Kandidatexamen

Undersökning av säkerhet vid kontroll och konfiguration av ett fordon från en mobilapplikation

Av

Emil Nielsen och Mhd Rami Alhoumsi

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Sammanfattning

Att kunna konfigurera och/eller kontrollera ett fordon genom en mobilapplikation är något som blir allt vanligare i den värld vi lever i. Det finns därav ett behov av att undersöka hur säkerheten hanteras i just detta scenario.

Denna kandidatuppsats är en undersökning i form av en litteraturstudie kring vilka risker och hot som finns då ett fordon ska kunna konfigureras och kontrolleras genom en mobilapplikation. Utifrån dessa hot och risker har det tagits fram en lämplig säkerhetslösning. Denna säkerhetslösning har sedan implementerats och verifierats i form av en prototyp.

I denna fallstudie används en Bluetoothuppkoppling för att kommunicera mellan en mobilapplikation och ett fordon. Säkerhetslösningen integrerar Google Firebase framförallt för att hantera systemets användare.

Resultatet av detta examensarbete är en fördjupad förståelse för vilka hot och risker som finns då ett fordon kontrolleras och konfigureras genom en mobilapplikation samt hur dessa rimligtvis bör hanteras. Utöver det har den säkerhetslösning som har tagits fram genom detta examensarbete förbättrat säkerheten för OMotion, deras fordon och deras kunder.

Nyckelord: Säkerhet, Fordon, Mobilapplikation, Litteraturstudie, Bluetooth, Google Firebase.

Abstract

To be able to control and/or configure a vehicle through a mobile app is something that is becoming more usual in the world we live in. Because of this there is an inherent demand to further explore how to handle the security in this scenario.

This Bachelor thesis is an investigation in the shape of a literature study into which risks and threats that appear when a vehicle is to be controlled and configured through a mobile app. Based on the risks and threats identified a reasonable security solution has been provided. This solution has then been implemented and verified through a prototype.

In this case study a Bluetooth connection is used to connect and communicate between a mobile app and a vehicle. The security solution integrates Google Firebase above all to handle the system's users.

The result of this degree project is an improved understanding of which risks and threats that appear when a vehicle is to be controlled and configured through a mobile app and how these should be handled. In addition to that the security solution that has been provided has improved the security for OMotion, their vehicles and their customers.

Keywords: Security, Vehicle, Mobile app, Literature Study, Bluetooth, Google Firebase.

Förord

Vi vill tacka OMotion AB och Ola Svensson för möjligheten att göra vårt examensarbete hos dem. Vi vill även tacka Ola för den hjälp och vägledning vi har fått under arbetet med att förstå hur OMotions fordon är uppbyggda och hur ECU:n fungerar samt lånet av en ECU för att kunna utveckla och testa hemifrån.

Vi vill även tacka vår handledare Christian Gehrmann för den vägledning och hjälp vi har fått under arbetet samt Thomas Johansson som var examinator för detta examensarbete.

Emil Nielsen, Mhd Rami Alhoumsi

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Detta examensarbete sker i samarbete med företaget OMotion AB. OMotion grundades 2013 av Ola Svensson och befinner sig strax utanför Lund. OMotion har byggt ett 3-hjuligt elektriskt fordon. Till fordonet har OMotion utvecklat en mobilapplikation för att hantera grundläggande funktioner i fordonet. Mobilapplikationen använder i sin tur Bluetooth för att koppla upp sig mot fordonet. Den mobilapplikation som redan finns har ingen säkerhet, det finns ingen hantering av användare och därmed inte heller någon hantering av vilka fordon och funktioner en specifik användare skall ha tillgång till. Det finns inte heller någon säkerhet kring Bluetooth uppkopplingen implementerad från fordonets sida. Det innebär i praktiken att i dagsläget kan vem som helst ansluta sig till ett OMotion fordon och komma åt känslig data. Det finns därav ett behov av att närmare undersöka vilka risker som finns och den säkerhet som krävs när ett fordon konfigureras och kontrolleras genom en mobilapplikation. I detta arbete kommer vi undersöka detta problem närmare för att förstå de viktigaste hoten och hur de kan undvikas. Som fallstudie kommer vi att använda systemet som OMotion har tagit fram.

1.2 Syfte

Syftet med examensarbetet är att utveckla en mobilapplikation som kan kommunicera med ett fordon där en lämplig säkerhetslösning skall tas fram och implementeras. Det förväntade resultatet är en mobilapplikation som skall kunna kopplas till fordonet och hantera säkerhet och åtkomstnivåer genom en molndatabas, där endast auktoriserade enheter kan komma åt fordonets data baserat på användarens åtkomstnivå. Exempelvis OMotion personal, ägaren av bilen eller en gäst.

1.3 Målformulering

Examensarbetet är en undersökning i vad som kan vara en lämplig säkerhetslösning då ett fordon skall kontrolleras och konfigureras genom en mobilapplikation. Utifrån den

undersökning som görs skall en lämplig säkerhetslösning tas fram i förhållande till OMotions situation. Denna säkerhetslösningen skall sedan verifieras med en prototyp.

1.4 Problemformulering

Vår uppgift är att hitta en säkerhetslösning då ett fordon ska konfigureras och kontrolleras av en mobilapplikation. Det ska finnas olika roller som har olika behörigheter. Säkerheten består till stora delar av autentisering (kontroll av att användaren är den som de uppger sig att vara) och auktorisering (kontroll av vilka behörigheter den specifika användaren har).

Frågor som kommer att besvaras genom examensarbetet är följande:

1. Hur ska säkerhet och åtkomst på fordonsdata hanteras på ett smart sätt?
2. Vilken information och data behövs i databasen?
3. Hur kan säkerhetslösningen vi tar fram förbättra säkerheten för OMotions fordon, mobilapplikation och användare?

Dessa frågor tas upp och diskuteras i diskussionsdelen av denna rapport.

1.5 Motivering

OMotions arbete med att ta fram en 3-hjulig elbil som ger friheten av en motorcykel och stabiliteten av en bil är något som är ganska unikt. Företaget är fortfarande ungt och är i en tillväxtfas. Vi ser detta som en perfekt möjlighet för oss att hjälpa ett ungt företag i sin resa mot toppen samtidigt som vi får en chans att ta vår examen.

Vi har valt att göra detta examensarbetet då det ger oss en möjlighet att kombinera alla kunskaper vi har fått ta del av under utbildningen samtidigt som projektet i sig är väldigt intressant.

1.6 Angreppssätt och metoder

För att uppnå våra mål har vi gjort en litteraturstudie där vi har analyserat vilka hot och risker som finns i vårt scenario. Utifrån dessa hot och risker har vi tagit fram en lämplig säkerhetslösning. Denna lösning har sedan implementeras och verifierats i form av en prototyp.

1.7 Avgränsingar

- Vårt projekt innefattar endast utveckling av en mobilapplikation och uppbyggning/konfiguration av databasen som hanterar säkerheten, *inte* en desktop applikation eller någon utveckling på fordonet.
- Applikationen kommer att utvecklas i en multiplattform miljö och kommer därför troligtvis att fungera på andra enheter också men begränsningar i test möjligheter gör att vi endast kan garantera mobilapplikationens funktion på Android 9 och nyare versioner.

2 Metod

I detta kapitel beskrivs de olika delarna av detta examensarbete samt hur vi har arbetat med dessa. Delarna är inte nödvändigtvis i kronologisk ordning då vi delvis undersökte och utvärderade olika lösningsalternativ samtidigt som vi gjorde vår litteraturstudie.

2.1 Litteraturstudie

En stor del av detta examensarbete bestod av litteraturstudier för att närmare undersöka vilka säkerhetsrisker och hot som finns vid det scenario som behandlas i detta arbete. Det har även gjorts en undersökning kring vilka olika tekniker och resultat som finns som går att tillämpa på detta scenario. Utöver det krävdes det en hel del studier för att sätta oss in i arbetsmiljön Flutter och Google Firebase.

2.2 Undersökning av lösningsalternativ

Utifrån de krav som finns på systemet samt de hot och risker som identifierades genom litteraturstudien har det gjorts en undersökning för att ta fram lösningsalternativ för att hantera de olika hot och risker som har identifierats.

2.3 Utvärdering av lösningsalternativ

De alternativ som har tagits fram genom undersökningen har kontinuerligt utvärderats genom teoretisk analys. Därefter har de olika delarna kopplats ihop och tillsammans bildat säkerhetslösningen. Denna säkerhetslösning har sedan implementerats och verifierats i form av en prototyp.

2.4 Avslutande analys och rekommendation till lösning

Det har gjorts en avslutande analys av prototypen samt det arbete som har gjorts. Utifrån denna analys har en rekommendation till framtida lösning presenterats.

3 Teknisk Bakgrund

Detta kapitel består av en övergripande teknisk bakgrund av de system som berörs av detta arbete. Längre fram i rapporten beskrivs dessa tekniker i detalj och hur de används.

3.1 OMotion 2

Fallstudien som detta arbete bygger på använder till stora delar det fordon som har tagits fram av OMotion AB. Fordonet är den andra generationens OMotion fordon och har därav fått namnet OMotion 2. Fordonet är en elektrisk 3-hjuling, se appendix 1 för bild på fordonet.

3.1.1 Styrenhet

OMotion 2 består precis som alla andra fordon av en mängd olika delar som fyller diverse olika funktioner. Styrenheten är den s.k. “hjärnan” i fordonet. Det är den som får alla andra delar att fungera tillsammans som en enhet. Det är också den del som vi kommer att arbeta mot i detta examensarbete. På styrenheten sitter ett Bluetoothchip som ger oss möjligheten att kommunicera med fordonet via Bluetooth. Styrenheten kommer hädanefter refereras till som ECU:n (Engine Control Unit).

3.2 Bluetooth

I detta arbete använder vi Bluetooth för att koppla en mobilapplikation till ett fordon. Bluetooth 1.0 släpptes ursprungligen 1999 och har sedan dess utvecklats genom åren. Den senaste versionen Bluetooth 5.0 släpptes 2016 och de stora utvecklingsområdena har varit räckvidd, överföringshastighet och batterianvändning. Bluetooth 1.0 hade en maximal överföringshastighet på 1 Mb/s och en räckvidd på max 10 meter. Bluetooth 5.0 erbjuder en rad olika överföringshastigheter som i sin tur påverkar räckvidden där den maximala är 2 Mb/s som används vid kortare avstånd medan en lägre överföringshastighet kan uppnå en maximal räckvidd på 240 meter. Bluetooth kommer generellt sett i tre olika klasser som är anpassade för olika räckvidd och överföringshastighet (Klass 1, ca 100m; Klass 2, ca 10m; Klass 3, ca 5m) [1].

3.3 Google Firebase

Google Firebase[1] är ett ramverk för att hantera diverse back-end funktioner, exempelvis erbjuds funktioner för autentisering och hantering av användare, lagring av data i databaser, användning av push notiser samt en mängd andra funktioner. Dessa funktioner och Firebase som helhet diskuteras vidare i vår litteraturstudie under kapitel 4.3.1 Google Firebase. Google Firebase föreslogs av OMotion som ett alternativ för att hantera autentiseringen av användare i mobilapplikationen och är även det alternativ vi har valt för att implementera autentiseringen av användare.

3.4 Flutter

Som en del av arbetet med att ta fram en säkerhetslösning har vi även utvecklat en prototyp av mobilapplikationen. Denna prototyp har utvecklats i Flutter[2]. Flutter är ett front-end ramverk som bygger på programmeringsspråket Dart. Flutter erbjuder genom sin "MaterialApp" klass ett enkelt och logiskt sätt att utveckla applikationer för flera olika plattformar samtidigt, däribland Android, IOS och Webb samt desktop.

3.5 Fota-Tool

Fota-tool är ett signerings- och krypteringsverktyg för inbyggda system som använder RSA-PSS och AES-128-CBC. RSA-PSS är en algoritmsvit som bygger på RSA algoritmen och som endast stöder signering, verifiering och nyckelgenerering med Probabilistic signature scheme(PSS) padding. Fota-tool krypterar enligt AES-128-CBC algoritmen. Cipher block chaining (CBC) är en metod som används för att kryptera data i block för att producera en krypterad text som kallas ciphertext med hjälp av en kryptografisk nyckel och algoritm. I detta fall används CBC med 128 bitar för Advanced Encryption Standard(AES) algoritmen. I denna fallstudie används Fota-tool för att hantera kryptering och signering av de mjukvaruuppdateringar som skall kunna göras på fordonet genom mobilapplikationen. Då Fota-tool verktyget endast är något vi har integrerat, *inte* implementerat själva, kan vi endast beskriva hur det fungerar utifrån den dokumentation som finns. Det betyder att vi inte kan beskriva exakt i detalj hur det fungerar. Det vi kan beskriva är hur det integreras

[1] https://firebase.google.com/
[2] https://flutter.dev/

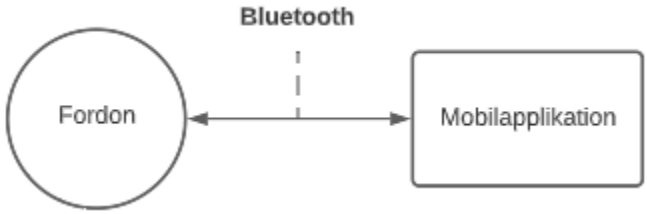
och används i vår säkerhetslösning. Detta görs under kap. 5.4 Mjukvaruuppdatering på fordonet. Fota-tool är ett egenutvecklat verktyg av OMotion. Detta gör att det inte finns någon publik information som vi kan hänvisa till gällande dokumentation av Fota-tool.

4 Litteraturstudie

Innan vi kunde börja ta fram en säkerhetslösning gjorde vi en fördjupande litteraturstudie kring vilka hot och risker som finns samt hur man vanligtvis hanterar dessa. Vi har analyserat ett antal olika förslag till delar av lösningen och jämfört hur de passar in i vår situation. Vi har analyserat vilka risker och hot som finns vid användning av Bluetooth för att koppla enheterna. Därefter har vi analyserat hur de olika delarna av lösningen skall se ut och implementeras.

4.1 Översikt över systemet

För att kunna identifiera vilka hot och risker som finns mot ett system måste man först förstå hur systemet är uppbyggt, vilka entiteter som finns samt hur dessa är sammankopplade. Figur 1 visar en översikt över det system som finns sedan tidigare.

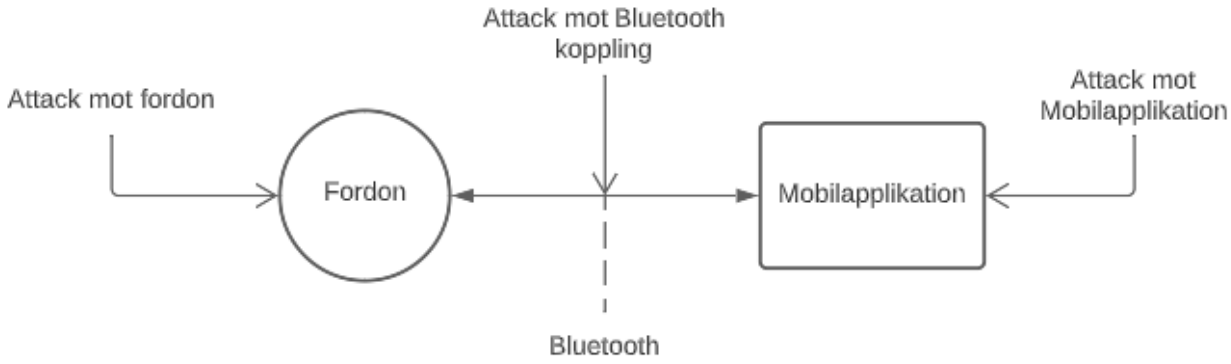


Figur 1: Översikt över det system som finns sedan tidigare

Mobilapplikationen används av OMotion och deras kunder för att kunna kommunicera med OMotions fordon. Detta uppnås genom en Bluetooth uppkoppling. Det finns sedan tidigare ingen implementerad säkerhet eller hantering av användare i detta system varken i fordonet eller mobilapplikationen. Alltså kan alla användare (inkl. hackare) ansluta till alla fordon. Detta är förstås ingen bra situation alls. Genom denna litteraturstudie tittar vi närmare på mer exakt vilka hot och risker som finns samt hur dessa kan hanteras.

4.2 Hot & risker

4.2.1 Översikt



Figur 2: Översikt över de attacker som kan ske mot systemet

För att förstå hur en säkerhetslösning skall byggas upp behöver vi först förstå vilka hot och risker som finns. I figur 2 visas en översiktlig bild av vilka attacker som teoretiskt kan ske mot vårt system. I detta kapitel kommer vi närmare undersöka hur dessa attacker sker samt hur vi kan skydda vårt system mot dem.

4.2.2 Bluetooth

När det kommer till Bluetooth så kan man generellt tänka sig att det finns två olika typer av attacker. En attack skulle i teorin kunna ske mot en Bluetoothuppkoppling mellan två andra enheter. Detta är en så kallad “man in the middle” attack. Detta hanteras delvis genom att Bluetooth 2.1(släpptes 2007) gjorde krypteringen av Bluetoothkopplingen mellan två enheter till obligatorisk istället för valfri [1]. Utöver det krävs det försiktighet vid användning, exempelvis bör man stänga av Bluetoothkopplingen när man inte aktivt använder den för att minimera risken att bli hackad.

Det andra hotet när det kommer till Bluetooth är de attacker som sker mot en annan enhet via en Bluetoothuppkoppling. Dessa attacker finns i en mängd olika former och utförande, under de kommande kapitlena undersöker vi närmare några av de vanligaste av dessa attackerna samt analyserar hur dessa är kopplade till detta arbete.

4.2.2.1 BlueSmacking

BlueSmacking är en Denial of Service attack mot Bluetoothenheter. BlueSmack attacken äger rum i L2CAP(Logical Link Control and Adaption Protocol) lagret i Bluetooths protokoll stack. I L2CAP lagret finns en möjlighet att begära och ta emot ping från andra Bluetoothenheter. Detta används vanligtvis för att kolla om en koppling finns samt att mäta tiden det tar för ett paket att skickas till enheten och återvända tillbaka. Varje enhet har en gräns på hur stort det här paketet får lov att vara och om paketet är för stort kommer den mottagande enheten att krascha. Detta ger utrymme för en hackare att utföra en Denial of Service attack mot en enhet genom att skicka paket som är för stora [2].

4.2.2.2 BlueJacking

BlueJacking är en attack som bygger på förmågan att skicka meddelanden till andra Bluetoothenheter genom OBEX(OBject EXchange) protokollet [3]. Vanligtvis används OBEX för att skicka filer mellan två enheter och designades ursprungligen för infraröd kopplingar mellan enheter. OBEX har sedan adopterats av bl.a. Bluetooth [4]. BlueJacking använder OBEX protokollet för att skicka t.ex. Spam eller länkar som vid klick kan vara skadliga för den mottagande enheten. Det är relativt enkelt att utföra en BlueJack attack men är inte så vanligt längre [5].

4.2.2.3 BlueSnarfing

BlueSnarfing är likt BlueJacking en attack som utnyttjar OBEX protokollet. Skillnaden här är att istället för att skicka filer så hämtar man filer och data. D.v.s man stjäler data från en annan Bluetoothenhet. Det kan vara allt från enstaka meddelanden eller kontaktlistor till att mer eller mindre kopiera hela enheten. Detta kan förstås vara väldigt skadligt vid attacker mot särskilt utsatta enheter. Det mest välkända fallet då användning av BlueSnarfing blev påkommet var då Google 2013 erkände att de samlade data från okrypterade trådlösa nätverk. Bland datan som samlades fanns både mailadresser och lösenord. Google betalade en uppgörelse på 7 miljoner USD efter sitt erkännande [6].

4.2.2.4 BlueBugging

BlueBugging är en Bluetoothattack som går ut på att installera en bakdörr till en annan Bluetoothenhet. En bakdörr kan implementeras på många olika sätt men är generellt ett sätt att skippa den autentisering av användare som finns på enheten. Det finns två typer av bakdörrar, "Object Code Backdoors" och "Asymmetric Backdoors". "Object Code Backdoors" bygger på att man integrerar bakdörren vid kompilering eller länkning. Det innebär att koden för mjukvaran förblir oförändrad medan den kompilerade maskinkoden innehåller en bakdörr. Det gör dessa väldigt svåra att upptäcka. Generellt är bakdörrar symmetriska vilket innebär att den som hittar bakdörren kan använda den. En asymmetrisk bakdörr kan endast användas av den som har installerat den, exempelvis genom att man har installerat den med ett lösenord [7]. Hackaren installerar en bakdörr ofta genom ett virus. Denna bakdörr kan sedan användas för att göra mer eller mindre allt som ägaren av den attackerade enheten kan, läsa och skriva meddelanden, ringa samtal eller ändra kontaktinformation. BlueBugging är egentligen en vidareutveckling av BlueJacking och BlueSnarfing där de föregående endast kan skicka eller hämta information kan BlueBugging ta full kontroll över den attackerade enheten [8].

4.2.2.5 Risker kopplade till vår Bluetooth applikation

De hot som uppstår vid en Bluetoothkoppling är i vårt fall också ganska skyddade genom att det är just en Bluetoothkoppling. Bluetooth har vanligtvis en räckvidd på ca 10 till uppemot 100 meter beroende på miljö och vilken klass av Bluetooth som används, för mobiltelefoner är det vanligaste klass 2 vilket då innebär en räckvidd på upp till 10 meter [9]. Detta innebär att attacker rimligtvis endast sker då hackaren kan gömma sig i en folkmängd eller liknande. Exempelvis kan man föreställa sig att denna typ av attacker sker på ett café eller i någon liknande miljö.

I vårt arbete används Bluetooth för att koppla en mobilapplikation till ett fordon. Fordonet innehåller en ECU(Engine Control Unit) som har ett inbyggt Bluetoothchip. Detta gör att man i teorin kan hacka fordonet i sig enligt de attacker vi har nämnt ovan. Det betyder att vår säkerhetslösning behöver någon sorts autentisering och auktorisation inbyggd i fordonet, detta diskuteras vidare under kap. 4.2.3 Autentisering av användare. Som ett

extra lager av skydd kan man också tänka sig att den data som lagras på fordonet bör krypteras. Hur detta hotet kan hanteras tittar vi närmare på under kap. 5.2 Autentisering av användare.

De attacker som kan ske mot mobiltelefonen kan i teorin ske vid två olika scenarion. Det första då mobiltelefonen inte är kopplad till fordonet och det andra då mobiltelefonen är kopplad till fordonet. I det första fallet då mobiltelefonen inte är kopplad till fordonet behöver vi hantera och eventuellt skydda den data som lagras lokalt i mobiltelefonen. Detta hanteras i vårt fall genom att det *inte* lagras någon känslig data lokalt på mobiltelefonen. Det andra fallet då mobiltelefonen är kopplad till fordonet kan det eventuellt finnas känslig data att hämta genom mobiltelefonen och en hackare skulle i teorin kunna komma åt både datan och fordonet. Detta hanteras genom den autentisering av användare vi diskuterar vidare under kap. 4.2.3 Autentisering av användare.

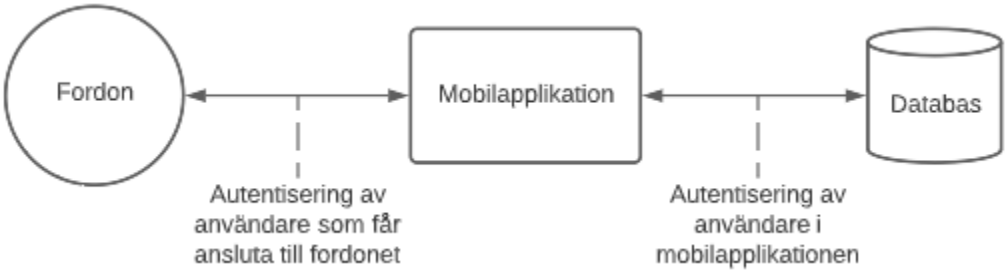
4.2.3 Autentisering av användare

Detta kapitel går igenom de risker & hot som berör autentiseringen av användare i vårt scenario. Autentiseringen av användare är den största delen av den säkerhetslösning vi skall ta fram. Då mobilapplikationen skall erbjuda olika många funktioner beroende på användarens auktorisationsnivå är denna del av applikationen extremt viktig. En potentiell användare måste autentiseras och auktoriseras mot den databas av giltiga användare som skapas och hanteras av OMotion. Vid en bristande säkerhet här skulle en användare kunna komma åt information som de egentligen inte skall ha tillgång till. Det som skulle kunna hända då är att användaren kanske konfigurerar något den inte har någon kunskap om och därav skadar fordonet. Ett annat stort hot är risken för stöld då fordonet skall kunna låsas och låsas upp genom mobilapplikationen. Utöver det finns även en risk för stöld av information om en användare kan komma åt information de inte skall ha tillgång till. Under kap. 4.2 Alternativ för autentisering av användare, diskuterar och analyserar vi ett antal olika tekniker för hur detta skulle kunna implementeras och under kap. 5 Arkitektur, diskuterar vi närmare hur vi har valt att lösa detta problem.

Utöver de risker som finns vid hantering av användare och olika auktorisationsnivåer finns det även hot som är direkt riktade mot fordonet. Då ECU:n i fordonet har ett inbyggt Bluetoothchip kan man i teorin även hacka fordonet i sig enligt alla de Bluetoothattacker vi har diskuterat under kap. 4.2.2 Bluetooth. En hackare skulle alltså kunna komma åt all fordonsdata genom en attack mot fordonet. Det innebär att även fordonet behöver någon sorts inbyggd autentisering av vilka enheter som tillåts ansluta till fordonet. Hur detta kan hanteras och implementeras diskuterar vi vidare under kap. 5.2 Autentisering av användare.

4.3 Alternativ för autentisering av användare

Den största delen av säkerhetslösningen är autentiseringen av användare. I detta scenario behövs autentiseringen av användare utföras i två steg. Först behövs användaren av mobilapplikationen autentiseras mot en databas av giltiga användare. Utöver det behövs anslutningen till ett fordon autentiseras, d.v.s den som försöker ansluta till ett fordon måste autentiseras så att den faktiskt får lov att ansluta.



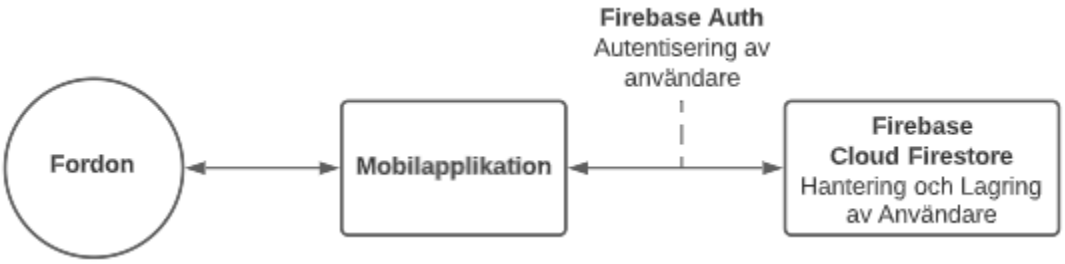
Figur 3: Översikt över autentisering av användare

Vi föreslår en lösning där autentiseringen mot fordonet sker genom mobilapplikationen, d.v.s att endast de användare som använder OMotions mobilapplikation kan anslutas till OMotions fordon. Därav kopplas autentiseringen mot fordonet ihop med den databas av giltiga användare som krävs för autentisering av användare i mobilapplikationen. Hur exakt detta skall fungera i praktiken tittar vi närmare på under kap. 5.2 Autentisering av användare.

Under detta kapitel har vi valt att närmare undersöka ett antal tekniker för att implementera hanteringen och autentiseringen av användare i mobilapplikationen. Detta kräver en databas av giltiga användare samt någon metod för att autentisera dessa genom mobilapplikationen, för detta system skulle det fungera utmärkt med autentisering genom mailadress och lösenord. Databasen behöver dessutom innehålla information om vilka fordon en användare ska ha tillgång till samt vilken roll användaren har på respektive fordon. Databasen skapas och konfigureras i samband med detta examensarbete men kommer därefter att hanteras av OMotion.

Det huvudsakliga kravet vi fick från OMotion var att mobilapplikationen skulle vara kompatibel med både Android och IOS telefoner. Utifrån det kravet har vi kollat på alternativ för autentiseringen av användare som är lätta att kombinera med multi-plattform front-end utvecklingsmiljöer som t.ex. React.js och Flutter. React.js är ett JavaScript-bibliotek med öppen källkod som används för att bygga användargränssnitt till både mobil och webbapplikationer. [10]. Flutter är ett ramverk framtaget av Google som används för att bygga användargränssnitt till både mobil och webbapplikationer [11].

4.3.1 Google Firebase



Figur 3: Översikt över hur systemet skulle kunna se ut med Google Firebase

Det första alternativet vi har analyserat är Google Firebase[3], detta då Firebase föreslogs av OMotion som ett alternativ för att hantera autentiseringen av användare. Firebase är en back-end utvecklingsplattform som erbjuder en mängd olika funktioner. Vi kommer i detta kapitel främst att titta närmre på de funktioner som skulle kunna användas i vårt system.

[3] https://firebase.google.com/

För vår lösning krävs en databas för att lagra och hantera användare samt en metod för att autentisera dessa användare i mobilapplikationen. Firebase erbjuder olika typer av databaser i form av Cloud Firestore[4], Realtidsdatabas[5] och Storage[6]. Storage är en databas för lagring av filer. Cloud Firestore och Realtidsdatabasen är väldigt lika i att de är noSQL databaser som lagrar datan i JSON format, skillnaden är att Realtidsdatabasen kan uppdatera datan som lagras i realtid. Det är framförallt användbart i större system där olika användares ändringar påverkar andra användare [12]. Det databasalternativ som är aktuellt för vårt system är Cloud Firestore. Firestore skulle i vårt system fylla funktionen att hålla reda på vilka fordon en användare har tillgång till samt vilken roll användaren har på respektive fordon.

Utöver databaserna erbjuder Firebase ett antal olika autentiseringsalternativ genom Firebase Auth[7]. Exempelvis finns funktioner för hantera autentisering genom mailadress och lösenord eller Social login d.v.s autentisering med sociala medier konton som t.ex. Facebook eller Twitter. Firebase Auth använder standardprotokoll så som OAuth 2.0 och OpenID Connect för autentiseringen av användare [13].

OpenID Connect och OAuth 2.0 är båda protokoll för att hantera en form av autentisering. OAuth 2.0 används framförallt för skydd av resurser som t.ex. appar eller filer. OpenID Connect bygger på OAuth 2.0 protokollet och används framförallt för autentisering av användare på webbplatser och appar för konsumenter [14].

Generellt sett krypterar Firebase data under transport genom att använda HTTPS samt lagrar datan logiskt isolerat [15]. I vårt scenario innebär detta att den data som skickas mellan mobilapplikationen och Firebasedatabasen är krypterad. Utöver det krypteras den data om systemets användare som lagras i databasen genom en 256 bitars Advanced Encryption Standard och varje krypteringsnyckel krypteras i sin tur med ett antal huvudnycklar som roteras med jämna mellanrum [16].

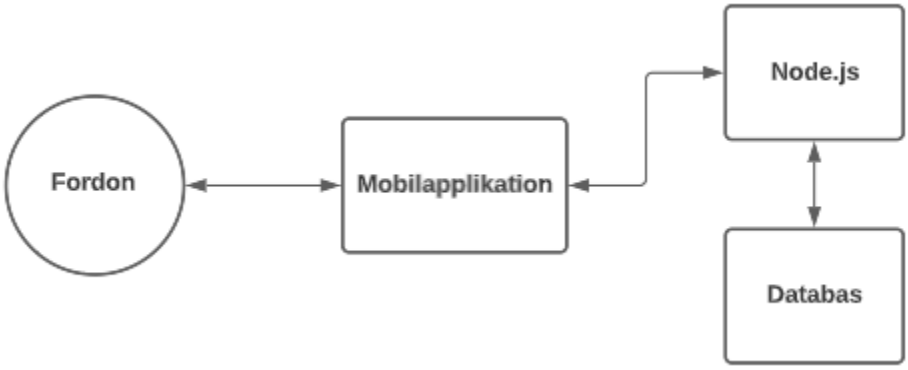
[4] https://firebase.google.com/products/firestore
[5] https://firebase.google.com/products/realtime-database
[6] https://firebase.google.com/products/storage
[7] https://firebase.google.com/products/auth

Den del av Firebase vi hittills har diskuterat kategoriseras under namnet “Build”, denna del innehåller den typ av funktioner som kan tänkas behövas under utveckling av en applikation. Utöver Build så erbjuder Firebase funktioner under kategorierna Release & Monitor samt Engage. Under Release & Monitor finns exempelvis Google Analytics och Performance Monitoring, dessa ger utvecklaren och/eller administratören en möjlighet att enkelt analysera sin applikation [17]. Under kategorin Engage finns verktyg som exempelvis Cloud Messaging vilket gör det enkelt att skicka ut push-notiser till användarna av applikationen [18]. Dessa funktioner är mindre relevanta för detta examensarbete men kan vara användbara vid vidareutveckling och användning av applikationen.

Vi har valt att undersöka Firebase närmare då det föreslogs av OMotion som ett alternativ för att bygga upp backend delen av säkerhetslösningen. Detta då Firebase har funktioner för att hantera autentiseringen av användare men även eftersom Firebase har en mängd andra funktioner som ger OMotion en möjlighet att enkelt kunna vidareutveckla mobilapplikationen i framtiden utan att behöva ändra den befintliga backend delen.

Firebase autentisering syftar till att göra det enkelt att integrera säkra autentiseringssystem för stora som små applikationer. Autentiseringssystemet kan konfigureras på några rader kod och kan hantera komplexa fall som exempelvis “account merging”. Firebase är lätt att kombinera med front-end ramverket Flutter då båda är framtagna av Google med ett syfte av att enkelt kunna kombineras. Då Firebase har ett stort urval av funktioner och metoder kan man med Firebase utveckla applikationer utan att behöva skriva någon dedikerad serverkod. Detta gör Firebase till ett unikt alternativ som endast kräver integrering, inte nödvändigtvis egen utveckling. Firebase diskuteras vidare och jämförs med andra alternativ under kap. 4.3.4 Sammanfattning av alternativ för autentisering av användare.

4.3.2 Node.js och databas



Figur 4: Översikt över hur systemet skulle kunna se ut med Node.js och databas

Det andra alternativet vi har analyserat är Node.js[8] i kombination med valfri databas. Node.js är ett asynkront eventbaserat ramverk för back-end utveckling byggt på JavaScript. Fördelen med att använda ett asynkront eventbaserat ramverk är att man nästan helt kan utesluta att koden går i baklås. Node.js används ofta tillsammans med men är inte exklusivt för det motsvarande front-end ramverket React.js samt någon databas för att utveckla mobil och webbapplikationer för flera olika plattformar samtidigt.

Node.js är lämpligt att använda för både mobil och webbapplikationer stora som små där Node.js då utgör serversidan som får applikationen att kommunicera med databasen. Detta gör Node.js till ett väldigt anpassningsbart verktyg. Node.js är öppen källkod och kan därför anpassas med en mängd olika tredjepartsbibliotek för att hantera diverse olika funktioner. Det finns även en möjlighet att utveckla ett eget autentiseringssystem som då är anpassat för detta examensarbete. Node.js går också att kombinera med valfri databas vilket ger en möjlighet att använda en databas som är anpassad för detta arbete [19].

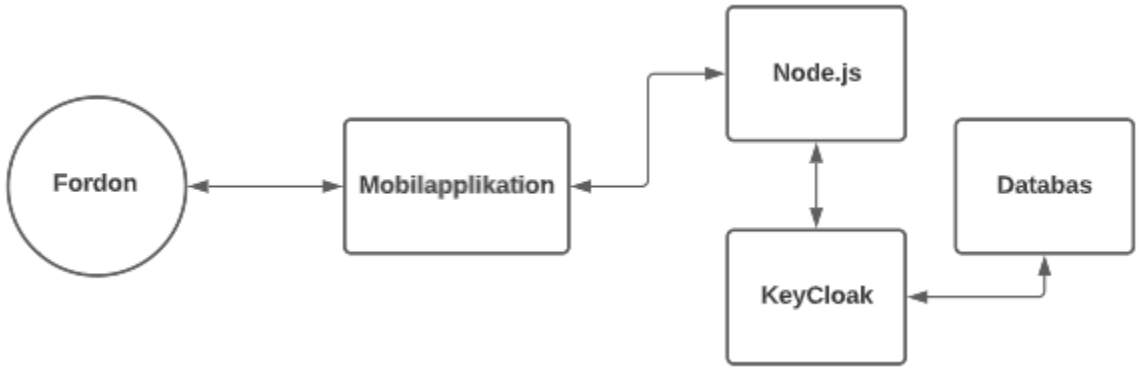
Node.js skulle i vårt system användas för att implementera funktionerna för autentisering av de användare som lagras i databasen. Node.js i sig har inga inbyggda funktioner för autentisering som exempelvis Firebase Auth erbjuder. Det innebär att detta systemet skulle

[8] https://nodejs.org/en/

kräva implementation av dessa autentiseringsfunktioner på egen hand. Detta skulle kräva mer tid och arbete men skulle ge en möjlighet att skräddarsy lösningen utifrån problemet.

Vi har valt att undersöka Node.js med databas närmare då det är ett ramverk som vi har arbetat med tidigare. Node.js i sig har inga funktioner för att hantera autentisering av användare vilket innebär att vi i detta fall skulle behöva implementera dessa funktioner själva. Detta kräver såklart mer tid och arbete men ger en möjlighet att specialanpassa säkerhetslösningen utifrån det problem som skall lösas. Node.js är väldigt anpassningsbart och skalbart. Däremot kräver Node.js mer arbete och undersökning av öppen källkod bibliotek för att kunna vidareutveckla mobilapplikationen i framtiden. Node.js diskuteras vidare och jämförs med andra alternativ under kap. 4.3.4 Sammanfattning av alternativ för autentisering av användare.

4.3.3 KeyCloak



Figur 5: Översikt över hur systemet skulle kunna se ut med Node.js och KeyCloak

Det tredje och sista alternativet vi har analyserat är KeyCloak[9]. KeyCloak är ett öppen källkod bibliotek byggt i JavaScript för att autentisera och hantera användare. I vårt system skulle KeyCloak kunna användas tillsammans med Node.js och en databas för att implementera autentiseringen och hanteringen av användare. Detta system skulle vara ganska likt det som diskuterades under föregående kap. Node.js med databas. KeyCloak erbjuder en inbyggd databas men det är inte rekommenderat att använda denna i

[9] https://www.keycloak.org/

produktion [20]. Istället erbjuder KeyCloak support för att integrera databaser som exempelvis MariaDB och MySQL [21]. KeyCloak skulle i detta system erbjuda inbyggda autentiseringsfunktioner istället för de som i det föregående alternativet skulle implementerats på egen hand. Node.js skulle i detta system endast fungera som en sammankoppling mellan KeyCloak och mobilapplikationen.

KeyCloak erbjuder inbyggda autentiseringsfunktioner för att autentisera de användare som lagras i databasen. Det finns alternativ för att autentisera användare med mailadress och lösenord samt Social Login d.v.s autentisering med sociala medier konton som t.ex. Facebook eller Twitter. KeyCloak använder standardprotokoll för autentisering och har support för OpenID Connect, OAuth 2.0 och SAML [22].

SAML är likt OpenID Connect ett protokoll för autentisering av användare. Däremot är SAML oberoende av OAuth och används vanligen för att ge en användare tillgång till flera appar med endast en inloggning [14].

Vi har valt att undersöka KeyCloak närmare då det föreslogs av handledare Christian Gehrmann som ett alternativ för att hantera autentiseringen av användare. KeyCloak skulle fungera bra som ett komplement till Node.js. KeyCloak är ett öppen källkod bibliotek som är byggt för Node.js med funktioner för att hantera autentisering av användare och erbjuder tillsammans med Node.js en anpassningsbar och skalbar backend. Då detta är ett alternativ som bygger på en Node.js backend gäller även här att vidareutveckling skulle kräva mer arbete och undersökning av vilka öppen källkod bibliotek som finns. KeyCloak diskuteras vidare och jämförs med andra alternativ under kap. 4.3.4 Sammanfattning av alternativ för autentisering av användare.

4.3.4 Sammanfattning av alternativ för autentisering av användare

Vi har valt att undersöka tre olika alternativ för att implementera den del av säkerhetslösningen som berör autentisering av användare. Google Firebase, Node.js med databas och KeyCloak. Till att börja med vill vi säga att alla tre alternativ är bra alternativ

och skulle alla kunna användas för implementation i vårt arbete. Under detta kapitel jämför vi de olika alternativen och försöker motivera varför vi har valt ett av dem framför de andra.

Om man tänker rent säkerhetsmässigt har både Firebase och KeyCloak en bra inbyggd säkerhet medan Node.js med databas kräver en mer genomtänkt uppbyggning av säkerhetslösningen utifrån vårt perspektiv. Exempelvis erbjuder både Firebase och KeyCloak ett färdigt system för att hantera användare där Firebase även erbjuder en inbyggd kryptering av data både under transport men även vid lagring. Node.js med databas skulle kräva att vi själva implementerar dessa bitar.

Node.js med databas skulle ge en större möjlighet att skräddarsy lösningen till det problem som skall lösas medan Firebase och KeyCloak trots goda anpassningsmöjligheter kräver att man arbetar inom vissa ramar. Detta innebär också att autentiseringsfunktionerna måste implementeras på egen hand medan Firebase och KeyCloak erbjuder inbyggda autentiseringsfunktioner.

Man skulle även kunna tänka sig att man kombinerar Node.js med KeyCloak och därav jämför det med Firebase. I det fallet får man en bra inbyggd lösning för att hantera användare i båda fallen. Firebase skulle ge en extra säkerhet med sin inbyggda kryptering medan Node.js med KeyCloak skulle ge en större anpassningsmöjlighet. Autentiseringsfunktionerna som finns inbyggda i Firebase och KeyCloak är väldigt lika, båda erbjuder olika typer av autentisering som exempelvis mailadress och lösenord samt Social Login. Dessutom använder både Firebase och KeyCloak standardprotokoll för autentisering i form av OAuth 2.0, SAML och OpenID Connect. Den stora skillnaden mellan Firebase och KeyCloak är databasen av användare. KeyCloak tillåter systemet att använda olika typer av databaser medan Firebase är låst till noSQL databaser där datan lagras i JSON format.

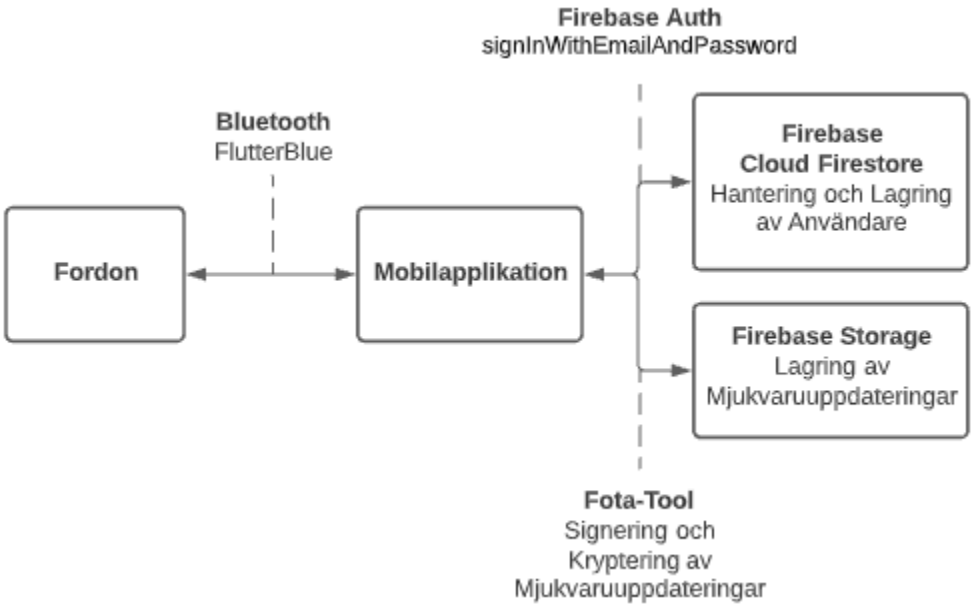
Vi har valt att använda oss av Firebase för att bygga upp den del av säkerhetslösningen som berör autentisering av användare. Delvis då det var Firebase som föreslogs av OMotion men även då vi väger den inbyggda krypteringen i Firebase högre än de extra

anpassningsmöjligheterna som finns med Node.js och KeyCloak. Firebase och KeyCloak är väldigt lika i hur autentiseringen av användare fungerar, båda använder standardprotokoll och båda erbjuder väldigt liknande alternativ för att autentisera användare. Detta innebär att både Firebase och KeyCloak är utmärkta alternativ för att integrera autentisering och hantering av användare i vårt system. Det blir därav de andra mindre faktorerna som gör Firebase till ett bättre alternativ för vårt system. En noSQL databas fungerar utmärkt i vårt system, vilket betyder att Firebase begränsning i databasalternativ inte är något problem för vårt system. Vi kan därav ta vara på den extra inbyggda krypteringen av data som erbjuds med Firebase.

5 Arkitektur

Detta kapitel är en fördjupande beskrivning av den säkerhetslösning vi har tagit fram genom detta arbete. Vi går även igenom vilka delar som är implementerade och vilka som inte är implementerade samt vilka delar vi enbart har integrerat.

5.1 Översikt



Figur 6: Översikt över säkerhetslösningen

Den säkerhetslösning som har tagits fram genom detta examensarbete beskrivs översiktligt i figur 6. Säkerhetslösningen är framtagen utifrån de hot och risker vi har identifierat genom vår litteraturstudie. Mobilapplikationen har vi implementerat själva medan Firebase, Fota-tool och FlutterBlue enbart är integrerade för att hantera vissa funktioner i lösningen.

Säkerhetslösningen har tagits fram utifrån de hot och risker som har identifierats och består till stora delar av autentisering av systemets användare. Både i mobilapplikationen men även vid anslutning till ett fordon. En användare måste autentisera sig med mailadress

och lösenord när den vill logga in i mobilapplikationen. Därefter sker autentiseringen mot fordonet helt i bakgrunden. Användaren behöver alltså inte ha något separat lösenord för sitt fordon.

5.2 Autentisering av användare

Autentisering av användare är den största delen av säkerhetslösningen och som vi tidigare under kap. 4.3 Alternativ för autentisering av användare, har diskuterat finns det i vårt system ett behov av autentisering av användare i två steg. Det behövs autentisering av den användare som vill logga in i mobilapplikationen och det behövs autentisering av den användare som vill ansluta till ett fordon. Vi föreslår en lösning där autentiseringen mot fordonet sker genom mobilapplikationen, d.v.s endast de användare som använder OMotions mobilapplikation kan anslutas till OMotions fordon. Hur detta fungerar tittar vi närmare på under detta kapitel.

5.2.1 Autentisering av användare i mobilapplikationen

För den första delen då användaren ska autentiseras i mobilapplikationen har vi valt integrera Firebase Auth och Firebase Cloud Firestore. Auth hanterar själva autentiseringen, där har vi valt att använda metoden som bygger på inloggning med mailadress och lösenord. Cloud Firestore hanterar vilka fordon en användare skall ha tillgång till samt vilken roll användaren har på respektive fordon.

5.2.1.1 Databasen

Databasen som hanterar användare består av två delar. Firebase Auth, se bild 1, som hanterar autentiseringen av användare och Firebase Cloud Firestore, se bild 2, som hanterar vilka fordon en användare ska ha tillgång till samt vilken roll användaren har på respektive fordon. Endast administratören av databasen kan lägga till och ta bort användare. Åtkomst till databasen skyddas genom en autentisering med mailadress och lösenord. I detta system är det OMotion som administrerar databasen. Det innebär att OMotion enskilt bestämmer vilka användare som är auktoriserade, vilka fordon en användare har tillgång till samt vilken roll en användare har på ett fordon.

En användare registreras genom “Add user” knappen på bild 1. Där skrivs då in en mailadress och ett lösenord, detta görs av OMotion. Lösenordet kan senare ändras av användaren genom mobilapplikationen. Ett unikt id genereras automatiskt när en användare skapas. Därefter läggs användaren till i Firestore databasen, se bild 2, och OMotion kan skriva in vilka fordon den användaren ska ha tillgång till samt vilken roll användaren ska ha på respektive fordon.

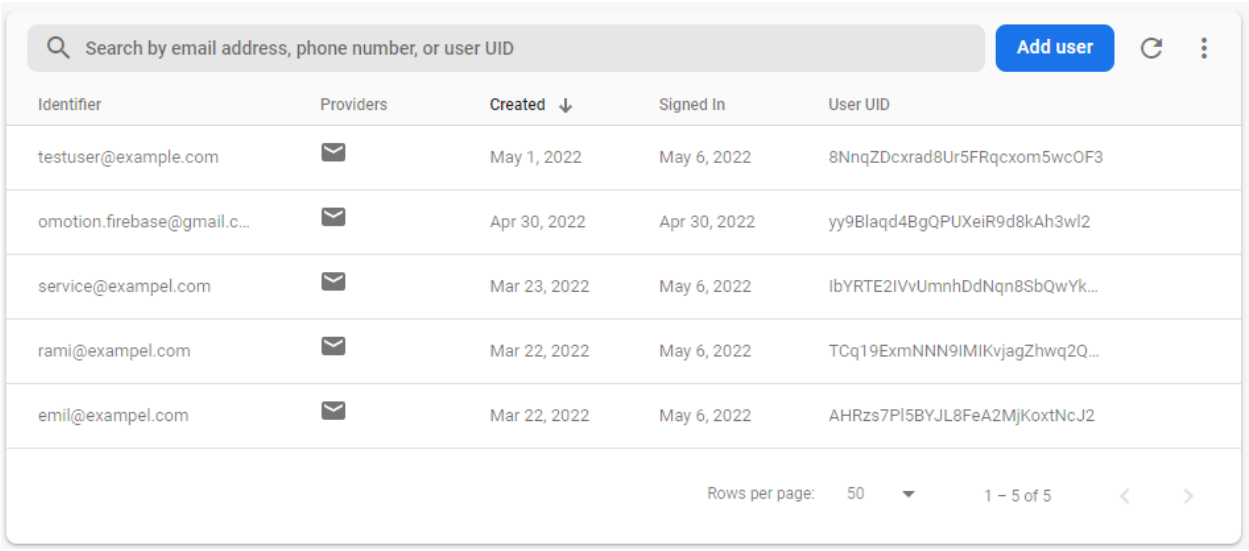


Bild 1: Firebase Auth

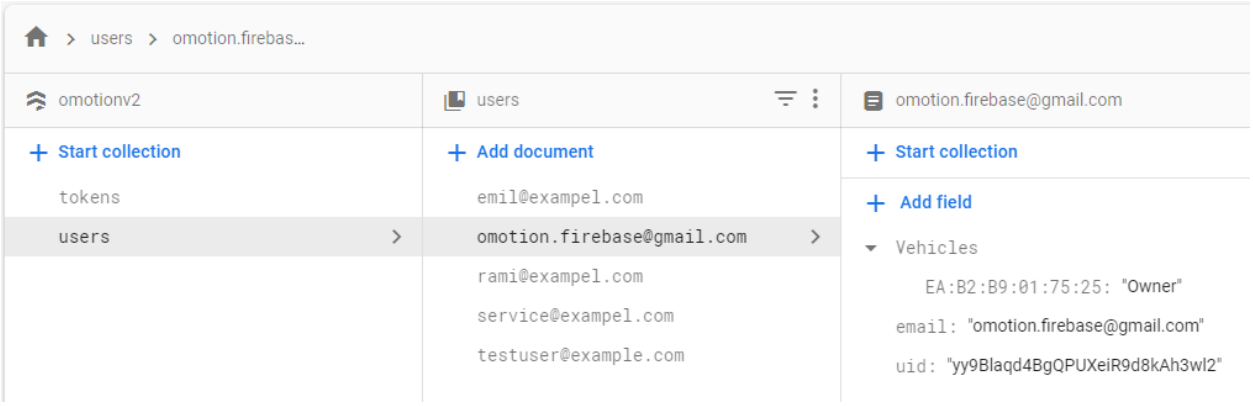


Bild 2: Firebase Cloud Firestore

5.2.1.2 Inloggning

Vi vet nu hur en användare läggs till i systemet och vad som gör en användare till en auktoriserad användare. Under detta kapitel tittar vi närmare på hur autentiseringsprocessen ser ut när en användare skall logga in i mobilapplikationen.

När en användare trycker på “Login” knappen på bild 3, körs den kod som visas på bild 6. Den mailadress och det lösenord som matats in av användaren skickas till FirebaseAuth genom metoden “signInWithEmailAndPassword” och i retur fås en “userCredential”. userCredential är en klass i Firebasepaketet, denna klass innehåller information om användaren men även vilken typ av operation som gjorts. I vårt fall skulle denna operation vara “Sign-in”. Om “userCredential” objektet inte är null har användaren blivit inloggad och vi kan nu hämta information om den inloggade användaren. Vi använder oss av användarens unika id för att säkerställa att vi hämtar rätt information. Den information som hämtas är användarens mailadress och den map som innehåller de fordon användaren har tillgång till samt vilka roller användaren har till respektive fordon. Därefter skickas användaren vidare till sidan för att skapa en anslutning till ett fordon. Om inloggningen misslyckas, d.v.s om mailadressen inte finns i databasen eller om lösenordet är felaktigt visas en popup som berättar för användaren att inloggningen har misslyckats.

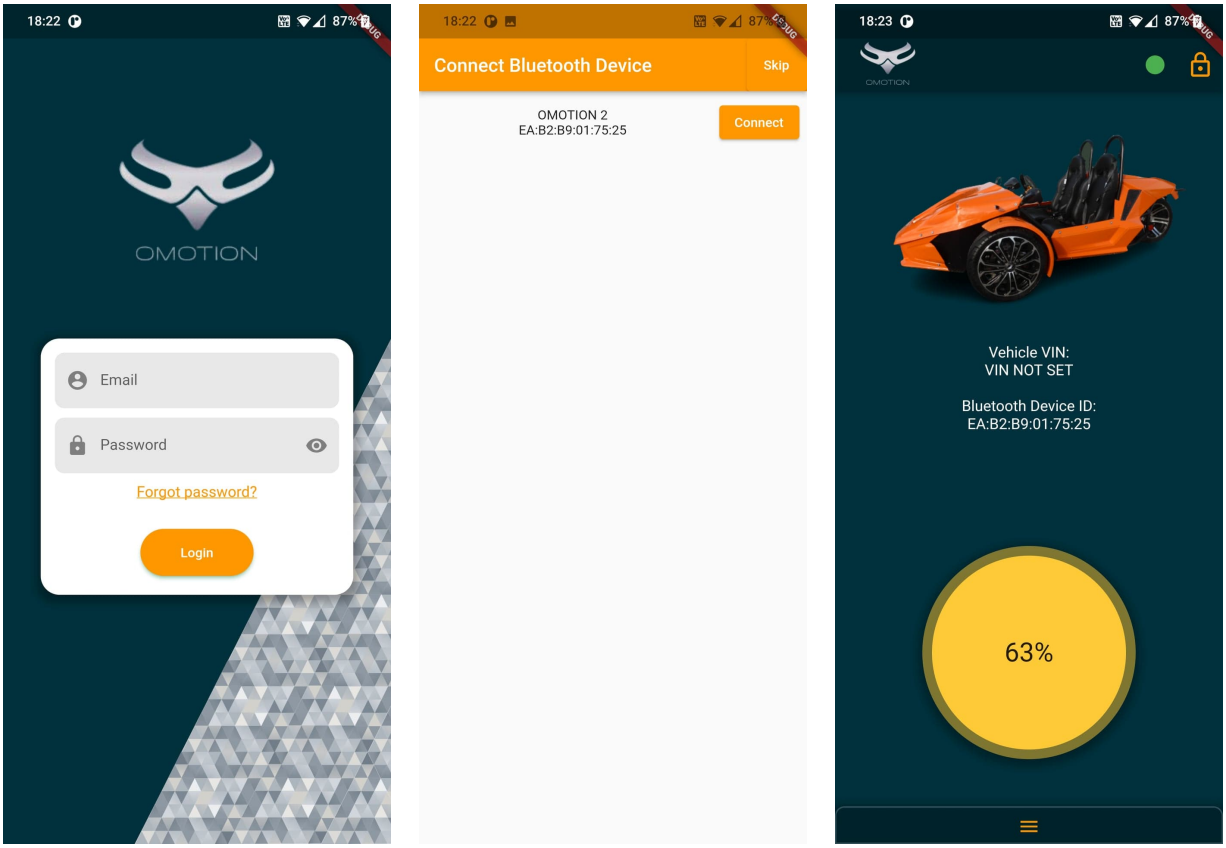


Bild 3-5: Översikt över inloggningsprocessen från användarens perspektiv

```
onPressed: () async {
  try {
    userCredential = await FirebaseAuth
        .instance
        .signInWithEmailAndPassword(
            email: email, password: password);
    if (userCredential != null) {
      var user =
          FirebaseAuth.instance.currentUser;
      FirebaseFirestore.instance.collection('users').where('uid', isEqualTo: user?.uid).get().then((value) {
        for (int i = 0;
            i < value.docs.length;
            i++) {
          //Set user credentials based on role in database
          context.read<FirebaseUserData>().setUserEmail(value.docs[i].data()['email']);
          context.read<FirebaseUserData>().setAvailableVehicles(value.docs[i].data()['Vehicles']);
        }
        Navigator.pushReplacementNamed(
            context, "bluetooth");
      });
```

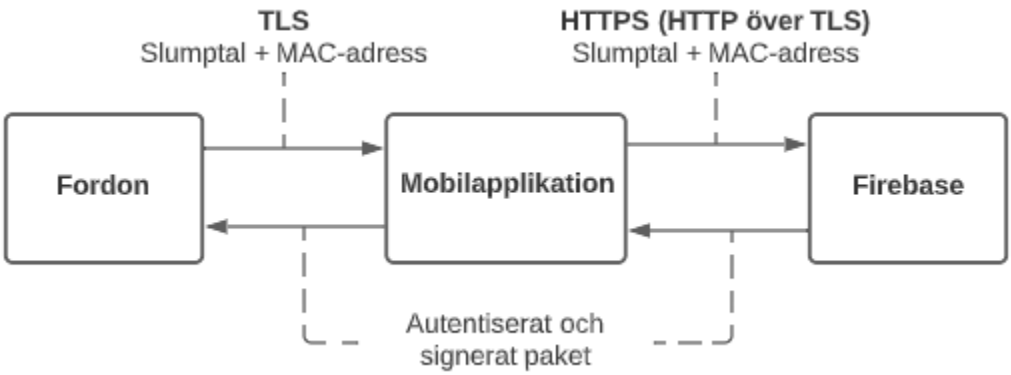
Bild 6: Kod för inloggning

5.2.2 Autentisering mot fordonet

Den andra delen av autentiseringen av användare, alltså den del som ska autentisera användaren som vill ansluta till ett fordon har inte implementerats genom detta arbete. Detta då lösningen kräver konfigurationer på fordonets ECU vilket bryter mot de avgränsningar som har gjorts under kap. 1.7 Avgränsningar. Då denna del inte har implementerats är detta kapitel enbart vårt förslag på hur problemet kan lösas.

5.2.2.1 Förslag på autentisering mot fordonet

Vi föreslår en lösning som bygger på att användarens tillgång till fordonet autentiseras och signeras av den Firebase databas som redan är integrerad. Detta skulle innebära att det endast behövs *en* databas av användare och fordonet i sig behöver inte hålla reda på vilka användare den har utan endast att autentiseringen är signerad av vårt Firebase system. Figur 7 visar översiktligt hur detta skulle kunna fungera. Denna lösning kräver att fordonet konfigureras med en publik nyckel motsvarande den privata nyckel som används för signering i Firebase.



Figur 7: Översikt över en digital signatur

När en användare försöker ansluta till ett fordon skickar fordonet en sträng som består av ett slumptal och fordonets MAC-adress. Detta behövs skickas på ett säkert sätt, vi föreslår att detta skickas med TLS(Transport Layer Security) protokollet [23]. TLS är ett standardiserat protokoll för att skicka data på ett säkert sätt mellan två enheter.

Mobilapplikationen skickar därefter strängen vidare till Firebase för autentisering och signering. Detta skulle kunna implementeras med Firebase Functions. Firebase Functions är ett sätt att kunna köra egen serverkod på Firebase. Denna serverkoden skulle då behöva autentisera fordonets MAC-adress mot de MAC-adresser som den inloggade användaren skall ha tillgång till enligt databasen. Vid en lyckad autentisering där, alltså att användaren skall ha tillgång till fordonet, kan serverkoden därefter signera den sträng (slumptal+MAC-adress) som skickades från fordonet med den privata nyckel som lagras i vårt Firebase system. Vi föreslår att detta görs med en JWS(JSON Web Signature) [24]. JWS är ett protokoll för att enkelt kunna signera JSON objekt för att garantera att objektet kommer från en trovärdig källa. Det signerade JSON objektet skickas sedan till mobilapplikationen som vidarebefordrar det till fordonet.

Fordonet kan därefter kontrollera signaturen på JSON objektet med dess publika nyckel. Därefter kontrolleras slumptalet och MAC-adressen som skickades med. Om alla kontroller är okej så kan en Bluetoothanslutning upprättas mellan fordonet och mobilapplikationen.

5.2.2.2 Kryptering av data

Den lösningen som har föreslagits fungerar i praktiken bra men det finns alltid risker. Beroende på hur den privata signeringsnyckeln lagras skulle denna eventuellt kunna hamna i händerna på obehöriga användare som då kan autentisera sig mot ett OMotion fordon utan att fordonet märker någon skillnad. Vi föreslår därför att den privata nyckeln lagras i en krypterad databas som är byggd för att just lagra hemlig information, exempelvis Google Cloud Secret Manager som enkelt kan kombineras med Firebase. Utöver detta rekommenderar vi som ett extra lager av skydd att den data som lagras på fordonet krypteras.

Vi föreslår att krypteringen av den data som lagras på fordonet sker enligt algoritmen Advanced Encryption Standard med 128 bitars nycklar (AES-128). Algoritmen är välkänd och väldokumenterad, vi kommer därav inte gå in i detalj på hur AES-128 fungerar då det är en ganska komplicerad algoritm. AES-128 är en symmetrisk algoritm vilket innebär att samma hemliga nyckel används för kryptering och dekryptering. Denna nyckel är i vårt fall

endast känd för fordonet och mobilapplikationen. I mobilapplikationen kan nyckeln lagras på samma sätt som den privata nyckeln vid digital signering i Google Cloud Secret Manager. I fordonet skulle nyckeln behöva lagras på ett säkert sätt, *inte* i koden som körs på ECU:n. Hur exakt detta ska göras kan vi inte rekommendera då vi inte har tillräcklig kunskap om hur fordonets hårdvara är uppbyggd men man skulle kunna tänka sig att man har något separat minne på hårdvaran som kan lagra privata nycklar.

5.3 Mobilapplikation

Som en del av arbetet med att ta fram säkerhetslösningen har vi även implementerat en prototyp av mobilapplikationen. Mobilapplikationen är den enda del som vi helt har implementerat själva. Denna mobilapplikation är utvecklad i multi-plattform ramverket Flutter och skrivet i programmeringsspråket Dart. Detta för att man enkelt ska kunna använda appen på både Android och IOS. Det finns då även utrymme för att vidareutveckla en webbapplikation i samma miljö. Under detta kapitel går vi igenom hur mobilapplikationen är uppbyggd, vilka funktioner som finns samt hur den kopplar ihop de andra delarna av lösningen.

5.3.1 Struktur

Strukturen på mobilapplikationen visualiseras i appendix 2. Användaren möts först av en inloggningssida. Vid en lyckad inloggning skickas användaren vidare till sidan där man kan ansluta till ett fordon. Om användaren har tillgång till något fordon som är inom Bluetooths räckvidd visas de upp här och användaren kan välja att ansluta till fordonet. När användaren har anslutit till fordonet skickas den vidare till startsidan i appen. Dessa tre sidor visas på bild 3-5. När användaren ansluter till fordonet får användaren en roll, denna roll hämtas från Firebase databasen. Baserat på användarens roll (Owner, OMotion, Service eller Guest) finns det olika många funktioner. Exempelvis kan vi på bild 7-9 se de olika konfigurationerna som kan göras på ett fordon baserat på användarens roll. Inloggade roller på bild 7-9 från vänster: Owner, Service, OMotion. Hur inloggningen sker, vad som avgör vilka fordon en användare skall ha tillgång till och hur användarens roll avgörs har redan diskuterats under kap. 5.2 Autentisering av användare

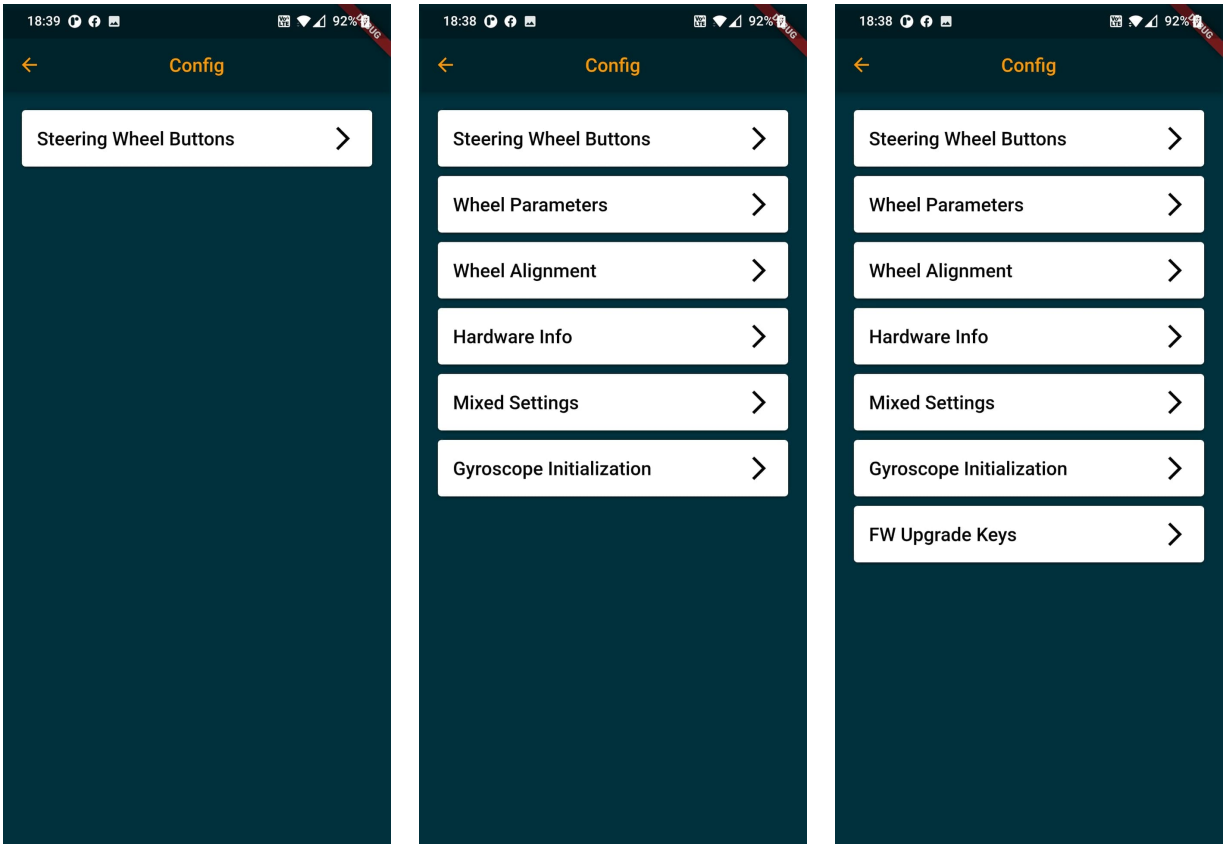


Bild 7-9: Översikt över vilka konfigurationer som kan göras baserat på användarens roll

Mobilapplikationens funktioner är uppdelade i två kategorier. “Status” är de funktioner som kan läsa data från fordonet och “Settings” är de funktioner som kan skriva data till fordonet. Utöver dessa kategorier finns två funktioner som kan nås från startsidan “Immobilizer” som tillåter användaren att låsa och låsa upp fordonet samt “Connection Status” som tillåter användaren att byta fordon, logga ut eller byta lösenord. Dessa två funktioner kan nås genom den gröna/röda cirkeln och hänglåset längst upp på startsidan. Appens meny och “Connection Status” sidan visas på bild 10 och 11. Fler skärmdumpar på mobilapplikationen visas under appendix 3.

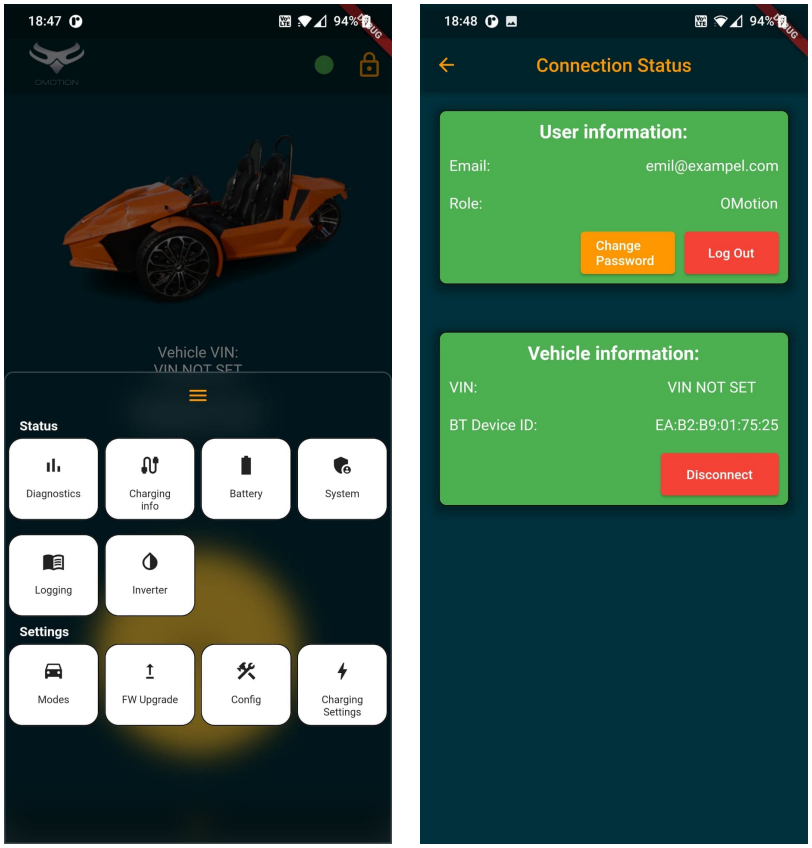


Bild 10 & 11: Appens meny & "Connection Status"

5.3.2 Bluetooth

Mobilapplikationen kommunicerar med fordonets ECU via Bluetooth, detta är implementerat med flutterbiblioteket FlutterBlue. Hur exakt en Bluetoothuppkoppling skapas är inte helt tydligt då dokumentationen för FlutterBlue enbart fokuserar på hur paketet används, *inte* vad som sker i bakgrunden när de olika metoderna anropas. Vi har inte heller tillräcklig kunskap för att diskutera hur uppkopplingen skapas från fordonets sida. Hur Bluetoothanslutningen autentiseras har vi diskuterat under kap. 5.2.2 Autentisering mot fordonet.

Det vi kan beskriva är hur en Bluetoothuppkoppling används för att skicka data mellan fordonet och mobilapplikationen. En Bluetoothenhet består generellt av services och varje service består av characteristics. Det är dessa characteristics som används för att skicka data mellan fordonet och mobilapplikationen. Connect metoden som visas på bild 12 och

13 är den metod som anropas när användaren väljer att ansluta till ett fordon enligt bild 2. Bild 12 visar hur vi ansluter till fordonet med FlutterBlues inbyggda connect metod. Därefter kan vi lyssna på Bluetoothenhetens services. BLE(Bluetooth Low Energy) protokollet kräver att man vid anslutning till en Bluetoothenhet skriver en gång till varje characteristic innan man kan läsa data från dessa characteristics. När detta är gjort kan man sedan börja lyssna på varje characteristic, varje gång fordonet skickar data anropas "listen" metoden. De nya värdena sparas i en map och initieras sedan genom "initBluetoothData" metoden. Detta visas på bild 13. Beroende på vilken characteristic det är och vilken typ av data det är som skickas så läses datan in, tolkas och skickas till de variabler som visas i appen. Hur exakt datan tolkas är något som OMotion inte aktivt vill dela med sig av, vi kan därav inte visa upp ett exempel på hur det görs i denna rapport.

```
void connect(BluetoothDevice device) async {
  try {
    await device.connect();
  } catch (e) {
    print(e.toString());
  } finally {
    _services = await device.discoverServices();
  }
```

Bild 12: Bluetooth connect metod, del 1

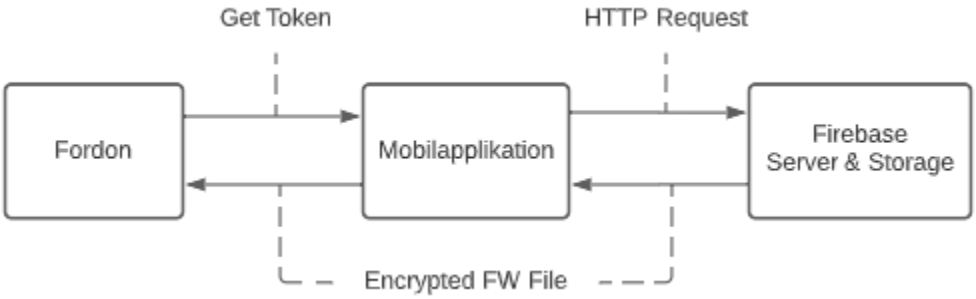
```
      // Have to write to characteristic to be able to get notified on value changes.
      characteristic.write([1, 0]);
      await Future.delayed(const Duration(milliseconds: 100));

      // Listen to value changes on each characteristic,
      // upon changes read values and call initBluetoothData
      await characteristic.setNotifyValue(true);
      characteristic.value.listen((value) {
        widget.readValues[characteristic.uuid] = value;
        initBluetoothData(characteristic);
      });
```

Bild 13: Bluetooth connect metod, del 2

5.4 Mjukvaruuppdatering på fordonet

Mjukvaruuppdatering på fordonet är en av funktionerna som OMotion önskade kunna göra genom mobilapplikationen. Då denna funktion har en stor påverkan på säkerheten för systemet har vi valt att beskriva hur den har implementerats ur ett säkerhetsperspektiv under detta kapitel. Figur 8 visar översiktligt hur flödet ser ut.



Figur 8: Översikt över mjukvaruuppdateringen

För att hantera mjukvaruuppdateringarna på ett säkert sätt har vi integrerat ett verktyg som heter Fota-tool. Vi har även integrerat Firebase Storage och Firebase Cloud Functions. Firebase Storage är en databas för lagring av filer, det är där mjukvaruuppdateringsfilerna lagras. Firebase Cloud Functions är ett sätt att köra serverkod genom Firebase och Fota-tool är serverkoden som körs genom Firebase Cloud Functions. Fota-tool är en tredjeparts mjukvara som vi endast integrerat, vi kan därav endast beskriva hur det fungerar utifrån den dokumentation som finns vilket inte nödvändigtvis är in i minsta detalj.

Mjukvaruuppdateringsprocessen startar med att mobilapplikationen hämtar en token från fordonet. Denna token är en hexadecimal sträng som är unik för varje fordon och varje förfrågan. Varje mjukvaruversion har en unik nyckel “model key” och varje fordon har en unik nyckel. Den token som hämtas från fordonet byggs upp av en hopslagning av den aktuella mjukvarans “model key” och fordonets unika nyckel. Denna kombinerade nyckel krypteras sedan med den Firebase serverns publika RSA nyckel och omvandlas därefter till hex format.

När en token har hämtats skickas den sedan vidare till servern genom en http request. När servern får en http request körs Fota-tool mjukvaran. Fota-tool tar den mjukvarufil som matchar den “model key” som skickades med i token, signerar paketet med den token som kommer från fordonet med RSA-PSS och krypterar sedan hela paketet med AES-128-CBC. Efter krypteringen läggs det på en HMAC(Hash-Based Message Authentication Codes) på hela paketet. HMAC är en teknik för att skicka information över internet mellan två parter privat. Detta uppnås genom att de två parterna kommer överens om en hemlig nyckel som används för kryptering samt en hash algoritm som ska användas. HMAC är i detta fall konfigurerad mellan fordonet och Firebase servern. Detta innebär att fordonet kan verifiera att paketet kommer från rätt källa och att det inte har påverkats på vägen.

Därefter skickas det signerade och krypterade paketet tillbaka till mobilapplikationen som i sin tur skickar paketet vidare till fordonet. Fordonet kan därefter verifiera HMAC så att paketet kommer från rätt källa. Därefter dekrypteras paketet och signaturen kontrolleras mot den token som skickats. Om alla kontroller stämmer kan mjukvaruuppdateringen installeras på fordonet.

Detta innebär att det krypterade mjukvarupaket som genereras kommer vara unikt för varje fordon och varje mjukvara. Det betyder i sin tur att detta specifika mjukvarupaketet som genereras endast kan installeras på det fordon som token hämtades ifrån.

Integrationen av Fota-tool i denna funktion tillåter användarna av OMotions fordon att säkert kunna installera ny mjukvara på sina fordon. Den kod som genererar en token på fordonet samt Fota-tool mjukvaran fanns redan och har endast integrerats. Det vi har gjort är att koppla ihop dessa genom mobilapplikationen och Firebase.

6 Slutsats

I detta kapitel presenteras resultatet av detta examensarbete. Det diskuteras även huruvida den målsättning som gjordes i början av arbetet har uppfyllts och hur lösningen skulle kunna vidareutvecklas.

6.1 Återkoppling på frågor

Här diskuteras de frågor som togs upp under kap. 1.4 Problemformulering

6.1.1 Hur ska säkerhet och åtkomst på fordonsdata hanteras på ett smart sätt?

Fordonsdatan bör enligt vår analys skyddas i två steg. Först måste fordonet skyddas så att endast auktoriserade enheter kan ansluta sig till fordonet. Därefter bör fordonsdatan skyddas genom någon sorts kryptering. Under kap. 5.2.2 Autentisering mot fordonet, ger vi vår rekommendation för hur detta kan hanteras och implementeras.

6.1.2 Vilken information och data behövs i databasen?

Den data som lagras i databasen diskuteras i sin helhet under kap. 5.2.1.1 Databasen. Översiktligt kan vi säga att information som en användares mailadress, lösenord, unika id samt vilka fordon användaren har tillgång till och vilken roll användaren har på respektive fordon är det som lagras i databasen.

6.1.3 Hur kan säkerhetslösningen vi tar fram förbättra säkerheten för OMotions fordon, mobilapplikation och användare?

Vi har under kap. 1.1 Bakgrund berättat lite kort om den mobilapplikation som fanns tidigare. Vi har då även pratat lite om den bristande säkerheten som följde med den gamla mobilapplikationen. Kort och enkelt, det fanns ingen säkerhet med den mobilapplikation som fanns tidigare. Vem som helst kunde ansluta till ett fordon och läsa/skriva data. Det fanns inte heller någon hantering av användare vilket gjorde att vem som helst kunde använda den gamla mobilapplikationen.

Det vi har gjort är en genomgående analys av vilka hot och risker som finns och utifrån dessa tagit fram en säkerhetslösning för OMotions system. Vår lösning, som beskrivs under kap. 5 Arkitektur, ger en heltäckande säkerhet mot alla de hot som har identifierats. Detta innebär en integrerad hantering av användare med olika roller samt skydd mot de externa hot som kan ske mot både mobilapplikationen och fordonet.

6.2 Resultat

Resultatet av detta examensarbete är en fördjupad förståelse för vilka hot och risker som finns då ett fordon kontrolleras och konfigureras genom en mobilapplikation samt hur dessa rimligtvis bör hanteras. Utöver det har den säkerhetslösning som har tagits fram genom detta examensarbete förbättrat säkerheten för OMotion, deras fordon och deras kunder. Det finns nu en integrerad autentisering av användare både i mobilapplikationen men även vid anslutning till ett fordon. Tidigare kunde alla användare anslutas till alla fordon, nu kan endast de användare som har auktoriserats av OMotion komma åt OMotions fordon.

Huruvida resultatet uppfyller den målsättning som sattes under kap. 1.3 Målformulering, diskuteras under kap. 6.3.1 Måluppfyllelse.

6.2.1 Experimentella resultat

Ett antal tester har gjorts med syfte att testa hur lång tid det tar att registrera en ny användare i databasen, autentisera en användare i mobilapplikationen samt börja läsa data från ECU:n. Det har gjorts 10 tester för varje scenario, de resultat som presenteras här är genomsnittet på dessa 10 tester. Alla tester som gjorts presenteras under appendix 4. Då mjukvaruuppdateringsdelen av lösningen var en av de sista att implementeras har vi inte hunnit utföra några tester på den. Det finns inte heller några motsvarande tester från den gamla mobilapplikationen att tillgå, vi har därav inget att jämföra dessa värden mot.

Test:	*Tid, genomsnitt i sekunder:*
Registrering av ny användare i databas	85.94
Autentisering av användare i mobilapplikation	2.16
Börja läsa data från ECU:n	2.34

Tabell 1: Översikt över testresultat

6.3 Diskussion

Detta är en diskussion kring examensarbetet utifrån de målsättningar och problemställningar som gjordes i början av arbetet.

6.3.1 Måluppfyllelse

Målet med examensarbetet har överlag uppfyllts på ett tillfredsställande sätt. En litteraturstudie har gjorts för att identifiera vilka hot och risker som finns i vårt scenario. En lämplig säkerhetslösning har tagits fram utifrån dessa hot och majoriteten av säkerhetslösningen har implementerats och verifierats i form av en prototyp. De frågor som ställdes i början av examensarbetet har besvarats under kap. 6.1 Återkoppling på frågor.

6.3.2 Vidareutveckling

Då säkerhetslösningen som har tagits fram fortfarande har vissa delar som inte är implementerade är den vidareutveckling som kan göras främst en fullföljning av den säkerhetslösning vi har tagit fram. Exempelvis krävs det fortfarande en implementation på den del av lösningen som beskrivs under kap. 5.2.2 Autentisering mot fordonet.

Utöver det är vissa delar av mobilapplikationen inte implementerade fullständigt. Exempelvis sparas de inställningarna som kan göras på fordonet genom mobilapplikationen endast lokalt i mobilapplikationen. Det behövs alltså implementeras att inställningen faktiskt skickas till fordonet. Funktioner som Logging och Immobilizer är inte heller implementerade. Alla dessa är pga tidsbrist.

För att säkerhetslösningen skall kunna lanseras till OMotions kunder krävs en

implementation på dessa funktioner i mobilapplikationen samt den del av säkerhetslösningen som beskrivs under kap. 5.2.2 Autentisering mot fordonet.

6.3.3 Reflektion över etiska aspekter

Examensarbetet innehåller ett antal olika etiska aspekter. Då information om systemets användare skall lagras i en databas måste detta göras i enlighet med personuppgiftslagen.

Då examensarbetet är en undersökning i säkerhet, finns det nästan alltid ett etiskt dilemma mellan säkerhet och användbarhet i systemet som utvecklas. Det handlar om att skydda OMotion och deras kunders integritet samtidigt som systemet är lätt att använda för just dessa.

Systemet innehåller i dagsläget inte någon datainsamling kring hur systemet används m.m. Däremot skulle man kunna tänka att det är något som behövs i framtiden för optimering av systemet. I det fallet krävs en noga avvägd analys kring vilken typ av data som skall lagras för att kunna bibehålla användarnas integritet.

7 Uppdelning av arbetet

I detta kapitel redogörs vem som har gjort vad under detta examensarbete.

7.1 Emil Nielsen

Utvecklat följande delar av säkerhetslösningen:

- All front-end av mobilapplikationen.
- Bluetoothkopplingen mellan mobilapplikationen och fordonet samt hantering av den data som hämtas från och skrivs till fordonet.

Skrivit följande delar av rapporten:

- Delar av kap. 1, 2, 3, 4, 5 och 6

7.2 Mhd Rami Alhoumsi

Utvecklat följande delar av säkerhetslösningen:

- Hantering och autentisering av användare genom Firebase
- Integration av verktyget Fota-tool för hantering av mjukvaruuppdateringar på fordonet genom Firebase.

Skrivit följande delar av rapporten:

- Delar av kap. 1, 2, 3, 4, 5 och 6

8 Referenser

[1] C. Woodford. “Bluetooth”, ExplainThatStuff, [Online]. Jun. 2021. Tillgänglig: https://www.explainthatstuff.com/howbluetoothworks.html (hämtad: 2022-05-17).

[2] A. Mitra, “What is BlueSmack Attack?”, The Security Buddy, [Online]. Mar. 2017. Tillgänglig: https://www.thesecuritybuddy.com/bluetooth-security/what-is-bluesmack-attack/ (hämtad: 2022-03-08).

[3] A. Mitra, “What is BlueJacking?”, The Security Buddy, [Online]. Mar. 2017. Tillgänglig: https://www.thesecuritybuddy.com/bluetooth-security/what-is-bluejacking/ (hämtad: 2022-05-17).

[4] A. Gusev. “Object Exchange (OBEX) Protocol Primer”, Developer, [Online]. Dec. 2005. Tillgänglig: https://www.developer.com/mobile/object-exchange-obex-protocol-primer/ (hämtad: 2022-05-17).

[5] M. Higgins, “What is bluejacking? How to avoid bluejacking attacks”, NordVPN, [Online]. Jan. 2022. Tillgänglig: https://nordvpn.com/blog/bluejacking/ (hämtad: 2022-03-08).

[6] Techslang, “What is Bluesnarfing?”, u. å. [Online]. Tillgänglig: https://www.techslang.com/definition/what-is-bluesnarfing/ (hämtad: 2022-03-08).

[7] A. Mitra, “What is a Backdoor?”, Tech Security Buddy, [Online]. Mar. 2017 Tillgänglig: https://www.thesecuritybuddy.com/malware-prevention/what-is-a-backdoor/ (hämtad: 2022-04-20).

[8] Techslang, “What is Bluebugging?”, u. å. [Online]. Tillgänglig: https://www.techslang.com/definition/what-is-bluebugging/ (hämtad: 2022-03-08).

[9] A. Peshin, “What is the range of Bluetooth and how can it be extended?”, ScienceABC, [Online]. Dec. 2021. Tillgänglig: https://www.scienceabc.com/innovation/what-is-the-range-of-bluetooth-and-how-can-it-be-extended.html (hämtad: 2022-03-08).

[10] N. Pandit, “What And Why React.js”, C#CORNER, [Online]. Feb. 2021. Tillgänglig: https://www.c-sharpcorner.com/article/what-and-why-reactjs/ (hämtad: 2022-05-03).

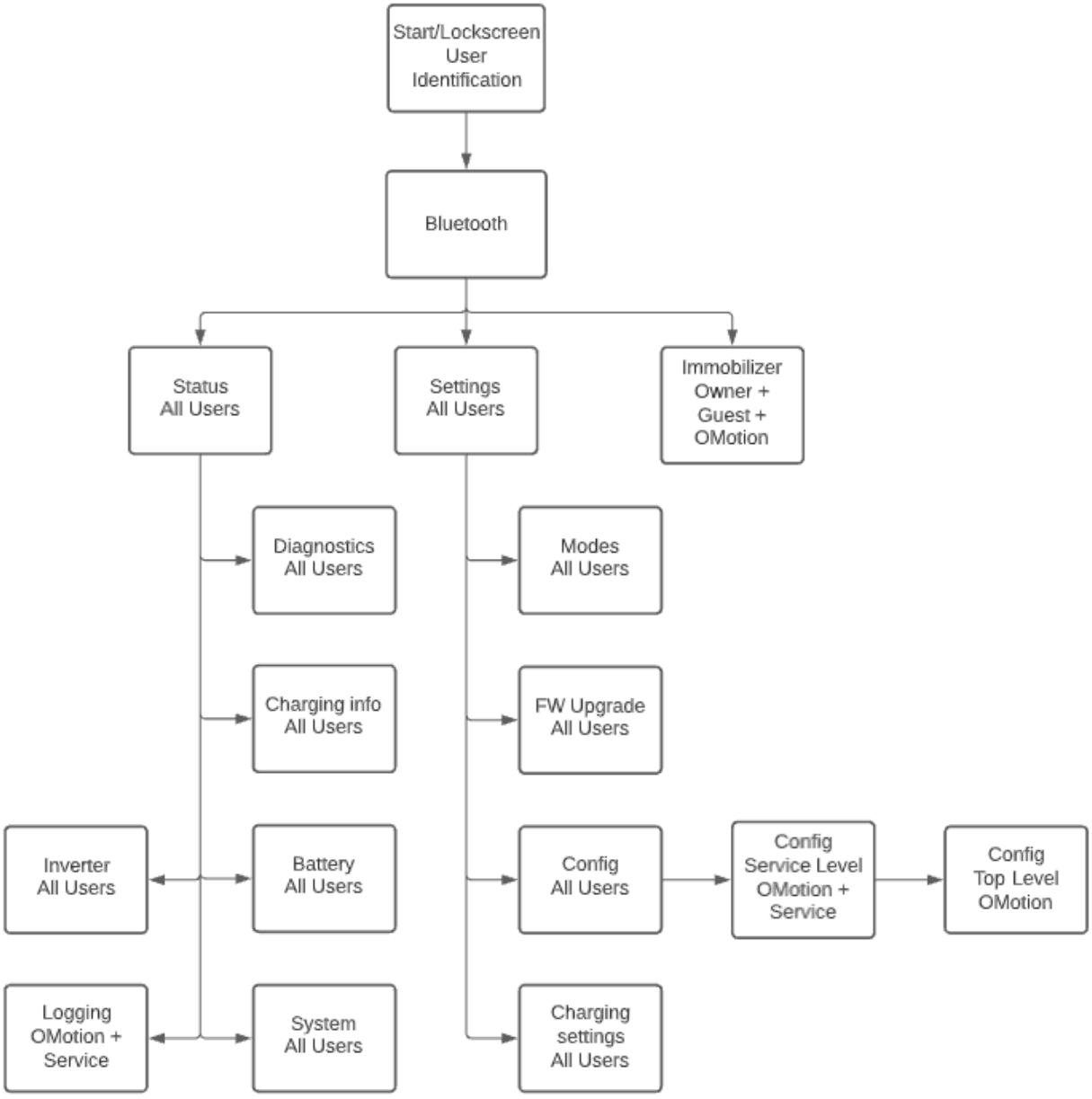
[11] Flutter, “What is Flutter?”, u. å. [Online]. Tillgänglig: https://docs.flutter.dev/resources/faq (hämtad: 2022-05-03).
[12] Google Firebase, “Google Firebase Products Build”, u. å. [Online]. Tillgänglig: https://firebase.google.com/products-build (hämtad: 2022-03-10).
[13] Google Firebase, “Firebase Authentication”, u. å. [Online]. Tillgänglig: https://firebase.google.com/docs/auth (hämtad: 2022-05-12).
[14] Okta, “Vad är skillnaden mellan OAuth, OpenID Connect och SAML?”, u. å. [Online]. Tillgänglig: https://www.okta.com/se/identity-101/whats-the-difference-between-oauth-openid-connect-and-saml/ (hämtad: 2022-05-12).
[15] Google Firebase, “Privacy and Security in Firebase”, u. å. [Online]. Tillgänglig: https://firebase.google.com/support/privacy (hämtad: 2022-03-10).
[16] Google Firebase, “Server-side encryption”, 2022. [Online]. Tillgänglig: https://cloud.google.com/firestore/docs/server-side-encryption (hämtad: 2022-03-21).
[17] Google Firebase, “Google Firebase Products Release”, u. å. [Online]. Tillgänglig: https://firebase.google.com/products-release (hämtad: 2022-03-10).
[18] Google Firebase, “Google Firebase Products Engage”, u. å. [Online]. Tillgänglig: https://firebase.google.com/products-engage (hämtad: 2022-03-10).
[19] NodeJS, “About Node.js”, u. å. [Online]. Tillgänglig: https://nodejs.org/en/about/ (hämtad: 2022-03-09).
[20] KeyCloak, “Setting up the relational database”, u. å. [Online]. Tillgänglig: https://www.keycloak.org/docs/latest/server_installation/#_database (hämtad: 2022-05-12).
[21] KeyCloak, “Configuring the database”, u. å. [Online]. Tillgänglig: https://www.keycloak.org/server/db (hämtad: 2022-05-12).
[22] KeyCloak, “Open source identity and access management”, u. å. [Online]. Tillgänglig: https://www.keycloak.org/ (hämtad: 2022-03-10).
[23] Cloudflare, “What is TLS (Transport Layer Security)?”, u. å. [Online]. Tillgänglig: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/ (hämtad: 2022-05-16).

[24] R. Broeckelmann, “DSig Part 2: JSON Web Signature (JWS)”, Medium, [Online]. Apr. 2016. Tillgänglig: https://medium.com/@robert.broeckelmann/dsig-part-2-json-web-signature-jws-f428d0b5ae40 (hämtad: 2022-05-17).

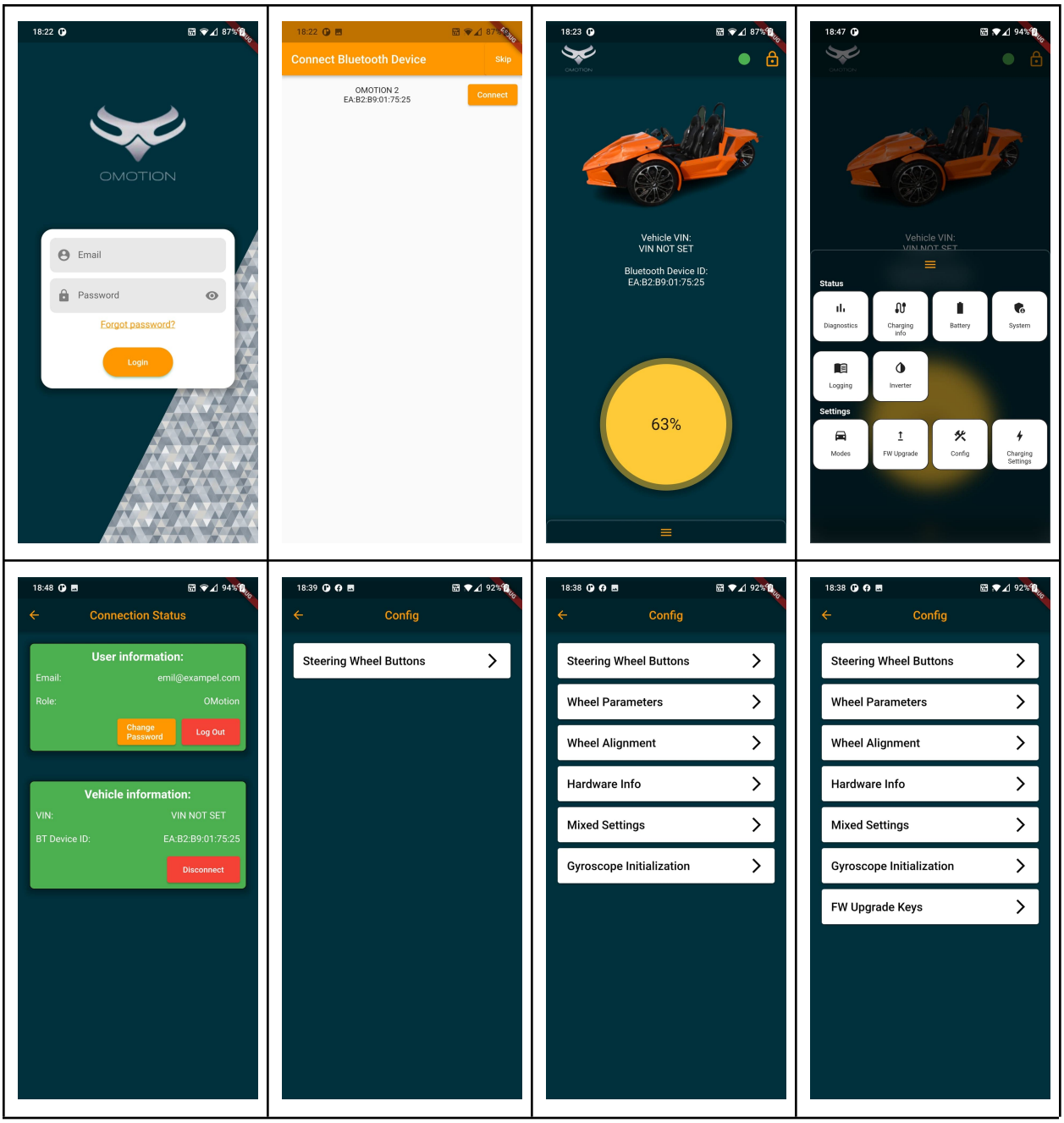
9 Appendix

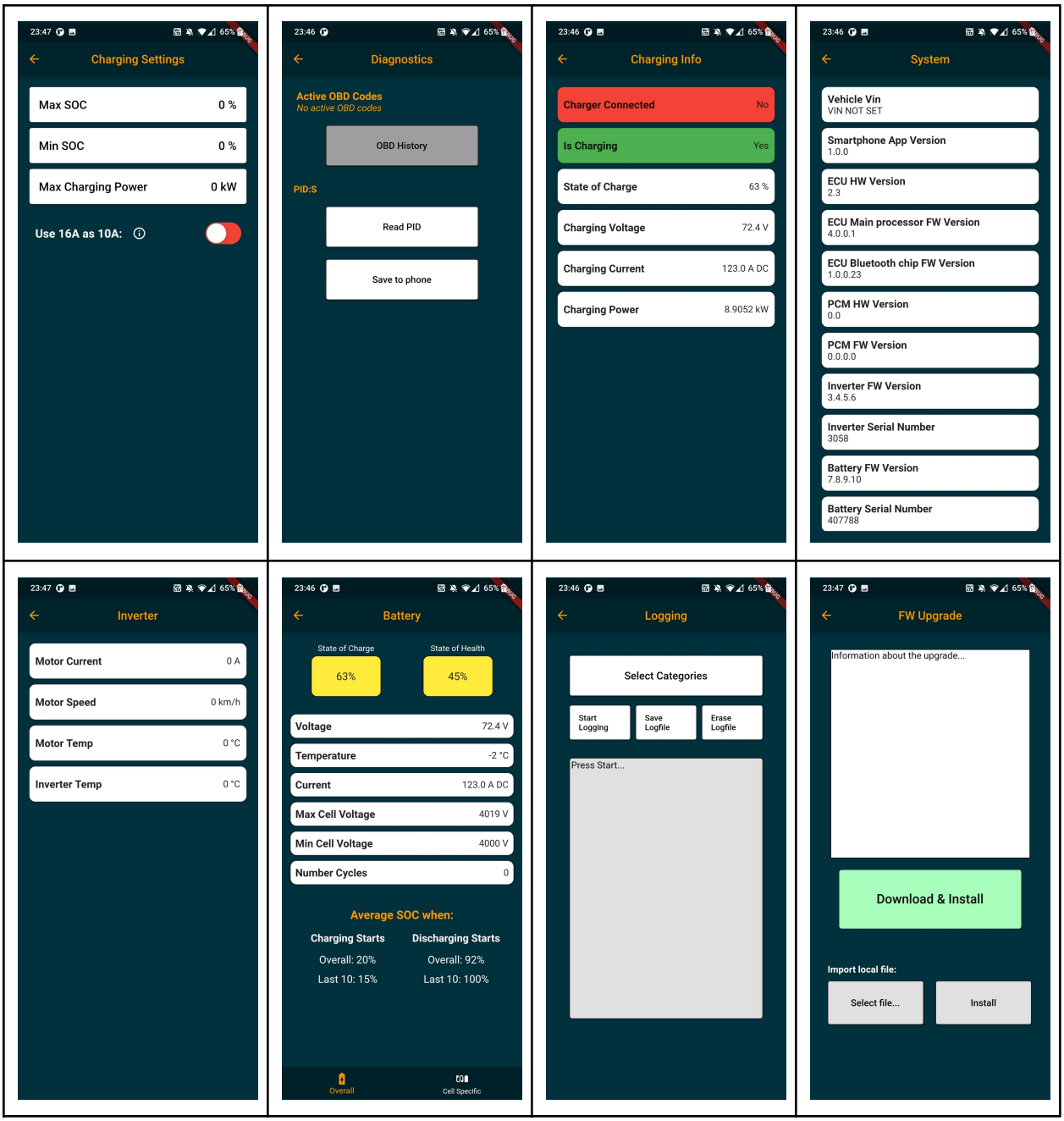
Appendix 1: Bild på OMotion 2

Appendix 2: Översikt över den generella strukturen på mobilapplikationen.



Appendix 3: Skärmdumpar av mobilapplikationen




Charging Settings
Max SOC 0 %
Min SOC 0 %
Max Charging Power 0 kW
Use 16A as 10A:
Diagnostics
Active OBD Codes
No active OBD codes
OBD History
PID:S
Read PID
Save to phone
Charging Info
Charger Connected No
Is Charging Yes
State of Charge 63 %
Charging Voltage 72.4 V
Charging Current 123.0 A DC
Charging Power 8.9052 kW
System
Vehicle Vin
VIN NOT SET
Smartphone App Version
1.0.0
ECU HW Version
2.3
ECU Main processor FW Version
4.0.0.1
ECU Bluetooth chip FW Version
1.0.0.23
PCM HW Version
0.0
PCM FW Version
0.0.0.0
Inverter FW Version
3.4.5.6
Inverter Serial Number
3058
Battery FW Version
7.8.9.10
Battery Serial Number
407788
Inverter
Motor Current 0 A
Motor Speed 0 km/h
Motor Temp 0 °C
Inverter Temp 0 °C
Battery
State of Charge
63%
State of Health
45%
Voltage 72.4 V
Temperature -2 °C
Current 123.0 A DC
Max Cell Voltage 4019 V
Min Cell Voltage 4000 V
Number Cycles 0
Average SOC when:
Charging Starts
Overall: 20%
Last 10: 15%
Discharging Starts
Overall: 92%
Last 10: 100%
Overall
Cell Specific
Logging
Select Categories
Start Logging
Save Logfile
Erase Logfile
Press Start...
FW Upgrade
Information about the upgrade...
Download & Install
Import local file:
Select file...
Install

Appendix 4: Testresultat, tid i sekunder

	Registrering av ny användare i databas	***Autentisering av användare i mobilapplikation***	***Börja läsa data från ECU:n***
Test 1	81.87	2.09	2.59
Test 2	92.32	1.98	2.25
Test 3	78.52	2.04	2.98
Test 4	85.66	2.38	2.34
Test 5	82.98	1.99	1.98
Test 6	90.36	2.00	2.07
Test 7	96.85	2.08	2.63
Test 8	78.82	2.33	2.41
Test 9	84.49	2.52	1.99
Test 10	87.57	2.21	2.14
Genomsnitt	***85.94***	***2.16***	***2.34***