Teemu Viljanen

IoT-sovelluskehyksen luotettavuuden parantaminen JWT-teknologian avulla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

12.3.2024

Tiivistelmä

Tekijä:	Teemu Viljanen
Otsikko:	IoT-sovelluskehyksen luotettavuuden parantaminen JWT-teknologian avulla
Sivumäärä:	38 sivua
Aika:	12.3.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Amir Dirin

Tämä opinnäytetyö käsittelee luotettavuutta IoT-systeemien näkökulmasta ja esittää keinon IoT-sovelluskehyksen luotettavuuden parantamiseksi TPM-turvapiiriä ja JWT-teknologiaa hyödyntäen. Pohjustuksena on esitetty katsaus luotettavuuden käsitteeseen ja siihen, mitä se merkitsee IoT-teknologian kontekstissa. Projektissa toteutettiin proof-of-concept-malli IoT-sovelluskehyksestä, joka mahdollistaa skaalautuvan ja luotettavan sensorisysteemin nopean ja helpon kehittämisen.

Toteutettu sovelluskehys perustuu innovaatioprojektissa syksyllä 2022 tehtyyn sovelluskehykseen. Innovaatioprojektin Python-ohjelmointikielellä tehdyn toteutuksen pohjalta on sovelluskehyksestä tehty uusi versio Go-ohjelmointikielellä. Sovelluskehys sisältää sensorien toiminnallisuudesta ja sensorien keräämän datan käytöstä vastaavat ohjelmat sekä web-pohjaisen käyttöliittymän systeemin toiminnan seuraamiseksi. Toteutettu versio sisältää uutena ominaisuutena laitteiden kaksivaiheisen autentikoinnin, joka on toteutettu TPM- ja JWT-teknologioita hyödyntäen. Lisäksi toteutus mahdollistaa usean sensorin käytön yhdellä systeemiin kiinnitetyllä laitteella, mikä ei aiemmassa versiossa ollut mahdollista.

Sovelluskehyksen toimintaa testattiin kattavasti useilla erilaisilla käyttötapauksilla. Testitapaukset osoittivat systeemin toimivan halutulla tavalla. Kun sovelluskehyksen toimintaa ja ominaisuuksia verrattiin vuoden 2022 projektissa tehtyyn toteutukseen, voitiin todeta, että tehdyt muutokset sovelluskehyksen toimintaan ja rakenteeseen paransivat sen luotettavuutta ja turvallisuutta. Sovelluskehys kokonaisuudessaan osoittaa myös lupausta jatkokehitysmahdollisuuksia ajatellen.

Avainsanat: IoT, Luotettavuus, Tietoturva, TPM, JWT, MQTT, Golang

Abstract

Author:	Teemu Viljanen
Title:	Improving Trustworthiness of IoT Framework Using JWT Technology
Number of Pages:	38
Date:	12 March 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Amir Dirin, Senior Lecturer

This thesis focuses on trust in the context of IoT systems and offers a possible way to improve the trustworthiness of an IoT framework, using TPM and JWT technologies. The paper first discusses the concept of trustworthiness and what it means in the context of IoT technology. The project work consisted of building a proof-of-concept model of an IoT framework that enables the fast and easy development of scalable and trustworthy sensor systems.

The framework is based on a model that was made in an innovation project in the fall of 2022. The implementation of that project was made with the Python programming language and formed the architectural basis for the developing of a new implementation of the framework, which was made using the Go programming language. The framework includes programs responsible for sensor functionality and the use of the data the sensors gather, as well as a web-based management interface for observing the functioning of the system. A new feature in the updated version is that for each device in the system, there is a two-step authentication process, which is implemented using TPM and JWT technologies. In addition, the new version allows for multiple sensors to run on a single device, which was not possible in the previous implementation.

The functioning of the framework was tested with several different use cases. The testing showed the system to function as expected. When comparing the differences between the implementation from 2022 and the new one, it is possible to assert that the changes made to the functioning and structure of the framework increased its trustworthiness and security. On the whole, the framework also shows promise for further development.

Keywords: IoT, Trust, Security, TPM, JWT, MQTT, Golang

Sisällys

Lyhenteet

IoT: *Internet of Things*. Fyysisten internetiin yhdistettyjen laitteiden luoma verkosto.

TPM: *Trusted Platform Module*. Fyysinen turvapiiri tai sen virtuaalinen versio, jota käytetään laitteiden turvallisuuden parantamiseen

MQTT: *Message Queue Telemetry Transport*. Verkkoprotokolla laitteesta laitteeseen tapahtuvan julkaisu-/tilausviestinnän jonotuspalvelu.

JWT: *JSON Web Token*. Standardi turvallisesti kapseloituun tiedonvälitykseen kahden tahon välillä.

1 Johdanto

Yksi suuri kysymys teknologian saralla on luottamus. Teknologiayhtiöiden jargonintäytteiset lehdistötiedotteet uusista innovaatioista ovat täynnä lupauksia ja ylisanoja uusista mahdollisuuksista. Ne kuitenkin hukuttavat alleen paljon epävarmuutta teknologian luotettavuudesta ja turvallisuudesta. Ihmiset ovat huolissaan maailman tilasta ja toivovat teknologiasta apua erilaisten ongelmien korjaamiseen. On kyse sitten vihreästä siirtymästä tai eriarvoisuuden kitkemisestä, teknologisen kehityksen niskaan ladataan suuria odotuksia, vaikka muutokset tapahtuvatkin aina kulttuurisista lähtökohdista [1].

Teknologia kehittyy koko ajan nopeaa vauhtia, ja kun esimerkiksi koneoppimisyritys OpenAI:n toimitusjohtaja Sam Altman vakuuttelee, että ”kukaan ei halua tuhota maailmaa” [2], voimme vain toivoa, että hän on oikeassa. Jotta teknologiasta puhuttaessa kuultaviin lupauksiin voitaisiin luottaa, on pystyttävä luottamaan ensin itse teknologiaan.

IoT-teknologiat ovat levinneet yhteiskunnan joka kolkkaan ja mullistaneet niitä tapoja, miten asioita maailmassa pyöritetään. Internetiin liitettyjä laitteita on joka paikassa. Melkein mistä tahansa sähkölaitteesta on nykyään saatavilla internetiin liitetty versio.

Tässä opinnäytetyössä tutkimusaiheena on IoT-systeemien luotettavuus ja turvallisuus. Tavoitteena on parantaa sensoreiden ja niiden tuottaman datan luotettavuutta. Vuoden 2022 syksyllä tehdyssä Metropolian innovaatioprojektissa toteutettiin ohjelmistokehys, joka mahdollistaa luotettavan ja turvallisen IoT-sensorisysteemin nopean kehittämisen [3]. Tämän opinnäytetyön tarkoituksena on jatkokehittää systeemiä lisäämällä siihen uusi turvallisuusominaisuus, joka tekee IoT-sensorisysteemistä entistä luotettavamman.

Tavoitteena on saada aikaan proof-of-concept-versio järjestelmästä, joka perustuu suurelta osin vuoden 2022 projektin arkkitehtuuriin, ja parantaa sen

luotettavuus- ja turvallisuusominaisuuksia. Uusi versio toteutetaan Go-kielellä ja siihen kuuluu kaksivaiheinen laitteiden autentikointivaihe, jossa yhdistyvät TPM-sirun avulla tehty etäautentikointi ja JWT-pohjainen salaus [4].

Tässä raportissa pohjustetaan aluksi hieman sitä, mitä luotettavuudella tarkoittaa teknologian yhteydessä ja miten luotettavuus pitäisi ottaa huomioon IoT-teknologioiden yhteydessä. Aihe on tietysti laaja, ja IoT-järjestelmien turvallisuuden ja luotettavuuden alueissa on paljon kiinnostavaa. Kerätyn datan suojaaminen ja turvallinen säilytys on oma alueensa, josta on kirjoitettu useita opuksia, mutta tämän opinnäytetyön piiriin se ei mahdu. Sensoridatan täsmällisyys ja virheettömyys ovat myös kiinnostavia osa-alueita, mutta nekin on jätettävä pois tämän tutkimuksen piiristä. Tässä opinnäytetyössä keskitytään tutkimaan sitä, miten on mahdollista parantaa IoT-sensorisysteemin luotettavuutta järjestelmän arkkitehtuurisuunnittelun kautta.

Taustan pohjustuksen jälkeen raportissa esitellään projektissa toteutetussa IoT-sovelluskehyksessä käytetyt teknologiat, sovelluskehyksen rakenne ja toiminta. Sen jälkeen käydään läpi systeemin testaus ja siitä saadut tulokset. Lopuksi pohditaan vielä jonkin verran sovelluskehyksen toteutuksen onnistumista ja sen käyttö- ja jatkokehitysmahdollisuuksia.

2 Taustaa

Mitä luotettavuus tarkoittaa teknologian suhteen? Mikä tekee jostain teknologiasta luotettavan?

Se että käyttäjä luottaa teknologiaan näkyy käyttäjän uskomuksissa ja käyttäytymisessä. Ensinnäkin käyttäjän täytyy uskoa, että teknologialla on kyky toteuttaa ne toiminnot, jotka sen halutaan toteuttavan. Käyttäjän on oltava valmis asettamaan itsensä haavoittuvaiseen asemaan ja luotettava siihen, että teknologiaa käytetään vain siihen tarkoitukseen, mihin sitä sanotaan käytettävän, eikä sillä ole muita, mahdollisesti kyseenalaisia, tarkoitusperiä. Tärkeää on myös, että voi uskoa, että teknologian käyttämä ja keräämä data on täsmällistä

ja virheetöntä. Kun käyttäjällä on tarpeeksi vahva uskomus teknologian luotettavuuteen, näkyy se käyttäytymisessä, jolloin käyttäjä voi antaa teknologian tehdä tehtävänsä ilman, että yrittää kontrolloida sitä. [5.]

Yleisesti käytetty määritelmä teknologian ja datan luotettavuuden pohjana on kolmen kohdan niin sanottu CIA-kolmikko: *Confidentiality, Integrity, and Availability*, eli luottamuksellisuus, eheys ja saatavuus [6]. Luottamuksellisuus tarkoittaa käytännössä sitä, että tietoon pääsee käsiksi vain sellaiset tahot, joilla on oikeanlainen valtuutus. Eheys tarkoittaa sitä, että tieto pitää muotonsa koko elinkaarensa ajan, eikä sitä voi muokata ilman oikeanlaisia valtuutuksia. Saatavuus tarkoittaa sitä, että tieto on saatavilla silloin kun sitä tarvitaan. [7.]

Näiden asioiden hoitamiseksi on kehitelty monenlaisia menetelmiä niin laitteiden kuin ohjelmistojenkin osalta. Muun muassa erilaiset salausmenetelmät, autentikointi- ja pääsynhallintaprotokollat sekä palomuurit ovat keinoja systeemien turvallisuuden lisäämiseksi ja luottamuksellisuus-, eheys- ja saatavuustavoitteiden saavuttamiseksi. [8.]

IoT, eli *Internet of Things*, tarkoittaa yhteen liitettyjen laitteiden verkostoa, jonka laitteet voivat kommunikoida keskenään ja jotka keräävät ja liikuttavat dataa internetin välityksellä [9]. IoT-kontekstissa teknologian luotettavuudella on omat erityisongelmansa. Koska laitteita on useita ja niissä voi olla monenlaisia erilaisia ohjelmistoja, mitään helppoa yhden konstin menetelmää ei ole olemassa.

Yksi metodi IoT-systeemien luotettavuuden arvioinniksi on TM, eli *Trust Management*, eli luotettavuudenhallinta. TM-prosessin tarkoituksena on arvioida verkoston luotettavuutta antamalla jokaiselle verkoston noodille arvo, joka kertoo sen luotettavuuden tasosta [9]. Hyvä luotettavuudenhallintaprosessi on tärkeä osa systeemin toimintaa, jotta voidaan muodostaa käsitys koko systeemin turvallisuus- ja luotettavuustilanteesta ja voidaan saada selville, mitä on tehtävissä sen parantamiseksi [10].

Selkeä luotettavuudenhallintaprosessi parantaa myös käyttäjien käsitystä siitä, mitä riskejä laitteissa on ja miten hyvin ja missä määrin laitteisiin voi luottaa. Käyttäjän luottamus laitteisiin on monimutkainen yhtälö, jossa yhdistyvät kokemukset, uskomukset ja oletukset teknologiasta ja sen tekijöistä. Myös teknologian, ohjelmistojen ja laitteiden maine vaikuttaa käyttäjien luottamukseen. Mitä paremmin niiden tekijät hommansa hoitavat, sitä helpommin käyttäjät uskaltavat teknologiaa käyttää. [10.]

Monet IoT-laitteiden valmistajat eivät kuitenkaan ota tietoturvaa tarpeeksi vakavasti. Valmistajia on yhtenään kehotettu ottamaan enemmän turvallisuuskäytäntöjä käyttöön, jotta yritykset ja yksityiset käyttäjät eivät joudu alttiiksi vahingoille [11]. Silti tutkimuksissa on havaittu, että turvallisuus on ensimmäinen asia, josta leikataan, kun etsitään säästöjä tuotantoprosesseihin [8].

Turvallisuuteen panostava ohjelmistotuotanto, *SecDevOps* [12], voi osaltaan parantaa laitteiden turvallisuutta, mutta vastuu systeemien luotettavuudesta kuuluu kaikille prosessissa mukana oleville tekijöille kaikissa tuotantovaiheissa. Kuten Chevuru et al sanovat: ”[IoT-ekosysteemin] turvallisuussuunnittelu ei voi alkaa sovelluskehittäjästä, sen pitää alkaa silikoniosien suunnittelusta, osien valmistusvaiheesta, ja sen pitää jatkua alustojen ja sovellusten suunnittelussa ja installoinnissa ja käyttötuessa” [13, xxvii].

2.1 IoT-systeemien turvallisuushaasteita

IoT-systeemeissä on monia piirteitä, jotka lisäävät niiden turvallisuuden takaamisen haasteellisuutta ja jotka ovat tärkeitä ottaa huomioon systeemejä suunnitellessa. Näitä piirteitä voivat olla systeemin suuri koko, laitteiden heterogeenisyys ja monimuotoisuus, resurssien rajoitteet, funktionaalisuuden korostaminen turvallisuuden sijasta ja korkeat vaatimukset yksityisyydensuojalle. [8.]

Resurssien rajoitteilla tarkoitetaan monien IoT-laitteiden rajallista laskentatehoa ja vähäistä muistia verrattuna tavallisiin tietokoneisiin. Tämä rajoittaa mahdollisuuksia käyttää perinteisiä tunnistautumiskeinoja. Huonosti suunniteltujen

turvallisuusominaisuuksien takia IoT-laitteita onkin usein helpompi ottaa haltuun kuin perinteisiä tietokoneita. [8.]

IoT-teknologian arvo on pitkälti sen monipuolisuuden ja mukautuvuuden ansiota. IoT-systeemi voi pitää sisällään monenlaisia laitteita, joilla on erilaisia konfiguraatioita ja ohjelmistoja [8]. Mitä monimutkaisempi järjestelmä on ja mitä enemmän erilaisia laitteita siihen on kiinnitetty, sitä enemmän sillä on myös turvallisuushaasteita. Kun monet IoT-laitteet käyttävät erilaisia ohjelmistoja ja kommunikaatiokanavia, perinteisten turvallisuusratkaisujen soveltaminen voi olla mahdotonta. Usein seurauksena on se, että eri systeemin osissa on eritasoinen turvallisuus. Tällaisissa tapauksissa siitä systeemin laitteesta, jossa on heikoin turvataso, tulee koko systeemin heikoin lenkki. Se määrittää koko systeemin turvatason. [8.]

Toisiinsa yhteydessä olevien laitteiden määrä lisää mahdollisten hyökkäyskohteiden määrää, ja systeemin kasvaessa jokainen järjestelmään lisätty uusi laite on uusi mahdollinen hyökkäyspiste. Täten dynaamisissa systeemeissä on erityisen tärkeää löytää ja havaita alhaisen tietoturvan kohteet ja pystyä paikantamaan hyökkäyksille alttiit noodit. [9.]

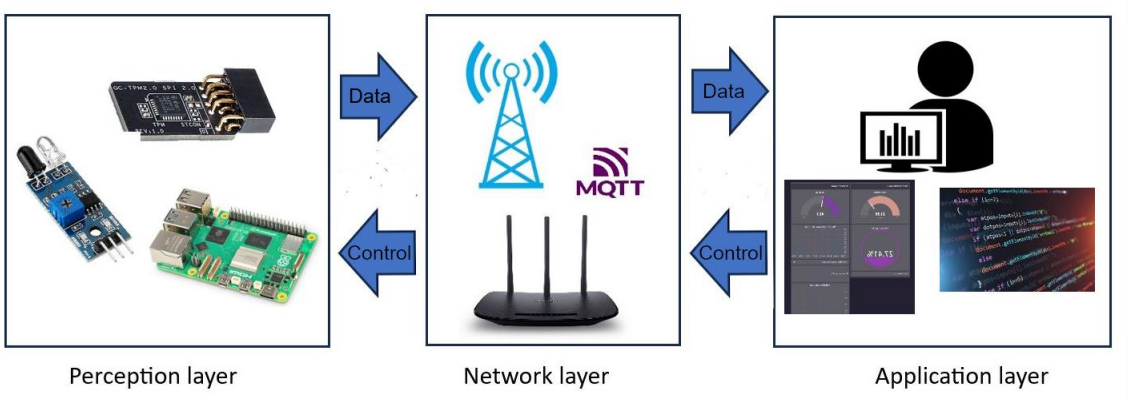
Monet IoT-systeemit ovat kriittisessä osassa toimintaa eikä niiden käyttöä voi tuosta vain lopettaa. Jos kyseessä on esimerkiksi teollisuudessa usein käytössä oleva SCADA-systeemi (*Supervisory Control And Data Acquisition*), on usein mahdotonta käyttää uudelleenkäynnistys- ja resetointiprosesseja, koska tuotantoprosessia ei ole mahdollista tuosta vain tauottaa [8].

IoT-systeemeissä on aina sekä fyysinen että verkon kautta toimiva osa, joiden molempien täytyy toimia oikein, jotta systeemi toimii. Tämä tarkoittaa myös sitä, että jos mahdollinen hyökkääjä saa verkon kautta toimivan osan hallintaansa, saa hyökkääjä myös fyysisen laitteen käyttöönsä. Vaarassa voi olla tietojen lisäksi myös systeemiin liitetyt fyysiset laitteet. [8.]

Esimerkiksi vuonna 2015 hakkerit saivat murrettua Jeepin salaukset ja saivat sen ohjausjärjestelmän käyttöönsä kesken ajon [14]. Heillä oli käytössään ohjelmisto, joka salli hakkereiden lähettää käskyjä Jeepin viihdejärjestelmän kautta kaikkiin kojelaudan funktioihin ohjauksesta jarruihin. Vuonna 2023 Tesla järjesti hackaton-tapahtuman, jossa testattiin, onko Teslan turvajärjestelmät mahdollista ohittaa, mikä onnistui niin ikään keskittämällä hyökkäys ajoneuvon viihdejärjestelmään [15]. Aiemmin alalla hyvänä esimerkkinä IoT-systeemien haavoittuvaisuudesta on käytetty ydinvoimaloiden SCADA-systeemeihin hyökännyttä stuxnet-virusta, jonka avulla onnistuttiin ottamaan käyttöön ydinvoimalan sentrifugien toiminta [13, s. 4-5].

Jos hyökkääjä pääsee käsiksi laitteille, on niille helppo myös asentaa haittaohjelmia. Kaikenlaisien internetiin liitettyjen laitteiden haavoittuvuuksia hyväksi käyttävä Mirai-haittaohjelma voi ottaa suuren joukon laitteita hallintaansa. Hyökkääjä voi koota Mirain avulla kaapatuista laitteista bottiarmeijan, jota voi käyttää palvelunestohyökkäyksissä muihin kohteisiin. [16.]

Tavallisesti IoT-systeemi koostuu kolmesta tasosta: fyysistä havaintotasosta (*Perception layer*), verkkotasosta (*Network layer*) ja ohjelmistotasosta (*Application layer*) [9; 10].



Kuva 1. IoT-systeemin tasot: havaintotaso, verkkotaso ja ohjelmistotaso.

Kuvassa 1 on havainnollistettu IoT-systeemin tasoja ja niiden suhdetta toisiinsa. *Perception layer*, eli havaintotaso koostuu fyysisitä laitteista ja sensoreista.

Network layer, eli verkkotaso on vastuussa laitteiden yhdistämisestä toisiinsa ja ohjelmistoihin. Tähän tasoon lasketaan kuuluvaksi myös käytetyt kommunikaatioprotokollat. *Application layer*, eli ohjelmistotaso koostuu ohjelmistoista, jotka hyödyntävät laitteista kerättyä dataa ja mahdollisesti ohjaavat systeemiin kuuluvien laitteiden toimintaa. [9.]

Laitteiden yhdistäminen yhdeksi toiminnalliseksi kokonaisuudeksi on hyväksi toiminnan tehokkuudelle, mutta kuten jo todettua, se tuo mukanaan lisähaasteita luotettavuuden takaamiseen. Yhden tason turvaaminen ei kerro koko systeemin luotettavuudesta [10; 17]. Hyvän ja kattavan luotettavuudenhallintaprosessin pitäisikin kattaa kaikkien kolmen tason toimijat.

Kun laitteiden käyttöä ohjaavat viestit kulkevat systeemissä, on kulkuradalla monta mahdollista kohtaa, jossa voi olla haavoittuvaisuuksia. Mahdollinen hyökkäys voi kohdistua mihin systeemin tasoon hyvänsä [8]. Koska laitteet ovat yhteydessä toisiinsa, voi esimerkiksi kommunikaatioprotokollien haavoittuvaisuuksia hyödyntämällä saada systeemiin kuuluvan laitteen käyttöön tai katkaista tiedonkulun ohjelmistoihin [17]. Molemmissa tapauksissa seuraukset voivat olla vakavia.

2.2 IoT-systeemin laitteiden ja sensorien luotettavuudesta

Luotettavat laitteet ovat olennaisia IoT-järjestelmien toimivuuden ja turvallisuuden kannalta [21]. Ylempänä mainittua CIA-kolmikkoa mukaillen Loi et al. [11] nimeävät neljä osa-aluetta IoT-laitteiden turvallisuuden arvioimiseksi: 1) laitteiden lähettämän ja vastaanottaman datan luottamuksellisuus, 2) laitteiden välisten yhteyksien eheys ja autentikointi, 3) laitteiden saavutettavuus ja pääsynhallinta, 4) laitteiden kyky osallistua refleksiivisiin hyökkäyksiin, jolla tarkoitetaan sellaisia laitteiden haavoittuvaisuuksia, jotka mahdollistavat niiden käytön esimerkiksi DDoS- eli palvelunestohyökkäyksissä.

IoT-systeemien suunnittelussa pitäisi olla lähtökohtana, että vain luotetuista laitteista voidaan lähettää ja vastaanottaa dataa. Laitteen luotettavuuden

perustekijänä on laitteiden täsmällinen tunnistaminen. Hyvin suunnitellussa systeemissä on tapa pitää kirjaa tunnistetuista laitteista, ja niiden toiminnallisista oikeuksista. Kun laite on tunnistettu luotettavasti, seuraava vaihe on pääsynhallinta, jolla varmistetaan, minkälainen pääsy systeemin tietoihin ja mahdollisiin toimintoihin kullakin laitteella on. Laitteiden luotettavuutta pitäisi myös pystyä tarkkailemaan prosessien aikana, ja jos jotain epäilyttävää huomataan, olisi hyvä, että niiden käyttäytymistä ja pääsyoikeuksia datan ja ohjelmistojen pariin rajoitetaan. [9.]

Jokainen systeemin laite on yhtä tärkeässä osassa koko systeemin luotettavuuden kannalta, ja hyvin suunnitellussa systeemissä vain luotetut laitteet voivat vaikuttaa toistensa toimintaan. Kunkin laitteen luotettavuuden tason pitäisi määrittää se, miten kyseinen laite voi olla vuorovaikutuksessa systeemin muiden laitteiden kanssa. [9.]

On kyse sitten minkälaisesta IoT-systeemistä tahansa, siinä on todennäköisesti jonkinlaisia sensoreita. Sensorit ovat erityisen hankalia tietoturvan kannalta, koska ne ovat usein rakenteeltaan yksinkertaisia, eikä niillä ole paljoa prosessointikykyä tai muistia monimutkaisten turvaratkaisujen käyttöön [17]. Samalla on kuitenkin erityisen tärkeää, että sensoreista kerättyyn dataan voidaan luottaa. Jos havaintotason laitteiden keräämä data ei ole oikeaa ja luotettavaa, esimerkiksi viallisten tai vääriin käsiin joutuneiden sensorien takia, koko systeemin toiminta voi olla vaakalaudalla, vaikka verkkotason ja ohjelmistotason tietoturva olisikin hyvällä tolalla. [10.]

Skaalautuvassa IoT-systeemissä yksi ongelmista on uusien toisilleen tuntemattomien laitteiden mukaan tulo. Systeemien suunnittelussa onkin aina syytä kehittää protokollat uusien laiteiden mukaan tuontia varten. Usein se tarkoittaa sitä, että suoritetaan jonkinlaiset esivalmistelut ennen kuin laite kytketään systeemiin mukaan. [18.]

Yksi yleinen tapa hallita uusien laitteiden tunnistamista on käyttää satunnaisavaimien joukkoa, jolloin käytössä on salattujen avaimien pooli, josta jokainen

uusi laite saa avaimen esivalmisteluvaiheessa. Sen avulla se tunnistetaan luotettavaksi ja se pääsee osaksi systeemin toimintaa [18]. Keskitetty avainpooli sallii systeemin skaalautumisen, mutta on kuitenkin rajallinen [8].

Yksi satunnaisavainten poolia parempi tapa parantaa laitteiden ja siten koko systeemin luotettavuutta on etäautentikointi (*Remote Attestation*), jossa systeemiin tulevan uuden laitteen tilasta tehdään sellainen tunniste, jonka perusteella varmentajataho voi päätellä onko laite luotettava vai onko sen asetuksia ehkä muokattu [19]. TPM-turvapiirin (*Trusted Platform Module*) ominaisuuksiin pohjautuva attestointi on oiva tapa varmistaa laitteen tila ja luotettavuus. Tämän opinnäytetyön osana tehty IoT-sovelluskehys käyttää TPM:n ominaisuuksiin pohjautuvaa attestointia ja sitä avataan enemmän myöhemmin.

TPM-turvapiirin ongelmana on, että se vaatii resursseja. Voi olla kallista, jos jokaiseen pieneen sensoriin halutaan oma turvapiiri. Lisäksi vanhojen laitteiden uudelleenkonfiguroiminen voi olla työlästä ja epäkäytännöllistä [20]. Kuitenkin jos turvallisuutta ja luotettavuutta pidetään tärkeinä, niin niihin on syytä panostaa resursseja.

Weisong et al huomauttavat myös, että usein IoT-systeemeiltä puuttuu hyvä keskitetty hallinnointi-infrastruktuuri, jonka avulla voitaisiin pitää kirjaa laitteiden käyttäytymisestä [8]. Tämän opinnäytetyön osana tehtyyn sovelluskehykseen kuuluu käyttöliittymä, josta voidaan seurata kaikkia systeemissä kiinni olevia laitteita ja niiden luotettavuustilannetta reaaliajassa. Toteutetusta sovelluskehyksestä enemmän seuraavaksi.

3 IoT-sovelluskehys

Tämän opinnäytetyön osana toteutettu ohjelmisto on IoT-sovelluskehys, joka mahdollistaa skaalautuvan sensorisysteemin nopean ja helpon kehittämisen.

Projektissa toteutetun järjestelmän arkkitehtuuri perustuu suurelta osin vuoden 2022 syksyllä toteutetun innovaatioprojektin tuloksena syntyneeseen IoT-

sovelluskehykseen [3]. Edellisessä toteutuksessa oli tehty sovelluskehys, jossa jokaisen systeemiin liitetty laite todennettiin sen tunnistearvon perusteella. Mikäli laitteen tunnistearvo löytyi erillisestä attestointitietokannasta, laite todettiin turvalliseksi ja sen keräämää dataa saatettiin käyttää. Systeemiin kuului myös selaimessa toimiva hallintaohjelma, josta oli mahdollista nähdä systeemissä olevien laitteiden määrä ja niiden luotettavuusaste. Systeemin päälle tehtiin vielä testisovellus, joka tunnisti ihmisen läsnäolon tilassa infrapunakameran ja etäisyysanturin avulla ja otti kuvan webkameralla aina, kun anturit havaitsivat oikeanlaiset arvot.

Tämän projektin tarkoituksena on kehittää systeemin luotettavuutta entisestään. Laitteen autentikoinnista on tehty kaksiosainen. Käytössä on TPM:n avulla luotu avain, joka perustuu laitteen tilaan, jolloin laitteen tunnistearvo kertoo vain sen, onko laite järjestelmässä, ja avain kertoo, onko laite halutussa tilassa. Tämä parantaa systeemin luotettavuutta huomattavasti, koska systeemin laitteisiin tehdyt muutokset huomataan.

Koska uusi järjestelmä on toteutettu Go-kielellä, se poikkeaa hieman aikaisemmasta Pythonilla tehdystä toteutuksesta. Go ei esimerkiksi tue periytymistä samalla tavalla kuin Python, joten ohjelmien toteutus eroaa niiltä osin.

Ainoa aspekti, joka on lähes muuttumaton vuoden 2022 projektista on websivuna toimiva lokitietoja ja näyttävä ja tulkitseva ohjelma. Siihen tehdyt muutokset olivat minimaalisia ja liittyivät lähinnä JSON-muodossa kulkevien viestien muotoiluun, jotta ohjelma osaa lukea ne oikein.

3.1 Käytetyt teknologiat

Tässä osiossa esitellään lähemmin tärkeimmät projektissa käytetyt teknologiat.

Ohjelmisto on toteutettu Go-kielellä. Sen käyttöön päädyttiin, koska Gon käyttö on yleistynyt ohjelmistomaailmassa ja IoT-projekteissa [21] ja haluttiin tietää, miten se soveltuisi tähän tehtävään. Gon etuja erityisesti IoT-projekteihin liittyen

ovat sen tehokkuus, kielen sisäänrakennettu rinnakkaisajotoiminnallisuus, hyvä skaalautuvuus sekä yhteensopivuus monien alustojen kanssa [22]. Esimerkiksi Java- ja Python-kieliin verrattuna Gon on havaittu olevan tehokkaampi ja nopeampi, mikä on erityisen tärkeää varsinkin teollisuuden IoT-sovelluksissa, joissa toiminnan optimointi ja nopeus ovat arvokkaita aspekteja [23].

3.1.1 TPM ja etäautentikointiprotokolla

TPM (*Trusted Platform Module*) [24] on fyysinen turvapiiri, joka voidaan asentaa elektroniseen laitteeseen. TPM-sirulle on mahdollista tallentaa tietoja laitteen tilasta (PCR), ja sen avulla on mahdollista luoda kryptografisia avaimia. TPM:n ominaisuudet mahdollistavat laitteiden autentikoinnin ja niillä voi esimerkiksi saada selville, onko laitteiden asetuksia muutettu.

TPM-sirua on mahdollista käyttää myös virtuaalisessa muodossa. Esimerkiksi IBM on kehittänyt TPM-ohjelmiston, jota voi käyttää testaustarkoituksissa. Tämän projektin kehityksessä ja testauksessa käytettiin vaihtoehtoista SWTPM-kirjastoa [25].

Etäautentikoinnin tarkoituksena on se, että mitään järjestelmään liitettyä laitetta ei oteta järjestelmän käyttöön ennen kuin se on autentikoitu ja todettu luotettavaksi. Etäautentikointiprotokollan logiikka on seuraavanlainen: TPM-sirun avulla luodaan identifioiva tunnus, käytännössä avain, jonka julkinen osa tallennetaan ulkoiseen tietokantaan. Niin kauan kuin laitteen kokoonpano ja tila pysyvät samanlaisina TPM-sirulta haettu avain vastaa tietokantaan tallennettua julkista avainta. Etäautentikointi toteutetaan vertaamalla TPM-sirulta saatavaa avainta toisessa paikassa sijaitsevaan julkiseen avaimeen. Etäautentikoinnin voi suorittaa niin usein kuin haluaa, ja onkin yleisesti ottaen hyvä idea tarkastaa, onko laitteella tapahtunut muutoksia tietyin väliajoin.

3.1.2 JWT

JWT, eli *JSON Web Token*, [26] on tapa välittää tietoa salattuna systeemin eri osien välillä JSON-objektina. JWT:n tärkeä ominaisuus on, että jokainen JWT voidaan allekirjoittaa avaimella, joka avaajan on tiedettävä, jotta JWT:n sisältämä tieto saadaan esiin.

JWT koostuu kolmesta osasta, joita ovat *header* ja *payload* ja *signature*. Header-osa pitää yleensä sisällään tiedot tokenin tyypistä ja käytetystä salausalgoritmista. Payload-osa sisältää niitä tietoja, jotka halutaan välittää systeemin osien välillä. Yleensä payload-osassa ilmoitetaan myös, esimerkiksi, tokenin julkaisija ja tokenin voimassaolon päättymisaika. Signature-osa luodaan allekirjoittamalla tiedot header- ja payload-osista käyttämällä header-osassa mainittua salausalgoritmia ja salasanaa. Allekirjoitus varmistaa, ettei viesti ole muuttunut matkallaan, ja yksityisen salasanan käyttö varmistaa, että viesti on luotettavasta lähteestä. [26.]

3.1.3 MQTT

MQTT on yksinkertainen ja kevyt kommunikaatioprotokolla, jonka avulla on mahdollista lähettää ja vastaanottaa viestejä. Se vaatii toimiakseen palvelimen, jota kautta viestit kulkevat. Viestejä julkaisevien ohjelmien tulee nimetä käyttämänsä julkaisukanava, ja kuuntelevien ohjelmien tulee nimetä kanava, josta ne voivat vastaanottaa viestejä. Kaikki samaa palvelinta käyttävät ohjelmat voivat kommunikoida keskenään, kun ne käyttävät samoin nimettyjä kommunikaatiokanavia. [27.]

Tämän projektin toteutuksessa systeemin sisäinen ja toimintaa ohjaava kommunikaatio tapahtuu MQTT-kanavalla ”management”. Lisäksi käytössä on sensorin data-arvojen lähetykseen tarkoitetut kanavat, jotka voidaan nimetä kunkin sensorin käyttötarkoituksen mukaisesti.

3.1.4 Tietokannat

Toteutuksessa käytetään kahta tietokantaa. Toinen on attestointi-tietokanta, joka kehitysvaiheessa toteutettiin yksinkertaisena relaatiotietokantana. Relaatiotietokanta on tietokanta, jossa yksi tietue koostuu yhden tai useamman sarakkeen tiedoista. Se on yksinkertainen tapa järjestää tietoa helposti ymmärrettävään muotoon [28]. Kehitysvaiheen alustava testitietokanta toteutettiin MariaDB-ohjelmistolla, se sisältää yhden taulun, jossa on kolme saraketta: *name*, *itemid* ja *pubKey*, jotka vastaavat laitteen nimeä, itemid-tunnistetta ja TPM-sirun avulla tehtyä julkista avainta.

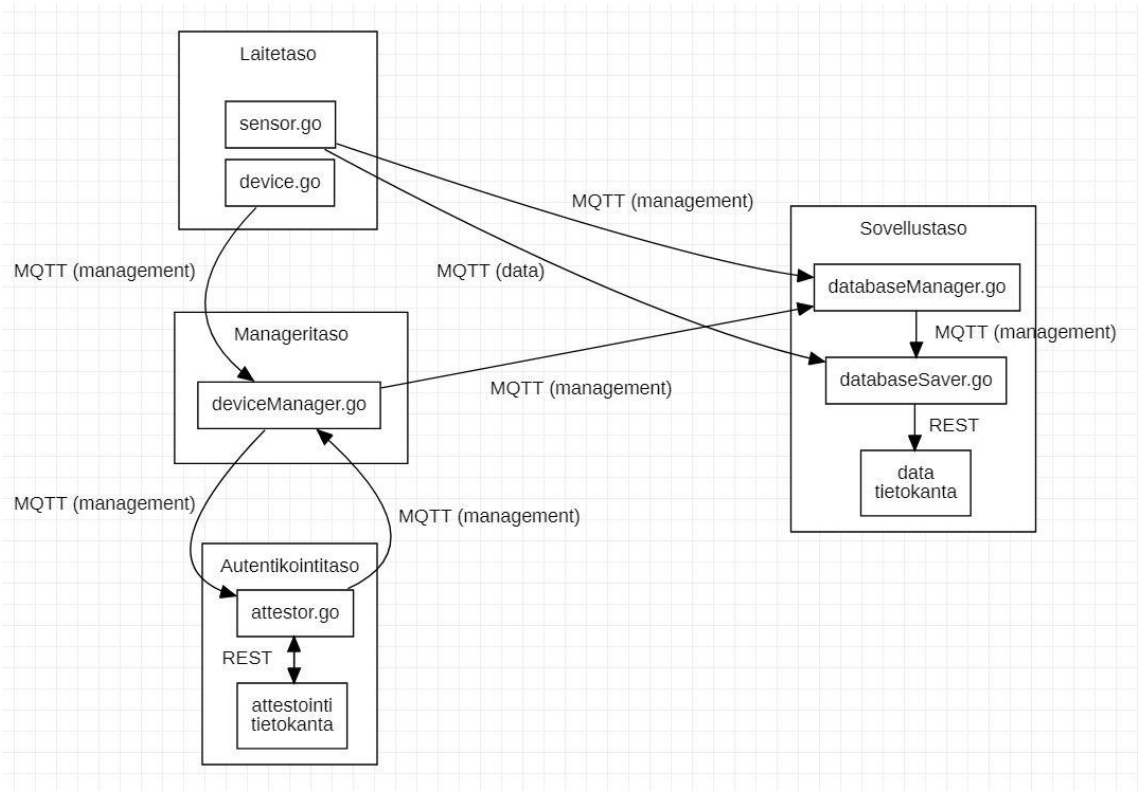
Toinen tietokanta on datan tallennukseen käytetty tietokanta, joka on toteutettu aikasarjatietokantana InfluxDB-ohjelmistolla. Aikasarjatietokanta on tietokanta, joka on optimoitu aikaleimatun datan keräämiseen. Tietokantaan kerätään data-arvoja, joiden kehitystä ja muutoksia voidaan helposti seurata ajan kuluessa. [29.] Tässä toteutuksessa jokainen tallennettu data-arvo saa tunnisteena sensorin datakanavan nimen, ja influxDB lisää automaattisesti aikaleiman jokaiseen tallennettuun arvoon. Kerättyä dataa on helppo tarkastella esimerkiksi tekemällä hakuja sensorin datakanavatunnisteen perusteella.

3.1.5 Hallinnallinen web-sivusto

Ohjelmistoon kuuluva selaimessa toimiva hallintaohjelma on toteutettu käyttäen Node.js-palvelinta ja JavaScript-kieltä käyttävää web-sivustoa. Sivusto näyttää lokitiedot kaikista systeemin ”management”-kanavalla tapahtuvasta liikenteestä sekä näyttää tiedot systeemissä olevista laitteista ja niiden luotettavuustiloista.

3.2 Yleiskuva systeemin rakenteesta

Järjestelmä koostuu viidestä erillisestä osa-alueesta. Samaan osa-alueeseen kuuluvat ohjelmat tulee olla toiminnassa samalla laitteella, mutta eri osa-alueet voivat, ja niiden tulisikin, toimia eri laitteilla. Osa-alueet ovat 1) laitetaso, 2) manageritaso, 3) autentikointitaso, 4) sovellustaso sekä 5) UI-taso.



Kuva 2. Systeemin osat ja niiden välinen kommunikaatio.

Kuvassa 2 on esitetty, mitkä ohjelmat tasoihin kuuluvat ja mitä protokollaa ne käyttävät kommunikointiin.

Laitetason ohjelmat pyörivät yksittäisillä systeemiin liitetyillä laitteilla. Ne toimivat laitteiden ohjausohjelmina ja niiden toiminnallisuuden toteuttajina. Laitetasoon kuluu ohjelmat device.go, sensor.go.Device.go on vastuussa laitteen toiminnallisuudesta ja sensor.go on vastuussa sensorin toiminnallisuudesta. Laitetason ohjelmat kommunikoivat manageri- ja sovellustason ohjelmien kanssa.

Manageritason ohjelmat voivat olla joko omalla laitteellaan, tai millä tahansa laitteella, joka on MQTT-yhteydessä systeemiin. Manageritason ohjelma(t) vastaa laitteiden ja sensorien liittämisestä systeemiin ja kommunikoi laite-, autentikointi- ja sovellustasojen välillä. Manageri-tasoon kuuluu ohjelma deviceManager.go, joka on vastuussa laitteiden luotettavuuden määrittelystä.

Autentikointitason ohjelmat toimivat omalla erillisellä laitteellaan. Ne ovat vastuussa attestoinnista ja laitteiden luotettavuuden varmistamisesta. Autentikointitasoon kuuluu ohjelma attestor.go sekä attestointitietokanta, jotka ovat yhteydessä toisiinsa REST-kutsujen kautta. Autentikointitaso kommunikoi manageritason kanssa.

Sovellustaso koostuu ohjelmista, jotka hyödyntävät systeemin laitteiden ja sensorien keräämää dataa. Sovellustaso kommunikoi laite- ja manageritasojen kanssa. Sovellustasoon kuuluvat ohjelmat databaseManager.go ja databaseSaver.go sekä tietokanta datan tallennukseen.

UI-taso eli käyttöliittymätaso koostuu hallinnallisesta ohjelmistosta, josta voi seurata systeemin toimintaa reaaliaikaisesti. UI-tasoon kuuluu selaimessa toimiva hallintaohjelmisto, joka kuuntelee liikennettä systeemin MQTT-verkossa ja ilmoittaa siitä käyttäjälle visuaalisesti.

3.3 Systeemin kommunikointi

Järjestelmän osat kommunikoivat keskenään MQTT:n avulla. Toimintaa ohjaavat viestit kulkevat kanavalla nimeltä ”management”, ja sensorin lähettämä data kulkee sen omalla kanavallaan. Taulukosta 1 näkyy, miten ohjelmat käyttävät MQTT-kanavia.

Taulukko 1. Järjestelmän osien käyttämät kommunikaatiokanavat.

Ohjelmatiedosto	MQTT-kanava kuuntelu	MQTT-kanava lähetys	REST
device.go		management	ei
sensor.go		management, datakanava	ei
deviceManager.go	management	management	ei
attestor.go	management	management	kyllä
databaseManager.go	management	management	ei

databaseSaver.go	management, datakanavat		kyllä

Kuten taulukosta 1 näkyy, kaikki järjestelmän ohjelmat käyttävät ”management”-MQTT-kanavaa. Sensor.go-ohjelma ja dataSaver.go-ohjelma käyttävät erillistä datakanavaa. Attestor.go-ohjelma käyttää REST-kutsua hakiessaan autentikointitietoja tietokannasta, ja databaseSaver.go-ohjelma käyttää REST-kutsua tallentaessaan data-arvoja tietokantaan.

Management-kanavan viestit seuraavat standardimuotoa, joka on näytetty esimerkkikoodissa 1.

```
type ManagementMessage struct {
    DeviceName       string    `json: "name"`
    Itemid           string    `json: "itemid"`
    Message          string    `json: "message"`
    Event            string    `json: "event"`
    Time             time.Time `json: "time"`
    Jwt              string    `json: "jwt"`
    SensorName       string    `json: "sensorName"`
    SensorHostDevice string    `json: "sensorHostDevice"`
    SensorChannel    string    `json: "sensorChannel"`
    Misc             string    `json: "misc"`
}
```

Esimerkkikoodi 1. Management-kanavan viestin sisältö

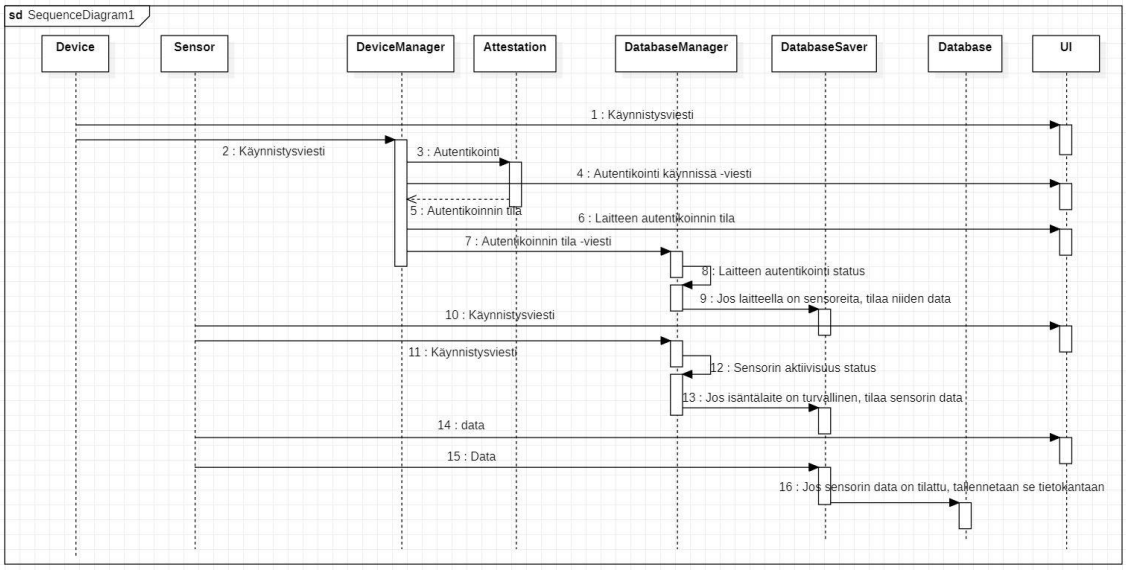
”DeviceName” viittaa fyysisen laitteen nimeen. ”Itemid” viittaa laitteen tunnistuskoodiin, jota käytetään laitteen identifioimiseen. ”Message” on selkokielinen viesti siitä, mistä tapahtumasta viestissä on kyse ja on tarkoitettu ihmisen luettavaksi. ”Event” on viestiin liittyvän tapahtuman tunniste, jonka perusteella systeemi voi tehdä toimintoja: mahdollisia ”Event”-kentän tapahtumia ovat device-startup, sensor-startup, attest, attest-ok, attest-fail, validation-ok, validation-fail, save. “Time” viittaa viestin lähetyshetkeen. ”Jwt” viittaa mahdolliseen JWT:iin, joka kulkee viestin mukana. ”SensorName” viittaa sensorin nimeen. ”SensorHostDevice” viittaa sensorin isäntänä toimivan laitteen nimeen. ”SensorChannel” viittaa siihen MQTT-kanavaan, jolla sensori lähettää mittausdataa. ”Misc” on yleisen viestinnän lokero, informaatiolle, joka ei sovellu muihin aiheisiin.

Aikaa lukuun ottamatta kaikki management-viestin osa-alueet ovat tekstimuodossa, ja jos viestiin ei ole tarpeellista liittää osa-alueen tietoa, voi kentän aina jättää tyhjäksi. Käytännössä pakollisia kenttiä, jotka pitäisi aina olla käytössä, ovat ”Event”, ”Time” ja ”Message”. Muut täytetään tilanteen ja tarpeen mukaan.

Viestit kulkevat MQTT-kanavalla JSON-muodossa. Ne puretaan esimerkkikoodissa 1 näkyvään go-kielen struct-muotoon käytännön ohjelmoinnin helpottamiseksi.

3.4 Systeemin toiminta

Tässä osiossa selvennetään, miten systeemi käytännössä toimii. Kaikki tarvittavat ja oleelliset vaiheet on käyty läpi. Ennen kuin laite käynnistetään, on hyvä varmistaa, että kaikki manageritason, autentikointitason, sovellustason ja UI-tason ohjelmat ovat käynnissä ja kuuntelevat ”management”-MQTT-kanavaa samalla MQTT-palvelimella.



Kuva 3. Sekvenssikaavio systeemin toiminnasta.

Kuvassa 3 on esitetty systeemin toiminta ja siitä näkyy, miten eri osat kommunikoivat MQTT:n välityksellä toistensa kanssa. Kuvassa Device ja Sensor ovat laitetason ohjelmia, jotka toimivat omalla laitteellaan. DeviceManager on

manageritason ohjelma, joka toimii omalla laitteellaan. Attestation kuvaa autentikointitason ohjelmistoa. DatabaseManager, DatabaseSaver ja Database ovat sovellustason ohjelmia. UI kuvaa UI-tason ohjelmistoa.

Seuraavaksi käydään läpi tarkemmin koko toimintaprosessi ja selvennetään kunkin osan toimintaa.

3.4.1 Avaimen luonti

Ennen kuin systeemi voidaan ottaa käyttöön, on laitteelta saatava avain, jota voi käyttää tunnistuksessa. Jokaisella valmistetulla TPM-sirulla on oma uniikki avainten luontiin tarkoitettu lähdearvonsa. Sen avulla voi luoda erilaisia avaimia, kuten EK (*Endorsement Key*) ja AK (*Attestation Key*). EK on tarkoitettu salauksen tekemiseen ja purkuun, ja AK on tarkoitettu allekirjoitustarkoituksiin. EK ja AK ovat rajattu toimimaan vain TPM:n sisällä: avainten yksityistä osaa ei voida lukea TPM:ltä. Vain julkiset osat voidaan lukea. [30.]

Avaimen luonti tapahtuu tpm2tools-ohjelmiston [31] avulla, joka täytyy olla valmiiksi asennettuna laitteelle. Avaimen luontia varten tehtiin skripti, jossa vaiheet on automatisoitu.

```
echo "creating ek key"
tpm2_createek -c 0x81010002 -G rsa -u ek.pub
tpm2_readpublic -c 0x81010002 -o ek.pem -f pem

echo "creating ak key"
tpm2_createak -C 0x81010002 -c ak.ctx -G rsa -g sha256 -s rsassa -u
ak.pub -f pem -n ak.name
tpm2_evictcontrol -c ak.ctx 0x81010003
tpm2_readpublic -c 0x81010003 -o ak.pem -f pem
```

Esimerkkikoodi 2. Avaimenluontiskripti.

Kuten esimerkkikoodista 2 näkyy, ensiksi luodaan EK, ja sen jälkeen luodaan AK, jonka pohjana käytetään ensin tehtyä EK:ta. AK laitetaan tiettyyn TPM2-sirun muistipaikkaan (0x8101003), jotta laitteen ohjelmisto pääsee myöhemmin käyttämään sitä, ja sen julkisesta osasta tehdään luettava ak.pem-tiedosto.

Ak.pem-tiedosto on mahdollista avata ja siitä voi nähdä, mikä on avaimen julkinen arvo. Tämä avaimen julkinen arvo tallennetaan sitten autentikointitasolle, jollain toisella laitteella sijaitsevaan attestointi-tietokantaan, mistä se voidaan hakea autentikointiprosessin aikana.

3.4.2 Autentikointiprosessi

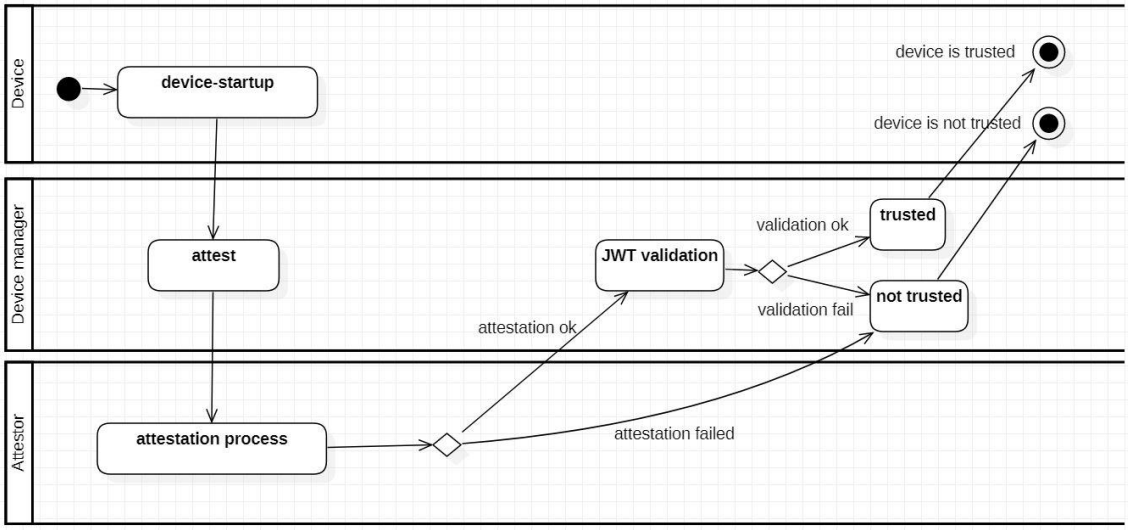
Kun laite käynnistetään ja avain on luotu ja sijoitettu määritettyyn TPM:n muistipaikkaan, device.go-ohjelma hakee muistipaikasta julkisen avaimen ja käyttää sitä JWT:n allekirjoittamiseen. Sitten ohjelma lähettää tervehdysviestin management-kanavalle, jolla se ilmoittaa olevansa käynnissä ja kytkettynä systeemiin. Tervehdysviesti sisältää allekirjoitetun JWT:n ja viestin ”Event”-kentän arvo on ”device-startup”.

DeviceManager.go-ohjelma vastaanottaa viestin ja nähdessään ”Event”-kentän tapahtuman olevan ”device-startup”, se käynnistää attestointi-prosessin lähettämällä viestin management-kanavalla, jossa ”Event”-kentän tapahtuma on ”attest”.

Kun attestor.go-ohjelma vastaanottaa management-kanavalla vietin, jonka ”Event”-kentän tapahtuma on ”attest”, se etsii viestin mukana tulevan laitteen ”Itemid”-kentän tunnisteen perusteella attestointi-tietokannasta aikaisemmin luodun julkisen avaimen. Mikäli tunnisteella löytyy tietokannasta avain, ohjelma lähettää avaimen management-viestin mukana ”Misc”-kentässä, jolloin ”Event”-kentän arvo on ”attest-ok”. Mikäli tietokannasta ei löydy laitteen tunnisteella avainta, lähettää ohjelma viestin management-kanavalla, jossa ”Event”-kentän arvo on ”attest-fail”.

Mikäli attestointi onnistuu ja deviceManager.go-ohjelma vastaanottaa management-kanavalla viestin, jonka ”Event”-kentän arvo on ”attest-ok”, yrittää deviceManager.go-ohjelma avata JWT:n viestin mukana saamallaan avaimella. Mikäli avain on oikea ja JWT:n avaus onnistuu, on koko validointi prosessi onnistunut ja ohjelma ilmoittaa siitä management-kanavalla viestillä, jossa ”Event”-kentän

arvo on ”validation-ok”. Mikäli JWT ei avaudu avaimen avulla, ilmoittaa ohjelma siitä management-kanavalla viestillä, jossa ”Event”-kentän arvo on ”validation-fail”.



Kuva 4. Aktiviteettidiagrammi laitteen autentikointiprosessista

Kuvassa 4 on esitetty todennusprosessin kaksivaiheisuus. Ensin varmistetaan, että laite löytyy attestaatiotietokannasta, ja sen jälkeen vielä varmistetaan, että laitteen TPM-sirulta suoraan saatu julkinen avain vastaa attestaatiotietokannan avainta. Näin saadaan varmistettua, että kukaan ei ole muuttanut laitteen asetuksia tai kokoonpanoa niistä arvoista, jotka sillä pitäisi olla.

3.4.3 Datan tallennus

Sovellustason databaseManager.go-ohjelma pitää kirjaa järjestelmän todennetuista ja turvallisiksi määritetyistä laitteista sekä järjestelmässä toimivista sensoreista. Käytännössä se siis ylläpitää kahta listaa, toisessa on turvallisten laitteiden nimitieto ja toisessa on sensoreiden tiedot: sensorin nimi, sensorin isäntälaite ja sensorin datakanavan nimi.

Kun laitteen kaksivaiheinen todennus on suoritettu, databaseManager.go-ohjelma vastaanottaa management-kanavalla viestin, jossa ”Event”-kentän arvo

on ”validation-ok”. Tämä kertoo ohjelmalle, että viestin mukana tulevan laitteen tiedot, käytännössä sen nimi, voidaan tallentaa todennettujen laitteiden listaan.

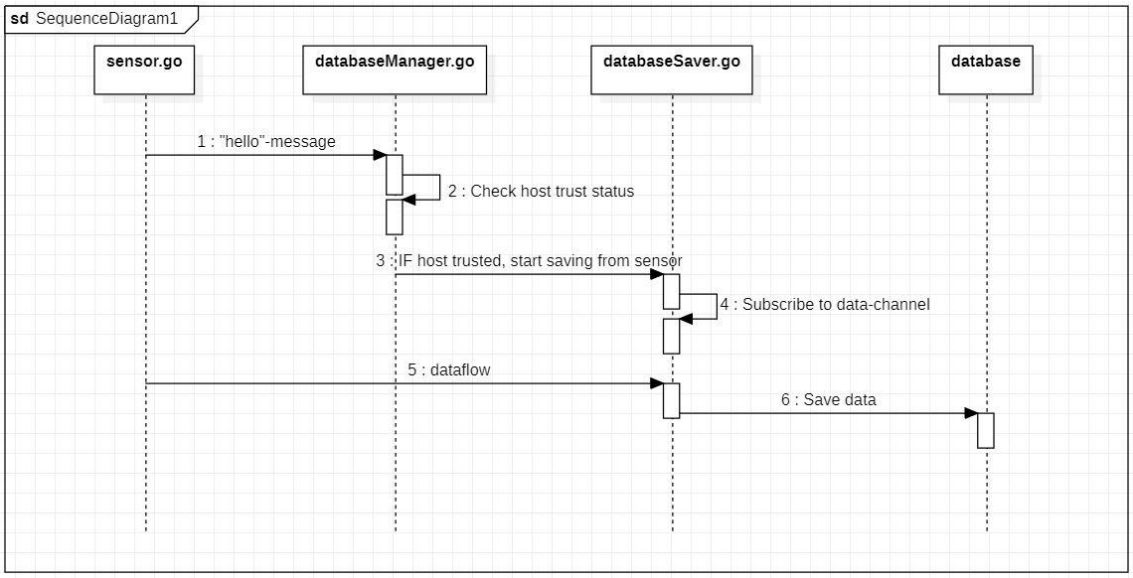
Kun systeemissä käynnistyy uusi sensori, sensor.go-ohjelma lähettää management-kanavalla viestin, jossa ”Event”-kentän arvo on ”sensor-startup”. Kun databaseManager.go-ohjelma vastaanottaa viestin, se lisää sensorin tiedot sensoreiden listaan.

Aina kun databaseManager.go-ohjelma vastaanottaa viestin, jossa ”Event”-kentän arvo on ”validation-ok”, se vertaa uutta laitenimeä sensorilistan sensoreiden isäntälaitteiden nimiin. Vastaavasti jos ”Event”-kentän arvo on ”sensor-startup”, ohjelma vertaa sensorin isäntälaitteen nimeä todennettujen laitteiden listassa oleviin nimiin.

Mikäli saman laitteen nimi löytyy todennettujen laitteiden listasta ja on sensorilistassa jonkin sensorin isäntälaitteena, ohjelma lähettää viestin management-kanavalla, jossa ”Event”-kentän arvo on ”save” ja ”SensorChannel”-kentän arvo on sensorin data-arvoja lähettävän MQTT-kanavan tunniste.

DatabaseSaver.go-ohjelma vuorostaan tilaa myös käynnistyessään management-kanavan. Kun se vastaanottaa viestin, jossa ”Event”-kentän arvo on ”save”, se tilaa sensorin data-arvoja lähettävän kanavan, jonka nimi tulee myös viestin mukana. Kun databaseSaver.go on tilannut datakanavan, aina kun se vastaanottaa viestin siltä kanavalta, se tallentaa arvon datatietokantaan.

Tiedot tallentuvat tietokantaan kanavan nimen perusteella. Tallennettu tietue sisältää aina datakanavan nimen, data-arvon sekä tallennusajan.



Kuva 5. Sekvenssidiagrammi sensorin datan tallennuksesta.

Kuvassa 5 on kuvattu sensorin datan tallennuksen toiminta. Mikäli sensorin isäntälaite on todettu turvalliseksi, sensorin datakanavan tietoja aletaan tallentaa. Ylempänä kuvasta 4 voi nähdä, miten laitteen luotettavuuden varmistus tapahtuu.

Go sallii kätevän moniajosysteemin, joka mahdollistaa datakanavien dynaamisen tilaamisen. Jokainen systeemiin tuleva uusi sensori, jonka dataa tallennetaan aloittaa käytännössä uuden prosessin, joka kuuntelee sensorin MQTT-kanavaa.

```
// Channel for receiving messages
var messageChannel = make(chan string)

// Start goroutine to handle messages from the channel
go func() {
    for {
        select {
        case msg := <-messageChannel:
            topic := msg
            if topic != "" {
                subscribeToTopic(client, topic)
            }
        }
    }
```

```
}()
// Subscribe to initial topic
subscribeToTopic(client, "management")

// Wait for new messages to arrive
select {}
```

Esimerkkikoodi 3. Useiden kanavien tilaaminen dynaamisesti

Prosessi on toteutettu Gon viestikanavatoiminnon avulla, minkä toiminta on kuvattu esimerkkikoodissa 3. Ohjelma tilaa käynnistyessään ”management”-kanavan ja aina kun se saa oikeanlaisen viestin tehdään uusi MQTT-kanava tilaus.

4 Testaus ja tulokset

Tässä osiossa perehdytään siihen, miltä systeemin toiminta näytti testausvaiheessa ja minkälaisia tuloksia testauksesta saatiin.

Tässä projektissa on pitkälti kyse luotettavuudesta. Testauksen tarkoituksena oli varmistaa, että systeemi toimii oikein ja tekee sen, mitä sen on tarkoituskin tehdä. Tämän varmistamiseksi kokeiltiin erilaisia mahdollisia testitapauksia ja katsottiin, miten systeemi suoriutuu niistä.

Systeemin kehitys ja testaus tapahtui virtuaalisesti. Käytetty virtuaalikone toteutettiin Oracle VM VirtualBox -ympäristössä toimivana Ubuntu-käyttöjärjestelmää käyttävänä koneena. Virtuaalikonetta päädyttiin käyttämään, koska se piti prosessin yksinkertaisena. Testausvaiheessa olisi ollut mahdollista käyttää esimerkiksi docker-pohjaista ratkaisua, mutta koska virtuaalisella TPM:llä avainten luonti tapahtuu aina laitteen käynnistyessä ja niiden siirtäminen attestointitietokantaan on tehtävä manuaalisesti, prosessia ei ole mahdollista automatisoida täysin. Tämän takia virtuaalikoneen käyttö oli yksinkertaisempi ja helpompi ratkaisu.

Testivaiheessa laite-, manageri-, applikaatio ja UI-tasojen ohjelmat pyörivät virtuaalikoneella ja autentikointi-tason ohjelmat, eli attestointitietokanta ja

attestor.go-ohjelma, pyörivät eri ympäristössä kannettavalla tietokoneella. Näin oli mahdollista simuloida kahden eri laitteen yhteistoimintaa.

Sensoriohjelman, sensor.go, datalähteenä käytettiin yksinkertaista satunnaislukugeneraattoria.

MQTT-palvelimena käytettiin Mosquitto-projektin testitarkoituksiin soveltuvaa palvelinta osoitteessa test.mosquitto.org.

4.1 Testauksen valmistelu

Jotta sovelluskehys toimisi, on aluksi tehtävä muutama toiminnalle tärkeä alustava toimenpide. Ensimmäinen vaihe testauksen valmistelussa oli virtuaalisen TPM-sirun aktivointi. Prosessin helpottamiseksi tehtiin sitä varten shell-skripti, joka sisälsi kaikki tarvittava komennot ja joka on helppo ajaa aina virtuaalikoneen käynnistyessä.

Seuraava vaihe oli avaimien luonti TPM-sirulle ja julkisen avaimen lisääminen attestointitietokantaan. Koska attestointitietokanta oli toisessa tietokoneessa, avaimen arvo siirrettiin sinne manuaalisesti.

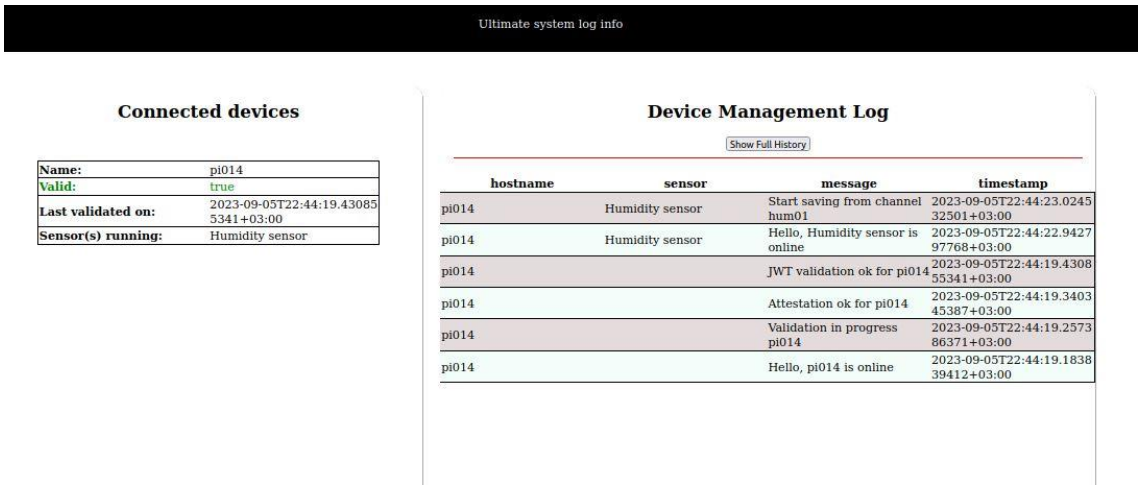
Näiden toimenpiteiden jälkeen on aika käynnistää kaikki manageri-, autentikointi-, applikaatio- ja UI-tasojen ohjelmat. Tämän jälkeen on mahdollista aloittaa varsinainen testaus siitä, miten systeemi suoriutuu erilaista tilanteista laitteiden validoinnin suhteen.

Testitilanteet toimivat käytännössä niin, että laitetason ohjelmat käynnistetään ja katsotaan, mitä tapahtuu missäkin tapauksessa, erilaisilla asetuksilla. Kunkin testiajon jälkeen toiminta arvioitiin käyttöliittymänäkymän antamien tietojen ja datatietokantaan tallennettujen tietojen pohjalta.

4.2 Yksi laite ja yksi sensori

Ensimmäinen testisvaihe oli sellainen, jossa systeemiin kuuluu yksi laite ja yksi sensori. Testitilanteita oli tässä vaiheessa kolme: 1) laitteen tunniste löytyy attestointitietokannasta ja tietokannassa oleva avain vastaa TPM:n AK:ta, 2) laitteen tunniste löytyy attestointitietokannasta mutta tietokannan avain ei vastaa TPM:n AK:ta, ja 3) laitteen tunnistetta ei löydy attestointitietokannasta lainkaan.

Ensimmäisessä tilanteessa kaiken pitäisi mennä hyvin, ja systeemi voi pitää laitetta luotettavana. Niin testitilanteessa myös tapahtui.



Kuva 6. Onnistunut attestointi ja validointi.

Kuvassa 6 on käyttöliittymän näkymä, kun kaikki on mennyt hyvin. Näkymän oikealla puolella näkyy kaikki systeemin tapahtumat, joista ilmoitetaan ”management”-kanavalla. Ensin laite ilmoittaa olevansa päällä. Sitten validointiprosessi käynnistyy. Sitten attestor.go-ohjelma ilmoittaa attestoinnin onnistuneen, minkä jälkeen myös deviceManager.go-ohjelma ilmoittaa JWT:n validoinnin onnistuneen. Validoinnin onnistuttua kokonaisuudessaan, kun samalla laitteella oleva sensori tulee systeemiin, datan tallennus voidaan aloittaa. Tästäkin näkymässä näkyy ilmoitukset.

Näkymän vasemmassa reunassa on lista systeemissä olevista laitteista ja niillä olevista sensoreista. Mikäli laite on luotettava, ”Valid”-kentässä näkyy arvo

”true” ja sen väri on vihreä. Validointiaika näkyy myös näkymässä ”Valid”-kentän alapuolella.

Toisessa tilanteessa validoinnin ei pitäisi onnistua, koska avaimet eivät ole samat. Näin testitilanteessa kävikin.

Kuva 7. JWT:n validoinnin epäonnistuminen.

Kuvassa 7 näkyy, mitä tapahtuu, kun JWT:n validoiminen ei onnistu, eli attestointitietokannasta saatu julkinen avain ei vastaa sitä, mikä on saatu laitteen TPM-sirulta. DeviceManager.go-ohjelma ilmoittaa JWT:n validoinnin epäonnistuneen. Laitteen ”Valid”-kentän arvoksi tulee ”false”, joka näkyy punaisena. Validointiajan kentässä on arvo ”null”, joka kertoo siitä, että laitetta ei ole koskaan validoitu. Käyttöliittymän näkymä ilmoittaa myös, että sensori on olemassa, mutta viestiä tallennuksen aloittamisesta ei tule, koska tallennusta ei aloiteta laitteelta, johon ei luoteta.

Kolmannessa tilanteessa validoinnin pitäisi myös epäonnistua, koska laitteen tunnistetta ei löydy attestointitietokannasta. Näin testitilanteessa myös kävi.

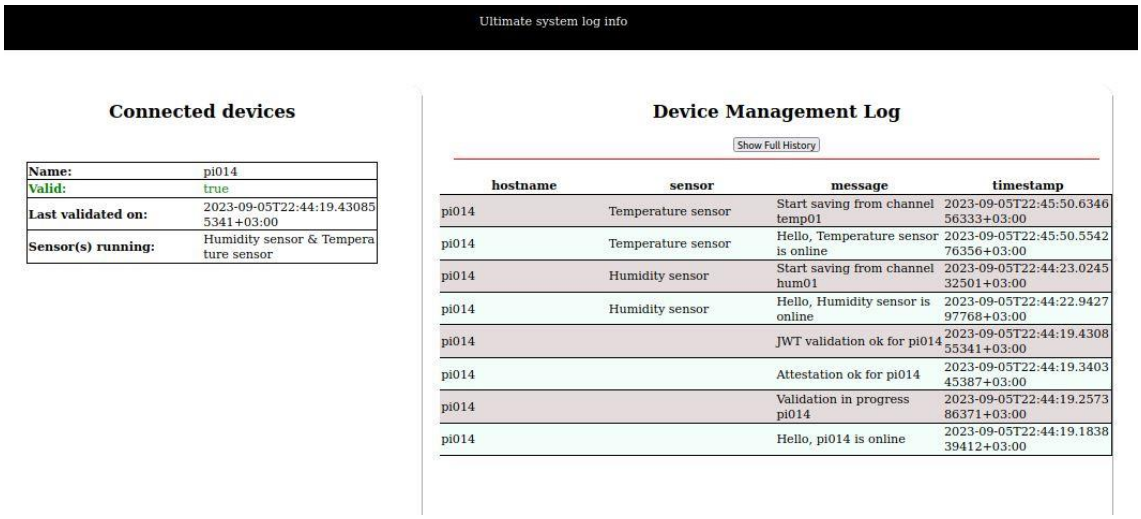
Device.go-ohjelma lähettää käynnistysviestinsä, minkä jälkeen deviceManager.go-ohjelma ilmoittaa validointiprosessin aloittamisesta. Kun attestointi epäonnistuu, eli attestointitietokannasta ei löydy tietuetta laitteen tunnisteella, lähettää attestor.go-ohjelma viestin attestoinnin epäonnistumisesta. Laitteen

”Valid”-arvoksi tulee punainen ”false”, eikä sensorin lähettämää dataa aleta tallentaa.

4.3 Yksi laite ja useita sensoreita

Toisessa testivaiheessa testattiin, mitä tapahtuu, jos yhdellä laitteella on useampi sensori. Tähän vaiheeseen kuului kaksi testitilannetta: 1) laitteen validointi onnistuu ja laitteella on kaksi sensoria, 2) laitteen validointi ei onnistu ja laitteella on kaksi sensoria. Testit suoritettiin siten, että käynnistettiin laiteohjelma ja sitten kaksi sensoriohjelmaa, joista toinen kuvasi lämpötilasensoria ja toinen kosteussensoria, mutta joilla oli samanniminen isäntälaite.

Ensimmäisessä testitilanteessa kaiken pitäisi mennä hyvin. Laitteen validoinnin onnistuttua systeemin pitäisi alkaa tallentaa dataa molemmista sillä olevista sensoreista. Näin testitilanteessa myös kävi.



Kuva 8. Kaksi sensoria samalla luotetulla laitteella.

Kuvassa 8 näkyy mitä tapahtuu, kun luotetulla laitteella on kaksi aktiivista sensoria. Käyttöliittymän näkymässä voi nähdä oikealla laitteen käynnistymisen ja validointiprosessin viestit ja vasemmalla validoinnin onnistumista kuvaavan vihreän ”true”-arvon Valid-kentässä. Koska laite on luotettava, molemmilta sensoreilta aletaan tallentaa data-arvoja. Näkymässä voi nähdä, miten sekä kanavalta

”temp01” että kanavalta ”hum01” aletaan tallentaa dataa. Näkymän vasemmassa reunassa voi nähdä, miten sensorit listautuvat luotetun laitteen alle.

Toisessa testitilanteessa kummankaan sensoreiden dataa ei pitäisi alkaa tallentaa, koska niiden isäntälaitteen luotettavuudesta ei ole takeita. Näin testitilanteessa myös kävi.

DeviceManager.go-ohjelma ilmoitti JWT:n validoinnin epäonnistuneen ja laitteen Valid-kentän arvoksi tuli punainen ”false”. Koska laite ei ollut validoitu, kummankaan sensorin dataa ei alettu tallentamaan. Käyttöliittymän näkymässä näkyi ilmoitus sensorien käynnistymisestä, mutta ei tallennuksen aloittamisesta, koska tallennusta ei koskaan aloitettu.

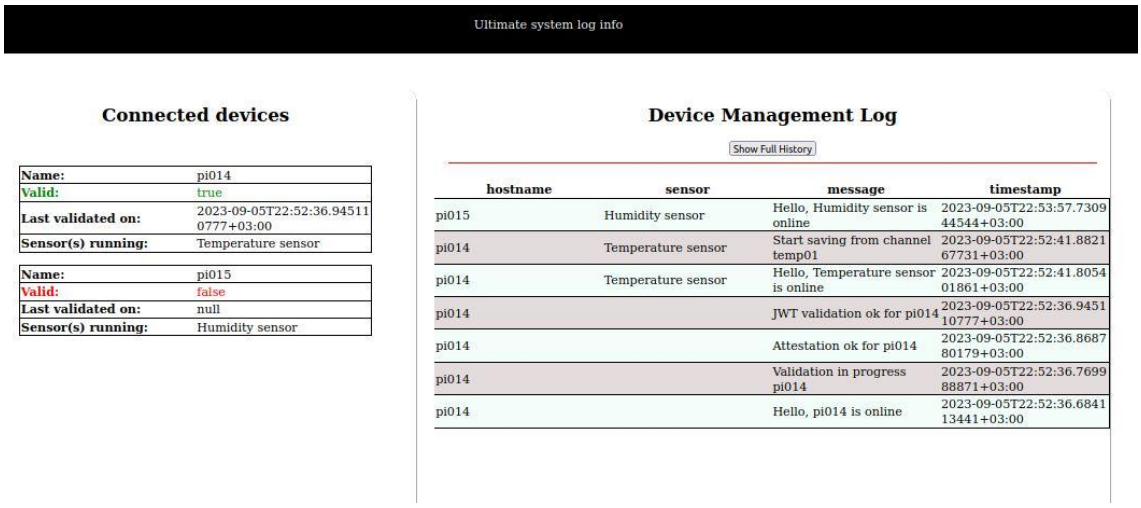
4.4 Useampi laite

Kolmannessa testivaiheessa testattiin, mitä tapahtuu, jos käytössä on useampi laite, joilla kullakin on sensori. Käytännössä testitilanteissa käytettiin kahta laiteohjelmaa, joilla kummallakin oli yksi sensori. Tähän vaiheeseen kuului kolme testitilannetta: 1) molempien laitteiden validointi onnistui, 2) toisen laitteen validointi onnistui, toisen ei, 3) molempien laitteiden validointi epäonnistui.

Ensimmäisessä tilanteessa kaiken pitäisi mennä hyvin ja molemmilla laitteilla olevilta sensoreilta pitäisi alkaa tallentamaan dataa. Näin testitilanteessa myös kävi.

Käyttöliittymästä voitiin nähdä laitteiden käynnistysviestit ja validointiprosessin onnistunut kulku. Näkymästä nähtiin myös molempien sensorien käynnistymisviestit ja viestit siitä, että niiden dataa alettiin tallentamaan. Näkymän vasemmasta reunasta saattoi nähdä myös listauksen kahdesta validista laitteesta systeemissä, molemmilla ”Valid”- kentän arvona vihreä ”true”.

Toisessa tilanteessa pitäisi toisella laitteella olevan sensorin dataa alkaa tallentaa ja toisella olevan dataa ei. Näin testitilanteessa myös kävi.



Kuva 9. Sensori luotettavalla ja ei-luotettavalla laitteella.

Kuvassa 9 voi nähdä, miten systeemi toimii, kun toisen sensorin isäntälaite on luotettava ja toisen ei. Näkymän oikealla puolella tapahtumalistassa näkyy, että luotettavalla laitteella olevan lämpötilasensorin dataa aletaan tallentamaan, mutta ei luotetulla laitteella olevan kosteussensorin dataa ei aleta tallentamaan.

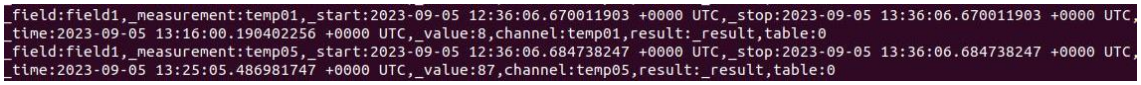
Näkymän vasemmassa reunassa listataan molemmat laitteet ja niiden ”Valid”-kentän arvot vastaavat haluttuja, toinen on vihreä ”true” ja toinen punainen ”false”. Jos toinenkin laite myöhemmin autentikoidaan ja arvioidaan luotettavaksi, aletaan myös toisen sensorin dataa tallentamaan tietokantaan.

Kolmannessa tilanteessa kummallakaan laitteella olevien sensorien ei pitäisi alkaa tallentaa tietoa, koska niiden validointi ei onnistunut. Näin testitilanteessa myös kävi.

Käyttöliittymästä voitiin nähdä, että laitteet käynnistyivät, ja kun niiden validoinnit epäonnistuivat sensoreilta ei alettu tallentamaan dataa tietokantaan. Molemmat laitteet ilmestyivät näkymän vasemman reunan palstalle ja niiden ”Valid”-kentän arvot olivat molemmilla punainen ”false”.

4.5 Data tietokannassa

Jokaisen testiajon jälkeen tarkastettiin datatietokannasta, oliko data tallentunut oikein. Sensorien tallennettua dataa on mahdollista etsiä tietokannasta datakanavan nimellä.



Kuva 10. Data-arvoja influx-tietokannasta haettuna.

Kuvassa 10 näkyy, minkälaista dataa tietokantaan tallentuu, ja minkälaista tietoa siitä saa ulos. Yhden tietueen tiedoissa on aikaleima tallennushetkestä sekä kentät ”value” ja ”channel”. Channel-kentän arvoksi tulee sensorin datakanavan nimi, ja value-kentän arvoksi tulee sensorin lähettämä data-arvo kokonaislukuna.

5 Pohdintaa tuloksista

Tämän opinnäytetyön tarkoituksena oli tuottaa proof-of-concept-malli sovelluskehyksestä, joten testauksen laajuus oli rajoitettu vain pääasiallisen toiminnallisuuden testaamiseen. Testitapausten tarkoituksena oli osoittaa, että systeemi toimii niin kuin sen pitikin, eikä toiminnallisia puutteita ole havaittavissa. Suurelta osin voidaankin sanoa, että systeemi toimi juuri niin kuin sen pitikin ja toiminnallisuus vastasi haluttua.

Kaksivaiheinen autentikointi tuo lisäarvoa sovelluskehykseen parantamalla laitteiden ja koko systeemin luotettavuutta. TPM:n toiminnallisuuden yhdistäminen JWT:n kanssa on uusi ja toimiva tapa hyödyntää TPM:n turvallisuusominaisuuksia. Edellisessä innovaatioprojektin yhteydessä tehdyssä versiossa [33] TPM:n toiminnallisuus ei ollut keskeisessä osassa. Innovaatioprojektin versiossa tarkoitus oli vain sisällyttää TPM-siru osaksi systeemiä. Tässä uudessa versiossa TPM:n turvallisuusominaisuuksia käytetään oikeasti hyödyksi, ja se on integroitu syvällisemmin ohjelmiston toiminnallisuuteen.

Toinen selkeä parannus innovaatioprojektin versioon on useiden sensorien dynaaminen lisääminen, mikä tuo uusia mahdollisuuksia systeemien kasvattamiseen ja dynaamiseen skaalautuvuuteen. Tässä Go-kielen erityisominaisuudet osoittautuivat erityisen hyväksi. Gossa on sisäänrakennettu kanavapohjainen rinnakkaisajo-ominaisuus, joka mahdollistaa rinnakkaisprosessien tehokkaan toiminnan. Python-kielellä rinnakkaisprosessien toiminta tehdään säikeitä käyttäen, joka vaatii enemmän laskentatehoa kuin Gon sisäänrakennettu rinnakkaisuus [23]. Systeemin testivaiheessa useiden sensorien rinnakkaisuutta testattiin vaatimattomasti kahdella sensorilla, joten on vaikea arvioida, miten toiminta suurten satoja tai tuhansia sensoreita käyttävien systeemien kanssa sujuisi. Periaate on kuitenkin toimiva.

Luotettavuudenhallinnan (*Trust Management*) kannalta laitteiden varma autentikointi on erittäin hyvä asia, ja ohjelmistokehyksen käyttöliittymä on hyvä ja toimiva ratkaisu systeemin tilanteen tarkkailuun. Selaimessa toimivasta sovelluksesta näkee reaaliajassa systeemissä olevat laitteet ja niiden validointistatukset. Käyttöliittymä myös pitää lokia kaikista tapahtumista seuraamalla ”management”-kanavaa, ja lokitietoja on mahdollista selata pitkältäkin aikaväliltä etsintätyökalun avulla.

Tämä sovelluskehys on myös hyvin muovautuva ja vastaa myös haasteeseen erilaisten laitteiden ja ohjelmistojen yhteen tuomisesta. Go on yhteensopiva monien alustojen kanssa [34], ja hallinnallinen toiminta tapahtuu MQTT-protokollaa käyttäen, jonka toiminnasta go vastaa. Myös TPM on yhteensopiva monien järjestelmien kanssa. TPM käyttää tietoja laitteen tilasta luodakseen turvallisen avaimen. Riippumatta laitteen käyttämistä järjestelmistä ja ohjelmistoista TPM huomaa laitteen toimintaan vaikuttavat muutokset ja laitteen tilan muutokset.

Sovelluskehys parantaa lähinnä havaintotason (*perception laye*r) laitteiden ja välillisesti ohjelmistotason (*application layer*) luotettavuutta. Mitä tulee CIA-kolmion, *luottamuksellisuuden*, *eheyden* ja *saatavuuden* suhteen, laiteiden kaksivaiheinen todennusprotokolla on erinomainen peruspilari, jonka päälle voi

rakentaa turvallisen systeemin. Koska vain luotettuja laitteita käytetään, kerätty data on varsin luotettavaa.

Systeemin turvallisuuden suhteen voisi tehdä paljon lisääkin. Tämän projektin tarkoitus oli vain demonstroida TPM:n turvallisuusominaisuuksien yhdistämistä JWT:n kanssa ja niiden käyttömahdollisuutta autentikoinnissa. Tämä systeemi ei ole mitenkään lopullisen varma.

Jos systeemistä haluttaisiin tehdä käytännössä toimiva versio, lisättäviä turvallisuuskomponentteja voisi olla esimerkiksi jonkinlaisten pääsynhallinta- ja viestien salausmetodien käyttö. Yksi mahdollinen tapa toteuttaa viestinnän salaus olisi käyttää MQTT:n mahdollistamia tapoja, kuten SSL-sertifikaatteja kullakin laitteella ja viestien erillistä salausta [32].

Luotettavuutta voisi myös lisätä tekemällä autentikointia käytön aikana. Nykymuodossa sovelluskehys tekee laitteiden autentikoinnin aina niiden käynnistyksen yhteydessä. Mikäli laitteen toimintaa tai asetuksia muutetaan käytön aikana, ei attestointiprosessi sitä huomaa. Voisi olla hyödyllistä tehdä autentikointia tietyin väliajoin laitteen ollessa käynnissä, esimerkiksi tunnin välein riippuen systeemin käyttötarkoituksesta ja resursseista.

Yksi tapa miten Go-toteutus on innovaatioprojektin Python-toteutusta heikompi, on se, että Pythonin perintäominaisuudet sallivat tehdä sensoritiedostosta erittäin helppokäyttöisen kehittäjän näkökulmasta. Muutokset, jotka kehittäjän tarvitsi tehdä uuden sensorin tullessa mukaan, olivat hyvin vähäisiä. Golla toteutettuna sensoritiedosto on isompi ja ”rumempi”, mutta siinäkin varsinaiset muutokset, jotka jäävät kehittäjän tehtäväksi uuden sensorin yhteydessä, ovat suhteellisen pienet. Myös Golla olisi kuitenkin mahdollista abstrahoida joitain toimintoja näkymättömiin, mikä voisi olla hyödyllistä kehyksen käytön helppouden kannalta.

Toinen asia sensoreihin liittyen, mikä vaikutti osaltaan myös projektin testausvaiheeseen, on se, että jos testausta olisi halunnut tehdä Rapberry Pi -tietokoneilla niin kuin Python-toteutuksen kanssa tehtiin, olisi se vaatinut lisää työtä.

Koska Python on yleisemmin käytetty ohjelmointikieli, perussensoreille löytyy yleensä valmiit kirjastot niiden käyttöön. Golla tällaisia kirjastoja ei ole, joten jos sensoreita haluaa käyttää, on käytännössä ohjelmoitava niille ajurit itse.

Kuten jo sanottu, tarkoituksena oli tehdä toimiva proof-of-concept-malli. Lisätestaus olisi kuitenkin mielenkiintoista. Varsinkin olisi hauska nähdä, miten systeemi suoriutuisi suuresta määrästä sensoreita.

Sovelluskehyksen sekä suurin vahvuus että heikkous lienee lopulta TPM:n käyttö. Se on systeemin luotettavuuden ja turvallisuuden perusta, mutta se on myös asia, jolle on varattava resursseja, niin prosessointiresursseja kuin ajallisiakin. Jokainen käytettävä laite on oltava sellainen, johon voi laittaa TPM:n ja joka kykenee käyttämään sitä. Jokaiselle laitteelle pitää myös tehdä alustavat toimet: sen tiedot luotuine avaimineen on lisättävä attestointitietokantaan. Lopulta se on kuitenkin pieni vaiva verrattuna siihen, miten paljon systeemin luotettavuus kasvaa TPM:aa käyttämällä.

Taulukossa 2 on vielä tiivistelmä sovelluskehyksen ominaisuuksista ja siitä, miten uusi Go-toteutus eroaa aiemmasta innovaatioprojektin Python-toteutuksesta.

Taulukko 2. Kahden toteutuksen ominaisuuksia ja eroja.

	Go-toteutus (2023)	Python-toteutus (2022)
Ohjelmointikieli	Go	Python
Kommunikaatioprotokolla	MQTT	MQTT
TPM-integraatio	Kyllä	Kyllä
TPM:n käytetyt ominaisuudet	Itemid-tunnistus, avaimen luonti	Itemid-tunnistus
Etäattestointi	Kyllä	Kyllä
Autentikointi	Kaksivaiheinen: attestointitietokanta,	Yksivaiheinen: attestointitietokanta

	JWT-varmennus	
Useiden laitteiden dynaaminen yhdistäminen systeemiin	Kyllä	Ei

Mahdollisia käyttötapauksia sovelluskehyksellä on paljon. Koska kyseessä on kehys, jonka tarkoituksena on esittää laitteiden kaksivaiheinen autentikointimetodi, se on mallina abstrakti ja taipuu moneen suuntaan. Potentiaalisia käyttötapauksia voi löytyä esimerkiksi teollisuuden, lääketieteen ja liikenteen prosesseista. Käytännössä mikä tahansa erillisistä laitteista koostuva IoT-systeemi voidaan rakentaa tätä sovelluskehystä pohjana käyttäen.

6 Yhteenveto

Tämän opinnäytetyön tarkoituksena oli tarkastella mahdollista tapaa parantaa IoT-sovelluskehyksen luotettavuutta. Luotettavuus ja turvallisuus ovat tärkeitä asioita teknologian suhteen. IoT-systeemien tullessa yhä enemmän osaksi maailman rakennetta on se, että niihin voi luottaa yhä tärkeämpää.

Opinnäytetyöprojektin tarkoituksena oli toteuttaa proof-of-concept-malli luotettavasta ohjelmistokehyksestä, joka sallii helposti skaalautuvan IoT-sensorisysteemin nopean kehittämisen. Ohjelmistokehyksen pohjana käytetiin vuoden 2022 syksyllä innovaatioprojektissa tehtyä ohjelmistoa, jonka luotettavuus- ja turvallisuusominaisuuksia parannettiin. Tuotettu ohjelmisto käyttää TPM-turvapiirin ominaisuuksia yhdessä JWT-teknologian kanssa uudella tavalla, ja jokainen systeemiin liitetty laite käy läpi kaksivaiheisen autentikointiprosessin.

Go-ohjelmointikielen käyttö uudessa versiossa toi mukanaan sekä haasteita että uusia hyödyllisiä ominaisuuksia. Kun sovelluskehyksen toimintaa ja ominaisuuksia verrattiin vuoden 2022 projektissa tehtyyn Python-pohjaiseen toteutukseen, voidaan todeta, että tehdyt muutokset sovelluskehyksen toimintaan ja

rakenteeseen parantavat sen luotettavuutta ja turvallisuutta. Ohjelmistokehyksen sisältämä selaimessa pyörivä hallintaohjelma tarjoaa myös toimivan tavan pysyä kartalla laitteiden luotettavuuden tilasta systeemissä reaaliajassa.

Systeemin toiminnallisuutta testattiin kattavasti erilaisissa tilanteissa ja sen perusominaisuudet havaittiin toimiviksi. Käytännölliseen sovellukseen, jota voisi käyttää oikean maailman tilanteissa on vielä matkaa, mutta toteutettu malli esittää toimivan tavan parantaa IoT-systeemien luotettavuutta.

Kaiken kaikkiaan voidaan sanoa, että projektin toteutus oli onnistunut. Valmistunut ohjelmisto toimii odotetulla ja halutulla tavalla. Sovelluskehys tarjoaa myös hedelmällisen maaperän jatkokehitystä ajatellen.

Lähteet

1 Green, Lelia. 2022. Will technology save us from the climate emergency? Verkkoaineisto. Edith Cowan University. <https://www.ecu.edu.au/news-room/articles/opinion/will-technology-save-us-from-the-climate-emergency>. 22.8.2022. Luettu 8.11.2023.

2 Klein, Naomi. 2023. AI machines aren't hallucinating. But their makers are. Verkkoaineisto. The Guardian. <https://www.theguardian.com/commentisfree/2023/may/08/ai-machines-hallucinating-naomi-klein>. 8.5.2023. Luettu 8.5.2023.

3 Vainio, Matias; Peltonen, Jani; Viljanen, Teemu; Soininen, Joonas. 2022. IoT-sovelluskehys. <https://github.com/teemvil/iot>.

4 Viljanen, Teemu. 2023. IoT-sovelluskehys. <https://github.com/teemvil/gost>.

5 McKnight, D. Harrison. 2005. Trust in Information Technology. Teoksessa Davis, G.B. (toim.). The Blackwell Encyclopedia of Management. Vol. 7 Management Information Systems. Hoboken: Blackwell.

6 Samonas, Spyridon; Coss, David. 2014. The CIA Strikes Back: Redefining Confidentiality, Integrity and Availability in Security. Journal of Information Systems Security. Vol. 10, s. 21–45.

7 Information Security. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Information_security>. Luettu 31.10.2023.

8 Sha, Kewei; Wei, Wei; Yang, T. Andrew; Wang, Zhiwei; Shi, Weisong. 2018. On security challenges and open issues in Internet of Things. Future Generation Computer Systems. Vol. 83, s. 326–337.

9 Konsta, Alyzia Maria; Lafuente, Alberto Lluch; Dragoni, Nicola. 2023. Trust Management for Internet of Things: A Systematic Literature Review. Verkkoaineisto. Arxiv. <https://arxiv.org/abs/2211.01712>. 25.9.2023. Luettu 31.10.2023.

10 Yan, Zheng; Zhang, Pen; Vasilakos, Athanasios V. 2014. A survey on trust management for Internet of Things. Journal of Network and Computer Applications. Vol. 42, s. 120-134.

11 Loi, Franco; Sivanathan, Arunan; Gharakheili, Hassan Habibi; Radford, Adam; Sivaraman, Vijay. 2017. Systematically Evaluating Security and Privacy for Consumer IoT Devices. Verkkoaineisto. University of South

Wales. <https://www2.ee.unsw.edu.au/~hhabibi/pubs/conf/17iotsnp.pdf>. Luettu 31.10.2023.

12 SecDevOps: What it is and why it matters. 2023. Verkkoaineisto. Pluralsight. <https://www.pluralsight.com/blog/software-development/secdevops> 5.5.2023. Luettu 12.2.2024.

13 Cheruvu, Sunil; Kumar, Anil; Smith, Ned; Wheeler, David M. 2020. Demystifying Internet of Things Security. Berkeley: Springer Nature.

14 Hackers Remotely Kill a Jeep on the Highway - With Me in It. 2015. Verkkoaineisto. Wired. <https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/>. 21.7.2015. Luettu 17.5.2023.

15 Leffer, Lauren. 2023 Hackers Render Tesla Car Unsafe to Drive, Win Themselves a Model 3. Verkkoaineisto. Gizmondo. <https://gizmodo.com/tesla-hack-hackers-model-3-elon-musk-hackathon-1850263319>. 24.3.2023. Luettu 17.5.2023.

16 What is the Mirai Botnet? Verkkoaineisto. Cloudflare. <https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/>. Luettu 3.12.2023.

17 Tsiknas, Konstantinos; Taketzis, Dimitrios; Demertzis, Konstantinos; Skianis, Charalabos. 2021. Cyber Threats to Industrial IoT: A Survey on Attacks and Countermeasures. Verkkoaineisto. <https://www.preprints.org/manuscript/202102.0148/v1>. Luettu 8.11.2023.

18 Chan, Haowen; Perrig, Adrian; Song, Dawn. 2003. Random Key Predistribution Schemes for Sensor Networks. Verkkoaineisto. Carnegie Mellon University. <https://www.cs.cmu.edu/~haowen/chan_randomkey.pdf>. Luettu 8.11.2023.

19 Banks, Alexander Sprogø; Kisiel, Marek; Korsholm, Philip. 2021. Remote Attestation: A Literature Review. Verkkoaineisto. Arxiv. <https://arxiv.org/abs/2105.02466>. 12.5.2021. Luettu 8.11.2023.

20 Dusku, Edlira. Remote Attestation. Verkkoaineisto. Scholarly Community Encyclopedia. <https://encyclopedia.pub/entry/7912>. 31.1. 2022. Luettu 8.5.2023.

21 Narasimhan, Poornima. 2019. Golang for Internet of Things - A perspective. Verkkoaineisto. Medium. <https://talktopoorni.medium.com/my-2-cents-golang-for-internet-of-things-596de9b1f8df>. 23.12.2019. Luettu 28.11.2023.

22 Shah, Nehal. 2019. Major Reasons to use Golang for IoT Platform. Verkkoaineisto. Medium. <https://medium.com/@nehal.shah2131/major-reasons-to-use-golang-for-iot-platform-45fa1f4ca348>. 19.4.2019. Luettu 28.11.2023.

23 Dymora, Paweł; Paszkiewicz, Andrzej. 2020. Performance Analysis of Selected Programming Languages in the Context of Supporting Decision-Making Processes for Industry 4.0. Applied Sciences 10, no. 23: 8521.

24 Arthur, W.; Challener, D. 2015. A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security. Berkeley: Apress.

25 Berger, Stefan. 2022. Software TPM. <https://github.com/stefanberger/swtpm>.

26 Introduction to JSON Web Tokens. Verkkoaineisto. Auth0. <https://jwt.io/introduction> Luettu 8.5.2023.

27 MQTT. 2022. Verkkoaineisto. MQTT.org. <https://mqtt.org/faq/>. Luettu 28.9.2023.

28 What is a Relational Database (RDBMS)? Verkkoaineisto. Oracle. <https://www.oracle.com/database/what-is-a-relational-database/>. Luettu 28.9.2023.

29 What is a time series database? Verkkoaineisto. Influxdata. <https://www.influxdata.com/time-series-database/>. Luettu 29.9.2023.

30 TPMCourse. 2023. Verkkoaineisto. Nokia. <https://github.com/nokia/TPMCourse> Luettu 22.5.2023.

31 TPM2 Tools ohjelmisto. 2022. <https://github.com/tpm2-software/tpm2-tools>.

32 Cope, Steve. 2023 Introduction to MQTT Security Mechanisms. Verkkoaineisto. Steve's Internet Guide. <http://www.steves-internet-guide.com/mqtt-security-mechanisms/>. 16.6.2023. Luettu 28.11.2023.

33 Dirin, Amir; Oliver, Ian; Laine, Teemu H. 2023. A Security Framework for Increasing Data and Device Integrity in Internet of Things Systems. Sensors 23(17), 7532.

34 Building Go Applications for Different Operating Systems and Architectures. 2019. Verkkoaineisto. DigitalOcean. <https://www.digitalocean.com/community/tutorials/building-go-applications-for-different-operating-systems-and-architectures> 9.10.2019. Luettu 12.2.2024.