

2. Skaliert die Software? Müssen mehrere Instanzen der Software gleichzeitig laufen, und wenn ja, wie viele? Wie viel Ressourcen verbraucht die Komponente? Ist die Performance einer Skriptsprache ausreichend hinsichtlich Arbeitsspeicher und Ausführungszeit oder sollte ein Programm in binärform genutzt werden?

3. Muss die Komponente vor dem Start der Emulation präpariert werden? Dazu könnte das Erstellen von Schlüsseln oder Zertifikaten zählen.

4. Zu welchem Netzwerk (intern, extern) gehört die Komponente und benötigt sie eine statische IP-Adresse?

5. Welche Dateien sollen auf dem Docker Image vorhanden sein und welche sollen später zur Laufzeit eingesetzt werden? Volumes in Docker erleichtern es, Dateien zwischen Containern zu verteilen, z.B. um Schlüssel zu verteilen oder Logs auszulesen. Volumes können auch genutzt werden, um Skripte in einen Container zu laden. Ändert sich der Code, muss nicht das Image neu gebaut werden.

Es ergeben sich 9 Container: setup, dns_server, dns_resolver, oem, supplier, ecu, wait, update_service, und exit. Diese sind verteilt auf zwei Netzwerke: vehicle und internet.

Container	Image	Network
setup	unbound	-
dns_server	bind9	internet
dns_resolver	unbound	vehicle + internet
oem	oem	internet
supplier	flask	internet
ecu	ecu	vehicle
wait	ubuntu	-
update_service	flask	vehicle
exit	unbound	vehicle + internet

Tabelle 5.1: Verwendete Images und Netzwerkkonfiguration

Der Container setup hat zur Aufgabe, die Umgebung für die anderen Container zu erstellen. Für alle Volumes gibt es zwei Versionen: „template“ und „mount“. Ersteres ist eine Vorlage, zweiteres beinhaltet die Verzeichnisse, die als Volumes in die Container gemounted werden. Am Anfang des Prozesses löscht der setup-Container alle Verzeichnisse in „mount“ und kopiert alle Verzeichnisse aus „template“ in „mount“. Dadurch bleibt die Umgebung reproduzierbar. Der setup-Container erstellt zusätzlich ein Schlüsselpaar, das genutzt werden kann, um den DNS Resolver herunter zu fahren. Dafür benötigt er Software, die sich auf dem unbound Image befindet.

Für den autoritativen DNS Server wird Bind9 benutzt. Der Server benötigt eine statische IP-Adresse, damit andere Container DNS Requests an ihn senden können. Um dynamische DNS Updates zu nutzen, muss vorher ein Passwort gesetzt werden. Dieses ist bereits in der Konfiguration gesetzt.

Der DNS Resolver wird mit Unbound betrieben. Unbound verbraucht wenig Ressourcen und prüft DNSSEC Signaturen. Unbound verfügt über einen Cache, dieser wird aber nicht persistent betrieben. Einträge werden nicht über den Prozess hinaus gespeichert, so wie es in vorherigen Kapiteln als Requirement hervorgehoben wurde. Grund hierfür ist, dass es in den Szenarien, die die Implementierung abbildet, nicht relevant ist. Die

Konfiguration, bestehend aus root-key, root hint und remote-control-key, werden zur Laufzeit gemountet. Der DNS Resolver benötigt ebenfalls eine statische IP Adresse.

Der HTTP Server des OEM nutzt ein eigenes oem-Image, das eine modifizierte Version des flask-Image ist. Es beinhaltet zusätzliche Software, mit der dynamische DNS Updates gesendet werden können. Der Container erhält ein Volume, welches das Python Skript mit Flask und die Schlüssel für den DNS Server beinhaltet.

Der Supplier Container fasst die Funktionalität des OTA Servers und des HTTPS Servers des Supplierts zusammen. Das Protokoll OTA wird bewusst nicht genutzt in der Implementierung, sondern durch eine simple HTTP Request ersetzt. Es würde zusätzliche Komplexität hinzufügen, ohne dass es eine relevante Funktion erfüllt. Der Fokus der Implementierung ist das Lifecycle Management, nicht das Herunterladen und Installieren einer Binärdatei. Die Aufrufe an die OTA Schnittstellen werden durch HTTP Schnittstellen ersetzt, die ihren Aufruf loggen. Bei den Zeitmessungen, die als Teil der Evaluation stattfinden, gehen wir somit von einer Binärdatei mit Größe von 0 Bytes aus. Der Supplier Container nutzt das Flask Image und erhält ein Volume mit dem passenden Python Skript.

Die ECU-Container können über die EC_AMOUNT Variable skaliert werden. Die Implementierung ist ausgelegt auf eine Anzahl von ECUs zwischen 1 und 1000. Um effizient zu skalieren, muss der Arbeitsspeicher der Container limitiert sein. Um die Limitierung einzuhalten, nutzt das ECU-Image nicht Flask, sondern eine eigens entwickelte Rust Binärdatei, die einen Single Threaded HTTP Server implementiert. Der Container nutzt SoftHSM, um die Schlüssel zu erstellen und zu speichern und OpenSSL, um Zertifikate zu erstellen. Alle ECU-Container mounten dasselbe Volume, welches ein Bash Skript zum Erstellen der Schlüssel und Zertifikate enthält.

Der Wait Container führt ein sleep Befehl für eine Sekunde aus. Dies gibt den obigen Containern Zeit um hochzufahren, bevor der Update Service auf ihre Schnittstellen zugreift. Diese Vorgehensweise ist einfach, weist jedoch Schwächen auf: es ist nicht garantiert, dass die Container nach Ablauf der Zeit fertig sind. Ein besseres Vorgehen wäre es, die Container über eine Schnittstelle auf ihren Status zu prüfen.

Nachdem alle obigen Container erfolgreich hochgefahren sind, startet der Update Service. Dieser nutzt ein Python Skript, welches für jede Iteration, angegeben durch die ITERATIONS Variable, und jede ECU ein Update durchführt. Über eine DNS Request kann der Update Service alle IP Adressen der ECUs erhalten. Der Update Service setzt

während des Updates mehrere Zeitstempel und logt nach dem Update die Zeiten in eine Logfile auf dem Volume (Siehe Evaluation).

Der Exit-Container hat die Aufgabe das System herunter zu fahren. Dazu nutzt er die GET exit/ Routen in den HTTP Servern und die Remote Controls des DNS Servers und des DNS Resolvers. Dafür benötigt der Container die passenden Schlüssel, welche über ein Volume gemountet werden.

Um sicher zu stellen, dass das System korrekt funktioniert, werden mehrere Schritte zur Validierung durchgeführt. Bereits während eines Updates werden die Response Codes der HTTPS Request überprüft, um den korrekten Zustand des Systems zu garantieren. Die zusätzlich zu prüfenden Nachbedingungen sind, dass die ECU-Container Zugriff auf das eigene, aktuelle Schlüsselpaar und Zertifikat haben, dass veraltete Schlüsselpaare und Zertifikate gelöscht wurden, dass der neue TLSA Eintrag verfügbar ist und alte TLSA Einträge nicht. Das überprüft der Update Service nach einem Update in einem zusätzlichen Schritt. Dieser beinhaltet das Aufrufen einer GET verify/ Route der jeweiligen ECU, welche ihr Schlüsselpaar und Zertifikat überprüft. Der Update Service sendet anschließend eine DNS Request an den DNS Resolver, mit einer Query an den alten TLSA Eintrag. Für die Validierung muss diese Request fehlschlagen, um sicher zu stellen, dass weder der DNS Resolver, noch der DNS Server, über den alten TLSA verfügen. Es wird keine zusätzliche DNS Request über den neuen TLSA Eintrag verschickt, weil dies schon als Teil des Updates geschah.

Zusätzlich gibt es einen Versuchsablauf, in welchem zufällig Fehler auftreten können. In diesem Fall muss das System die korrekte Fehlerbehandlung aufrufen, um erneut in einen validen Zustand zu gelangen. Anschließend wird erneut versucht das Update durchzuführen. Die Fehler können an den Schnittstellen zum Update Service auftreten und für den Versuch hat jeder Aufruf einer Schnittstelle eine Chance von 1% einen Fehler auszulösen. Dieser Versuch soll prüfen, ob das System sich nach einem oder mehreren Fehlern erholen kann. Dieser Versuch geschieht getrennt von Versuchen, welche die Performance prüfen. In diesen können keine zufälligen Fehler auftreten.

Das alternative Szenario ist realisiert durch einen Git-Branch. In dieser Version sind der Update Service und die Routen des Suppliers angepasst, sowie das Image der ECU. Die Credentials werden nicht in SoftHSM geladen, sondern als Dateien im ECU Container gespeichert. SoftHSM wird in diesem Fall nicht verwendet, da die Funktion externe Credentials zu laden, nicht von allen Chips unterstützt wird [1].

6 Evaluation

Während der Emulation wurden im Laufe der Verifikationsschritte keine Fehler festgestellt, demnach sind die Nachbedingungen als erfüllt zu betrachten. Diese sind, dass dem Service auf der ECU das entsprechende Schlüsselpaar zur Verfügung steht, der DNS-Resolver den entsprechende TLSA-Eintrag geladen hat und die Credentials der vorherigen Version sowohl von der ECU als auch vom autoritativen DNS-Server gelöscht wurden. Im Versuch mit zufällig auftretenden Fehlern hat das System es erfolgreich geschaft, alle Updates zu installieren und sich von Fehlern zu erholen.

Um die Implementierung zu bewerten, betrachten wir die Installationszeit, die Aktualisierungszeit und den genutzten Speicherplatz. Installationszeit beschreibt die Zeit, die gebraucht wird, um Credentials für einen Service zu erstellen, den es vorher noch nicht gab. Die Aktualisierungszeit gibt an, wie lange es dauert, Credentials für die neue Version eines Service zu erstellen, sowie die alten Credentials zu löschen. Wir betrachten auch wie diese Zeiten mit der Anzahl der ECUs skalliert. Für den Versuch werden eine feste Anzahl an Services nacheinander installiert. Sobald alle Services installiert sind werden alle nacheinander aktualisiert. Der Versuch wird durchgeführt mit 1, 50, 100, 250, 500 und 1000 Services. Während des Versuchs können keine zufälligen Fehler auftreten.

Um die Zeiten zu messen, werden im Supplier und im OEM die Zeiten der Routen geloggt und im Update Service mehrere Zeitstempel gespeichert und die Differenzen dieser in einer CSV-Datei geloggt. Die Abschnitte die gemessen werden sind: das Erstellen des Schlüsselpaares sowie des Zertifikates, das Hochladen des Zertifikates zum Supplier, das Laden und Verifizieren des TLSA Eintrags und das Löschen der veralteten Credentials. In dem alternativen Szenario werden dieselben Abschnitte gemessen und mit dem selben Spaltennamen gespeichert. Da der Ablauf jedoch unterschiedlich ist, entstehen dadurch unterschiedliche Zeiten. Z.B. wird in dem Abschnitt namens „credential_creation“ bei Vehicle-Generated Credentials die Schnittstelle der ECU zum Erstellen von Schlüsselpaar und Zertifikat aufgerufen. Bei Supplier-Generated hingegen, ruft der Update-Service die Schnittstelle des Suppliers zum Erstellen der Credentials auf, welcher diese dann auch

an den OEM weiterleitet. Welcher Abschnitt welchem Schritt entspricht, lässt sich der Tabelle 6.1 entnehmen.

Abschnitt	Vehicle-Generated	Supplier-Generated
certificate_creation	3 - 4	1 - 7
certificate_upload	5 - 11	12 - 13
dns_query	12 - 15	8 - 11

Tabelle 6.1: Abschnitte in unterschiedlichen Szenarien.
Schritte in Vehicle-Generated entsprechen den Schritten in Abbildung 4.6, die in Supplier-Generated der Abbildung 4.11

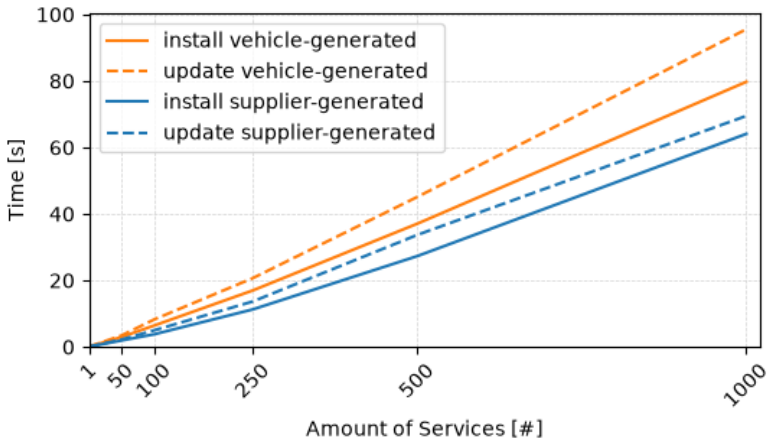

Abbildung 6.1: Installationszeit und Aktualisierungszeit für X Services

Abbildung 6.1 zeigt, wie die Installationszeit und die Aktualisierungszeit mit der Anzahl der Services skaliert. Es handelt sich hierbei um die Gesamtzeit X Services zu installieren oder zu aktualisieren. Es ist zu erkennen, dass die Zeiten linear mit der Anzahl der Services wachsen und dass die Variante Supplier-Generated weniger Zeit benötigt als Vehicle-Generated.

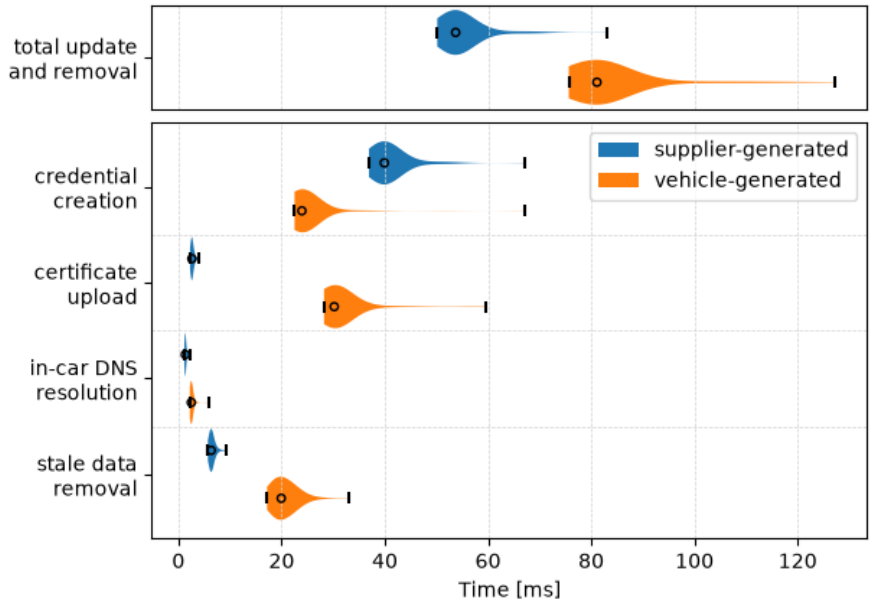

Abbildung 6.2: Abschnitte der Aktualisierungszeit

Abbildung 6.2 zeigt die unterschiedlichen Abschnitte der Aktualisierungszeit, in beiden Szenarien. Da es sich um einen Violin Plot handelt, sieht man darauf nicht nur einzelne Werte, sondern einen Datensatz, mit Minimum, Maximum und Durchschnitt. Der Datensatz, der hier gezeigt wird, stammt aus einem Updatezyklus mit 250 Services. Auch in diesem Diagramm ist zu erkennen, dass die Version Vehicle-Generated mehr Zeit benötigt als Supplier-Generated. Grund hierfür ist, dass Supplier-Generated kein SoftHSM verwendet und somit die Abschnitte, die Zugriff auf die Credentials der ECU benötigen deutlich schneller sind. Aus den Logs geht hervor, das ein Aufruf von SoftHSM ungefähr 10 ms zusätzlich dauert. Das deckt sich mit den Zeiten, die hier zu erkennen sind, da Vehicle-Generated ungefähr 30 ms länger braucht und in einem Durchlauf drei mal SoftHSM aufruft. Im Abschnitt „credential_creation“ hat Vehicle-Generated zwar einen niedrigeren Wert, das liegt jedoch daran, dass der Upload dort nicht einbezogen wird. Diesen kann man bei „certificate_upload“ sehen.

	Vehicle-Generated		Supplier-Generated	
Role	1 Update	250 Update	1 Update	250 Update
Vehicle	25ms (51%)	30ms (50%)	5ms (13%)	6ms (13%)
Supplier	3ms (5%)	3ms (5%)	11ms (29%)	13ms (26%)
OEM	21ms (43%)	27ms (45%)	23ms (58%)	29ms (60%)
Total	50ms (100%)	59ms (100%)	39ms (100%)	48ms (100%)

Tabelle 6.2: Zeiten gerundet und summiert nach Teilnehmer.

Tabelle 6.2 zeigt die Aktualisierungszeit nach Komponenten verteilt. Es handelt sich hier um durchschnittliche Werte. Man kann ablesen, dass sich, wie zu erwarten, die Zeiten von Vehicle im Falle von Vehicle-Generated deutlich höher sind, als im Supplier-Generated Szenario. Dasselbe gilt für den Supplier, jedoch im umgekehrten Sinne. Der Supplier verbraucht mehr Zeit im Szenario Supplier-Generated, als im Szenario Vehicle-Generated. Die Zeiten des OEMs sind szenarienübergreifend gleichbleibend.

Um den Verbrauch des Speicherplatzes zu bewerten, habe ich die Anzahl der Einträge zusammen mit der Größe der Zonefile des autoritativen Nameserver gespeichert. Aus dem ersten und dem letzten Wert habe ich mit einer Geradengleichung die Bytes pro Eintrag sowie den Overhead berechnet. Es ergibt sich 1952 Bytes pro Eintrag, sowie 7401 Bytes Overhead. Rechnet man dies nun auf 1000 Services für ein Fahrzeug, so erhält man ungefähr 2 Megabyte pro Fahrzeug.

Da der Versuch auf limitierter Hardware durchgeführt wurde und es durch die Nutzung bzw. die Nicht-Nutzung von SoftHSM zu größeren Unterschieden zwischen den Szenarien kam, ist es schwer zu sagen, wie die Zeiten eines realen Systems aussehen würden. Es ist jedoch davon auszugehen, dass auch in einem realen System die Zeit unter 200 ms liegen würde. Die Zeiten, die nicht betrachtet wurden, OTA-Download und OTA-Install, transportieren größere Datenmengen und würden die Zeiten merkbar erhöhen. Zusätzlich sollte es für einen Fahrzeughersteller kein Problem darstellen 2 Megabyte pro Fahrzeug bereitzustellen.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

In dieser Arbeit wurde eine Emulation einer Zertifikatsverwaltung basierend auf dem Design von Salomon et al. implementiert und evaluiert. Es wurden die Umstände betrachtet, die ein solches System notwendig machen. Es wurde analysiert, welche Akteure daran teilnehmen und wie ihre Anwendungsfälle aussehen. Das Kapitel Design zeigt die Komponenten und ihre Schnittstellen, sowie Abläufe und Fehlerszenarien. Im Kapitel Implementierung wurde auf die Realisierung der Emulation eingegangen und auf die verwendeten Docker-Images, Docker-Container und die Konfiguration des Systems. In der Evaluation wurden die Installationszeiten und die Aktualisierungszeiten in beiden Szenarien betrachtet. Diese blieben unter 100 ms. Unterschiede in den Szenarien sind auf SoftHSM zurückzuführen. Der Speicherplatzverbrauch des DNS beschränkt sich auf ungefähr 2 MB für 1000 Services.

7.2 Ausblick

Da das DNS-Verzeichnis eine detaillierte Auskunft über die in einem Fahrzeug installierte Software gibt, ist es wichtig zu regulieren, wer Zugriff auf dieses Verzeichnis hat. Ein Angreifer könnte andernfalls nach Software mit bekannten Sicherheitslücken suchen, um diese anschließend auszunutzen. Aus diesem Grund müsste eine Implementierung in einem Produktivsystem DNS-Queries an den autoritativen Nameserver nur dann zulassen, wenn sie vom Resolver des Fahrzeugs oder vom OEM selbst stammen.

Der Update Service bietet durch seine Berechtigungen Potenzial für Denial-of-Service-Angriffe. Sollte ein Angreifer Remote-Code-Execution auf dem Update Service erreichen, könnte er die Schnittstellen zum Löschen von Credentials ausnutzen, um die Authentifikation innerhalb des Fahrzeugs zu stören. Zusätzlich müssen die Lebensdauern der

Zertifikate betrachtet werden. Das beinhaltet die Lebensdauern der Zertifikate selbst und die Lebensdauern der entsprechenden RRSIGs. Fahrzeughersteller müssen passende Lebensdauern wählen, um Sicherheit zu gewährleisten und die Offline-Fähigkeit des Systems zu wahren. Für den Fall, dass ein Fahrzeug über längere Zeiträume vom Internet abgeschnitten ist, muss es eine Fallback-Option geben.

Weitere Forschung könnte betrachten, wie sich die Performance mit tatsächlicher Hardware, d.h. mit echten ECUs verhält und wie sich die Fehlerbehandlung auf die Performance auswirken.

Des weiteren könnten auch andere Systeme diese Art des agilen Credentials Managements verwenden. Es eignet sich für geschlossene Systeme mit mehreren Komponenten, die sich gegenseitig authentifizieren müssen und von unterschiedlichen Zulieferern stammen.

Literaturverzeichnis

[1] Anderson, Julie: *ATECC608A CryptoAuthentication Device Summary Data Sheet*. 2018. – URL https://ww1.microchip.com/downloads/aemDocuments/documents/SCBU/ProductDocuments/DataSheets/ATECC608A-CryptoAuthentication-Device-Summary-Data-Sheet-DS40001977B.pdf. – Zugriffsdatum: 31.09.2026

[2] Arends, R. ; Austein, R. ; Larson, M. ; Massey, D. ; Rose, S.: *DNS Security Introduction and Requirements*. RFC 4033. März 2005. – URL https://www.rfc-editor.org/info/rfc4033/

[3] Atkins, D. ; Austein, R.: *Threat Analysis of the Domain Name System (DNS)*. RFC 3833. August 2004. – URL https://www.rfc-editor.org/info/rfc3833/

[4] Autosar: *Explanation of Firmware Over-The-Air*. R25-11. 2025. – URL https://www.autosar.org/fileadmin/standards/R25-11/CP/AUTOSAR_CP_EXP_FirmwareOverTheAir.pdf. – Zugriffsdatum: 2026-08-27

[5] Autosar: *SOME/IP Protocol Specification*. R25-11. 2025. – URL https://www.autosar.org/fileadmin/standards/R25-11/FO/AUTOSAR_FO_PRS_SOMEIPProtocol.pdf. – Zugriffsdatum: 2026-09-16

[6] Autosar: *SOME/IP Service Discovery Protocol Specification*. R25-11. 2025. – URL https://www.autosar.org/fileadmin/standards/R25-11/FO/AUTOSAR_FO_PRS_SOMEIPServiceDiscoveryProtocol.pdf. – Zugriffsdatum: 2026-09-16

[7] Barnes, R.: *Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE)*. RFC 6394. Oktober 2011. – URL https://www.rfc-editor.org/info/rfc6394/

[8] Häckel, Timo ; Meyer, Philipp ; Mueller, Mehmet ; Schmitt-Solbrig, Jan ; Korf, Franz ; Schmidt, Thomas C.: Dynamic Service-Orientation for Software-Defined In-Vehicle Networks. In: *Proc. of the IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, IEEE, June 2023, S. 1–5. – URL https://doi.org/10.1109/VTC2023-Spring57618.2023.10199712

[9] Ham, Jeroen Van D.: Toward a Better Understanding of "Cybersecurity". In: *Digital Threats* 2 (2021), Juni, Nr. 3. – URL https://doi.org/10.1145/3442445

[10] Hoffman, P.: *DNS Security Extensions (DNSSEC)*. RFC 9364. Februar 2023. – URL https://www.rfc-editor.org/info/rfc9364/

[11] Hoffman, P. ; Schlyter, J.: *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. August 2012. – URL https://www.rfc-editor.org/info/rfc6698/

[12] IEA: Vehicle software and software-defined vehicles. Paris : IEA, 2026. – Forschungsbericht. – URL https://www.iea.org/reports/vehicle-software-and-software-defined-vehicles. – Zugriffsdatum: 2026-08-27

[13] Iorio, Marco ; Reineri, Massimo ; Risso, Fulvio ; Sisto, Riccardo ; Valenza, Fulvio: Securing SOME/IP for In-Vehicle Service Protection. In: *IEEE Transactions on Vehicular Technology* 69 (2020), November, Nr. 11, S. 13450–13466

[14] International Organization for Standardization (ISO): Road vehicles – Cybersecurity engineering / ISO/SAE. August 2021 (ISO/SAE 21434:2021). – Standard

[15] Lee, Seulhui ; Choi, Wonsuk ; Lee, Dong H.: Protecting SOME/IP Communication via Authentication Ticket. In: *Sensors* 23 (2023), Juli, Nr. 14, S. 6293

[16] Miller, Charlie ; Valasek, Chris: Remote Exploitation of an Unaltered Passenger Vehicle. In: *Black Hat USA* (2015), S. 1–93. – URL https://ioactive.com/wp-content/uploads/2018/05/IOActive_Remote_Car_Hacking-1.pdf. – Zugriffsdatum: 2026-08-27

[17] NIST: Recommendation for Key Management: Part 1 – General / NIST. Washington, D.C., Mai 2020 (SP 800-57 Part 1 Rev. 5). – Forschungsbericht

[18] Object Management Group (OMG): Data Distribution Service / OMG. URL https://www.omg.org/spec/DDS/1.4, März 2015 (DDS 1.4). – Standard

[19] Salomon, Timo ; Meyer, Philipp ; Schmidt, Thomas C.: POSTER: DNSSEC in the Car – Towards an Agile Management of Automotive Service Security. In: *Proc. of ACM SIGCOMM: Posters and Demos.* New York : ACM, 2025, S. 43–45. – URL https://doi.org/10.1145/3744969.3748414

[20] Salomon, Timo ; Mueller, Mehmet ; Meyer, Philipp ; Schmidt, Thomas C.: Building Automotive Security on Internet Standards: An Integration of DNSSEC, DANE, and DANCE to Authenticate and Authorize In-Car Services / Open Archive: arXiv.org. URL https://arxiv.org/abs/2506.13261, June 2025 (2506.13261). – Technical Report

A Anhang

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments
DeepL	Übersetzung des Abstract
Plantuml	Erstellung von Diagrammen
Matplotlib	Erstellung von Diagrammen

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit in allen Teilen selbstständig angefertigt und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt habe. Die in meiner Arbeit verwendeten KI-basierten Hilfsmittel habe ich (ggf. mit Produktnamen) angegeben.

Ich verantworte die Übernahme jeglicher von mir verwendeter maschinell generierter Passagen vollumfänglich selbst und trage die Verantwortung für eventuell durch die KI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate.

Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschungsversuch behandelt werden können.

Ort	Datum	Unterschrift im Original