Ville Kuokkanen

Sulautettujen tietoturvakomponenttien käyttö pilviympäristössä

Tietotekniikan pro gradu -tutkielma

8. marraskuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Ville Kuokkanen

Yhteystiedot: ville.j.kuokkanen@student.jyu.fi

Ohjaaja: Timo Hämäläinen

Työn nimi: Sulautettujen tietoturvakomponenttien käyttö pilviympäristössä

Title in English: Usage of embedded secure elements in a cloud environment

Työ: Pro gradu -tutkielma

Opintosuunta: Kaikki opintosuunnat

Sivumäärä: 62+8

Tiivistelmä: Sulautetut järjestelmät ovat usein hyvin vähäresurssisia, jonka takia tavallisesti käytettyjä salausmenetelmiä ei ole mielekästä käyttää. Erilliset tietoturvakomponentit mahdollistavat salauksen käytön myös prosessointiteholtaan rajoitettujen laitteiden kanssa. Tässä tutkielmassa on vertailtu kolmen eri valmistajan tietoturvakomponenttien suorituskykyä, sekä toimintaa pilvipalvelun kanssa. Laitteet olivat suorituskyvyltään verrattaen samanlaisia. Suurimat erot laitteiden välillä olivat vaadittujen kirjastojen käytössä ja laitteiden käyttöönotossa tämän tutkielman tarpeisiin.

Avainsanat: sulautetut järjestelmät, tietoturvakomponentit, salaus, pilvipalvelut

Abstract: Embedded systems are often very resource limited, therefore commonly used encryption algorithms aren't feasible to use. Dedicated secure elements allow the use of encryption algorithms with resource constrained devices. In this thesis three secure elements from different manufacturers are compared by performance and how they can be used with a cluod computing service. The three devices were almost equal by performance. The biggest differences were in the usage of the needed software libraries and how the devices needed to be initialized for this use case.

Keywords: embedded systems, secure elements, encryption, cloud computing

Termiluettelo

IoT	Esineiden internet (Internet of things)
ECC	Elliptisiin käyriin liittyviin laskutoimituksiin perustuvia julkisen avaimen salausmenetelmiä.
RSA	Epäsymmetrinen julkisen avaimen salausalgoritmi (Rivest–Shamir–Adlem
TPM	Suojauskäsittelijä (Trusted platform module)
SE	Tietoturvakomponentti (Secure element)
CA	Certificate authority
MITM	Man-in-the-middle -hyökkäys
HSM	(Hardware security module)
SoC	Järjestelmäpiiri (System on a Chip)
MQTT	Pieniresurssisia laitteita varten suunniteltu sovelluskerroksen tiedonsiirtoprotokolla
AMQP	Sovelluskerroksen tiedonsiirtoprotokolla (Advanced Message Queuing Protocol)
TLS	Verkkoliikenteessä yleisesti käytetty salausprotokolla (Transport Layer Security)
DTLS	TLS versio UDP:lle
PKI	Julkisen avaimen järjestelmä (Public key infrastructure)

Kuviot

Sisällys

1 Johdanto

Sulautettujen järjestelmien suosio on noussut huomattavasti viime aikoina. Suuri määrä melko yksinkertaisia laitteita internetissä antaa todella suuren hyökkäyspinnan pahantahtoisille toimijoille. Sulautetuissa järjestelmissä pääosissa ovat energiatehokkuus, hinta, käyttöikä ja päivitysmahdollisuudet. Tavanomaisissa sulautetuissa järjestelmissä tietoturva saattaa jäädä taka-alalle mm. siitä syystä, että tehokas tietoturvan toteuttaminen vähällä prosessointiteholla vie laitteen resursseja tarpeettoman paljon muilta laitteen keskeisiltä toiminnoilta. Sulautettujen järjestelmien ja pilvipalveluiden yhdistäminen uudeksi kokonaisuudeksi on melko uusi käsite. Käyttötapauksia pilvi-IoT:lle ovat mm. “smart cities”, kodin automaatio ja terveyspalvelut. Kaikissa näissä käyttökohteissa IoT:lla voidaan parantaa palvelujen ja ympäristöjen hyötykäyttöä.

Pilvipalveluiden ja sulautettujen laitteiden välisen turvallisuuden parantamiseen on kirjallisuudessa esitetty useita keinoja, joista yksi on erillisen tietoturvakomponentin lisääminen osaksi kokonaisuutta. Tietoturvakomponentti voidaan lisätä olemassa olevaan laitteeseen, jolloin se suorittaa tavanomaiselle pieniresurssiselle laitteelle liian vaikeita kryptografisia operaatioita käyttäen laitteistotason toteutusta.

Laitevalmistajilta, kuten Microchip:ltä ja NXP:ltä on tulossa tai tullut markkinoille erillisiä tietoturvakomponentteja sulautetuille järjestelmille, jotka mahdollistavat laitteen turvallisen käytön, päivittämisen ja hyökkäyksien haittojen minimoinnin. Näiden laitteiden käyttöönotto on pyritty tekemään mahdollisimman helpoksi, jotta parempaan tietoturvaan ei tarvitsisi panostaa kohtuuttomia resursseja. IoT -laitteen ja pilven välinen autentikointi ja salaus parantaisi osaltaan pilvi-IoT:n tietoturvaa.

2 Sulautetut järjestelmät

Sulautetut järjestelmät ovat johonkin tiettyyn erityiseen tarkoitukseen tarkoitettuja laitteita. Sulautettuja järjestelmiä ovat mm. älypuhelimet, ohjauslaitteet ja muut vastaavat osasysteemit. IoT - laitteet, eli asioiden internet on käsite, jossa IoT-laite on yhteydessä toisiin vastaavanlaisiin fyysisiin laitteisiin internetin välityksellä. IoT -laite voi olla sulautettu järjestelmä, mutta kaikki sulautetut järjestelmät eivät ole IoT -laitteita. Tämän tutkielman puitteissa termit tarkoittavat kuitenkin samaa asiaa.

“Sulautetut järjestelmät ovat juuri tiettyyn tarkoitukseen olevia tietokonejärjestelmiä kohdeympäristössä tai tietokonejärjestelmiä, jotka ovat osana jotain toista järjestelmää. SoC:tä pystytään nykyisin kehittämään kustannustehokkaasti, jonka seurauksena sulautettuja järjestelmiä on kehitetty ja otettu käyttöön useisiin eri käyttötarkoituksiin johtuen prosessointitehon ja muistin halpenemisesta. Esimerkkejä sulautetuista järjestelmistä ovat verkkosovittimet tietokonejärjestelmiin ja mobiilialustoihin, ilmastoinnin ohjausjärjestelmät, teollisuuden järjestelmät, autot ja valvontajärjestelmät.” (Serpanos ja Wolf 2011).

2.1 Tietoturva

Riittävän turvallisuuden lisääminen laitteeseen vie prosessointitehoa muilta hyödyllisiltä ominaisuuksilta. Jotta saavutetaan järjestelmän käyttöön nähden tarpeeksi hyvä tietoturva, on välttämätöntä erityisesti sulautettujen laitteiden tapauksessa tasapainottaa laitteiden resurssit “hyödyllisen” käytön ja tietoturvan vaatimien resurssien välillä. Järeämpien salausalgoritmien käyttö vie luonnollisesti enemmän resursseja laitteelta kuin kevyempi salaus. Salausalgoritmien tarkempi kuvaus on luvussa 6.1.1. Niitä voidaan käyttää itse laitteen prosessorilla tai erillisellä salausoperaatioita suorittavalla komponentilla. (Saadatmand ja Leveque 2012).

Myös tutkimuksessa Zeeshan, Reed ja Siddiqui 2019 mainitaan, että monien IoT -laitteiden suorituskyky ei vain riitä riittävän salauksen (esim. PKI) toteuttamiseen.

Seuraava luokittelu on osa-alueista, joihin pyritään pääsemään turvallisessa järjestelmässä. CIA-kolmiota käytetään yleisesti kuvaamaan tietoturvahaasteita. Näitä kolmea ominaisuutta

(luottamuksellisuus, eheys ja saavutettavuus) pyritään tasapainottamaan, jotta hyvä tietoturva saadaan toteutettua. Seuraavasta luokittelusta kolme ensimmäistä kohtaa esiintyy useimmiten kirjallisuudessa (CIA). Jälkimmäiset osat on joissain tutkimuksissa otettu luokitteluun lisäksi.

- Luottamuksellisuus (Confidentiality): Informaatioon käsiksi pääsy vain sallituille tahoille.
- Eheys (Integrity): Tiedon muuttamisen estäminen ei-sallituilta tahoilta.
- Saatavuus (Availability): Tieto on saatavilla ja saavutettavissa.
- Autentikointi (Authentication): Tiedon lähettäjän tai vastaanottajan tunnistaminen.
- Pääsyn valvonta (Access control/Accountability): Tiedon saatavuus vain valikoiduille tahoille.
- Kieltämättömyys (Non-repudiation): Lähettäjä tai vastaanottaja ei voi kiistää toimintoja.
- Luotettavuus (Dependability): Järjestelmä pysyy käytettävänä.
- Turvallisuus (Safety): Järjestelmän käytöstä ei aiheudu haittaa käyttäjille.
- Yksityisyys (Privacy): Yksityiset tiedot ja resurssit on suojattu ulkopuolisilta tahoilta.

(Dhillon ja Kalra 2016; Serpanos ja Voyiatzis 2013; Neshenko ym. 2019).

IoT -järjestelmässä, jossa ei ole tunnistautumista rikkoo luottamuksellisuutta. Ilman salausta toimiva järjestelmä, joka mahdollistaa datan muokkaamisen tai väärentämisen rikkoo järjestelmän eheyttä.

IoT -järjestelmän tietoihin luvaton käsiksi pääsy rikkoo luottamuksellisuutta (C). Tietojen luvaton muokkaaminen, kuten datan väärentäminen rikkoo eheyttä (I). IoT -laitteeseen pääsyn tai sen toiminnan estäminen tai hidastaminen liittyvät saatavuuteen (A) (Neshenko ym. 2019). Tietoturvakomponenttien tapauksessa kaikki edellä mainitut kohdat ovat erittäin tärkeitä järjestelmien tietoturvan kannalta.

Jotta sulautetut järjestelmät olisivat yhtä turvallisia, kuin tietokonejärjestelmät yleensä, on tietoturvaan panostettava enemmän. Sulautettuja järjestelmiä on yhdessä käyttökohteessa yleensä useampia, niillä on käytössä vain rajoitettu määrä resursseja ja ne voivat olla epäsuotuisissa ympäristöissä turvallisuuden kannalta, kuten julkisissa tiloissa. Hyökkäyspinta-

ala sulautetuille järjestelmille on siis varsin suuri (Serpanos ja Voyiatzis 2013).

Yleisesti internetissä käytössä oleva IP-pino tarjoaa hyvän tietoturvan. Täyttä IP -pinoa on kuitenkin joissain tapauksissa hankala toteuttaa sulautetuissa laiteissa sellaisenaan (Garg ja Dave 2019). On kuitenkin mahdollista käyttää esimerkiksi REST-API:a väliabstraktiotasona tuomaan lisäturvallisuutta. Näin ollen itse IoT -laite ei ole suoraan yhteydessä internetiin tai pilvipalveluun.

Hassija ym. (2019) mukaan IoT -järjestelmän tietoturvan voi jakaa viiteen eri osa-alueeseen, jotka ovat itse laite, verkko, väliohjelmisto, yhdyskäytävä ja sovelluskerros. Toinen jaottelu on tutkimuksessa Neshenko ym. 2019 esille tuotu luokittelu: Laite, verkko ja sovellus.

IEEE 802.15.4 -standardia käyttävä IoT laitteen linkkikerroksen tietoturvan toteuttaminen ohjelmapuolella lisää energiankulutusta laitteessa jopa 85%. (Alharby ym. 2018)

IoT -ympäristön tietoturvaa ei saavuteta pelkästään parantamalla verkon turvallisuutta tai suojaamalla yksittäisiä laitteita, sillä kuten Keoh, Kumar ja Tschofenig 2014 tuovat esille sen, että yhden laitevalmistajan hyvin suojattu ja “turvallinen” laite ei ole suuremman yleiskuvan kannalta IoT -maailmassa kaikista tärkein asia. IoT -laitteiden kuuluisi kommunikoida keskenään turvallisesti ja tälle kommunikaatiolle olisi hyvä olla olemassa jokin oma standardi salatulle verkkoliikenteelle. Keoh, Kumar ja Tschofenig 2014 ehdottavat “constrained application protocol” (CoAP) ja “datagram transport layer security” (DTLS) käyttöä matalan prosessointitehon laitteissa. Avainmateriaalin asettaminen laitteeseen laitteen valmistuksen, käyttöönoton, ja käyttöoikeuksien lisäämisen aikana laitteen fyysiset turvallisuusmoduulit estävät pääsyn salausavaimiin tai niiden muuttamiseen (Keoh, Kumar ja Tschofenig 2014). Näitä paremman turvallisuuden takaavia keinoja on käytetty juuri esim. Microchip:n ATECC608A -laitteessa.

IoT-järjestelmien eri kerroksille on olemassa seuraavien alalukujen mukainen luokittelu (Hassija ym. 2019).

2.1.1 Laitteen tietoturva

Schaumont 2017 mukaan suoraan laitteeseen kohdistuvat hyökkäykset voidaan jakaa karkeasti kolmeen kategoriaan: I/O:n manipulointi, muistiin kohdistuva hyökkäys ja laitteen fyysinen manipulointi.

I/O:n manipulointi: Jos hyökkääjällä on pääsy laitteen syötteisiin, voidaan siinä tapauksessa toteuttaa mm. muistin ylivuotoja tai muistin korruptoimista (Schaumont 2017). Näitä hyväksikäyttämiseen voidaan käyttää ROP:ia (return-oriented programming) ja ajaa hyökkääjän omaa haluamaansa ohjelmakoodia laitteessa.

Muistiin kohdistuvat hyökkäykset: Jotta voi käsitellä samaa muistiavaruutta, kuin missä laitteen suojattu laiteohjelmisto sijaitsee, täytyy siihen päästä ensin käsiksi jollain keinolla. Tällainen keino voi olla esim. muunnellun ajurin tai muun laiteohjelmiston asentaminen laitteelle (Schaumont 2017). Tilanteessa, jossa ohjelman muistiin päästäisiin kuitenkin käsiksi, olisi hyvä, että siitä seuraisi mahdollisimman vähän vahinkoa itse järjestelmälle. Tästä syystä on erittäin tärkeää, että mahdolliset laitteella säilytettävät salausavaimet ovat tallessa siten, että niihin käsiksi pääseminen olisi mahdollisimman hankalaa. Erilliset tietoturvakomponentit muun muassa mahdollistavat salausavaimien tallentamisen yleisen muistiavaruuden ulkopuolelle.

Fyysinen manipulointi: Tässä tilanteessa hyökkääjällä on välitön fyysinen pääsy laitteeseen. Tähän kuuluu “Side-channel analysis” ja “fault-injection analysis”.

Sivukanavahyökkäykset ovat hyökkäyksiä, joissa hyökkääjä tarkastelee laitteen fyysisiä ominaisuuksia sen suorittaessa jotain operaatiota. Tällainen operaatio voi olla esimerkiksi salausalgoritmin suorittaminen, jolloin laitteen virrankulutus, elektromagneettinen säteily tai suoritusaika voi muuttua. Sivukanavahyökkäykset vaativat yleensä fyysisen käsiksi pääsyn itse laitteeseen.

Fyysistä manipulaatiota on myös se, jos sulautettuun laitteeseen on mahdollista yhdistää jokin siihen tavallisesti kuulumaton väline, kuten USB-tikku tai näppäimistö. Joissain tapauksissa voi olla, että laitteeseen käsiksi pääsy verkon kautta on tehty todella hankalaksi, mutta kuka tahansa tunnistautumaton taho voi käyttää näppäimistöä tai jotain muuta laitetta hyväk-

si.

Tutkimuksessa Yuce ym. 2017 RISC -pohjaiselle sulautetulle järjestelmälle on kehitetty useampaan abstraktiotasoon perustuva fault-injection hyökkäysviitekehys. Tällä menetelmällä on mahdollista päätellä laitteella olevan sovellustason salauksen salausavain käyttämällä eri metodeja. Ylimmillä tasoilla voidaan tarkastella laitteen toimintaa virhetilanteissa sovellustasolla ja alemmilla tasoilla tarkastellaan laitteen toimintaa aina lähempänä laitteistotasoa. Tutkimuksen mukaan fault-injection:in estämiseen kuuluu turvallisuuden oltava monitasoinen, jotta tämän kaltaiset hyökkäykset voitaisiin estää kokonaan.

Itse laitteen ohjainohjelmistossa voi olla haavoittuvuuksia. Tutkimuksessa Costin ym. 2014 selvitetään yleisesti ohjainohjelmistojen haavoittuvuuksia. Todella monessa tutkimuksessa tutkitussa ohjainohjelmistossa on tietoturvariskejä liittyen huonoon konfigurointiin ja alustukseen. Osassa ohjainohjelmistoja laitteen oletussalasana löytyy ohjainohjelmistosta sellaisenaan, kun taas osassa se löytyy salattuna.

Monissa ohjainohjelmistoissa on mukana nykyisiä tai vanhoja versioita kirjastoista ja ohjelmistoista, jotka on jo todettu haavoittuvaisiksi. Web-palvelimet ovat myös yleisiä sulautetuissa laitteissa. Näidenkin konfiguraatioissa on monessa tapauksessa todettu olevan haavoittuvuuksia.

Pahimmillaan ohjainohjelmistojen haavoittuvuudet voivat mahdollistaa hyökkääjälle helpon ja suoraviivaisen pääsyn käsiksi laitteeseen. Julkisesti saatavilla olevien ohjainohjelmistojen tapauksessa hyökkääjä voi joissain tapauksissa saada niistä kaiken tarvittavan informaation hyökkäyksen toteuttamiseen.

Vaikka ohjainohjelmisto itsessään ei olisi haavoittuvainen, on hyökkääjän joissain tilanteissa mahdollista päivittää laitteen ohjainohjelmisto haluamakseen. Tätä varten voidaan käyttää salausta varmistamaan päivitettävän ohjainohjelmiston eheys ja alkuperä (Dhobi ym. 2019).

2.1.2 Verkon haavoittuvuudet

Yksinkertaisimmat haavoittuvuudet (sulautetuissa) järjestelmissä ovat tehdasasetuksille jätetyt käyttäjätunnukset ja salasanat. Nämä tunnukset ovat yleensä helposti saatavilla, ja nii-

den jättäminen sellaisekseen aiheuttaa pahimmillaan helpon ja täyden pääsyn järjestelmään hyökkääjälle.

Sniffing -tyyppisissä hyökkäyksissä hyökkääjä voi monitoroida verkon liikennettä IoT -laitteen ja esimerkiksi pilven välillä. Tämänkaltainen hyökkäys on erityisen vakava, jos yhteys ei ole salattu, sillä hyökkääjä voi näin ollen päästä käsiksi mm. käyttäjätietoihin ja kirjautumistunnuksiin (Hassija ym. 2019).

Yksi tyypillisistä verkon haavoittuvuuksia hyödyntävistä hyökkäyksistä on Man-in-the-middle (MITM) hyökkäykset. Man-in-the-middle -hyökkäykset perustuvat siihen, että hyökkääjä pääsee käsiksi viestintäkanavaan. Näin ollen hyökkääjä voi vaikuttaa siihen, mitä hyökättävä kohde lähettää ja vastaanottaa. MITM -hyökkäykset ovat yleensä helppoja toteuttaa salaamattomissa yhteyksissä. Vaikka Man-in-the-middle -tyyppisiä hyökkäyksiä voi toteuttaa mm. sovellustasolla, ovat ne yleisempiä verkkotasolla TCP-liikenteessä.

Tutkimuksessa Roohi, Adeel ja Shah 2019 IoT -laitteiden DDoS -hyökkäyksille esitetään seuraava kategorisointi: Resurssien “depletion”-hyökkäys, kaistan “depletion” hyökkäys, infrastruktuurin hyökkäys ja nollapäivähyökkäys. DDoS/DoS -hyökkäyksessä hyökkääjä täyttää kohteen resurssit suurella määrällä pyyntöjä. Jos lähteitä on useampia, on kyseessä hajautettu palvelunestohyökkäys (DDoS). IoT -verkot ovat yleensä heterogeenisiä ja monimutkaisia, joten verkkokerros tässä tapauksessa on erityisen altis palvelunestohyökkäyksille (Hassija ym. 2019). Tutkimuksessa Angrishi 2017 todetaan, että pääsy suurimpaan osaan IoT-botnettiin kaapatuista laitteista on saatu hyödyntämällä heikkoja salasanoja.

Laitteesta ja ympäristöstä riippuen Denial-of-Service hyökkäys on mahdollista verkko- ja sovelluskerroksella. IoT -laitteen ollessa suoraan kiinni verkossa DoS -hyökkäyksiä on varsin hankala estää. Jos kuitenkin laitteen takana on kuitenkin jokin välitaso, jonka kautta kaikki liikenne esimerkiksi pilveen kulkee, voidaan välikerroksella käyttää järeämpää palomuuria ja muita havaitsemisjärjestelmiä.

Koska useimmiten IoT-laitteiden toiminta ei sijoitu ISO/OSI -mallissa samalla tavalla kuin tavanomaisten laitteiden kanssa, on uhka verkkokerroksen haavoittuvuudelle erityisen suuri. ZigBee, LoRaWAN ja LoWPAN verkot toimivat eri tavalla, kuin tavalliset verkkolaitteet. LoRaWAN tekniikka korvaa iso/osi ja tcp/ip pinoista kuljetus- ja verkkokerroksen (Adelantado

ym. 2017). ZigBee rakentuu osittain 802.15.4 -verkon päälle ottaen siitä mukaan fyysisen- ja mac-kerroksen. Myös bluetooth (BLE) pinon toiminta on erilainen verrattuna zigbee:hen (802.15.4) ja LoRaWAN:iin.

LoRa on LoRaWAN:ssa käytetty fyysinen kerros. Se tarjoaa pienen virrankulutuksen (noin 10 vuotta yhdellä paristolla), matalan tiedonsiirtonopeuden (27 kb/s levityskertoimella 7 ja kanavan leveydellä 500kHz tai 50 kb/s FSK:n avulla) ja pitkän välimatkan tiedonsiirtoa (2-5 kilometriä tiheään rakennetuilla alueilla ja 15 kilometriä asutusalueilla) (Adelantado ym. 2017).

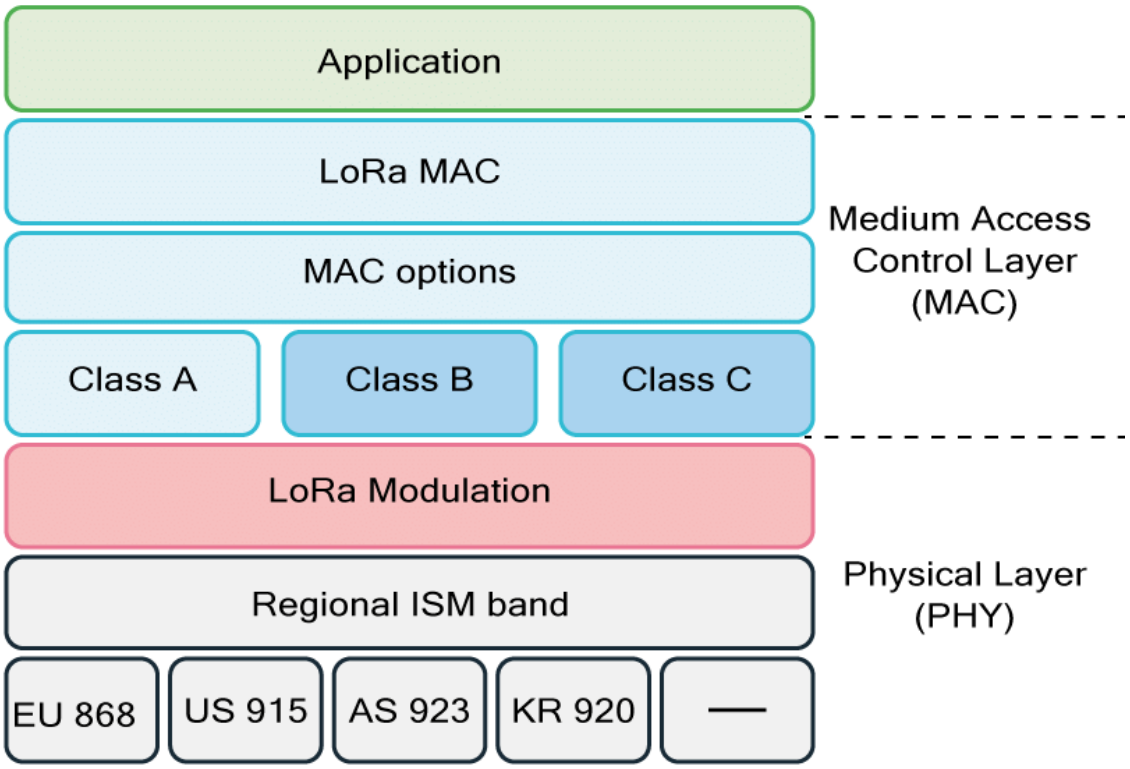
“Iot-laitteita varten sopivia kommunikointiteknologioita on useita. Lyhyen kantaman langattomia teknologioita kuten, Bluetooth, ZigBee ja Z-Wave on käytetty matalaresurssisien IoT-verkkojen kanssa niiden pienen virran kulutuksen vuoksi. Näitä langattomia teknologioita käyttäessä niistä tulee kuitenkin rajoite, kun tarvitaan tiedonsiirtoa pidemmälle matkalle. Tämänkaltaisia käyttökohteita ovat muun muassa älykaupungit. “Low Power Wide Area Network” (LPWAN) mahdollistaa pitkän kantaman kommunikaation matalalla virrankulutuksella niin, että matalatehoiset sensorit ja sulauteut järjestelmät voivat lähettää signaalia jopa kymmeniä kilometrejä ja tehdä niin jopa kymmenen vuotta ilman ulkoista virtalähdettä” (Butun, Pereira ja Gidlund 2018).

LoRaWAN on erityisesti tarkoitettu matalan virrankulutuksen laitteille. A- ja B- tason LoRaWAN luokitus on tarkoitettu sellaisen käyttötarkoituksen laitteille, jossa tiedon lähettäminen tapahtuu jaksoittain. C- tasolla laite voi joko vastaanottaa tai lähettää dataa jatkuvasti.

Tutkimuksessa Butun, Pereira ja Gidlund 2018 listataan seuraavat mahdolliset hyökkäykset LoRaWAN:in spesifikaatiolle 1.1: “RF jamming attack”, “Replay attack”, “Beacon (Class B) synchronization attack”, “Network traffic analysis” ja man-in-the-middle hyökkäykset.

MITM hyökkäys voi olla lorawan -verkossa erityisen vaarallinen, sillä osa “Application” palvelimen ja “Network” palvelimen välillä tapahtuvasta viestin välityksestä tapahtuu ilman salausta. Tätä kautta hyökkääjä voi päästä käsiksi session salausavaimeen (Butun, Pereira ja Gidlund 2018).

Kuvio 1: LoRaWAN pino (Kim, Lee ja Kim 2019).



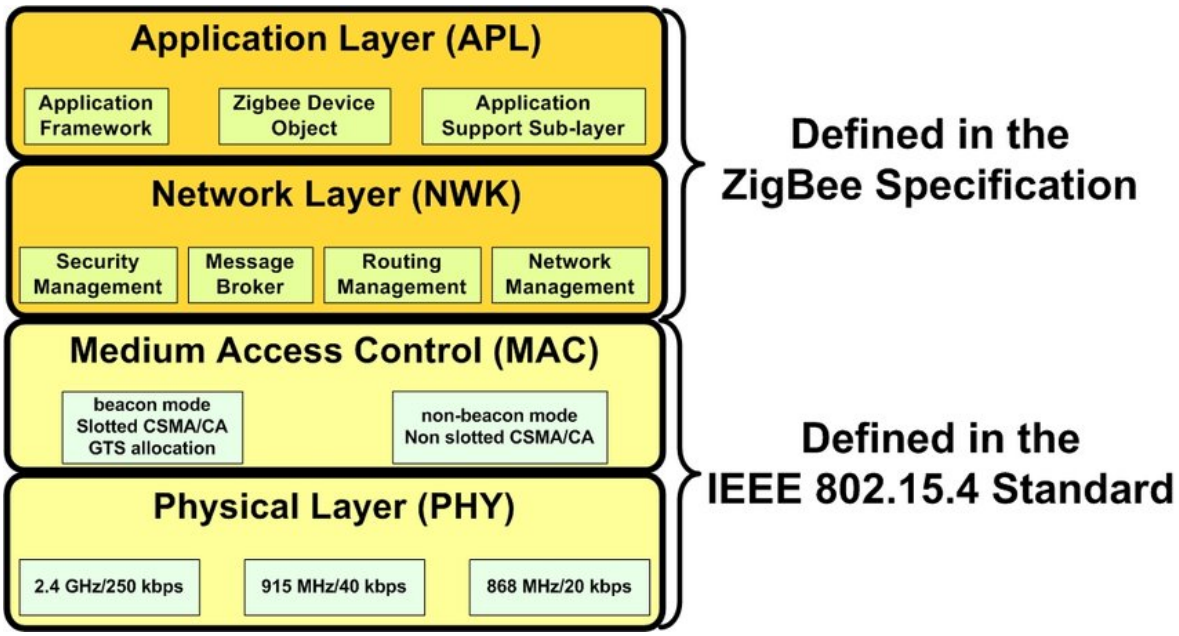
- ZigBee
- turhat avoimet portit
- DDoS hyökkäykset
- reitityshyökkäykset
- muut yleiset hyökkäykset verkossa

(Hassija ym. 2019)

(Garg ja Dave 2019) vähennetään verkon haavoittuvuuksia hyödyntämällä välirajapintaa (Chen, Guo ja Hu 2019): suurimpana haasteena sulautettujen laitteiden tietoturvaa on verkon haavoittuvuudet.

IoT-laitteissa on yleisesti käytössä jokin langaton tiedonsiirtomenetelmä. Tätä varten on kehitetty standardi IEEE 802.15.4, joka määrittelee LR-WPAN:in toiminnan. IEEE 802.15.4:n toimintaa on laajennettu standardeilla, joita ovat mm. ZigBee ja 6LoWPAN.

Kuvio 2: ZigBee pino ja IEEE 802.15.4 -standardi (Koubaa, Alves ja Tovar 2021).



ZigBee on yleisesti käytössä oleva vähävirtaisten laitteiden lähialueen langaton protokolla, joka perustuu 802.15.4:ään. ZigBee lisää 802.15.4:n päälle seuraavat kerrokset:

- Network
- Application Support
- Security Service Provider
- Zigbee Device Object
- Application Framework

ZigBee:n verkkokerros mahdollistaa verkkoihin liittymisen ZigBee:tä tukeville laitteille, sekä määrittelee ZigBee-verkon reitityksen. Application support -kerros on vastuussa pakettien hallinnoinnista ja verkkoon liittyneiden laitteiden tiedoista. “Security Service Provider”-kerros vastaa verkko- ja sovelluskerroksen turvallisuuden hallinnoinnista. ZigBee device object- kerros vastaa laitteiden löytämisestä sekä verkko, että sovelluskerroksen toiminnasta. Application framework -kerros sisältää ympäristön käyttäjän sovelluksille.(Khanji, Iqbal ja Hung 2019).

2.1.3 Väliohjelmiston haavoittuvuudet

Väliohjelmisto on taso lopullisen käyttökohteen ja verkkokerroksen välillä. Esimerkiksi pilvipalvelua voidaan käyttää väliohjelmistona pilvi-IoT:ssä siten, että data prosessoidaan ja välitetään jalostettuna eteenpäin esim. nettisivulle tai muuhun sovellukseen. Tämän kerroksen tulisi olla erityisen turvallinen, sillä yhden järjestelmän kaikki IoT-laitteet voivat olla yhdistettynä yhteen paikkaan (Hassija ym. 2019). Pahimmillaan väliohjelmistoon murtautumisesta hyökkääjä voi päästä käsiksi kaikkiin järjestelmän laitteisiin.

Seuraavat haavoittuvuudet ovat mahdollisia väliohjelmistossa.

- pilven malware injektio
- MITM
- pilven VM:n haavoittuvuudet
- “flooding attack”

(Hassija ym. 2019)

“Pilven malware-injektion tapauksessa hyökkääjä voi injektoida haitallista ohjelmakoodia tai haitallisen virtuaalikoneen pilveen. Hyökkääjä voi teeskennellä olevansa pätevä palvelu luomalla uuden virtuaalikoneinstanssin tai haitallisen palvelumoduulin. Tällä tavalla hyökkääjä voi saada pääsyn hyökättävän kohteen palvelupyyntöihin ja voi näin ollen päästä käsiksi arkaluontoiseen dataan” (Hassija ym. 2019).

Pilvipalvelusta riippuen virtuaalikoneen tai “containerin” toteutuksessa voi olla haavoittuvuuksia, joita hyökkääjä voi käyttää hyväksi päästäkseen käsiin jonkun toisen käyttäjän virtuaalikoneeseen tai jopa itse isäntälaitteeseen.

2.1.4 Yhdyskäytävän haavoittuvuudet

Yhdyskäytävällä tarkoitetaan tässä yhteydessä välikerrosta pilven ja IoT-laitteen välillä. Yhdyskäytävä voi yhdistää useampia laitteita toisiinsa tai sitä voidaan käyttää parantamaan koko järjestelmän tietoturvaa luomalla abstraktiotaso eri tasojen välille.

Tämän tason haasteita turvallisuudessa ovat muun muassa:

- “Secure on-boarding”
- Turhat rajapinnat gateway-laitteessa
- Päästä päähän salaus
- IoT -laitteen ohjainohjelmiston päivittäminen gateway-laitteen avulla

“Secure on-boarding”: Kun uusi laite liitetään osaksi IoT-järjestelmää, mahdolliset käytettävät salausavaimet kulkevat yhdyskäytävän kautta, jos sellainen on käytössä. Jos yhdyskäytävän lähtevää tai tulevaa verkkoliikennettä on mahdollista salakuunnella, on man-in-the-middle -tyyppisellä hyökkäyksellä mahdolista päästä käsiksi salausavaimiin, jotka päätelaite ja pilvipalvelu vaihtavat.

Ylimääräiset rajapinnat yhdyskäytävässä voivat vaikuttaa järjestelmän turvallisuuteen negatiivisesti. Yhdyskäytävässä tulisi olla vain ja ainoastaan ne rajapinnat mitä yhdistetyt (IoT) laitteet käyttävät. Ei ylimääräisiä portteja, protokollia tai muuta pääsyä yhdyskäytävään (Hassija ym. 2019).

2.1.5 Sovelluskerroksen haavoittuvuudet

- “Access control” hyökkäykset
- “Reprogram” hyökkäyset
- DDoS hyökkäyset
- Datan varastaminen

(Hassija ym. 2019)

Sovelluskerroksella tarkoitetaan tässä yhteydessä esimerkiksi pilvipalvelusta saadun datan loppukäyttöä käyttöympäristöissä, kuten nettisivuilla tai mobiilisovelluksessa. Sovelluskerroksen riskit liittyvät suurilta osin saatavuuteen ja loppukäyttäjän yksityisyyteen. Yksityinen data voi joutua vääriin käsiin, jos loppusovelluksen tietoturva on heikko.

Myös tämä kerros on haavoittuvainen joillekin samoille haavoittuvuuksille, joita on myös verkkokerroksella. Heikot salasanat ovat yksi myös yksi suurimmista hyökkäyskohteista tälläkin kerroksella. Loppukäyttäjän IoT-sovellukseen voidaan vaikuttaa DDoS hyökkäyksellä siten, että vaikutetaan johonkin järjestelmän osaan esimerkiksi johonkin sovelluksen kom-

ponenttiin, joka sijaitsee jossain välissä pilvipalvelua ja loppukäyttäjän sovellusta (Hassija ym. 2019; Neshenko ym. 2019).

2.2 Sulautetut tietoturvakomponentit

Erilliset tietoturvakomponentit (SE) ovat laitteita, jotka yksinkertaisimmillaan tallentavat turvallisesti muistiinsa yksityisen salausavaimen. Isäntälaite voi käyttää tietoturvakomponenttia suorittamaan kryptografisia toimenpiteitä ilman, että salausavaimet pääsevät missään vaiheessa itse isäntälaitteeseen. Serpanos ja Voyiatzis 2013 tutkimuksessa todetaan, että vaikka salatun liikenteen toteuttaminen laitteessa onnistuisi, on "tavallisessa" sulautetussa järjestelmässä hankala saada salausavainta niin hyvin talteen, että hyökkääjän olisi täysin mahdotonta päästä siihen käsiksi. Tämän epäkohdan SE:n kuuluisi korjata täysin tekemällä salausavaimiin käsiksi pääsyn fyysisesti mahdottomaksi.

HSM:n (Hardware Security module) ja SE:n (Secure element) eroavaisuudet ovat hyvin pienet. HSM:llä tarkoitetaan yleisesti ottaen laitetta, joka pystyy tallentamaan salausavaimia ja suorittamaan salaukseen liittyviä operaatioita. SE tarkoittaa samaa asiaa, mutta pienemmässä mittakaavassa. SE:t ovat yleensä samaan piirilevyyn juotettuja erillisiä komponentteja, tai ne ovat osana itse SoC:tä.

Salausoperaatioiden siirtämisessä pois isäntälaitteesta on merkittäviä etuja. On mahdollista tehdä normaalisti pienitehoiselle sulautetulle järjestelmälle hankalia kryptografisia operaatioita käyttäen laitetason toteutusta tietoturvakomponentissa. Parhaimmassa tapauksessa erillinen tietoturvakomponentti vie vähemmän virtaa ja on helpompi ottaa käyttöön, kuin mitä tehokkaampi laite, joka kykenisi suorittamaan operaatioita ohjelmistotasolla.

Yksi ominaisuus tietoturvakomponenteissa on ohjainohjelmiston eheyden tarkistaminen. Serpanos ja Voyiatzis 2013 tutkimuksessa juuri todetaan, että laitteen päivittämisestä voi olla enemmän haittoja kuin hyötyjä. "However, software upgrading leads to security risks, because it offers possibility for malicious software to replace legitimate software. To face these securityrisks, it is important to implement defences at both the communication protocols andthe internal organization of the system" (Serpanos ja Voyiatzis 2013).

2.3 Salausmenetelmät

Salauksella pyritään yleensä kahteen päätavoitteeseen: salassa pysyvään viestinvälitykseen ja tiedon lähteen autentikointi. On olemassa ns. symmetrisiä ja epäsymmetrisiä salaustapoja. Symmetrisessä salauksessa lähettävällä ja vastaanottavalla osapuolella on kummallakin sama yksityinen salausavain, jolla lähetettävä viesti aina salataan ja vastaanotettavan viestin salaus puretaan. Julkisen avaimen kryptografia (Public Key Cryptography (PKC)) on salausmenetelmä, joka toimii epäsymmetrisesti. Lähetettävä data voidaan aina salata julkisella salausavaimella, mutta sen purkamiseen tarvitaan yksityinen salausavain. Näitä julkisen salausavaimen menetelmiä ovat mm. (RSA, ECDH ja ECDSA), joista jälkimmäiset käyttävät hyväkseen elliptisiä käyriä (ECC). Julkisen salausavaimen menetelmää käytetään yleisesti autentikointiin, avainten jakoon, allekirjoitukseen ja sisällön salaukseen (Dhillon ja Kalra 2016)

IoT -laitteiden vähäisen prosessoritehon takia turvalliset salausmetodit ovat olleet taka-alalla. Nykyisin yleisimmin TLS:n kanssa käytössä olevat autentikointimenetelmät ovat RSA ja ECC. (Suárez-Albela ym. 2018). Samassa tutkimuksessa mainitaan, että juuri vähäresurssisia IoT-laitteita varten kehitetty DTLS ei voita virrankulutuksessa muuallakin käytössä olevaa TLS:ää sen jälkeen, kun siihenkin lisätään tarvittava turvallisuus.

RSA on jo 70-luvulla kehitetty epäsymmetrinen salausalgoritmi. RSA-avaimen luominen tapahtuu seuraavasti: (Rivest, Shamir ja Adleman 1978)

- valitaan luku n, joka on samalla RSA -avaimen pituus.
- Etsitään suuret alkuluvut p ja q.
- Löydetään luku d, jonka suurin yhteinen tekijä $(p-1)(q-1)$ kanssa on 1
- Lasketaan luku e, joka täyttää ehdon $e*d = 1(mod(p-1)(q-1))$

Tällä menetelmällä saadaan kaksi avainta. Yksityinen salausavain: lukupari (d,n) ja julkinen salausavain (e,n) (Rivest, Shamir ja Adleman 1978). RSA tukee allekirjoitusta, avainten vaihtoa ja salaus- ja purkuoperaatioita. Tyypillisimmin käytettyjä avaimen pituuksia RSA:ssa ovat 2048 ja 4096.

ECC (Elliptic Curve Cryptography) kryptografia perustuu elliptisiin käyriin. ECC avaimen

generointi toimii siten, että joltain elliptiseltä käyrältä tietyltä väliltä, esimerkiksi Secp256r1 tai vastaavalta valitaan satunnaisluku (private key), joka kerrotaan käyrän vakiopisteellä (zero point), josta saadaan x-y koordinaateilla oleva piste P, joka on julkinen salausavain. (Kapoor, Abraham ja Singh 2008; Gyamfi, Ansere ja Xu 2019). Yksityisen salausavaimen laskeminen julkisen salausavaimen perusteella on kohtuuttoman hankalaa. Tätä kutsutaan "Elliptic Curve Discrete Logarithm" "ongelmaksi".

ECC-pohjaisia salausmenetelmiä on laajasti käytössä IoT-laitteissa, sillä sen käyttö vie vain vähän resursseja ja vaatii huomattavasti vähemmän muistia, kuin yleisesti käytössä oleva RSA -järjestelmä. ECC:hen perustuvaa ECDSA:ta käytetään signaturen luomiseen ja samoin ECC:hen perustuvaa ECDHE:tä avainten vaihtoon (Suárez-Albela ym. 2018). Toisin kuin RSA, ECC:tä voi käyttää perustuen useisiin erilaisiin ja eri kokoisiin elliptisiin käyriin. Yleisesti käytettyjä käyriä ovat muun muassa:

- secp256r1
- secp256k1
- curve25519
- secp384r1

Salausalgoritmin turvallisuustaso voidaan kuvata aikana $2^X T$, jossa X on turvallisuustaso ja T on aikayksikkö, jossa yhden salatun tekstimuotoisen arvo voidaan verrata johonkin salattuun arvoon. Turvallisuustaso ei kuitenkaan yleensä ole sama kuin käytetyn avaimen pituus (Suárez-Albela ym. 2018; National Institute of Standards and Technology 2020).

Salausalgoritmien turvallisuustaso (bittiä)			
Turvallisuustaso	Symmetrinen salausalgoritmi	RSA	ECC
112	(3TDEA)	2048	secp224r1
128	AES-128	3072	secp256r1
192	AES-192	7680	secp384r1
256	AES-256	15360	secp512r1

(Suárez-Albela ym. 2018; National Institute of Standards and Technology 2020)

TLS: (Transport Layer Security) (IoT) -laitteiden täytyy tehdä salaus jollain protokollalla, ja yleisimpänä vaihtoehtona varsinkin MQTT:n AMQP:n ja HTTP:n kanssa käytetään TLS:ää.

TLS on sovelluskerroksen protokolla, joka koostuu kahdesta osa-alueesta “handshake”:sta ja “record”:sta. Handshake:n avulla kumpikin osapuoli voi todeta toisen osapuolen viestien olevan aitoja, sekä neuvotella käytettävästä salausalgoritmista. Yleisimpiä handshake algoritmeja ovat: ECDHE-ECDSA, DHE-RSA, ECDHE-RSA ja RSA. Tällä sovitulla salausalgoritmilla Record protokolla mahdollistaa salatun viestinvälityksen. Yleisin TLS:n kanssa käytetty salausalgoritmi on AES GCM (Suárez-Albela ym. 2018).

2.4 Tutkimuksia aiheesta

Tutkimuksia erilaisista keinoista parantaa IoT- laitteiden turvallisuutta on varsin paljon. Itse ohjelmistotason menetelmiä on tutkittu varsin paljon, joista suurin osa on jonkinlainen viitekehys tai menetelmämalli salausavainten turvallista vaihtoa varten. Tietoturvakomponenteista laitteistotasolla ei juurikaan löydy tutkimuksia, mutta jokaisella laitevalmistajalla on joka tapauksessa oma toteutuksensa tietoturvakomponentille.

2.4.1 IoT -laitteiden tietoturva

Laitteissa, joissa on käytössä jokin “suurempi” mikrokontrolleri, on ollut mahdollista käyttää ARM:in TrustZone teknologiaa. Tämä mahdollistaa TPM:n käytön itse mikrokontrollerissa.

IoT-laitteiden väliseen autentikointiin ja “access control”:iin on Das ym. 2019:n tutkimuksessa kehitetty menetelmä “LACKA-IoT”. Tämä menetelmä perustuu elliptisen käyrän kryptografiaan, jonka käyttäminen onnistuu myös pienen virrankulutuksen laitteissa. Tutkimuksen mukaan ehdotettu menetelmä on turvallinen seuraavia hyökkäyksiä vastaan:

- replay
- MITM
- impersonation attack
- malicious device deployment
- physical capture

- ESL leakage

Tutkimuksessa Durand ym. 2019 on testattu IoT-laitteiden turvaamista mm. tietoturvakomponentilla, ja suorituskykyä verrattu muihin vastaaviin toteutusratkaisuihin, joilla voidaan varmentaa laite.

Tutkimuksessa Suárez-Albela ym. 2018 on vertailtu ECDSA:n ja RSA:n eroavaisuuksia virrankäytössä ja suorituskyvyssä TLS:n kanssa. Tutkimuksessa salausmenetelmiä vertailtiin vähäresurssisella ESP32 -laitteella. Osoittautui, että tilanteissa, jossa ECDSA:ta ja RSA:ta käytetään samalla turvallisuustasolla, on ECDSA huomattavasti energiatehokkaampi käyttää. Paremman turvallisuustason salausmenetelmä voi olla jopa nopeampi kuin matalamman tason salaus, jos siihen käytettävät kirjastot on optimoitu paremmin (Suárez-Albela ym. 2018).

Toisaalta Sciancalepore ym. 2017 tutkimuksessa todetaan, että tavallisessa ECC:n toteutuksessa on omat haasteensa. Vaikka ECC:n käyttö on vähemmän vaativaa verrattuna RSA:han, ECC:n kanssa avainten vaihtoon menee merkittävästi kaistaa. Tutkimuksessa ehdotetaankin ECDH:n sijaksi ECQV:tä, jolla saadaan ECC:n käyttämää kaistanleveyttä vähennettyä merkittävästi.

Yksi trendikäs suunta esineiden internetin turvallisuuden parantamiseksi on tällä hetkellä lohkoketjujen käyttö (IoT) laitteiden välisen kommunikaation turvallisuuden parantamiseksi (Hassija ym. 2019). Tutkimuksen Singh, Singh ja Kim 2018 mukaan lohkoketjuja voidaan käyttää IoT laitteissa ainakin tunnistamaan viestien aitous ja varmentamaan laitteiden ohjainohjelmistot. Tässäkin menetelmässä on omat hankaluutensa. Lohkoketjujen ylläpitämiseen tarvittava louhinta on hankalaa toteuttaa pienitehoisilla laitteilla. Näin ollen tarvitaan jokin huomattavasti prosessoinniltaan tehokkaampi välikerros, joka suorittaisi kyseiset operaatiot. Ongelmaksi tulee se, miten IoT laitteet yhdistetään tähän välikerrokseen, sillä jos se ei ole turvallinen, on lohkoketjun tarjoamasta turvallisuudesta huomattavasti vähemmän hyötyä.

2.4.2 Tutkimuksia tietoturvakomponenteista

Tutkimuksessa Taher, Wei ja Yassin 2018 on tutkittu IoT -laitteiden ja pilvipalveluiden (pilviiot) välistä tietoturvaa. Tutkimuksessa todetaan, että aina vuodesta 2013 lähtien monissa tut-

kituissa keinoissa parantaa pilvi-IoT:n tietoturvaa on merkittäviä tietoturva-aukkoja. Tutkimuksessa esitetään toimintamalli, jolla pilven ja IoT -laiteen välinen tunnistautuminen voidaan tehdä turvallisesti. Menetelmässä käytetään ECC:tä myös IoT-laitteen puolella, joka tarkoittaa sitä, että tämänkin menetelmän kanssa on mahdollista käyttää erillistä tietoturvakomponenttia. Tämä menetelmä toimii ilman minkäänlaista välikerrosta.

Tutkimuksessa Durand ym. 2019 käytetään erillistä (sulautettua) tietoturvakomponenttia salaamaan liikenne käyttäen EDHOC:ia ja OSCORE:a. Salattu liikenne välitettiin “IoT -middleware” -välittäjälle, jossa on myös käytetty laitteistotason salausta (intel SGX). CoAP -viestit pyyntöineen ja vastauksineen veivät aikaa kukin 12-16 sekuntia. Tutkimuksessa todettiin myös, että mikrokontrolleri ja tietoturvakomponentti vievät hyvin vähän virtaa. Tavallinen nappiparisto on tarpeeksi varmistamaan 50 000:n tiedonvaihtosession onnistumisen.

3 Tietoturvakomponentit

Kaikki tähän valitut tietoturvakomponentit voivat toimia osana sulautettua järjestelmää. Yhteinen piirre näissä tietoturvakomponenteissa on, että ne tarjoavat “helpon” käyttöönoton osaksi järjestelmää. Niin sanottu “turn-key” -ratkaisu voidaan lisätä järjestelmään jälkikäteen. Tietoturvakomponenttien osalta se tarkoittaa sitä, että sulautetulle järjestelmälle vankempi tietoturva saadaan toteutettua jopa järjestelmän käyttöönoton jälkeen.

3.1 Toimintaperiaate

Satunnaisen salausavaimen luominen on tärkeä osa TLS:ää, jota suurin osa tietoturvakomponenteista tukee ainakin jollain tasolla. Satunnaislukugeneraattoreita on tietoturvakomponenteissa olemassa deterministisiä sekä aitoja. Kuitenkin satunnaisluvun generointiin ei tarvitse käyttää hirvittävästi resursseja, sillä jo tarpeeksi satunnaisesti tehty deterministinen satunnaisluku on tarpeeksi turvallinen salaustarkoituksiin. Tyyppillisesti tietoturvakomponentit yhdistetään isäntä-MCU:hun i2C:llä, SPI:llä tai SWI:llä. Nämä liitännät ovat hyvin yleisiä sulautetuissa järjestelmissä.

3.2 SSL/TLS kirjastot

OpenSSL, mbedTLS ja WolfSSL tls -kirjastot sulautetuille järjestelmille.

mbedTLS ja wolfssl on kumpikin enemmän tarkoitettu matalatehoisille iot-laitteille, joissa kevyt kirjasto on riittävä. OpenSSL on pääosin tehty muuhun tarkoitukseen, mutta käyttäminen varsinkin tehokkaammissa laitteissa onnistuu.

3.3 EAL-luokitus

Useissa tietoturvakomponenteissa mainitaan kyseiselle laitteelle saavutettu EAL-taso. EAL-luokitus (Evaluation Assurance Level) on määritelty standardissa ISO/IEC 15408-3:2008 (ISO 2014).

EAL-luokitukset on numeerinen taso sille, miten tarkasti jonkin järjestelmän tietoturvan toimivuus on testattu. Pelkkä EAL-luokitus ei kuitenkaan tarkoita sitä, että korkeamman tason järjestelmä olisi tietoturvallisempi. Laitteen täytyy ensin toteuttaa kaikki ominaisuudet, jotka edes teoriassa voisivat taata hyvän tietoturvan. Korkeampi EAL-luokitus siis takaa järjestelmien toimivan suuremmalla todennäköisyydellä sillä tavalla, miten niiden tietoturvallisesti kuuluisi.

- EAL1: funktionaalisesti testattu (functionally tested)
- EAL2: rakenteellisesti testattu (structurally tested)
- EAL3: menetelmällisesti testattu ja tarkistettu (methodically tested and checked)
- EAL4: menetelmällisesti suunniteltu, testattu ja katselmoitu (methodically designed, tested and reviewed)
- EAL5: semiformaalisti suunniteltu ja testattu (semiformally designed and tested)
- EAL6: semiformaalisti verifioitu suunnitelma ja testattu (semiformally verified design and tested)
- EAL7: formaalisti verifioitu suunnitelma ja testattu (formally verified design and tested)

3.4 Microchip ATECC608A

ATECC608A on Microchip:n kehittämä erillinen tietoturvakomponentti (SE). Laite on pääosin tarkoitettu oheislaitteiden ja verkon autentikointiin. Laitteen yksi käyttökohde on IoT -laitteiden verkon liikenteen salaaminen, laitteen käyttöjärjestelmän eheyden tarkistaminen, laitteen todentaminen ja turvallinen ohjainohjelmiston päivittäminen. Verkon laitteiden autentikointia varten laite tukee TLS 1.2:a ja TLS 1.3:a. Turvallisen käynnistämisen on tarkoitus toimia laskemalla tiiviste käytettävästä ohjainohjelmistosta. Viestien salaamista varten laite tukee AES:ää “pienten” viestien salaamiseen ja purkamiseen (Microchip 2021).

ATECC608A voidaan integroida laitteeseen jo valmistusvaiheessa, tai jälkikäteen melko helposti päivittämällä. Yksi tämän laitteen ominaisuuksista on, että käyttöönoton pitäisi olla todella helppoa, jolloin “hyvä” tietoturva pilvi-Iot:ssä saavutettaisiin mahdollisimman pienellä vaivalla.

3.4.1 Laitteen ominaisuudet

Tuetut salausalgoritmit:

- ECDH FIPS SP800-56A
- NIST P256 elliptinen käyrä
- SHA-256
- AES-128
- TLS 1.2 ja TLS 1.3 tuki (PRF/HKDF)
- ECDSA FIPS186-3

3.5 NXP EdgeLock SE050

NXP:n valmistama SE050 on IoT-laitteiden suojaamiseen tarkoitettu tietoturvakomponentti (SE). Tämä tietoturvakomponentti on suunniteltu seuraaviin käyttötarkoituksiin: turvalliseen yhdistämiseen yksityiseen tai julkiseen pilveen tai muuhun infrastruktuuriin, toisten “edge” -laitteiden kanssa tapahtuvaan tunnistautumiseen, datan salaamiseen, salausavaimien tallentamiseen ja ekosysteemin turvaamiseen. Laite on yritetty tehdä mahdollisimman helposti otettavaksi käyttöön jo olemassa olevan laitteiston kanssa. Laite on EAL6+ -luokiteltu.

3.5.1 Laitteen ominaisuudet

Laite tukee seuraavia salausalgoritmeja:

- ECC: ECDS, ECDH, ECDHE, ECDAA, EdDSA
- HMAC, CMAC
- SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
- AES-128, AES-192, AES-256
- 3DES
- RSA =< 4096

Tuetut ECC-käyrät:

- NIST 192-521 bit

- Brainpool 160-512 bit
- Twisted Edwards Ed25519/ Montgomery Curve25519
- Koblitz 192-256 bit
- Barreto-Naehrig Curve 256 bit

3.6 Infineon Optiga Trust M

Infineon Optiga Trust M on suunniteltu helposti käyttöön otettavaksi IoT -sovelluksiin teollisuuden automaatioon, kuluttajatuotteisiin ja kodin automaatioon. Laite tukee useita eri salausalgoritmeja viestien salaamiseen ja eheyden varmistamiseen. Laitteessa on alustettuna uniikki X.509 sertifikaatti, jonka avulla TLS:n käyttö pilven kanssa olisi helppoa. Laite on EAL6+ -luokiteltu.

3.6.1 Laitteen ominaisuudet

Tuetut Salausalgoritmit:

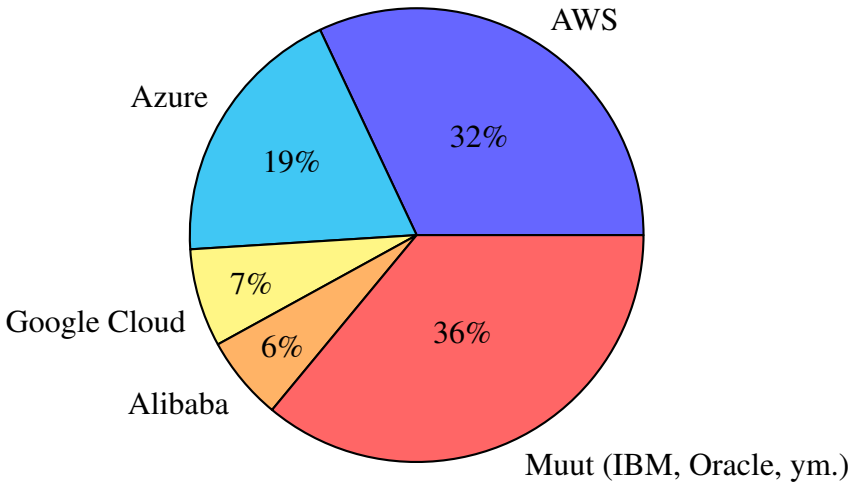
- RSA 1024-2048
- PRF, HKDF
- SHA-256, SHA-384, SHA-512
- AES-128, AES-192, AES-256
- FIPS 186-3

ECC:

- NIST P256, P384, P521
- Brainpool P256, P384, P512 r1

4 Pilviteknologiat

Nykyisin pilvipalveluiden suosion myötä monet eri tahot tarjoavat pilvipalveluita, mutta viime aikoina kolme tai neljä suurinta palveluntarjoajaa on alkanut erottua muusta tarjonnasta. Amazon AWS ja Microsoft Azure ovat syksyllä 2020 ylivoimaisesti suurimmat pilvipalveluiden tarjoajat.



Synergy, 2020 syksy, canalys

4.1 Pilvipalvelut

Käsitteellä pilvipalvelu tarkoitetaan palvelun tuottamista palveluntarjoajan palvelimilla. Tämä tarkoittaa sitä, että tiedon tallennus, prosessointi tai käsittely voidaan tehdä fyysisesti jossain toisessa sijainnissa ilman, että resursseja tarvitsee käyttää fyysisiin tietokoneisiin. Myöskään pilvipalveluntarjoajalta ei tarvitse käyttötarpeesta riippuen vuokrata kokonaista tietokonetta, vaan prosessointiresursseista voi maksaa erittäin joustavasti sekä tarvittavia resursseja voidaan skaalata käytön mukaan.

Pilvipalveluita on lukuisia erityyppisiä. Tyypillisimmin käytettyjä ovat: “infrastructure as a service”, “platform as a service” ja “software as a service” (Höfer ja Karagiannis 2011).

“Infrastructure as a service” on palvelu, jossa pilvipalveluntarjoaja tarjoaa resursseja, joiden avulla voidaan pilvipalvelussa ajaa haluttuja ohjelmistoja. Tyypillisiä IaaS-palveluita ovat

virtuaalikoneet, joilla on palveluntarjoajasta riippuva määrä tallennustilaa, prosessointitehoa ja verkkoyhteyksiä.

“Platform-as-a-service” tarjoaa käyttäjälle valmiin ympäristön ajaa ohjelmia. Tämä on IaaS:n verrattuna abstraktimpi taso, sillä tässä tilanteessa pilvipalveluntarjoaja tarjoaa kaikki resurssit laitteistotasosta käyttöympäristöön. Tämä tarkoittaa yleensä sitä, että esimerkiksi sovelluskehittäjän ei tarvitse huolehtia käytettävästä kehitysympäristöstä.

“Software-as-a-service” tarjoaa käyttäjälle valmiiksi käytettävissä olevan ohjelmiston, esimerkiksi sähköpostiohjelman tai tekstinkäsittelyohjelman.

4.2 IoT laitteet ja pilvet

IoT -laitteet ovat vuosien saatossa yleistyneet merkittävästi. Suurimpia käyttökohteita pilvi-IoT:lle ovat mm. terveyspalvelut, “smart cities” ja kodin automaatio. Koska IoT -laitteet ovat suorituskyvyltään hyvin rajoittuneita, on pilvipalveluiden käytöstä tässä kontekstissa merkittäviä hyötyjä. Näitä hyötyjä ovat mm. resurssien skaalautuvuus, allokointi kysynnän mukaan sekä mistä tahansa käsin käytössä oleva tallennustila (Almolhis ym. 2020).

Hassija ym. 2019 tutkimuksessa mainitaan, että IoT:n käyttökohteet ovat koko ajan lisääntymässä ja sitä otetaan jatkuvasti käyttöön uusiin tarkoituksiin. Tutkimuksessa listataan erityisesti turvallisuuskriittisiä käyttökohteita, joita ovat: älykaupungit, äly-ympäristöt, älysähköverkot, turvallisuus, jälleenmyynti, maanviljely ja kodin automaatio. Vaikka tutkimuksessa ei erikseen mainita pilviä tässä yhteydessä, ei kuitenkaan ole mitään estettä, etteikö pilvipalveluita voisi käyttää myös näiden käyttökohteiden tuottaman datan prosessointiin.

Pilvipalveluita hyödyntävät IoT-järjestelmät käyttävät yleensä hyvin laaja-alaisesti eri pilvipalvelutyyppejä. Laitteen yhdistämiseen tarvitaan jonkinlainen verkkopalveluita tarjoava kerros ja lisäksi jokin SaaS tai PaaS palvelu tiedon varastointia ja prosessointia varten.

IoT -laitteiden yhdistämiseen pilveen Amazon tarjoaa AWS IoT -core -palvelua. Tällä palvelulla Iot -laitteita voidaan yhdistää välittämään viestejä turvallisesti toisilleen tai muihin Amazon:in palveluihin. IoT-core tukee SE:n yhdistämistä käytettävään IoT -laitteeseen, jotta sen eheys ja alkuperä voidaan taata. Tässä palvelussa voidaan käyttää omaa CA:ta tai

AWS:n omaa toteutusta (Amazon 2021).

Microsoftin vastaava tuote IoT:tä varten on Azure IoT hub. IoT hub tarjoaa helpon laitteen yhdistämisen pilveen. Palveluun tukee suoraan yhdistetyn laitteen päivittämistä palvelusta käsin. Microsoft Azure IoTHub tukee monia eri kommunikaatioprotokollia kuten MQTT:tä, HTTPS:ää ja AMQP:tä (Microsoft 2021b).

Googlen IoT core tarjoaa suurilta osin samat ominaisuudet kuten Microsoftin ja Amazonin pilvipalvelut: Sertifikaattien käytön laitteen tunnistamisessa, useita kommunikaatioprotokollia ja laitteen ohjainohjelmiston turvallisen päivittämisen (Google 2021).

4.3 Pilvipalveluiden turvallisuus

IoT-laitteiden sekä pilvipalveluiden ja niiden välisen yhteyden turvallisuus on vain osa kokonaisvaltaisesta turvallisuudesta. Itse pilvipalvelun on myös oltava turvallinen ja luotettava. Pilvipalveluiden turvallisuutta on vertailtu Li ym. (2019) tutkimuksessa, jossa esitetään yksi tapa vertailla näiden turvallisuutta.

Yksi pilvipalveluihin liittyvä riskitekijä on muut käyttäjät. Pilvipalvelut on yleensä toteutettu virtuaalikoneilla, jolloin toisella pilvipalvelu käyttäjällä voi joissain tapauksissa olla mahdollista päästä käsiksi virtuaalikoneeseen, jossa käsitellään IoT -laitteista saatua dataa (Almolhis ym. 2020).

Koska data siirretään sulautetusta järjestelmästä pilveen verkon kautta, on olemassa riski, että jokin muu taho voi päästä dataan käsiksi tässä vaiheessa. Itse pilvipalvelun käyttäjä ei saa dataa käyttöön, kun vasta siinä vaiheessa, kun se on siirretty pilvipalveluun ja prosessoitu. Toinen riski datan siirtämisessä pilvipalveluun on siinä vaiheessa, kun datan jalostaminen ja prosessointi tapahtuu pilvessä (Almolhis ym. 2020).

Tutkimuksessa Zeeshan, Reed ja Siddiqui 2019 esitetään (hieman) turvallisempi malli pilvi-iot:n toteutukseen. Kaikki suoraan IoT-laitteiden kanssa tapahtuva liikenne reititetään pilvipalvelun kautta siten, että laitteeseen ei ole pääsyä muuta kautta. Tässä tilanteessa, jos iot-laitteen ja pilven välinen yhteys on turvallinen, niin ulkopuolelta siihen käsiksi pääsystä tulee huomattavan haasteellista.

4.3.1 Protokollat

MQTT on tehty laitteiden väliseen viestintään. MQTT toimii “publish/subscribe” -periaatteella, eli lähetettyä dataa julkaistaan välittäjälle, josta käyttäjät (client) voivat tilata jonkin lähettäjäryhmän viestejä. MQTT käyttää kuljetuskerroksena TLS:ää UDP:n sijasta. MQTT on hyvin vähän resursseja vaativa protokolla (Naik 2017).

HTTP on pääosin verkon viestitysprotokolla. Toisin kuin MQTT, HTTP käyttää toiminnassaan “request/response” -mallia, eli esimerkiksi käyttäjä voi lähettää verkkosivulle HTTP GET -pyynnön, johon palvelin myös vastaa. HTTP käyttää “topic”:n sijasta URI:a (Universal resource identifier (Naik 2017).

AMQP toimii kummallakin tavalla: “publish/subscribe” ja “request/response”. AMQP muistuttaa siltä osin MQTT:tä, että viestit eivät mene suoraan vastaanottajalle. AMQP:n tapauksessa viestit menevät lähettäjältä ensin viestin välittäjälle (exchange), josta sitten vastaanottajan viestijonoon (Uy ja Nam 2019).

CoAP käyttää HTTP:n tapaan URI:a viestien lähettämiseen, julkaisuun, vastaanottoon ja tilaamiseen. HTTP:stä poiketen CoAP:n avulla URI:n pystyy myös tilaamaan, jolloin siihen lähetetyt viestit tulevat sen tilanneelle. CoAP on suunniteltu alusta pitäen erittäin kevyeksi viestitysprotokollaksi (Naik 2017).

Turvallisuuden kannalta kaikki edellä mainitut protokollat ovat käyttökelpoisia useimmissa käyttötarkoituksessa sillä ne tukevat TLS:ää. AMQP kuitenkin tarjoaa kuitenkin kaikista parhaimman yhteyden suojauksen, kuten tutkimuksessa Naik 2017 mainitaan. Tämä kuitenkin tarkoittaa sitä, että käyttöönotto on monimutkaisempaa. MQTT ja CoAP tarjoavat hyvin vähän muita ominaisuuksia, kuten AMQP:n eri TLS-vaihtoehdot (Naik 2017). Tutkimuksessa Naik 2017 osoittautuu, että lähetettyjen viestien koko, laitteelta vaaditut resurssit, viive ja kaistanleveys menee näillä eri protokollilla samalla tavalla. Vaativin, hitain, ja protokolla suurimmalla viestin kokovaatimuksella on HTTP. HTTP:n jälkeen nopeus ja edullisuusjärjestys on: AMQP, MQTT ja CoAP. Näistä kevyin protokolla CoAP onkin hyvin soveltuva todella vähäresurssisiin IoT -laitteisiin. Muuten MQTT ja AMQP ovat ominaisuuksiltaan hieman edistyneempiä ja ovat suurelta osin käytössä IoT -sovelluksissa.

5 Tutkimus

Tässä tutkimuksessa vertailtavat laitteet on mainittu luvussa 3.

Tässä tutkielmassa vertaillaan kolmen eri laitevalmistajan tietoturvakomponentteja. NXP:n SE050, Microchip:n ATECC608A ja Infineon Optiga Trust M. Tutkimus koostuu erinäisistä mittauksista laitteiden suorituskykyyn liittyen, sekä prosessiin, miten kukin laite yhdistetään pilvipalveluun.

Tutkielmassa vertaillaan eri laitteiden välillä matalalla tasolla avaimen generointia ja "signature":n (allekirjoituksen) luontia, sillä nämä ovat yhdet oleellisimmat vaiheet TLS:n käytössä. Lisäksi vertaillaan pakettikaappauksista (Laite -> pilvi) saatua dataa havainnollistamaan laitteen koko toiminnallisuutta.

Nämä laitteet on valittu tähän tutkimukseen sillä perusteella, että ne ovat ominaisuuksiensa perusteella hyvin lähellä toisiaan.

5.1 Tutkimusmenetelmä

Tässä tutkimuksessa käytetään konstruktiivista tutkimusmenetelmää, joka tässä tapauksessa tarkoittaa useamman tietoturvakomponentin ominaisuuksien vertailua. Tässä tutkimuksessa vertailtavat asiat tietoturvakomponenteissa ovat ECC- ja RSA-operaatioiden suoritusaika sekä laitteen konfiguraatio ja suorituskyky yhdistettäessä pilvipalveluun. Tietoturvakomponenttien ominaisuuksista ECC ja RSA-operaatioiden suoritusaika on yksiselitteistä mitata määrällisesti.

Määrällinen tutkimusmenetelmä on tähän tutkimukseen varsin sopiva, sillä suoritusaika on helposti ymmärrettävissä, sekä se on yksi tärkeimmistä ominaisuuksista tietoturvakomponentissa tietoturvan lisäksi.

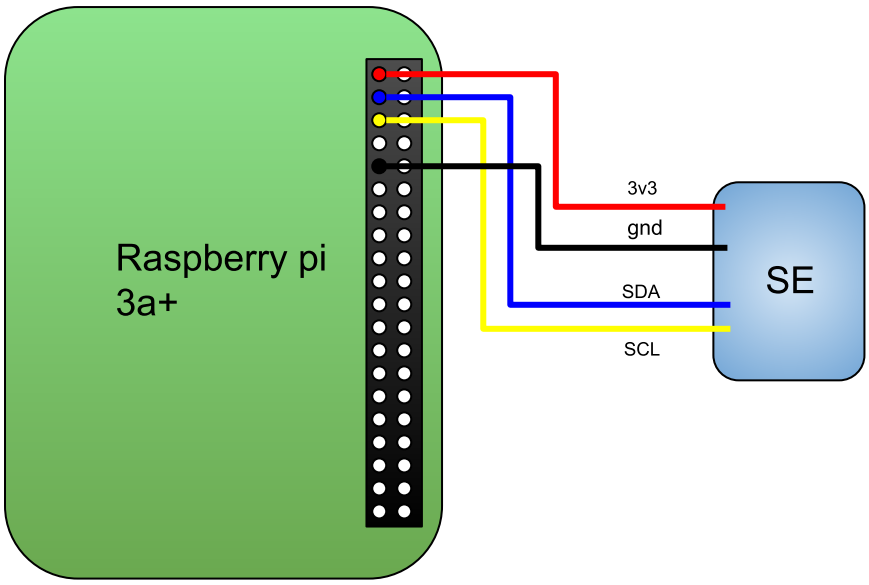
5.2 Tutkimusympäristö

5.2.1 Käyttöympäristö

IoT -laitteena on tässä tutkimuksessa käytetty Raspberry pi 3a+:aa. Raspberry pi on tähän tutkimukseen hyvin soveltuva sen muokattavuuden ansiosta. Myös tutkimuksissa, kuten Schläpfer ja Rüst (2019), Gyamfi, Ansere ja Xu (2019) ja Suárez-Albela ym. (2018) on käytetty samanlaista asetelmaa, jossa tietoturvakomponentti on liitetty osaksi olemassa olevaa järjestelmää.

Kaikki vertailtavat tietoturvakomponentit tukevat i2c tiedonsiirtoväylää, jonka ansiosta kaikkia laitteita on mahdollista vertailla samalla järjestelyllä. Raspberry pi:n käyttöjärjestelmänä on käytetty tässä tutkimuksessa Raspbian linux:n versiota 10. Linux kernel 5.4.51-v7+. Testiympäristön Raspberry pi 3a+ on konfiguroitu siten, että i2c -väylän käyttö on mahdollista. I2C -väylän saa otettua käyttöön seuraavalla tavalla:

- `sudo raspi-config`
  - interface options
  - I2C
  - Enable



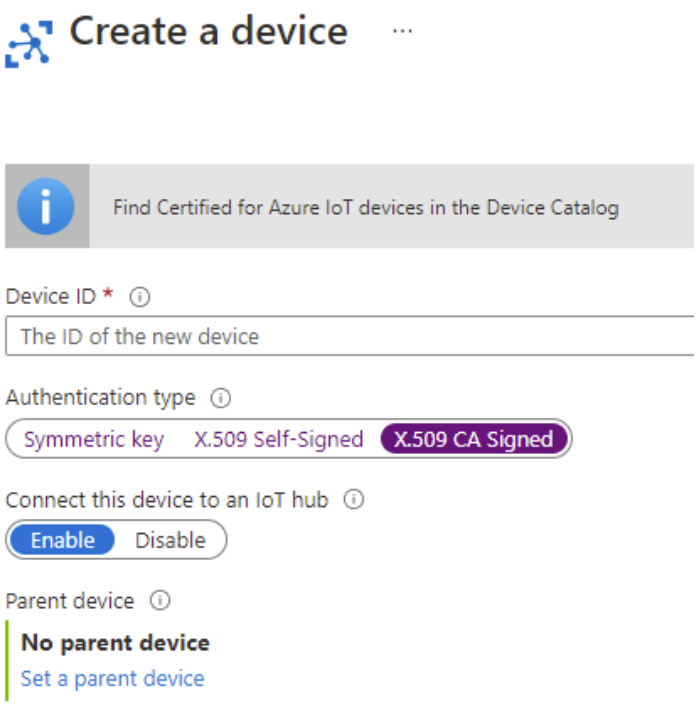
Kuvio 3: Raspberry pi:n yhdistys tietoturvaelementtiin

5.2.2 Pilvipalvelu, Azure iothub

Azure on Microsoftin tarjoama pilvipalvelu, jossa on muiden pilvipalveluntarjoajien tapaan lukuisia eri ominaisuuksia. Tässä tutkimuksessa kuitenkin käytetään vain Azuren IoTHub -palvelua, joka mahdollistaa iot -laitteen yhdistämiseen kyseiseen pilvipalveluun. Yhdistämisessä käytetään tässä tapauksessa x509 -sertifikaatteja, eikä “Connection String”:iä, jotta salausalgoritmeja voidaan käyttää ja tietoturvakomponenttien eri ominaisuudet tulevat esiin.

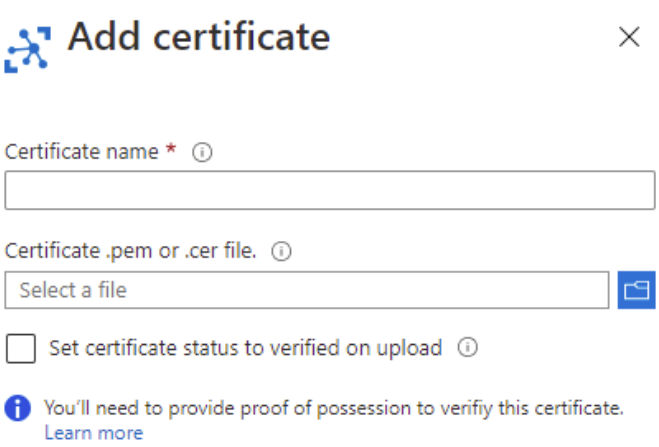
Uuden laitteen lisääminen IotHub:iin on erittäin yksinkertaista. Device ID:ksi voi valita sellaisen nimen, jolla haluaa kyseisen laitteen olevan tunnistettavissa. Seuraava valinta on “Authentication type” joksi valitaan tässä tapauksessa “X.509 CA signed”

Kuvio 4: Laitteen lisääminen IoTHub:iin.



X509 sertifikaatin Azureen tapahtuu kuvassa 5 osoitetulla tavalla. Sertifikaatin omistuksen voi jo varmentaa tässä näkymässä, tai osoittaa omistus thumbprint:llä. Thumbprint -varmennus tapahtuu siten, että alkuperäisen CA:n luojalla on hallussaan private key, jolla voi osoittaa, että sertifikaatti on luotu juuri sillä avaimella.

Kuvio 5: Sertifikaatin lisääminen IoTHub:iin.



5.3 Laitteiden käyttöönotto

Tässä kappaleessa kuvaillaan, miten kukin tietoturvakomponentti on otettu käyttöön ja mitä väliohjelmistoa ja sen versiota on käytetty.

5.3.1 NXP SE050

Kuten muutkin tässä tutkielmassa testattavat tietoturvakomponentit, NXP SE050 on mahdollista yhdistää isäntälaitteeseen i2c -väylää käyttäen. Tässä tapauksessa laitteeseen yhdistetään 3v3 -virtajohto, maa, SDA- ja SDL-liitännät kuvion 3 mukaisesti. Väliohjelmistoksi on SE050:n tapauksessa kaksi vaihtoehtoa: GitHub:sta löytyvä pelkistetty versio, tai NXP:n sivuilta löytyvä niin ikään suurimmaksi osaksi vapaan lähdekoodin versio, jota myös tässä tutkielmassa on käytetty (NXP 2021). Tässä tutkielmassa on käytetty väliohjelman versiota v03.01.00. Vain NXP:n sivuilta saatavissa oleva väliohjelmisto sisältää OpenSSL -toteutuksen. NXP:n laitteelle löytyy myös python -toteutus väliohjelmistosta.

Väliohjelmiston asennus on suoritettu seuraavalla tavalla raspberry pi:llä:

```
cd /home/pi/se_mw/simw-top/scripts
python create_cmake_projects.py
cd /home/pi/se_mw/simw-top_build/raspbian_native_se050_t1oi2c
ccmake .
```

```
cmake build .
sudo make install
```

(NXP 2021)

5.3.2 Infineon Optiga Trust M

Infineon tarjoaa ohjelmavaraston, jossa on tarvittavan väliohjelmiston lisäksi esimerkkejä linux -alustalle, sekä OpenSSL engine -toteutus. OpenSSL engine:iden toteutusta vertaillaan luvussa 5.4.1. Ennen kirjaston kääntämistä on syytä muokata tiedostoa "optiga-trust-m/optiga/include/optiga/optiga_lib_config_m_v1.h", jotta laitteen resetointia ja uudelleenkäynnistystä varten ei tarvitse lisätä laitteeseen muita johtoja 3v3, GND, SDA ja SDL:n lisäksi. Tässä tutkielmassa linux-optiga-trust-m väliohjelmistosta on käytetty haaraa "development_v3".

```
optiga_lib_config_m_v1.h

#define OPTIGA_COMMS_DEFAULT_RESET_TYPE      (1U)
```

Tämän jälkeen kirjasto kääntyy yksinkertaisesti komennoilla

```
make
sudo make install
```

(Infineon 2021)

5.3.3 Microchip ATECC608A

Microchip:n ATECC608:n kanssa on tässä tutkielmassa käytetty linux-ympäristössä toimivaa cryptoauthlib:n versiota v3.3.0 (Microchip 2021). Kirjaston käyttöohjeet ovat kuitenkin huonot, joten käyttäjän on itse pääteltävä miten kirjaston saa käännettyä ja tehtyä tarvittavat muutokset, jotta se toimisi.

Seuraava cmake -komento on ajettava ennen kirjaston kääntämistä, jotta se toimisi tämän tutkielman mukaisessa käyttötarkoituksessa:

```
cmake -D ATCA_HAL_I2C=ON -D ATCA_PKCS11=ON -D ATCA_OPENSSL=ON
-D ATCA_ATECC508A_SUPPORT=ON -D ATCA_ATECC608_SUPPORT=ON
-D ATCA_BUILD_SHARED_LIBS=ON -D ATCA_USE_ATCAB_FUNCTIONS=ON
```

Tässä tutkielmassa käytössä ollut DM320118 sisältää kolme ATECC608A -piiriä, joista yksi on valmiiksi konfiguroitu ja kaksi muuta on muokattavissa. Kullakin kolmella piirillä on oma i2c-osoite. Dokumentaatiossa ei kuitenkaan mainita, mikä kolmesta i2c-osoitteesta on mikäkin. Tämänkin käyttäjä saa päätellä itse. TrustFLEX -piirin muokkauksen käyttäjä voi tehdä itse, mutta kolmannen piirin: TrustCUSTOM:n muokkausta ja alustusta ei voi tehdä avoimen lähdekoodin sovelluksilla. Piirien muokkauksen voi tehdä cryptoauthlib:llä, mutta siihen ei ole juurikaan dokumentaatiota. Muokkauksen voi myös tehdä Microchip:n toisella sovelluksella: “Trust Platform Design Suite V2”.

5.4 Avaimen generointi ja signature

Jokaisen laitevalmistajan väliohjelmistoissa on luonnollisesti eroja, mutta siitä huolimatta tässä tutkielmassa laitteita on pyritty vertailemaan mahdollisimman tasapuolisesti. Esimerkiksi NXP:n toteutuksen kanssa “key store”:n avain on joka kerta poistettava, jotta uuden voi luoda samaan osoitteeseen. Aika on otettu kustakin operaatiosta kymmenen kertaa. Alla olevissa C-koodeissa ajan otto on toteutettu siten, että aika otetaan ohjelman sisällä olennaisimmista funktioista, eli juuri niistä, jotka liittyvät avaimen generointiin ja allekirjoitukseen (signature). Tämän tapaista ajan mittausta ohjelmassa kutsutaan CPU-ajaksi.

Seuraavassa osiossa on otteita ECC-salausavaimen generointiin käytetyistä C-koodeista jokaisesta laitteesta.

Microchip: avaimen generointi:

```
clock_t t = clock();
status = atcab_genkey(slot, &public_key);

t = clock() - t;
double gen_time = ((double)t)/CLOCKS_PER_SEC;
```

NXP: avaimen generointi:

```
time_t t;
t = clock();
status = sss_key_store_erase_key(&pCtx->ks, &keyPair);
status = sss_key_store_generate_key(&pCtx->ks, &keyPair,
   EC_KEY_BIT_LEN, 0);

ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);
t = clock() - t;
double gen_time = ((double)t)/CLOCKS_PER_SEC;
```

Infineon: avaimen generointi:

```
optiga_lib_status = OPTIGA_LIB_BUSY;
clock_t t = clock();
return_status = optiga_crypt_ecc_generate_keypair(me_crypt, keySize,
   keyType, FALSE, &optiga_key_id, (pubKey+i), &pubKeyLen);
trustm_WaitForCompletion(BUSY_WAIT_TIME_OUT);
t = clock()- t;
double gen_time = ((double)t)/CLOCKS_PER_SEC;
```

Signaturen tuotto on jokaisella laitteella toteutettu seuraavalla tavalla:

1. Alustetaan viesti, josta muodostetaan kooste
2. Joko SHA256 tai SHA384 koosteen (digest) tuotto
3. Allekirjoituksen (signature) muodostaminen

Jokaisen laitteen kanssa on käytetty samaa merkkijonoa koosteen luomiseen:

```
''temperature=6\ntestvariable=false\nserial=234j-f399-34jl-pp34\n
   another_variable=0x3442d\ntime=16:40\nmessage_received=true''
```

Microchip: allekirjoituksen (signature) luonti:

```
clock_t t = clock();
status = atcab_hw_sha2_256(&message, messageLen, &digest);
status = atcab_sign(slot, &digest, &signature);
t = clock() - t;
```

```
double sign_time = ((double)t)/CLOCKS_PER_SEC;
```

NXP signature:

```
time_t signt;
signt = clock();
memset(signature_2, 0, 256);
status = sss_digest_context_init(&digest_ctx, &pCtx->host_session,
    digest256_algorithm, kMode_SSS_Digest);
ENSURE_OR_GO_CLEANUP(status = kStatus_SSS_Success);
digest_256Len = sizeof(digest_256);
status = sss_digest_one_go(&digest_ctx, message, messageLen,
    digest_256, &digest_256Len);

status = sss_asymmetric_context_init(&ctx_asymm, &pCtx->session,
    &keyPair, kAlgorithm_SSS_SHA256, kMode_SSS_Sign);
ENSURE_OR_GO_CLEANUP(status == kStatus_SSS_Success);
signt = clock() - signt;
double sign_time = ((double)signt)/CLOCKS_PER_SEC;
```

Infineon signature:

```
clock_t t = clock();
SHA256_CTX sha256;
SHA256_Init(&sha256);
SHA256_Update(&sha256, message, messageLen);
SHA256_Final(digest, &sha256);
digestLen = sizeof(digest);
optiga_lib_status = OPTIGA_LIB_BUSY;
return_status = optiga_crypt_ecdsa_sign(me_crypt,
                                        digest,
                                        digestLen,
                                        optiga_key_id,
                                        signature,
                                        &signature_length);

trustm_WaitForCompletion(BUSY_WAIT_TIME_OUT);
return_status = optiga_lib_status;
```

```
t = clock() - t;
double sign_time = ((double)t) / CLOCKS_PER_SEC;
```

5.4.1 Pilvipalveluun yhdistämisen prosessi

Pilvipalveluun yhdistämisessä on olennaista, että laitteessa on olemassa jokin SSL/TLS toteutus. Näitä ovat mm. OpenSSL, mbedTLS ja wolfSSL. Kaikille tässä tutkielmassa vertailtavana oleville tietoturvakomponenteille on olemassa valmistajan puolesta valmis OpenSSL engine -toteutus, sekä osalle laitteista myös mbedTLS ja/tai wolfSSL toteutus. Tässä tutkielmassa käytetään kaikkien laitteiden kanssa OpenSSL:ää edellä mainitusta syystä.

NXP:n tietoturvakomponentille on suorat ja yksiselitteiset ohjeet Azure:n IoTHub:iin yhdistämiseen. Tämä prosessi on seuraavanlainen:

NXP:n tietoturvakomponentille on olemassa valmis python -skripti (ResetAndUpdate_AZURE.py, joka alustaa laitteen esimerkkidatalla.

- Tässä private key luodaan ja asetetaan laitteen suojattuun muistiin (tässä tapauksessa osoitteeseen 0x223344).
- Laitteeseen asetetaan osoitteeseen 0x223345 valmiiksi luotu laitteen tunnistamista varten oleva sertifikaatti, joka on allekirjoitettu NXP:n esimerkki-CA:n sertifikaatilla.
- Osoitteen 0x223344 private key:hin luodaan “reference private key”, joka tallennetaan myös isäntälaitteelle (RPi)

NXP:n “reference key” on muodoltaan täysin samanlainen kuin tavallinen yksityinen salausavain, paitsi että se toimii vain ja ainoastaan osoittimena NXP:n OpenSSL engine:lle varsinaiseen salausavaimeen, joka on tallennettuna itse tietoturvakomponenttiin. Tämän muotoinen salausavain käy myös suoraan liitteen A koodiin kohtaan “x509_private_key”. Tämän esimerkin referenssiavain on muotoa:

```
-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIBAAAAAAAAAAAAAAAAAAAAAAAAAiM0SlprW2paa1thAAoAoGCCqGSM49
AwEHoUQDQgAETvQbbZMknDCBgKJOSuxObDWu7cKIAp/9ATemYD1CnKVm5mDM1Yh2
AMBbNOb/1EH6nk8AUyMrbMRHYezqgnVl4A==
-----END EC PRIVATE KEY-----
```

Tämän salausavaimen toimimattomuuden normaalitilanteessa voi todeta komennolla:

```
openssl ec -in device_reference_key.pem -check
read EC key
EC Key Invalid!
1995845648:error:1010207B:elliptic curve
    routines:ec_key_simple_check_key:invalid private
    key:../crypto/ec/ec_key.c:326:
```

Python -skriptin luoma “rootCA_certificate.cer” viedään Azure:en kuvion 5 mukaisella tavalla. Sertifikaatin omistuksen varmennuksen voi halutessaan tehdä tai jättää välistä.

NXP:n SE050:n väliohjelmiston kääntäminen tuottaa OpenSSL engine:n ja kaiken siihen tarvittavan.

Liitteenä olevan Microsoftin “azure iot sdk”:n iothub_ll_client_x509 toimii SE050:n kanssa, kun “SAMPLE_OPENSSL_ENGINE”:ksi on määritelty “e4sss” tai ympäristömuuttujaksi asetettu SE050:n OpenSSL .cnf -tiedosto:

```
export OPENSSL_CONF=/simw-top/demos/linux/common/openssl11_sss_se050.cnf
```

Infineon Optiga Trust M -laitteen väliohjelmiston mukana asentuu myös tässä tutkielmassa käytettävä OpenSSL -engine (Infineon 2021).

Azure IoTHub:ia varten Infineon-trust-m -laitteella voi suoraan luoda tarvittavat sertifikaatit OpenSSL:llä.

Uuden private key:n (laitteen osoite 0xe0f2) ja CA -sertifikaatin luonti:

```
openssl req -keyform engine -engine trustm_engine -key
    0xe0f2:^:NEW:0x03:0x13 -new -x509 -days 365000 -out CA_e0f2.cer
```

Tällä tavoin laitetta itseään on käytetty luomaan CA-sertifikaatti, joka perustuu laitteella olevaan private key:hin, johon ei itsessään pääse käsiksi. Oikeassa käyttöympäristössä parempi lähestymistapa on saada CA-sertifikaatti joltain luotettavalta taholta ja käyttää sitä suoraan tai luoda “subordinate” CA, jolla voi itse allekirjoittaa käytettävien laitteiden sertifikaatteja.

Laitteen oman sertifikaatin luonti: Infineon 2021, private keyn alustaminen laitteen osoitteeseen 0xe0f3, voi vaihtoehtoisesti tehdä OpenSSL:llä.

```
trustm_ecc_keygen -g 0xe0f3 -t 0x13 -k 0x03 -o device_public_key.pem -s
```

Certificate sign request (csr)

```
openssl req -engine trustm_engine -key ``0xe0f3'' -keyform engine -new
    -out device_cert.csr
```

Sertifikaatin allekirjoitus CA-sertifikaatilla:

```
openssl x509 -req -in device_cert.csr -CA CA_e0f2.cer -CAkey ``0xe0f2:^''
-CAcreateserial -out device_cert.pem -days 500 -sha256 -force_pubkey
    device_public_key.pem
```

Liitteeseen A C-koodiin sertifikaatiksi muutetaan “device_cert.pem” ja private key: `0x0e0f3:^`

Microchip ei muiden laitevalmistajien tavoin tarjoa valmista OpenSSL -engine toteutusta, vaan PKCS#11 engine:ä voi käyttää siten, että kyseinen toteutus käyttää cryptoauthlib:iä.

PKCS11 käyttöönotto lyhyesti, (Microchip 2021).

```
git clone https://github.com/OpenSC/libp11.git
make
make install
```

/var/lib/cryptoauthlib/0.conf

```
# Reserved Configuration for a device
# The objects in this file will be created and marked as undeletable
# These are processed in order. Configuration parameters must be comma
# delimited and may not contain spaces

interface = i2c,0x6a,1
freeslots = 1,2,3

# Slot 0 is the primary private key
object = private,device,0
```

```
# Slot 2 for self CA
object = private,device2,2
```

Tässä tutkielmassa käytetty Microchip ATECC608A piiri on valmiiksi lukittu, eli laitteen suojatut tallennus-"slotit" on lukossa ja niitä ei voi päivittää. Lukittuihin muistiosoitteisiin voi tallentaa yksityisiä salausavaimia ja osaan myös sertifikaatteja. Laitteen slot 0:n salausavain käytetään laitteen omaa sertifikaattia varten, ja slot 2:n salausavainta CA -sertifikaatin luontia varten.

CA-sertifikaatin luonti

```
openssl req -keyform engine -engine pkcs11 -key
    "pkcs11:token=MCHP;object=device2;type=private" -new -x509 -days
    300000 -out slot2CA.cer
```

CSR laitteen sertifikaattia varten:

```
openssl req -engine pkcs11 -key
    "pkcs11:token=0123EE;object=device;type=private" -keyform engine -new
    -out slot0Device.csr
```

CSR:n allekirjoitus CA-sertifikaatilla:

```
openssl x509 -CAkeyform engine -engine pkcs11 -req -in slot0Device.csr
    -CA slot2CA.pem -CAkey
    "pkcs11:token=MCHP;object=device2;type=private" -CAcreateserial -out
    slot0Cert.pem -days 5000 -sha256
```

6 Tulokset

6.1 Vertailtavat ominaisuudet

Tässä tutkielmassa vertailtavat asiat laitteiden välisen suorituskyvyn osalta ovat:

- ECC:n (Secp256r1 ja Secp384r1) avaimen generointi ja allekirjoitus (signature)
- RSA 1024 ja 2048 avaimen generointi ja allekirjoituksen luonti

Muut vertailtavat kohteet:

- Laitteiden OpenSSL konfiguraatioiden erot pilvikontekstissa.
- Pakettikaappausten vertailu pilveen yhdistämisessä salausta käyttäessä.

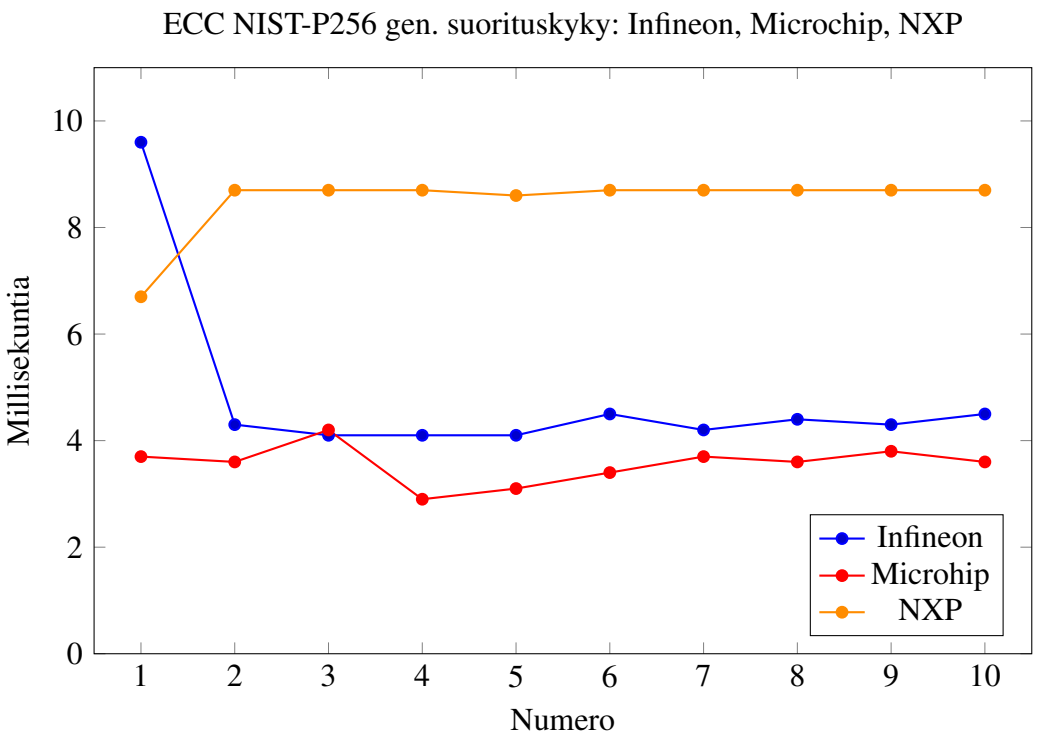
6.1.1 Salausalgoritmien toiminta ja suorituskyky

NXP(NIST-P256/Secp256r1), Microchip(P256), Infineon(Secp256r1) Kuviosta 6 nähdään, että yksittäisen Secp256r1 -avaimen generointi on kaikilla laitteilla erittäin nopeaa. Infineon Optiga Trust M:n keskiarvo 4.79ms ja ilman ensimmäisen koetuloksen poikkeamaa 4.26ms. Microchip:n ATECC608 on keskiarvoltaan 3.55 ms vertailun nopein ja nxp SE050:n keskiarvoltaan hitain 8.49 millisekunnilla.

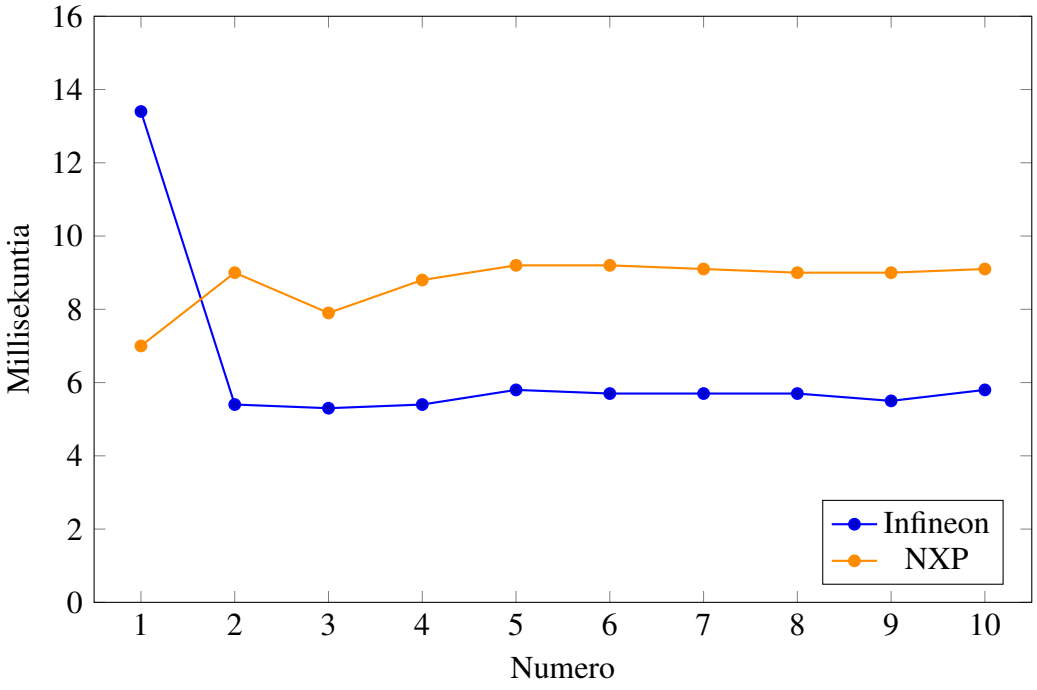
Infineonin ja NXP:n tietoturvakomponenttien välillä on hieman pienempi ero NIST-P384:ssä kuin 256:ssa. Kuvaajasta 6 ja 7 voidaan tulkita, että ero on 2.5 millisekuntia, kun 256:ssa se on 4.2 millisekuntia. Kuvaajissa 6 ja 7 Infineonin ensimmäinen koetulos on selvästi seuraavia hitaampi. On mahdollista, että tämä johtuu testikoodista, sillä laitteiden väliohjelmistot toimivat hieman eri tavoilla. Toinen mahdollisuus on, että Infineonin tietoturvakomponentin muistiosoitteen avaaminen tai varaus kestää ensimmäisellä kerralla pidempään.

Voidaan havaita, että ECC:n avaimen koon kasvattamisella 256:sta 384:ään avaimen generointi ja signaturen luonti hidastuvat hieman. Sama ilmiö on havaittu tutkimuksessa Suárez-Albela ym. (2018), vaikkakin ilman erillistä tietoturvakomponenttia. RSA:n tapauksessa tapahtuu huomattavasti suurempi muutos suorituskyvyssä, kun siirrytään 1024:stä 2048:aan.

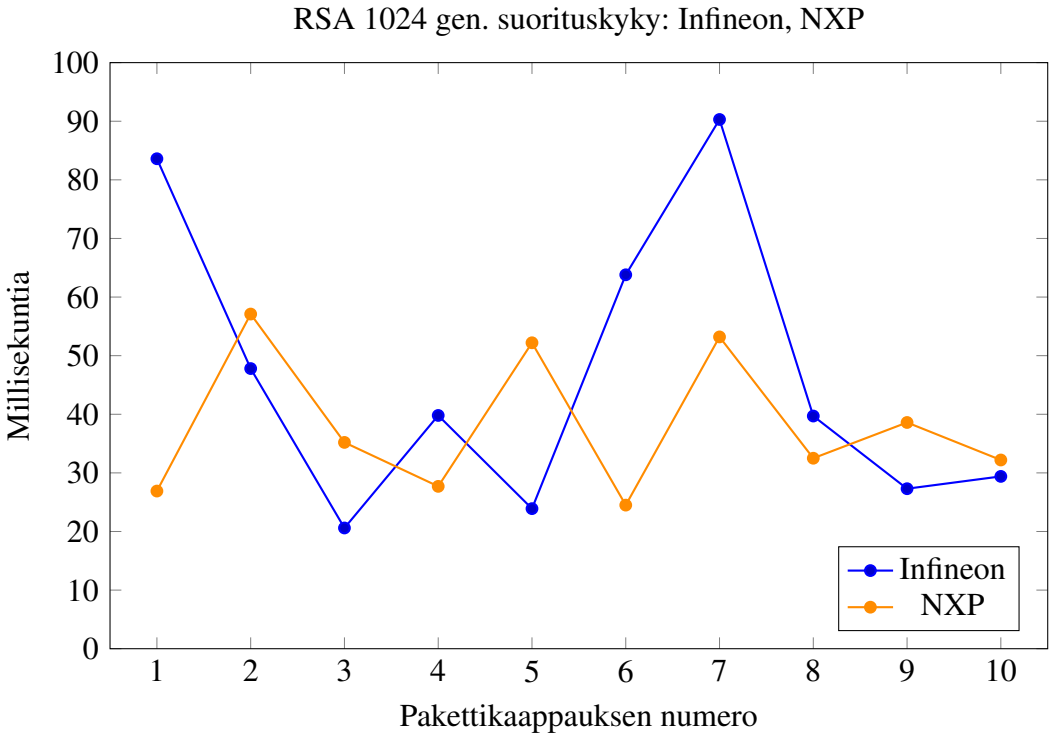
Kuvio 6: ECC NIST-P256 gen. suorituskyky: Infineon, Microchip, NXP



Kuvio 7: ECC NIST-P384 gen. suorituskyky: Infineon, NXP

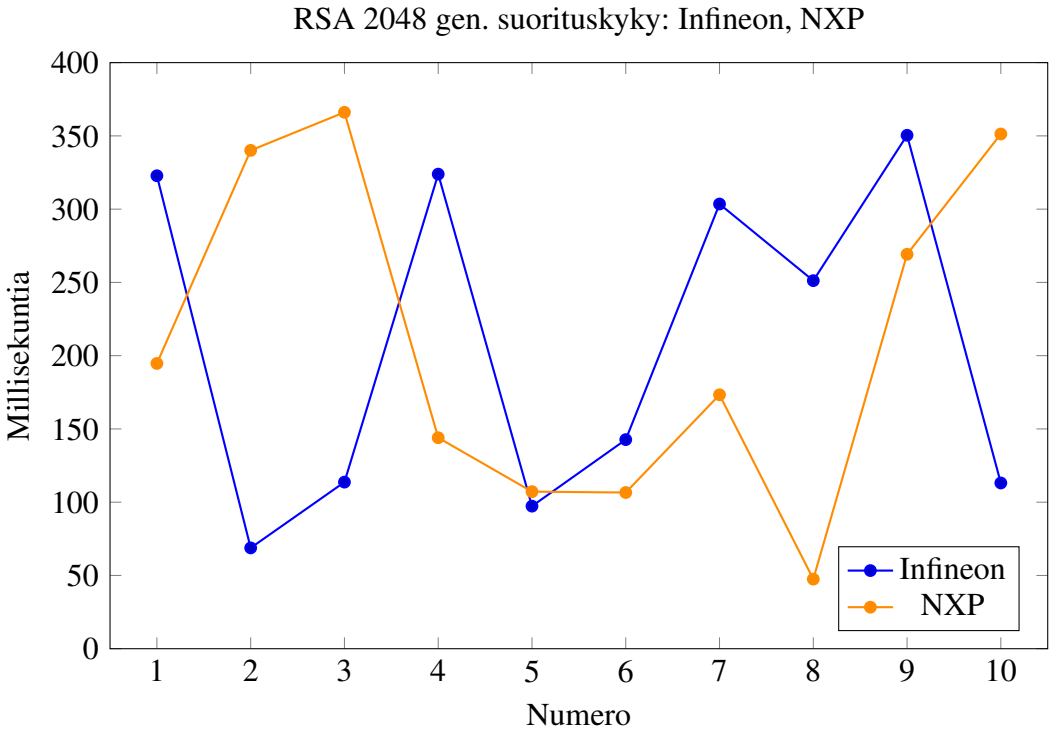


Kuvio 8: RSA 1024 gen. suorituskyky: Infineon, NXP



RSA:ta tukevat Infineonin ja NXP:n laitteet osoittavat, että RSA-tyyppisen salausavainparin luominen on huomattavasti hitaampaa kuin ECC -avainparin. Kuvaajista 6 ja 7 voidaan huomata, että ECC avainparin luominen kestää joka kerta lähes yhtä kauan. RSA avainparin luomisessa on huomattavasti enemmän vaihteluita, joka voidaan havaita kuvaajista 8 ja 9. Tämä johtuu siitä, että RSA-avainparin luomista varten täytyy löytää kaksi suurta alkulukua p ja q, jonka löytymisaika ei ole vakio (Rivest, Shamir ja Adleman 1978).

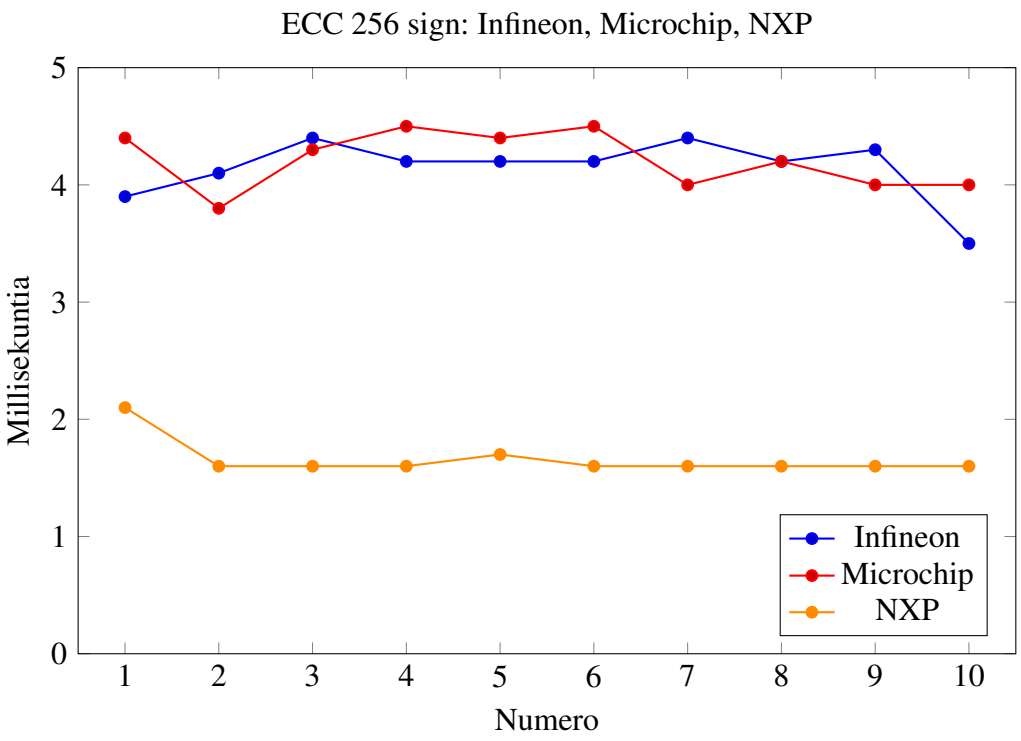
Kuvio 9: RSA 2048 gen. suorituskyky: Infineon, NXP



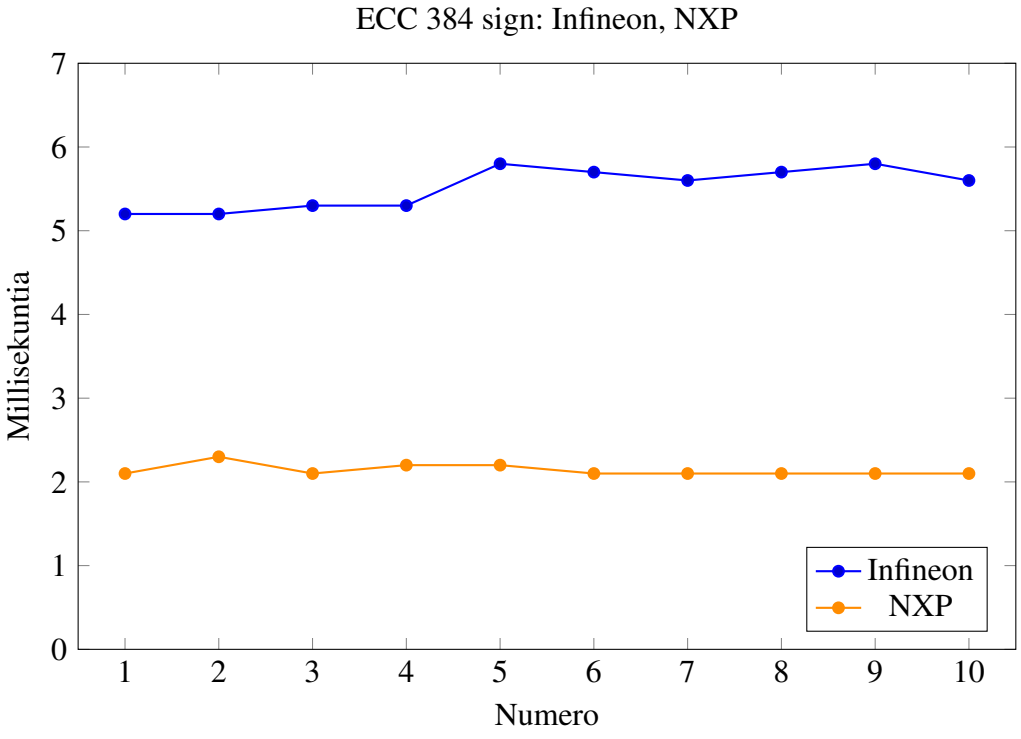
Tämän tutkielman allekirjoitusten (signature) vertailu on kappaleessa 5.4 esitellyllä tavalla lähetettävän viestin kooste, joka allekirjoitetaan salausavaimella. Kuviosta 10 ja 6 voidaan huomata, että Infineon Optiga Trust M ja Microchip ATECC608 ovat lähes yhtä nopeita niin avaimen generoinnissa, kuin viestin koosteen allekirjoittamisessa. Toisin kuin generoinnissa, NXP SE050 on signaturen luonnissa vertailtavista tietoturvakomponenteista hitain.

Kuvioista 12 ja 13 huomataan, että RSA signature ei ajallisesti niin satunnaista, kuin mitä kuvioiden 8 ja 9 kuvaama RSA avaimen generointi. RSA 2048 signaturen luonti on kuitenkin huomattavasti hitaampaa, kuin RSA 1024 signaturen.

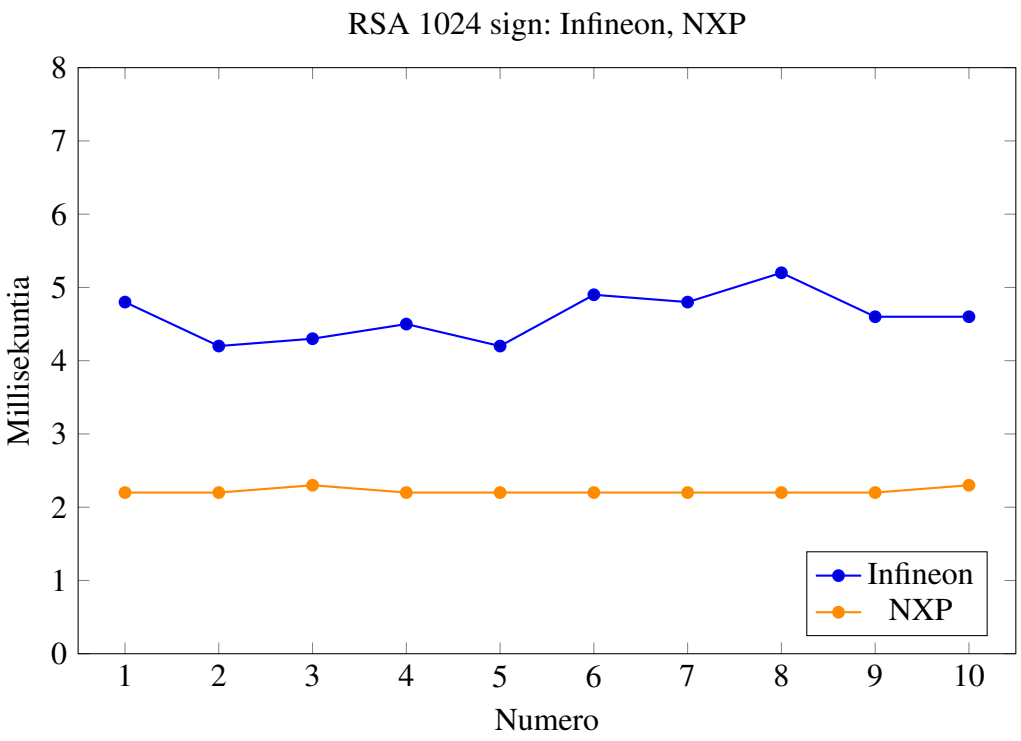
Kuvio 10: ECC 256 sign: Infineon, Microchip, NXP



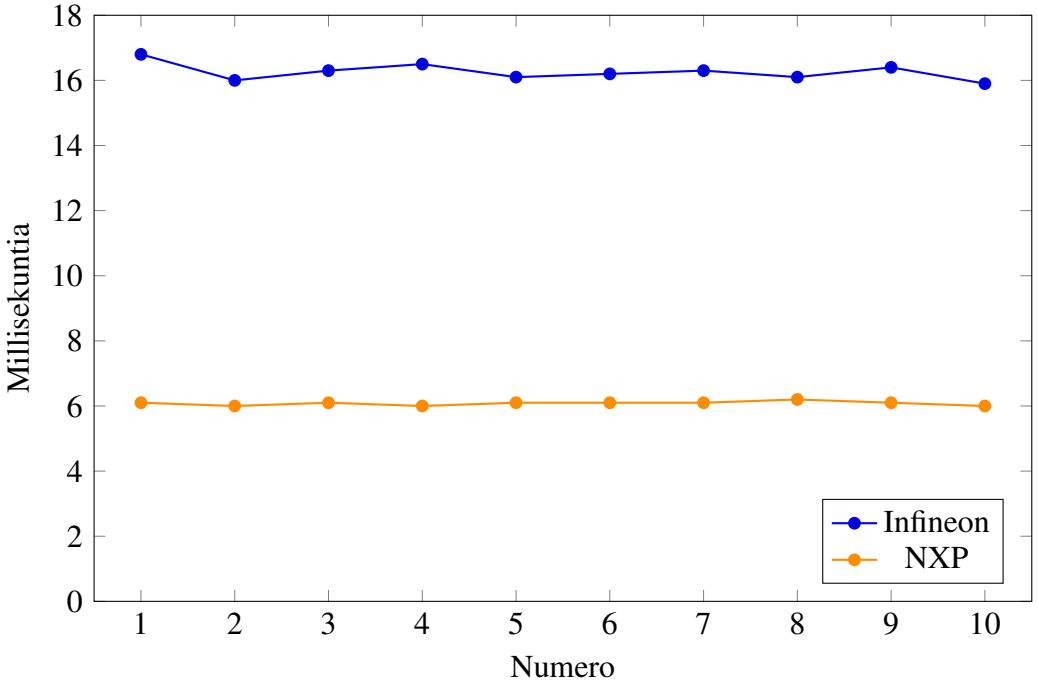
Kuvio 11: ECC 384 sign: Infineon, NXP



Kuvio 12: RSA 1024 sign: Infineon, NXP



Kuvio 13: RSA 2048 sign: Infineon, NXP



6.1.2 Pakettikaappaukset

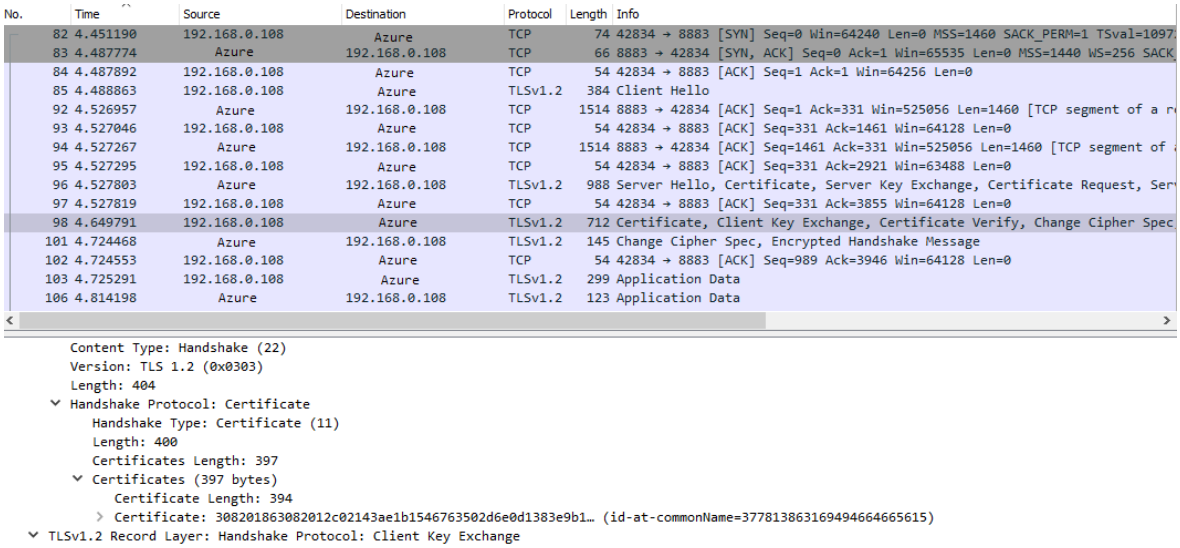
Pakettikaappaukset ovat Liitteenä olevan koodin Microsoft 2021a ajamisesta otettu Raspberry Pi:llä komennolla: `tcpdump -i wlan0 -w capture.pcap`.

Näissä pakettikaappauksissa on käytetty ecdsa_secp256r1_sha256 sertifikaattia Pakettikaappauksissa ei ole suuria eroja. Kunkin laitteen kanssa on käytetty omaa kappaleessa 5.4.1 luotua ECDSA-with-SHA256 -sertifikaattia. Azure iothub:iin yhdistämiseen on käytetty TLS 1.2:a, signaturen tuottoon ecdsa_secp256r1_sha256:sta ja avainten vaihtoon ECDHE:ta.

Vertailussa käytetään MQTT-protokollaa pilvipalveluun yhdistämiseen. Seuraavat kuviot on suodatettu näyttämään vain portin 8883 (MQTT) liikenne.

“Server hello” ja “Certificate verify” viestien välissä käyttäjän laitteen on tehtävä signature, joten tässä kaaviossa laitteiden erot tämän osalta tulevat hieman paremmin esille.

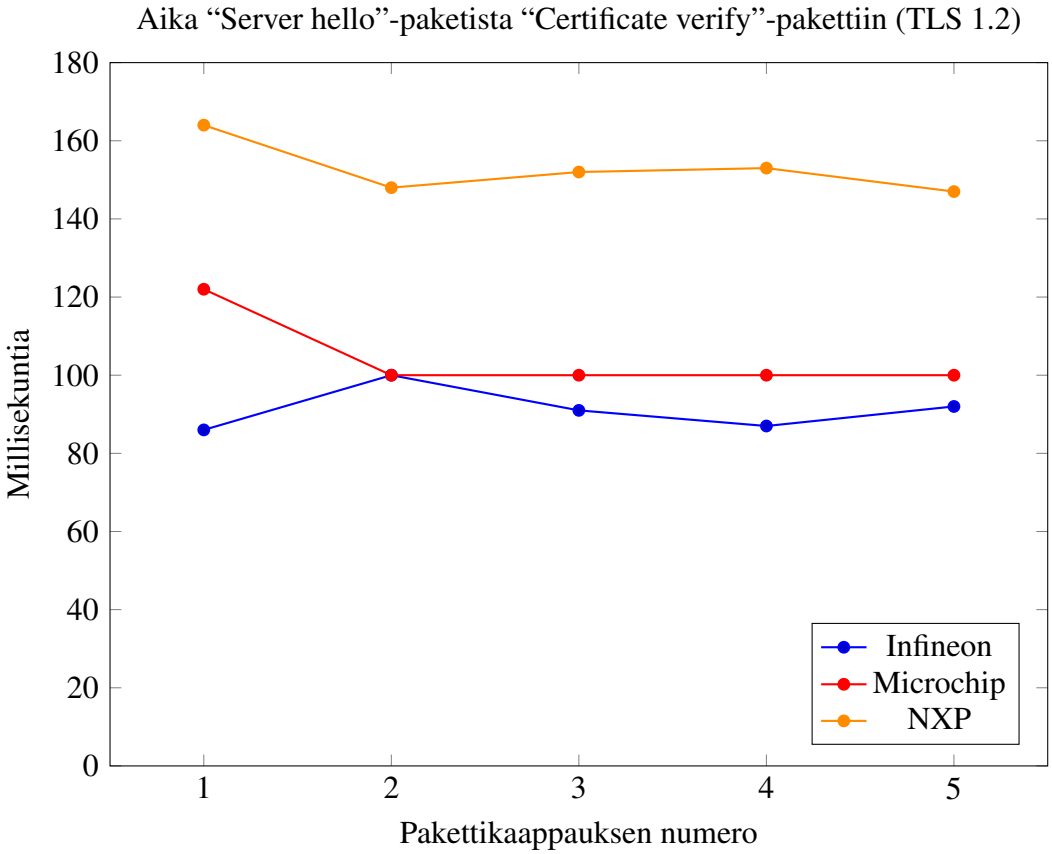
Kuvio 14: Pakettikaappaus (Microchip ATECC608A).



Kuvaajaan 15 on valittu pakettikaappauksesta paketit “Server hello” ja “Certificate verify” sen takia, että tässä vaiheessa käyttäjän eli tässä tapauksessa tietoturvakomponentin on lähetettävä kaikista aikaisemmista viesteistä muodostetun koosteen allekirjoitus (signature).

Kuvaajasta 15 voidaan huomata se mielenkiintoinen asia, että NXP SE050 on hitaampi kuin Infineon ja Microchip. Siitäkin huolimatta, että kuvaajasta 10 voidaan huomata, että signa-

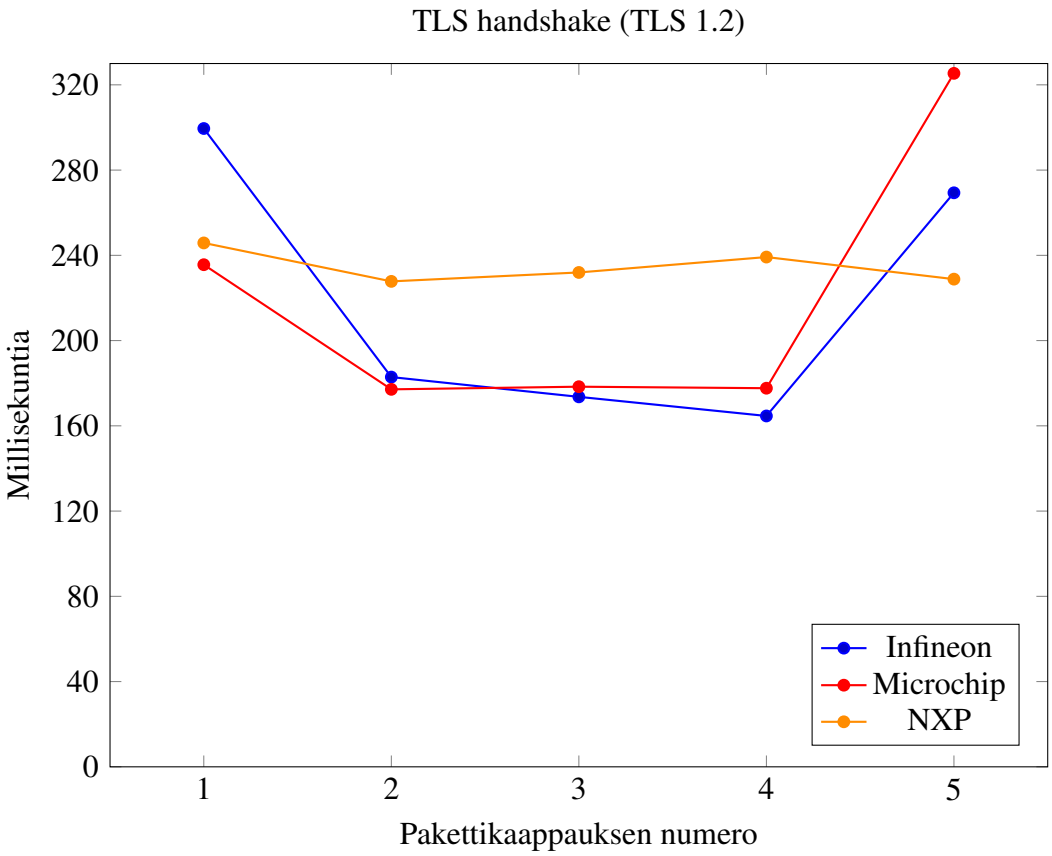
Kuvio 15: pakettikaappaukset



turen luomisessa NXP on selvästi muita nopeampi.

Kuvaajassa 16 on koko TLS1.2 handshake:en kulunut aika. Infineonin ja Microchip:n tapauksessa ensimmäisessä ja erityisesti viimeisessä pakettikaappauksissa on huomattava poikkeama muuhun dataan verrattuna. Tämä voi johtua useasta eri syystä, joista todennäköisin on käytetyn pilvipalvelun (Azure) tuoma viive.

Kuvio 16: TLS handshake



6.2 Tulosten yhteenveto

Kappaleen 6.1.1 mukaan kaikki tässä tutkielmassa mukana olleet tietoturvakomponentit suorittavat tietoturvaoperaatioita varsin nopeasti. Varsinkin ECC:n kanssa erot ovat vain muutamia millisekunteja niin avaimen luonti-, kuin allekirjoitusoperaatioissa. Toisessa vertailussa, jossa tietoturvakomponenttia käytetään avuksi yhdistämisessä pilvipalveluun, erot laitteiden ja testikertojen välillä on noin maksimissaan noin 100ms. NXP:n tapauksessa TLS handshake:ssa oli paljon vähemmän vaihtelua testikertojen välillä, kuin Microchip:lla ja Infineonilla.

Laitteiden käyttöönotossa oli kuitenkin huomattavia eroja eri laitteiden välillä. Microchip:n mukaan ATECC608 on helposti käyttöönotettava tietoturvakomponentti myös jo käytössä oleville laitteille. Kuitenkin tämän tietoturvakomponentin käyttöönotto tässä tutkielmassa oleviin käyttötapauksiin oli huomattavasti Infineonin ja NXP:n vastaavien tietoturvakomponentteja työläämpää, kuten luvussa 5.3 on mainittu. Tutkimuksessa Schläpfer ja Rüst (2019) on myös todettu, että tietoturvakomponentit tuovat huomattavia hyötyjä matalatehoisten IoT-laitteiden kanssa käytettynä, koska ne kuluttavat verrattaen vähän energiaa ja ovat suhteellisen nopeita suorittamaan kryptografisia operaatioita. Kuitenkin samassa tutkimuksessa mainitaan, että tietoturvakomponenttien käyttöönotto vaatii huomattavasti resursseja.

Kuitenkin Microchip:n ATECC608 on yksi tässä tutkielmassa vertailluista laittesta nopein. Onko kuitenkaan hankalasti käyttöönotettava tietoturvakomponentti mielekästä lisätä olemassaolevaan sulautettuun järjestelmään, vaikka se tarjoaisi huomattavan parannuksen tietoturvaan?

Tutkimuksessa Durand ym. (2019) on käytetty Microchip:n ATECC608:a apuna DTLS:n kanssa. Kuitenkin, jos käytössä on matalatehoinen mikrokontrolleri, niin kuin edellämainitussa tutkimukssa, on (D)TLS:n käyttö hidasta, vaikka itse tietoturvakomponentti suorittaisi operaatiot nopeasti. Kyseisessä tutkimuksessa DTLS:n avainten vaihtoon on kulunut 11.6s. Tässä tutkielmassa käytetyllä huomattavasti tehokkaammalla Raspberry pi 3a+:lla TLS handshake:en kuluu aikaa tietoturvakomponentista riippuen 180 - 320 ms.

Tutkimuksessa Suárez-Albela ym. (2018) käytetyn tietoturvakomponentin kanssa ECC ja RSA operaatioiden erot eivät ole niin suuria, kuin mitä tässä tutkielmassa on havaittu. Järjestys on kuitenkin sama, eli RSA on yleensä aina samalla turvallisuustasolla nopeampi ja

enemmän virtaa kuluttava, kuin mitä ECC.

Nämä laitteet ovat laitevalmistajiensa mukaan pääosin tarkoitettu hyvin matalan suoritustehon laitteisiin tuomaan parannusta tietoturvaan. Jos halutaan saavuttaa myös mahdollisimman matala virrankulutus, parhaimillaan laitevalmistajista Infineon mainostaa Optiga Trust M-laitteen tukevan lepotilaa, jossa laite ei kuluttaisi lainkaan virtaa.

7 Yhteenveto

Tässä tutkielmassa on käyty läpi sulautettujen tietoturvakomponenttien käyttöä pilvipalveluiden kanssa. Pilvi-IoT:n tietoturva koostuu monesta eri osa-alueesta, joita ovat muun muassa laitekerros, verkkokerros ja sovelluskerros. Vankan tietoturvan saaminen pilvi-IoT -järjestelmään vaatii panostusta kaikkiin osa-alueisin. Koska sulautetut järjestelmät ovat usein hyvin matalaresurssisia laitteita, on tärkeää, että niiden tietoturvan parannus ei veisi liikaa resursseja. Pilvipalvelun olisi turvallisuuden osalta hyvä tukea viestien salausta laitteelta pilveen, tukea turvallista ohjainohjelmiston päivittämistä sekä olla itse turvallinen alusta.

Tässä tutkielmassa on vertailtu kolmea eri laitetta, joita voidaan käyttää sulautetuissa järjestelmissä lisänä parantamaan tietoturvaa. Vaikka kaikki tässä tutkielmassa käytetyt laitteet olivat hyvin samanlaisia toimintojensa perusteella, niin laitteiden käyttöönoton prosessit erosivat toisistaan huomattavasti. Microchip:n ATECC608A oli vertailtavista laitteista kaikista hankalin ottaa käyttöön tämän tutkielman käyttötarpeet huomioon ottaen.

Microchip ATECC608A, Infineon Optiga Trust M ja NXP SE050 ovat erillisiä tietoturvakomponentteja, jotka pystyvät suorittamaan tavallisesti vähäresurssiselle laitteella haastavia kryptografisia operaatioita. Tässä tutkielmassa laitteiden suorituskykyä on verrattu ECC ja RSA operaatioilla sekä pilvipalveluun yhdistämisestä käyttäen TLS:ää. Osoittautui, että kaikki tietoturvakomponentit ovat verrattain nopeita, eikä niiden välillä ole käytännön kannalta kovin suuria eroja. ECC:n ja RSA:n avaimen generointi- ja signature-operaatioissa erot olivat muutaman millisekunnin luokkaa. TLS “handshake”:n tapauksessa Infineon Optiga Trust M ja Microchip ATECC608A olivat keskimäärin 80ms nopeampia kuin NXP SE050.

Modernit ECC:hen perustuvat ECDHE ja ECDSA -algoritmit ovat turvallisia käyttää salausavainten neuvotteluun ja viestien allekirjoitukseen, eikä niiden murtaminen ole nykypäivän menetelmillä mielekästä, varsinkaan, jos käytetään tarpeeksi pitkää salausavainta ja turvalliseksi todettua elliptistä käyrää. Pilven ja laitteen välissä esimerkiksi TLS:n käyttö on erittäin hyödyllistä. Sillä saadaan varmistettua luottamuksellisuus ja eheys viestien lähettämisessä ja vastaanottamisessa.

Lähteet

Adelantado, F., X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui ja T. Watteyne. 2017. “Understanding the Limits of LoRaWAN”. *IEEE Communications Magazine* 55 (9): 34–40. https://doi.org/10.1109/MCOM.2017.1600613.

Alharby, S., A. Weddell, J. Reeve ja N. Harris. 2018. “The Cost of Link Layer Security in IoT Embedded Devices ∗∗I wish to present my special thanks to Majmaah University in Saudi Arabia for their care and funding.” 15th IFAC Conference on Programmable Devices and Embedded Systems PDeS 2018, *IFAC-PapersOnLine* 51 (6): 72–77. ISSN: 2405-8963. https://doi.org/https://doi.org/10.1016/j.ifacol.2018.07.132. http://www.sciencedirect.com/science/article/pii/S2405896318308760.

Almolhis, N., A. M. Alashjaee, S. Duraibi, F. Alqahtani ja A. N. Moussa. 2020. “The Security Issues in IoT - Cloud: A Review”. Teoksessa *2020 16th IEEE International Colloquium on Signal Processing Its Applications (CSPA),* 191–196. https://doi.org/10.1109/CSPA48992.2020.9068693.

Amazon. 2021. *AWS IoT Core.* https://aws.amazon.com/iot-core/.

Angrishi, Kishore. 2017. *Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets.* arXiv: 1702.03681 [cs.NI].

Butun, Ismail, Nuno Pereira ja Mikael Gidlund. 2018. “Analysis of LoRaWAN v1.1 Security: Research Paper”. SMARTOBJECTS ’18. Los Angeles, California: Association for Computing Machinery. ISBN: 9781450358576. https://doi.org/10.1145/3213299.3213304. https://doi-org.ezproxy.jyu.fi/10.1145/3213299.3213304.

Chen, J., H. Guo ja W. Hu. 2019. “Research on Improving Network Security of Embedded System”. Teoksessa *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom),* 136–138. https://doi.org/10.1109/CSCloud/EdgeCom.2019.000-6.

Costin, Andrei, Jonas Zaddach, Aurelien Francillon ja Davide Balzarotti. 2014. “A Large-Scale Analysis of the Security of Embedded Firmwares”. Elokuu.

Das, A. K., M. Wazid, A. R. Yannam, J. J. P. C. Rodrigues ja Y. Park. 2019. “Provably Secure ECC-Based Device Access Control and Key Agreement Protocol for IoT Environment”. *IEEE Access* 7:55382–55397. https://doi.org/10.1109/ACCESS.2019.2912998.

Dhillon, P. K., ja S. Kalra. 2016. “Elliptic curve cryptography for real time embedded systems in IoT networks”. Teoksessa *2016 5th International Conference on Wireless Networks and Embedded Systems (WECON),* 1–6. https://doi.org/10.1109/WECON.2016.7993462.

Dhobi, Rahul, Sumit Gajjar, Dipen Parmar ja Tejas Vaghela. 2019. “Secure Firmware Update over the Air using TrustZone”. Teoksessa *2019 Innovations in Power and Advanced Computing Technologies (i-PACT),* 1:1–4. https://doi.org/10.1109/i-PACT44901.2019.8959992.

Durand, Arnaud, Pascal Gremaud, Jacques Pasquier ja Urs Gerber. 2019. “Trusted Lightweight Communication for IoT Systems Using Hardware Security”. Teoksessa *Proceedings of the 9th International Conference on the Internet of Things.* IoT 2019. Bilbao, Spain: Association for Computing Machinery. ISBN: 9781450372077. https://doi.org/10.1145/3365871.3365876. https://doi-org.ezproxy.jyu.fi/10.1145/3365871.3365876.

Garg, H., ja M. Dave. 2019. “Securing IoT Devices and SecurelyConnecting the Dots Using REST API and Middleware”. Teoksessa *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU),* 1–6. https://doi.org/10.1109/IoT-SIU.2019.8777334.

Google. 2021. *Cloud IoT Core.* https://cloud.google.com/iot-core.

Gyamfi, Eric, James Adu Ansere ja Lina Xu. 2019. “ECC Based Lightweight Cybersecurity Solution For IoT Networks Utilising Multi-Access Mobile Edge Computing”. Teoksessa *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC),* 149–154. https://doi.org/10.1109/FMEC.2019.8795315.

Hassija, V., V. Chamola, V. Saxena, D. Jain, P. Goyal ja B. Sikdar. 2019. “A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures”. *IEEE Access* 7:82721–82743. https://doi.org/10.1109/ACCESS.2019.2924045.

Höfer, CN, ja Georgios Karagiannis. 2011. “Cloud computing services: taxonomy and comparison”. *Journal of Internet Services and Applications* 2 (2): 81–94.

Infineon. 2021. *Linux tools and examples for OPTIGA™ Trust M1/M3 security solution.* https://github.com/Infineon/linux-optiga-trust-m.

ISO. 2014. *Information technology — Security techniques — Evaluation criteria for IT security — Part 3: Security assurance components.* ISO/IEC 15408-3:2008. Vernier, Geneva, Switzerland: International Organization for Standardization. https://www.iso.org/standard/46413.html.

Kapoor, Vivek, Vivek Sonny Abraham ja Ramesh Singh. 2008. “Elliptic Curve Cryptography”. *Ubiquity* (New York, NY, USA) 2008, numero May (toukokuu). https://doi.org/10.1145/1386853.1378356. https://doi-org.ezproxy.jyu.fi/10.1145/1386853.1378356.

Keoh, S. L., S. S. Kumar ja H. Tschofenig. 2014. “Securing the Internet of Things: A Standardization Perspective”. *IEEE Internet of Things Journal* 1 (3): 265–275. https://doi.org/10.1109/JIOT.2014.2323395.

Khanji, S., F. Iqbal ja P. Hung. 2019. “ZigBee Security Vulnerabilities: Exploration and Evaluating”. Teoksessa *2019 10th International Conference on Information and Communication Systems (ICICS),* 52–57. https://doi.org/10.1109/IACS.2019.8809115.

Kim, Dong-Hoon, Eun-Kyu Lee ja Jibum Kim. 2019. “Experiencing LoRa Network Establishment on a Smart Energy Campus Testbed”. *Sustainability* 11 (maaliskuu): 1917. https://doi.org/10.3390/su11071917.

Koubaa, Anis, Mário Alves ja Eduardo Tovar. 2021. “IEEE 802.15.4: a Federating Communication Protocol for Time-Sensitive Wireless Sensor Networks” (tammikuu).

Li, X., Q. Wang, X. Lan, X. Chen, N. Zhang ja D. Chen. 2019. “Enhancing Cloud-Based IoT Security Through Trustworthy Cloud Service: An Integration of Security and Reputation Approach”. *IEEE Access* 7:9368–9383. https://doi.org/10.1109/ACCESS.2018.2890432.

Microchip. 2021. *CryptoAuthLib - Microchip CryptoAuthentication Library.* https://github.com/MicrochipTech/cryptoauthlib.

Microsoft. 2021a. *Azure IoT C SDKs and Libraries.* https://github.com/Azure/azure-iot-sdk-c.

———. 2021b. *Azure IoT Hub.* https://azure.microsoft.com/en-us/services/iot-hub/.

Naik, Nitin. 2017. "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP". Teoksessa *2017 IEEE International Systems Engineering Symposium (ISSE),* 1–7. https://doi.org/10.1109/SysEng.2017.8088251.

National Institute of Standards and Technology. 2020. *Recommendation for Key Management: Part 1 – General.* U.S. Department of Commerce. https://doi.org/10.6028/NIST.SP.800-57pt1r5.

Neshenko, N., E. Bou-Harb, J. Crichigno, G. Kaddoum ja N. Ghani. 2019. "Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations". *IEEE Communications Surveys Tutorials* 21 (3): 2702–2733. https://doi.org/10.1109/COMST.2019.2910750.

NXP. 2021. *EdgeLock SE05x Plug Trust Middleware.* https://www.nxp.com/products/security-and-authentication/authentication/edgelock-se050-plug-trust-secure-element-family-enhanced-iot-security-with-maximum-flexibility:SE050.

Rivest, R. L., A. Shamir ja L. Adleman. 1978. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". *Commun. ACM* (New York, NY, USA) 21, numero 2 (helmikuu): 120–126. ISSN: 0001-0782. https://doi.org/10.1145/359340.359342. https://doi-org.ezproxy.jyu.fi/10.1145/359340.359342.

Roohi, A., M. Adeel ja M. A. Shah. 2019. "DDoS in IoT: A Roadmap Towards Security Countermeasures". Teoksessa *2019 25th International Conference on Automation and Computing (ICAC),* 1–6. https://doi.org/10.23919/IConAC.2019.8895034.

Saadatmand, M., ja T. Leveque. 2012. "Modeling Security Aspects in Distributed Real-Time Component-Based Embedded Systems". Teoksessa *2012 Ninth International Conference on Information Technology - New Generations,* 437–444. https://doi.org/10.1109/ITNG.2012.103.

Schaumont, P. 2017. “Security in the Internet of Things: A challenge of scale”. Teoksessa *Design, Automation Test in Europe Conference Exhibition (DATE), 2017,* 674–679. https://doi.org/10.23919/DATE.2017.7927075.

Schläpfer, Tobias, ja Andreas Rüst. 2019. “Security on IoT devices with secure elements” [kielellä en]. Teoksessa *Embedded World Conference 2019 - Proceedings.* Embedded World Conference, Nuremberg, Germany, 26-28 Februar 2019. München: WEKA. https://doi.org/10.21256/zhaw-3350. https://digitalcollection.zhaw.ch/handle/11475/16297.

Sciancalepore, S., G. Piro, G. Boggia ja G. Bianchi. 2017. “Public Key Authentication and Key Agreement in IoT Devices With Minimal Airtime Consumption”. *IEEE Embedded Systems Letters* 9 (1): 1–4. https://doi.org/10.1109/LES.2016.2630729.

Serpanos, Dimitrios, ja Tilman Wolf. 2011. “Chapter 1 - Architecture of network systems overview”. Teoksessa *Architecture of Network Systems,* toimittanut Dimitrios Serpanos ja Tilman Wolf, 1–9. The Morgan Kaufmann Series in Computer Architecture and Design. Boston: Morgan Kaufmann. https://doi.org/https://doi.org/10.1016/B978-0-12-374494-4.00001-3. http://www.sciencedirect.com/science/article/pii/B9780123744944000013.

Serpanos, Dimitrios N., ja Artemios G. Voyiatzis. 2013. “Security Challenges in Embedded Systems”. *ACM Trans. Embed. Comput. Syst.* (New York, NY, USA) 12, numero 1s (maaliskuu). ISSN: 1539-9087. https://doi.org/10.1145/2435227.2435262. https://doi-org.ezproxy.jyu.fi/10.1145/2435227.2435262.

Singh, M., A. Singh ja S. Kim. 2018. “Blockchain: A game changer for securing IoT data”. Teoksessa *2018 IEEE 4th World Forum on Internet of Things (WF-IoT),* 51–55. https://doi.org/10.1109/WF-IoT.2018.8355182.

Suárez-Albela, M., T. M. Fernández-Caramés, P. Fraga-Lamas ja L. Castedo. 2018. “A Practical Performance Comparison of ECC and RSA for Resource-Constrained IoT Devices”. Teoksessa *2018 Global Internet of Things Summit (GIoTS),* 1–6. https://doi.org/10.1109/GIOTS.2018.8534575.

Taher, Bahaa Hussein, Lu Hong Wei ja Ali A. Yassin. 2018. “Flexible and Efficient Authentication of IoT Cloud Scheme Using Crypto Hash Function”. Teoksessa *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence,* 487–494. CSAI ’18. Shenzhen, China: Association for Computing Machinery. ISBN: 9781450366069. https://doi.org/10.1145/3297156.3297220. https://doi-org.ezproxy.jyu.fi/10.1145/3297156.3297220.

Uy, Nguyen Quoc, ja Vu Hoai Nam. 2019. “A comparison of AMQP and MQTT protocols for Internet of Things”. Teoksessa *2019 6th NAFOSTED Conference on Information and Computer Science (NICS),* 292–297. https://doi.org/10.1109/NICS48868.2019.9023812.

Yuce, Bilgiday, Nahid Farhady Ghalaty, Chinmay Deshpande, Harika Santapuri, Conor Patrick, Leyla Nazhandali ja Patrick Schaumont. 2017. “Analyzing the Fault Injection Sensitivity of Secure Embedded Software”. *ACM Trans. Embed. Comput. Syst.* (New York, NY, USA) 16, numero 4 (heinäkuu). ISSN: 1539-9087. https://doi.org/10.1145/3063311. https://doi-org.ezproxy.jyu.fi/10.1145/3063311.

Zeeshan, N., M. Reed ja Z. Siddiqui. 2019. “Three-way Security Framework for Cloud based IoT Network”. Teoksessa *2019 International Conference on Computing, Electronics Communications Engineering (iCCECE),* 183–186. https://doi.org/10.1109/iCCECE46942.2019.8941877.

Liitteet

A iothub_ll_client_x509_sample.c

```
// Copyright (c) Microsoft. All rights reserved.
// Licensed under the MIT license. See LICENSE file in the project root
   for full license information.

// CAVEAT: This sample is to demonstrate azure IoT client concepts only
   and is not a guide design principles or style
// Checking of return codes and error values shall be omitted for
   brevity. Please practice sound engineering practices
// when writing production code.

#include <stdio.h>
#include <stdlib.h>

#include "iothub.h"
#include "iothub_client.h"
#include "iothub_device_client_ll.h"
#include "iothub_client_options.h"
#include "iothub_message.h"
#include "azure_c_shared_utility/threadapi.h"
#include "azure_c_shared_utility/shared_util_options.h"
#include "azure_c_shared_utility/crt_abstractions.h"
#include "azure_c_shared_utility/platform.h"
#include "iothubtransportmqtt.h"
#include "iothub_client_options.h"

// open ssl related includes
#include <openssl/crypto.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <openssl/engine.h>

/* This sample uses the _LL APIs of iothub_client for example purposes.
```

```
Simply changing the using the convenience layer (functions not having
   _LL)
and removing calls to _DoWork will yield the same results. */

// The protocol you wish to use should be uncommented
//
#define SAMPLE_MQTT

// If using an OpenSSL ENGINE uncomment and modify the line below
#define SAMPLE_OPENSSL_ENGINE "trustm_engine"

#ifdef SAMPLE_MQTT
   #include "iothubtransportmqtt.h"
#endif // SAMPLE_MQTT
#ifdef SAMPLE_MQTT_OVER_WEBSOCKETS
   #include "iothubtransportmqtt_websockets.h"
#endif // SAMPLE_MQTT_OVER_WEBSOCKETS
#ifdef SAMPLE_AMQP
   #include "iothubtransportamqp.h"
#endif // SAMPLE_AMQP
#ifdef SAMPLE_AMQP_OVER_WEBSOCKETS
   #include "iothubtransportamqp_websockets.h"
#endif // SAMPLE_AMQP_OVER_WEBSOCKETS
#ifdef SAMPLE_HTTP
   #include "iothubtransporthttp.h"
#endif // SAMPLE_HTTP

#ifdef MBED_BUILD_TIMESTAMP
   #define SET_TRUSTED_CERT_IN_SAMPLES
#endif // MBED_BUILD_TIMESTAMP

#ifdef SET_TRUSTED_CERT_IN_SAMPLES
   #include "certs.h"
#endif // SET_TRUSTED_CERT_IN_SAMPLES


static const char* connectionString = "connection string";
```

```
static const char* x509certificate =

"-----BEGIN CERTIFICATE-----""\n"
"MIIB6jCCAZACFDuRFiX8XSEs2J2epDnWUrKdc3NaMAoGCCqGSM49BAMCMIGGMQsw""\n"
"CQYDVQQGEwJGSTELMAkGA1UECAwCRkkxCzAJBgNVBAcMAkZJMQ0wCwYDVQQKDAR0""\n"
"ZXN0MQ0wCwYDVQQLDAR0ZXN0MT8wPQYDVQQDDDZDRDE2MzM4MjAxMDAxYzAwMDUw""\n"
"MDAwMGEwOTFiNWMwMDA3MDBiOTAwNzk4MDEwMTA3MTA4MDkwHhcNMjEwODE2MTAx""\n"
"MDUyWhcNMjkxMTAyMTAxMDUyWjBoMQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29t""\n"
"ZS1TdGF0ZTEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMSEwHwYD""\n"
"VQQDDBgzNzc4MTM4NjMxNjk0OTQ2NjQ2NjU2MTUwWTATBgcqhkjOPQIBBggqhkjO""\n"
"PQMBBwNCAASNduYLk5gD0wXIopmlD2X3TzNyvVoVKAPDVuOJ6p59dxOYhiaxb6/r""\n"
"4zIC5yKTOhZjU2IbnENOq6rvN7R2OVm9MAoGCCqGSM49BAMCA0gAMEUCIQDcZfNa""\n"
"Ws1FGVlOBYDHyOxjw6nmh/6sb0jz3pbPPL154gIgV1jR9SQg6AEEDDgzZkzCM8Jv""\n"
"6gebQxeUW1BdxpNRse8=""\n"
"-----END CERTIFICATE-----";



static const char* x509privatekey =

"0xe0f3:^";


static char device_certificate_pem [1300] = {0};
static uint16_t device_certificate_pem_length =
   sizeof(device_certificate_pem);

extern uint32_t pal_os_timer_get_time_in_milliseconds(void);

#ifdef SAMPLE_OPENSSL_ENGINE
static const char* opensslEngine = SAMPLE_OPENSSL_ENGINE;
static const OPTION_OPENSSL_KEY_TYPE x509_key_from_engine =
   KEY_TYPE_ENGINE;
#endif
```

```
#define MESSAGE_COUNT 5
static bool g_continueRunning = true;
static size_t g_message_count_send_confirmations = 0;

typedef struct EVENT_INSTANCE_TAG
{
   IOTHUB_MESSAGE_HANDLE messageHandle;
   size_t messageTrackingId; // For tracking the messages within the
      user callback.
} EVENT_INSTANCE;

static void send_confirm_callback(IOTHUB_CLIENT_CONFIRMATION_RESULT
    result, void* userContextCallback)
{
   (void)userContextCallback;
   // When a message is sent this callback will get envoked
   g_message_count_send_confirmations++;
   (void)printf("Confirmation callback received for message %zu with
      result %s\r\n", g_message_count_send_confirmations,
      MU_ENUM_TO_STRING(IOTHUB_CLIENT_CONFIRMATION_RESULT, result));
}



int main(void)
{


   IOTHUB_CLIENT_TRANSPORT_PROVIDER protocol;
   IOTHUB_MESSAGE_HANDLE message_handle;
   size_t messages_sent = 0;
   const char* telemetry_msg = "temperature=6""\n"
                        "testvariable=false""\n"
                        "serial=234j-f399-34jl-pp34""\n"
                        "another_variable=0x3442d""\n"
                        "time=16:40""\n"
                        "message_received=true";
```

```
   // Select the Protocol to use with the connection
#ifdef SAMPLE_MQTT
   protocol = MQTT_Protocol;
#endif // SAMPLE_MQTT
#ifdef SAMPLE_MQTT_OVER_WEBSOCKETS
   protocol = MQTT_WebSocket_Protocol;
#endif // SAMPLE_MQTT_OVER_WEBSOCKETS
#ifdef SAMPLE_AMQP
   protocol = AMQP_Protocol;
#endif // SAMPLE_AMQP
#ifdef SAMPLE_AMQP_OVER_WEBSOCKETS
   protocol = AMQP_Protocol_over_WebSocketsTls;
#endif // SAMPLE_AMQP_OVER_WEBSOCKETS
#ifdef SAMPLE_HTTP
   protocol = HTTP_Protocol;
#endif // SAMPLE_HTTP

   IOTHUB_DEVICE_CLIENT_LL_HANDLE device_ll_handle;

   // Used to initialize IoTHub SDK subsystem
   (void)IoTHub_Init();

   (void)printf("Creating IoTHub handle\r\n");
   // Create the iothub handle here
   device_ll_handle =
      IoTHubDeviceClient_LL_CreateFromConnectionString(connectionString,
      protocol);
   if (device_ll_handle == NULL)
   {
      (void)printf("Failure creating IotHub device. Hint: Check your
         connection string.\r\n");
   }
   else
   {
      // Set any option that are neccessary.
```

```
        // For available options please see the iothub_sdk_options.md
            documentation
        bool traceOn = true;
        IoTHubDeviceClient_LL_SetOption(device_ll_handle,
            OPTION_LOG_TRACE, &traceOn);

        // Setting the Trusted Certificate. This is only necessary on
            systems without
        // built in certificate stores.
#ifdef SET_TRUSTED_CERT_IN_SAMPLES
        IoTHubDeviceClient_LL_SetOption(device_ll_handle,
            OPTION_TRUSTED_CERT, certificates);
#endif // SET_TRUSTED_CERT_IN_SAMPLES

#if defined SAMPLE_MQTT || defined SAMPLE_MQTT_OVER_WEBSOCKETS
        //Setting the auto URL Encoder (recommended for MQTT). Please use
            this option unless
        //you are URL Encoding inputs yourself.
        //ONLY valid for use with MQTT
        bool urlEncodeOn = true;
        (void)IoTHubDeviceClient_LL_SetOption(device_ll_handle,
            OPTION_AUTO_URL_ENCODE_DECODE, &urlEncodeOn);
#endif
        // Set the X509 certificates in the SDK
        if (
#ifdef SAMPLE_OPENSSL_ENGINE
            (IoTHubDeviceClient_LL_SetOption(device_ll_handle,
                OPTION_OPENSSL_ENGINE, opensslEngine) != IOTHUB_CLIENT_OK) ||
            (IoTHubDeviceClient_LL_SetOption(device_ll_handle,
                OPTION_OPENSSL_PRIVATE_KEY_TYPE, &x509_key_from_engine) !=
                IOTHUB_CLIENT_OK) ||
#endif
            (IoTHubDeviceClient_LL_SetOption(device_ll_handle,
                OPTION_X509_CERT, x509certificate) != IOTHUB_CLIENT_OK) ||
            (IoTHubDeviceClient_LL_SetOption(device_ll_handle,
                OPTION_X509_PRIVATE_KEY, x509privatekey) != IOTHUB_CLIENT_OK)
            )
```

```
{
   printf("failure to set options for x509, aborting\r\n");
}
else
{
   do
   {
      if (messages_sent < MESSAGE_COUNT)
      {
         // Construct the iothub message from a string or a byte
            array
         message_handle =
            IoTHubMessage_CreateFromString(telemetry_msg);
         //message_handle =
            IoTHubMessage_CreateFromByteArray((const unsigned
            char*)msgText, strlen(msgText)));

         // Set Message property
         (void)IoTHubMessage_SetMessageId(message_handle,
            "MSG_ID");
         (void)IoTHubMessage_SetCorrelationId(message_handle,
            "CORE_ID");
         (void)IoTHubMessage_SetContentTypeSystemProperty(message_handle,
            "application%2Fjson");
         (void)IoTHubMessage_SetContentEncodingSystemProperty(message_handle,
            "utf-8");

         // Add custom properties to message
         (void)IoTHubMessage_SetProperty(message_handle,
            "property_key", "property_value");

         (void)printf("Sending messagee %d to IoTHub\r\n",
            (int)(messages_sent + 1));
         IoTHubDeviceClient_LL_SendEventAsync(device_ll_handle,
            message_handle, send_confirm_callback, NULL);
```

```
            // The message is copied to the sdk so the we can destroy
               it
            IoTHubMessage_Destroy(message_handle);

            messages_sent++;
         }
         else if (g_message_count_send_confirmations >= MESSAGE_COUNT)
         {
            // After all messages are all received stop running
            g_continueRunning = false;
         }

         IoTHubDeviceClient_LL_DoWork(device_ll_handle);
         ThreadAPI_Sleep(1);

      } while (g_continueRunning);
   }
   // Clean up the iothub sdk handle
   IoTHubDeviceClient_LL_Destroy(device_ll_handle);
}
// Free all the sdk subsystem
IoTHub_Deinit();

printf("Press any key to continue");
(void)getchar();

return 0;
}
```