Henri Murtonen

Datan eheyden varmistaminen IoT-ympäristössä ja siitä johtuvat tietoturvavaikutukset

Tietotekniikan pro gradu -tutkielma

14. joulukuuta 2016

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Henri Murtonen

Yhteystiedot: henri.j.murtonen@student.jyu.fi

Ohjaaja: Ari Viinikainen

Työn nimi: Datan eheyden varmistaminen IoT-ympäristössä ja siitä johtuvat tietoturvavaikutukset

Title in English: Protection of data integrity in IoT environment and resulting security impacts

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Sivumäärä: 52+0

Tiivistelmä: Tutkielmassa tarkastellaan IoT-ympäristön tietoturvaa ja sitä, millainen rooli datan eheyden varmistamisella on tietoturvan kannalta. Tutkielmassa testataan erilaisten datan eheyden varmistamismenetelmien toimintaa käytännössä. Tulosten perusteella analysoidaan, kuinka datan eheys kannattaa varmistaa IoT-ympäristössä ja kuinka nykyisiä menetelmiä voidaan parantaa.

Avainsanat: IoT, asioiden Internet, eheys, tietoturva

Abstract: This thesis reviews the information security of the IoT environment along with the role of data integrity protection in information security. Certain data integrity protection methods are tested in practice. Based on the results, it is proposed as to how integrity should be protected in the IoT environment and how current integrity protection methods can be enhanced.

Keywords: IoT, Internet of things, integrity, information security

Termiluettelo

6LoWPAN	"IPv6 over Low power Wireless Personal Area Networks", IPv6-verkkojen implementaatio IoT-laitteille.
CoAP	Constrained Application Protocol, IoT-laitteille suunniteltu kommunikaatioprotokolla, joka on suunniteltu integroitumaan HTTP-protokollaan.
Datan eheys	Datan muuttumattomana pysyminen, kun se säilötään tai siirretään kahden päätepisteen välillä.
ECC	Elliptic Curve Cryptography, joukko elliptisiin käyriin perustuvia julkisen avaimen salausmenetelmiä.
ECDSA	Elliptic Curve Digital Signature Algorithm, elliptisiin käyriin petustuva datan allekirjoittamisen menetelmä, ks. MAC
IoT	Internet of Things, asioiden Internet.
JSON	JavaScript Object Notation, formaatti, jolla data voidaan serialisoida merkkijonon muotoon, vrt. XML.
JSS	JSON Sensor Signatures, IoT-verkkoihin kehitetty datan eheyden varmistusmenetelmä, joka olettaa viestien olevan JSON-formaatissa.
M2M	Machine-to-Machine -kommunikaatio, kahden laitteen välinen suora kommunikaatio.
MAC	Message Authentication Code, datan eheystarkistuksissa käytetty sormenjälki, allekirjoitus tai tiiviste, jonka avulla voidaan selvittää, onko eheys säilynyt.
Man-in-the-middle	Tietoverkkohyökkäysmalli, jossa hyökkääjä pääsee kahden verkkosolmun väliin ja kykenee kuuntelemaan niiden välistä liikennettä tai muokkaamaan sitä.
NFC	Near-field Communication, teknologia, joka mahdollistaa kahden laitteen välisen kommunikaation lyhyillä etäisyyksillä.
RFID	Radio-frequency Identification, radiotaajuinen etätunnistus. Teknologia, jota käytetään tiedon etälukuun ja -tallentamiseen.

RSA	Ensimmäisiä julkiseen avaimeen perustuvia salausmentelmä. Nimetty keksijöidensä Ron Rivestin, Adi Shamirin ja Leonard Adlemanin mukaan.
TLS	Transport Layer Security, kryptografinen protokolla, joka suojaa tietoverkossa tapahtuvan liikenteen.
WSAN	Wireless Sensor and Actuator Networks, viittaa samaan käsitteeseen kuin WSN.
WSN	Wireless Sensor Networks, autonomisten sensorilaitteiden muodostamat verkot.
XML	Extensible Markup Language, formaatti, jolla data voidaan serialisoida merkkijonon muotoon, vrt. JSON.
ZigBee	Eräs IoT-verkoissa käytetty kommunikaatioprotokolla.

Kuviot

Kuvio 1. IoT-markkinoiden arvioitu jakautuminen vuonna 2020 (Shah ja Yaqoob 2016). . 4
Kuvio 2. Datan kulku esimerkkijärjestelmässä, jossa eheys on varmistettu päästä päähän. 27

Taulukot

Taulukko 1. Palvelinohjelmiston porttien konfiguraatio. 24
Taulukko 2. Datan eheyden varmistusmenetelmien testitulokset. 35

Sisältö

1 JOHDANTO 1

2 INTERNET OF THINGS 3
2.1 IoT-tietoturvan tila 5
2.2 IoT:iin liittyvät tietoturvauhat 6
2.3 IoT:iin liittyvät tietoturvahaasteet 7
2.4 Tutkitut ratkaisut IoT-tietoturvaongelmiin 8

3 DATAN EHEYDEN VARMISTAMINEN 11
3.1 IoT:iin liittyvät vaatimukset datan eheyden varmistamiselle 12
3.2 Päästä-päähän ulottuvan toteutuksen merkitys IoT-ympäristössä 14
3.3 ECC (Elliptic Curve Cryptography) 15
3.4 Datan eheyden varmistaminen osana IoT-ympäristön tietoturvaa 16

4 DATAN EHEYSTARKISTUKSET KÄYTÄNNÖN IOT-YMPÄRISTÖSSÄ 19
4.1 Testauksessa käytetyn mittausdatan rakenne 20
4.2 Testauksessa käytetty datan eheyden varmistusmenetelmä 20
4.3 Testatun palvelinohjelmiston arkkitehtuuri 23
4.4 Testatun IoT-laitteiston ohjelmistoarkkitehtuuri 25
4.5 Testattujen tilanteiden kuvaukset 26
4.6 Tilanteiden läpivienti 28

5 TULOKSET 30
5.1 Eri menetelmien käyttäytyminen testatuissa tilanteissa 30
5.1.1 Matkimishyökkäyksen analysointi 30
5.1.2 Man-in-the-middle -hyökkäyksen analysointi 32
5.1.3 Replay-hyökkäyksen analysointi 33
5.1.4 Datan virheellisen rakenteen analysointi 34
5.2 Menetelmien soveltuvuuden analysointi 35

6 YHTEENVETO 38

LÄHTEET 40

1 Johdanto

Internet of Things (IoT) eli asioiden Internet on vakiinnuttanut asemansa seuraavana globaalina kehitysaskeleena, jonka avulla tietotekniikkaa tuodaan loppukäyttäjien saataville. IoT yhdistää miljardit laitteet samaan maailmanlaajuiseen verkkoon, mikä luo aivan uudenlaisia haasteita tietoverkkojen ja palvelinten arkkitehtuureille. Eräs kriittisimmistä haasteista on tietoturvallisuuden takaaminen IoT-ympäristössä. Aihetta on tutkittu jo yli vuosikymmenen ajan, mutta tietoturvaratkaisujen saattaminen tuotantokäyttöön asti on ollut vajavaista. Tietoturvan huonoon tilaan on syynä esimerkiksi se, että monimutkaisten ja kokonaisvaltaisten tietoturvaratkaisujen käyttöönotto vaatii runsaasti kehittäjien resursseja. Lisäksi IoT-päätelaitteiden suorituskyky on heikko, eikä laitteiden suorituskapasiteettia yleisesti haluta lisätä pelkästään tietoturvan takia. Siksi tarvitaan metodeja, joiden käyttöönotto on yksinkertaista ja jotka parantavat tietoturvaa mahdollisimman pienillä suorituskykyvaatimuksilla.

Eräs keino, joka voisi täyttää edelliset vaatimukset, on datasiirron eheyden varmistaminen. Tämän tutkielman tarkoituksena on kartoittaa ja testata, kuinka tietoturvaa voidaan parantaa varmistamalla datasiirron eheys IoT-verkoissa. Tutkielmassa selvitetään, millaisia datan eheyden varmistamismenetelmiä on olemassa. Niitä suhteutetaan olemassa oleviin tietoturvaongelmiin IoT-kontekstissa, jolloin saadaan selville, miten datan eheyden varmistaminen voi vaikuttaa tietoturvaan. Tutkielmassa selvitetään myös, tuoko datan eheyden takaaminen muita konkreettisia etuja IoT-ympäristölle.

Rikkinäinen tai peukaloitavissa oleva dataliikenne, ts. dataliikenne, jonka eheyttä ei ole varmistettu, voi aiheuttaa tahattomia datan menetyksiä, palvelimien kaatuilua ja hyökkäyksille altistavia tietoturva-aukkoja. Näiden ongelmien eliminointi on ensiarvoisen tärkeää, ennen kuin voidaan ottaa käyttöön muita tietoturvamenetelmiä. Siispä tämän tutkielman tulokset ovat tärkeitä, sillä niitä voidaan käyttää apuna rakennettaessa pohjaa kokonaisvaltaiselle IoT-ympäristön tietoturva-alustalle. On syytä muistaa, että datasiirron eheyden varmistaminen ei ole sellaisenaan riittävä tietoturvamenetelmä, vaan sillä voidaan avustaa muiden tietoturvaratkaisujen toimintaa.

Datan eheys voidaan varmistaa useilla erilaisilla menetelmillä, joiden eroja ja soveltuvuutta

testataan tutkielmaa varten rakennetussa verkkoympäristössä. Tärkeimpänä muuttujana on tarkistuksen arkkitehtuurinen sijainti. Datan eheys voidaan tarkistaa joko kuljetuskerroksella tai sovelluskerroksella (Pöhls 2015). Tutkielmassa selvitetään, kumpi kerros sopii IoT-ympäristön tarpeisiin paremmin tai onko tarvetta molempien kerrosten hybridiratkaisulle.

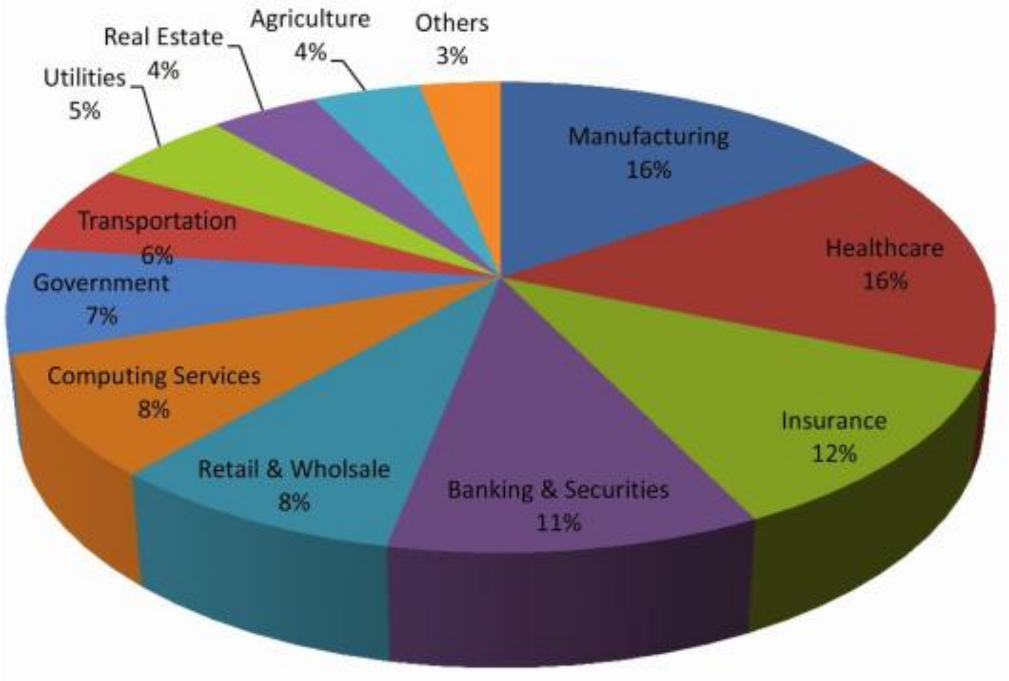
Toisessa luvussa luodaan yleiskatsaus IoT-tietoturvan tasoon ja listataan uhat, haasteet ja niihin liittyvät ratkaisut. Kolmannessa luvussa perehdytään datan eheyden takaamisen menetelmiin ja analysoidaan niiden suhdetta sekä IoT:iin että tietoturvaan. Neljännessä luvussa kuvaillaan tutkielmaa varten rakennettu verkkoympäristö palvelimineen ja laitteineen sekä listataan, millaisia tapauksia testausympäristössä toteutettiin ja selitetään niiden motiivit. Jokainen testaussessio on raportoitu yksityiskohtaisesti ja step-by-step -tyyppisesti. Viidennessä luvussa kootaan tutkielman tulokset ja analysoidaan niistä ratkaisut tutkimusongelmille. Kuudennessa luvussa kootaan yhteen tutkielman löydökset ja ehdotetaan jatkotutkimuksen aiheita.

2 Internet of Things

Internet of Things viittaa skenaarioon, jossa fyysisen maailman objektit ovat tietoverkkojen kautta yhteydessä Internetiin ja pystyvät kommunikoimaan minkä tahansa verkossa olevan entiteetin kanssa. Käytännössä tämä tarkoittaa, että laitteiden välisen kommunikaation tulee tapahtua automaattisesti ilman ihmisen tai muun manuaalisen prosessin väliintuloa (Shah ja Yaqoob 2016). Poikkeuksiakin toki on: Asplund ja Nadjm-Tehrani (2016) mainitsevat, että ainakin tietyissä lääketieteen sovelluksissa on toistaiseksi hyvä olla ihminen mukana silmukassa, jotta poikkeavuuksiin voidaan reagoida. Tällä tavoin saavutetaan korkeampi monipuolisuuden taso prototyypeissä ja IoT:iin liittyvissä tuotteissa.

IoT:in käyttökohteet ovat lähes rajattomat, mutta käytännön toteutuksia ajavat useimmiten samat motiivit. Henkilökohtaiseen käyttöön tulevat IoT-ratkaisut tavoittelevat elämän helpottamista, arkisten toimenpiteiden nopeuttamista ja henkilökohtaisen hyvinvoinnin varmistamista (Kraijak ja Tuwanut 2015; Laplante ja Laplante 2016). Liiketoimintasektorilla IoT:ia halutaan hyödyntää, jotta saavutetaan parempia palveluita, tehokkaampi tuotanto ja korkeampi laatu (Shah ja Yaqoob 2016). IoT:in käyttökohteiden moninaisuudesta kertoo liiketoimintasektorien ennustettu jakautuminen vuonna 2020. Kuviosta 1 havaitaan, että eri sektorit ovat hyvin tasaisesti jakautuneet, ja jokaiselle liiketoiminnan alueelle pystytään löytämään käyttökohteita IoT:ille.

Internet of Things ei perusperiaatteeltaan ole yksi uusi teknologia, vaan sen konsepti perustuu erilaisten uusimpien teknologioiden yhdistämiseen (Shah ja Yaqoob 2016). IoT hyödyntää uusimpia kehitysaskeleita kommunikaatioteknologioissa, älykkäissä antureissa, Internet-protokollissa ja radioteknologiassa. Kun IoT esiteltiin ensimmäistä kertaa, RFID-tunnistusta pidettiin kaikkein olennaisimpana osana IoT:in toimintaa (Shah ja Yaqoob 2016). Nykyään monet muut teknologiat ovat nousseet RFID-tunnistuksen rinnalle tarkoituksenaan toteuttaa IoT:in visio korkeasta yhdistettävyydestä. RFID:n lisäksi IoT-ympäristöön soveltuvia tai sitä varta vasten kehitettyjä teknologioita ovat Near Field Communications (NFC), Wireless Sensor and Actuator Networks (WSAN), Wireless Identification and Sensing Platforms (WISP), 6LoWPAN, Machine to Machine (M2M), Wireless Hart ja ZigBee (Atzori, Iera ja Morabito 2010).



Kuvio 1. IoT-markkinoiden arvioitu jakautuminen vuonna 2020 (Shah ja Yaqoob 2016).

Shah ja Yaqoob (2016) ovat listanneet, mitkä teknologiat soveltuvat erityisen hyvin kuhunkin käyttötarkoitukseen. RFID soveltuu parhaiten esimerkiksi älykkääseen parkkeeraukseen, turistikohteiden karttojen täydentämiseen, logistiikkaan, datan keräykseen esimerkiksi lääketieteessä, älykkääseen vedenjakeluun sekä älykoteihin ja -toimistoihin. RFID:n taustalla olevan RF-teknologian suurena haasteena on lyhyt kantama, sillä sensorit eivät sisällä omaa virtalähdettä, vaan ottavat virran saapuvasta signaalista (Atzori, Iera ja Morabito 2010). Kehityksessä on kuitenkin tapahtunut selkeätä edistystä, sillä vuonna 2010 maksimikantamaksi ilmoitettiin 2 metriä, kun vuonna 2013 saavutettiin jo 30 metrin maksimikantama tietyillä sovelluksilla (Atzori, Iera ja Morabito 2010; Chen ym. 2013). M2M- ja V2V (Vehicle-to-Vehicle) -kommunikaatiolle ja NFC:lle otollisia käyttökohteita ovat teollinen ylläpito, äykkäät autot, älykkäät sähköverkot, matkustus ja terveydenhuolto (Shah ja Yaqoob 2016).

IoT:in arkkitehtuurisesta määritelmästä on erilaisia mielipiteitä, mutta useiden tutkimusten mukaan IoT:in toiminta voidaan jakaa kolmeen eri kerrokseen (Mahmoud ym. 2015; Leo ym. 2014; Gou ym. 2013):

- Havaintokerros: Sensorit ja muut laitteet havainnoivat ympäristöä ja keräävät siitä tie-

toa. Havaintokerros havaitsee, kerää ja prosessoi tietoa ja sitten siirtää sen verkkokerrokselle (Mahmoud ym. 2015). Tämän kerroksen toimijoita ovat IoT-päätelaitteet ja sensorikeskittymät (Leo ym. 2014).

- Verkkokerros: Datan reititys ja lähetys erilaisiin IoT-keskittimiin ja laitteisiin tapahtuu verkkokerroksella. Verkkokerroksella operoidaan erilaisilla nykyaikaisilla verkkoteknologioilla, joihin kuuluvat esimerkiksi WiFi, LTE, Bluetooth, 3G ja ZigBee (Mahmoud ym. 2015). Tällä kerroksella toimivat pilvilaskenta-alustat, Internet-datakeskukset, langattomat viestintäteknologiat ja kiinteät yhteydet (Leo ym. 2014).
- Sovelluskerros: IoT-järjestelmän tarkoitus toteutetaan sovelluskerroksella (Mahmoud ym. 2015). Käytännössä tämä tarkoittaa esimerkiksi älykkään kodin eri sensorien yhdistävää järjestelmää. Sovelluskerroksen toimijoita ovat kaikki IoT-ympäristössä toimivat sovellukset, joista esimerkkejä ovat älykkäät kodit, älykkäät kaupungit ja telelääketiede (Leo ym. 2014).

2.1 IoT-tietoturvan tila

Shah ja Yaqoob (2016) Mainitsevat IoT:in suurimmaksi ongelmaksi tietoturvan. Ongelman suuruudesta kertoo esimerkiksi se, että vasta sen jälkeen mainitaan standardisoinnin ja skaalautuvuuden tuomat ongelmat. Kun otetaan huomioon, että IoT-laitteiden nykyinen lukumäärä on noin 6 miljardia ja sen arvioidaan nousevat noin 20-21 miljardiin laitteeseen vuoteen 2020 mennessä (Meulen 2015), Ciscon korkeimpien arvioiden ollessa jopa 50 miljardia (Shah ja Yaqoob 2016), on selvää, että myös standardisoinnin ja skaalautuvuuden haasteet ovat merkittävät — mutta ne ovat kuitenkin pienemmässä roolissa kuin tietoturvan aiheuttamat haasteet.

IoT-tietoturvan yleisestä tilasta on tehty lukuisia tutkimuksia (Kraijak ja Tuwanut 2015; Mahmoud ym. 2015; Leo ym. 2014; Ray ym. 2016; Wurm ym. 2016; Xu, Wendt ja Potkonjak 2014; Zhang ym. 2014). Wurm ym. (2016) ovat havainneet, että tietoturva IoT-laitteistoissa on usein joko laiminlyöty tai käsitelty muun kehityksen jälkeen "jälkiviisautena". Tämä johtuu siitä, että kehitys tehdään lyhyellä aikavälillä ja kokonaiskustannusten laskeminen on tärkeä tavoite valmistajien liiketoiminnassa. Leo ym. (2014) korostavat, että IoT-ympäristössä on erityisen tärkeää varmistaa tietoturvan toiminta jokaisessa verkon solmussa, sillä yhden

solmun heikko tietoturva vaarantaa kaikkien verkossa olevien laitteiden turvallisuuden. Jokaisen solmun yksittäinen varmistaminen on hyvin hankalaa, sillä IoT:in luonne on hyvin heterogeeninen ja yleensä tietyn laitteen toimittajalla ei ole tarkkaa tietoa, millaisten laitteiden kanssa se toimii yhdessä (Mahmoud ym. 2015). Kraijak ja Tuwanut (2015) huomauttavat, että IoT-ympäristön tietoturvavaatimukset eroavat perinteisestä Internetistä siltä osin, että IoT:iä sovelletaan kaikkein merkittävimpiin osiin globaalista taloudesta. Esimerkkeinä Kraijak ja Tuwanut (2015) mainitsevat kuljetuksen, terveydenhuollon, älykkäät kaupungit ja kodit sekä henkilökohtaiset ja sosiaaliset käyttötarkoitukset.

2.2 IoT:iin liittyvät tietoturvauhat

Gou ym. (2013) ovat analysoineet IoT:in kolme kerrosta tietoturvan näkökulmasta:

- Havaintokerroksen tietoturvauhat liittyvät fyysiseen haavoittuvuuteen ja kerätyn tiedon suojaamiseen. Koska havainnointi tapahtuu useimmiten olosuhteissa, joissa ei ole jatkuvaa valvontaa, laitteet ovat haavoittuvaisia tunkeilijoita ja hyökkääjiä vastaan. Tällä kerroksella käytetyistä teknologioista sekä WSN (Wireless Sensor Network) että RFID:ssä käytetty RF-kommunikaatio sisältävät tunnettuja haavoittuvuuksia.
- Verkkokerroksella tietoturvaongelmat ovat vastaavia kuin tietoverkoissa yleensä: Uhkana ovat man-in-the-middle -hyökkäykset, palvelunestohyökkäykset, datan salakuuntelu, eheyshyökkäykset ja virushyökkäykset.
- Sovelluskerroksen tietoturvauhat liittyvät siihen, että toimiva sovellus itsessään tarvitsee monia eri kerroksia, joiden välillä dataa siirretään. Monesta osasta koostuvaan sovellukseen voidaan hyökätä pilvilaskennan, tiedonlouhinnan, väliohjelmistojen, tietokantojen tai varmuuskopioiden kautta hyödyntäen niissä olevia tietoturva-aukkoja. Sovellus voi myös kärsiä haavoittuvuuksista esimerkiksi yksityisyydensuojaa tai aineettomien oikeuksien suojaa vastaan.

Xu, Wendt ja Potkonjak (2014) esittävät IoT-ympäristön suurimmaksi tietoturvaan liittyväksi uhaksi sen, että IoT-kontekstissa vihamielinen hyökkääjä voi Internetin kautta hallita fyysistä maailmaa. Esimerkiksi autojen, lentokoneiden tai teollisuuden laitteistojen kautta voidaan aiheuttaa merkittäviä vahinkoja. Kuten perinteisenkin tietoturvan kontekstissa, hyökkääjä

voi olla missä päin maailmaa tahansa. Oleellisena erona on kuitenkin se, että hyökkääjällä saattaa olla fyysinen pääsy laitteeseen, mikäli se on julkisessa tilassa tai muulla tavalla riittämättömästi fyysiesti suojattu. Tällöin hyökkääjä voi hyödyntää sellaisia hyökkäysväyliä, jotka eivät ole aiemmin olleet saavutettavissa. Tällaisia väyliä ovat boot-prosessit, laitteisto-hyökkäykset, siru-tason hyökkäykset ja debug-käyttöön tarkoitetut väylät ja takaportit (Ly ja Jin 2016).

2.3 IoT:iin liittyvät tietoturvahaasteet

Mahmoud ym. (2015) jakavat IoT:iin liittyvät tietoturvahaasteet kahteen eri luokkaan, teknologisiin haasteisiin ja turvallisuushaasteisiin. Teknologiset haasteet johtuvat siitä, että päätelaitteet ovat heterogeenisiä ja kaikkialla läsnäolevia. Teknologiset haasteet liittyvät yleensä langattomiin teknologioihin, skaalautuvuuteen, energiankulutukseen ja verkon hajautettuun luonteeseen. Turvallisuushaasteet puolestaan liittyvät siihen, kuinka periaatteita ja toiminnallisuuksia tulisi valvoa, jotta saavutetaan turvallinen verkko. Turvallisuushaasteisiin kuuluvat autentikaatio, luottamuksellisuus, päästä-päähän -tietoturva ja eheys. Zhang ym. (2014) puolestaan jakavat tietoturvahaasteet hieman eri tavalla, laitteistohaasteisiin ja kommunikaatiohaasteisiin. Periaatteena on se, että laitteistohaasteet liittyvät laitteisiin, "Things", ja niissä ajettaviin ohjelmiin, kun kommunikaatiohaasteet liittyvät laitteiden väliseen yhteydenpitoon ja verkon haavoittuvuuksiin. Aiemmin esitettyihin IoT:in toimintakerroksiin sovellettuna laitteistohaasteet liittyvät havainto- ja sovelluskerrokseen, ja kommunikaatiohaasteet liittyvät verkkokerroksen toimintaan.

Xu, Wendt ja Potkonjak (2014) esittävät IoT-ympäristön suurimmaksi tietoturvahaasteeksi sen, että vaikka energiatehokkuus nähdään yhtenä tärkeimmistä piirteistä IoT-järjestelmissä (Kraijak ja Tuwanut 2015), niin erilaisilla laitteistoilla on hyvin erilaiset resurssit energiankäytön suhteen. Siispä tavoite on vaikea: Samalla kerralla on saavutettava sekä geneerisyys että laajat kustomointimahdollisuudet (Xu, Wendt ja Potkonjak 2014).

Monet tutkimukset ovat esittäneet tarpeen keskitetylle ja sulautetulle tietoturvaratkaisulle, jotta IoT-ekosysteemin heterogeenisyyden tuomat ongelmat saataisiin karsittua (Babar ym. 2011; Leo ym. 2014; Lincke, Kuntze ja Rudolph 2015; Liu, Zhang ja Zhang 2013; Mah-

moud ym. 2015). Tätä haastetta on lähestytty hyvin erilaisilla keinoilla. Babar ym. (2011) ehdottavat tietoturvakehystä, jonka konseptien avulla kuhunkin käyttötarkoitukseen ja ympäristöön voidaan tiettyjen vaatimusten pohjalta räätälöidä tietoturvatoteutus. Leo ym. (2014) ovat suunnitelleet hajautetun tieturva-arkkitehtuurin, joka pohjautuu SOA-järjestelmiin. Myös Lincke, Kuntze ja Rudolph (2015) ehdottavat hajautettua ratkaisua, mutta se perustuu P2P-verkkojen arkkitehtuuriin, jossa pystytään saavuttamaan laaja dynaamisuus ratkaisun ollessa silti riittävän geneerinen. Liu, Zhang ja Zhang (2013) esittävät täysin uuden lähestymistavan, jonka perustana on "Artificial Immune System (AIS)", joka puolestaan pohjautuu biologisen immunisaation periaatteisiin. Mahmoud ym. (2015) ovat havainneet, että suurin osa IoT-tietoturvaa käsittelevistä tutkimuksista keskittyy vain autentikaatioon, identiteetin luomiseen ja pääsyn hallintaan unohtaen kokonaisvaltaiset hyödyt, joita esimerkiksi IPv6 ja 5G voivat tuoda IoT-ympäristön tietoturvalle.

2.4 Tutkitut ratkaisut IoT-tietoturvaongelmiin

IoT:in tietoturvaongelmiin on tutkittu ratkaisuja erittäin monesta eri näkökulmasta. Osa tutkimuksista ei ota lainkaan kantaa käytettäviin teknologioihin, vaan esittää abstraktin toteutuksen, johon voidaan liittää sopivat teknologiat kuhunkin käyttötarkoitukseen sopivasti. Osa keskittyy yhden potentiaalisen teknologian testaamiseen ja osa yhdistää olemassaolevia teknologioita jonkin uuden paradigman tai konseptin tutkimiseksi.

Riahi ym. (2013) ja Riahi ym. (2014) esittävät systeemisen lähestysmistavan IoT-tietoturvan toteuttamiselle. Esitetty toteutus perustuu eri toimijoiden roolittamiseen ja toimijoiden välisten suhteiden tarkkaan analysointiin. Eri toimijat ovat henkilö, älykäs objekti, prosessi ja teknologinen ekosysteemi. Jokaisen toimijan välillä on suhde, johon voidaan liittää jokin tietoturvan konsepti: Esimerkiksi yksityisyys on se suhde, joka liittää ihmisen ja teknologisen ekosysteemin toisiinsa. Lähteissä selvennetään myös, kuinka lähestymistapaa sovelletaan kolmessa todellisessa IoT-käyttökohteessa.

Lincke, Kuntze ja Rudolph (2015) esittävät vertaisverkkoihin perustuvan Tusted P2P Management (TP2M) -ratkaisun, jossa on hajautettu tietoturvan hallinta. Ratkaisua perustellaan sillä, että laitteiden kesken hajautettu työmäärä soveltuu hyvin tilanteeseen, jossa vähintään

osalla laitteista on hyvin rajallinen suorituskyky. Osalla laitteista voi olla korkeampi suorituskyky kuin toisilla, jolloin ne toimivat hallintasolmuna verkostossa. Matalan tason tietoturvaa ratkaisussa hoitaa TPM (Trusted Platform Module).

Liu, Zhang ja Zhang (2013) ovat kehittäneet ratkaisun, joka pohjautuu immunologiaan. Toisin kuin nykyiset IoT-tietoturvamenetelmät, jotka ovat passiivisia, kehitetty ratkaisu tarjoaa aktiivisen suojautumisen hyökkäyksiä vastaan. Menetelmä perustuu aktiiviseen luokitteluun ja poikkeavuuksien havaitsemiseen, ja toteutuksesta on esitetty yksityiskohtainen kuvaus.

CoAP on protokolla, joka sovittaa HTTP:n toiminnan IoT-ympäristöön sopivaksi. Se perustuu REST:iin ja siinä on tiivistetyt otsakkeet sekä tuki multicast-pyynnöille (Rahman ja Shah 2016). CoAP-protokollaa on tutkittu tietoturvan suhteen (Rahman ja Shah 2016; Capossele ym. 2015). Tutkimusten perusteella CoAP:n avulla saavutetaan suorituskykyetu verrattuna HTTP-protokollaan, mutta raskaiden tietoturvamenetelmien käyttöönotto kutistaa eron. Haasteena on siis säilyttää korkea suorituskyky samalla kun ylläpidetään turvallisuusstandardeja ja suojausta (Rahman ja Shah 2016). Capossele ym. (2015) esittävät ratkaisun, jossa DTLS-protokollalla tarjotaan CoAP:lle optimoitu päästä-päähän ulottuva tietoturva. DTLS (Datagram Transport Layer Security) on TLS-turvallisuusprotokollan implementaatio UDP-liikenteelle, ja se on yhteensopiva CoAP:n kanssa (Rahman ja Shah 2016). DTLS:lle on tutkittu myös globaalia käytännön toteutusta hyödyntäen julkisia sertifikaatteja (Panwar ja Kumar 2015), mikä puoltaa CoAP:n käyttöönottoa. Lisäksi DTLS:n voi sulauttaa SIM-kortille (Urien 2016), mikä kasvattaa käyttökohteiden mahdollista lukumäärää.

ZigBee-kommunikaatioprotokolla on vakiinnuttanut paikkansa yhtenä johtavista kommunikaatioprotokollista IoT-verkoissa (Razouk 2014). Siitä huolimatta sen tietoturva ei ole riittävän korkealla tasolla, vaan sisältää heikkouksia. Razouk (2014) on tutkinut ZigBee:tä ja esittää oman ratkaisunsa tietoturvaongelmien korjaamiseksi. Hernández-Ramos ym. (2015) esittävät PANA (Protocol for Carrying Authentication for Network Access) -tietoturvaratkaisun, joka voisi soveltua ZigBee:n lisäksi muihin yleisesti IoT-ympäristössä käytettäviin kommunikaatioprotokolliin.

Sahraoui ja Bilami (2014) esittävät 6LoWPAN-ympäristössä toimivan HIP (Host Identity Protocol) -teknologian jatkeen, jonka avulla saadaan tiivistettyä otsakkeita ja jaettua lasken-

nan kuormitusta. Tutkimuksen perusteella ratkaisulla voidaan parantaa 6LoWPAN-ympäristössä toimivien IoT-järjestelmien tietoturvaa vähäisellä kommunikaatiomäärän kasvulla.

Tellez, El-Tawab ja Heydari (2016) ovat tutkineet IoT-laitteiden fyysistä tietoturvaa ja onnistuneet löytämään firmware-tason haavoittuvuuksia. Löytyneiden haavoittuvuuksien kautta he kykenivät raakaa voimaa hyödyntäen varastamaan laitteen WSN-salausavaimen. Esitetyn ratkaisun avulla raa'an voiman käyttäminen tulee mahdottomaksi, mikä estää kaikki siitä aiheutuneet fyysiset haavoittuvuudet.

Sarigiannidis, Karapistoli ja Economides (2015) ovat kehittäneet VisIoT-työkalun, jolla voidaan monitoroida ja visualisoida IoT-liikennettä ja havaita tietoturvahyökkäyksiä. Käytännössä kehitetty työkalu on tunkeutumisen havaitsemisjärjestelmä, jossa on hyödynnetty visuaalisen analytiikan keinoja. Tutkimuksen perusteella ei voida vielä tehdä johtopäätöksiä visualisoinnin hyödyistä tässä kontekstissa, mutta järjestelmää kehitetään jatkuvasti ja sitä aiotaan testauttaa verkkoasiantuntijoilla.

3 Datan eheyden varmistaminen

Datan eheydellä tarkoitetaan sitä, että data ei ole korruptoitunut tai muusta syystä tahattomasti muuttunut pisteiden A ja B välillä. Se on tunnettu konsepti esimerkiksi tiedostojärjestelmissä, joissa se viittaa siihen, että data pitää voida lukea levyltä täsmälleen samassa muodossa kuin se on levylle kirjoitettu. Datan eheyden merkitys on noussut entisestään pilvipalvelujen suosion myötä (Sevis ja Seker 2016), sillä samaa dataa saattaa käsitellä huomattavasti suurempi joukko eri toimijoita. Eheyden vastakohta on korruptoituminen, ja se saattaa aiheutua monenlaisista syistä, kuten inhimillisestä erehdyksestä, ohjelmistobugista, laitteiston toimintahäiriöstä tai tarkoituksellisesta hyökkäyksestä (Sevis ja Seker 2016).

Tämän tutkielman kontekstissa datan eheydellä viitataan liikenteen eheyteen: Datasiirroissa ja kommunikaatiossa on varmistuttava siitä, että informaatio ei muutu lähettämisen ja vastaanottamisen välillä. Tällä tavoin varmistetaan, että käsiteltävä informaatio on todenperäistä, ja että käsittelyprosessissa voidaan riittävällä kohtuudella olettaa, että datan rakenne noudattaa sovittua määrittelyä. IoT-ympäristössä datan eheyden varmistamisella on erityinen rooli, sillä fyysisen tietoturvan pettäessä datan eheystarkistuksilla voidaan vielä estää hyökkääjältä datan manipuloinnin mahdollisuus.

Datan eheyden tarkistamiseen ja varmentamiseen on olemassa erilaisia keinoja, joista kukin sopii tietynlaiseen sovellusskenaarioon. Laitteistotason eheystarkistuksiin käytetään usein hash-puita, mobiilijärjestelmissä on käytössä teknologiat nimeltä Forward Integrity, Detection Object ja Mutual Itinenary Recording (Hou ym. 2004). Kaikkein useimmin käytetty datan eheyden varmistamismenetelmä on kuitenkin MAC (Message Authentication Code) eli tietyn mittainen kryptografinen sormenjälki datasta. Tätä sormenjälkeä kutsutaan myös allekirjoitukseksi tai hashiksi eli tiivisteeksi, ja sen laskennassa tulisi pyrkiä siihen, että erilaiset lähtödatat tuottavat mahdollisimman harvoin saman tiivisteen. Toiminta perustuu siihen, että lähettäjä antaa tietylle tiivistefunktiolle parametrina lähetettävän datan ja funktio palauttaa tiivisteen, joka liitetään lähetettävään viestiin. Lähettämisen ja vastaanottamisen jälkeen vastaanottaja muodostaa uudelleen tiivisteen samaa funktiota käyttäen, ja tarkistaa, että datasta uudelleen muodostettu tiiviste vastaa viestin mukana tullutta tiivistettä. Jos tapahtuu niin, että tiivisteet eroavat, niin vastaanottaja tietää viestin muuttuneen matkalla ja pyytää uudel-

leenlähetystä.

On syytä huomioida, että täydellistä eheyttä ei voida varmistaa pelkällä lähetettävästä datasta muodostetulla MAC:lla. Vaikka lähetettävä ja vastaanotettu data olisivat samat, voi eheys rikkoutua esimerkiksi siten, että sama data lähetetään useampaan kertaan kuin on ollut tarkoitus, tai että vastaanottaja on väärä. Edellä mainitut tapaukset johtuvat yleensä tarkoituksellisista tietoturvahyökkäyksistä, ja uudelleenlähetykseen perustuvia hyökkäyksiä kutsutaankin nimellä *replay attack*. Ratkaisuna ongelmaan on se, että MAC-tiivisteeseen liitetään vastaanottajan tunniste, esimerkiksi ID-koodi, sekä juokseva numerointi tai aikaleima, jolla estetään uudelleenlähetetyn datan pääsy käsittelyyn saakka. Varta vasten kehitetyillä menetelmillä voidaan varmistaa, että hyökkääjä ei kykene selvittämään ja kopioimaan juoksevan numeroinnin tai aikaleimojen logiikkaa. (Hou ym. 2004)

3.1 IoT:iin liittyvät vaatimukset datan eheyden varmistamiselle

IoT-ympäristössä datan eheyden varmistus toteutetaan joko kuljetus- tai sovelluskerroksella (Pöhls 2015). On syytä huomioida, että Pöhls (2015) ei viittaa yleisesti käytössä olevan OSI-mallin mukaisiin kommunikaatiokerroksiin, vaan TCP/IP-mallille räätälöityihin kerroksiin (Goralski 2009). Kuljetuskerroksen eheystarkitus on mahdollista toteuttaa esimerkiksi TLS:n (Transport Layer Security) avulla (Modadugu ja Rescorla 2004), jolloin se on luonteeltaan sisäänrakennettu ja automaattinen. Kuljetuskerroksen eheyden varmistaminen perustuu siihen, että kahden toimijan välisen kanavan eheyttä ei voi rikkoa ilman, että osapuolet huomaavat tapahtuman (Pöhls 2015). Sovelluskerroksen eheystarkistus on täysin sovelluskohtainen ja voidaan toteuttaa joustavasti käyttötarkoitukseen sopivaksi (Pöhls 2015). Lisäksi sovelluskerroksen eheystarkistus voidaan suorittaa riippumattomana muista tietoturvatekijöistä (Mössinger ym. 2016). Usein sovelluskerroksen eheystarkistus toteutetaan MAC-tarkistuksella.

IoT asettaa tiettyjä erityisvaatimuksia datan eheyden varmistamisessa käytettävien teknologioiden ja algoritmien suhteen. Nämä vaatimukset juontuvat suoraan IoT-ympäristön fyysisistä ja konseptuaalisista rajoitteista:

- Suorityskyky: Tämän vaatimuksen alle asettuu useita alavaatimuksia, joista osa on

yleisesti voimassa ja osa riippuu käyttötarkoituksesta ja käytössä olevista laitteistoresursseista. IoT-laitteiston suorituskyky ja muistikapasiteetti on yleisesti ottaen heikko, mistä johtuen osa algoritmeistä ei ole lainkaan saatavilla ja osa vaatii huonosta suoritustehosta johtuen liian kauan aikaa suorittaa. Tutkimukset ovat osoittaneet, että ECC (Elliptic Curve Cryptography) -algoritmeihin perustuvat ratkaisut pystyvät tarjoamaan suhteellisen lyhyet suoritusajat IoT-ympäristössä (Mössinger ym. 2016). ECC-algoritmejä käsitellään tarkemmin luvussa 3.3.

- Heterogeenisyys: IoT-laitteilla on käytössään hyvin erilaiset suoritusresurssit, niissä ajetaan hyvin erilaisia ohjelmia ja niiden käyttöjärjestelmät sekä arkkitehtuurit eroavat toisistaan. Heterogeenisyyden haaste nousee esille jatkuvasti IoT-järjestelmistä puhuttaessa, ja keskitetyn ratkaisun kehittäminen on siksi vaikeaa. Mitä laajemmaksi IoT-järjestelmä kasvaa, sitä vaikeampaa on toteuttaa kaikilla alustoilla toimiva ratkaisu (Babar ym. 2011).
- Automaattisuus: Määritelmänsä mukaan IoT yhdistää erilaisia laitteita, jotka kommunikoivat keskenään. Koska kaikki kommunikaatio tapahtuu laitteelta laitteelle, järjestelmän täytyy osata havaita ja korjata vikatilanne ilman ulkopuolista puuttumista. Kriittisissä järjestelmissä on ensiarvoisen tärkeää, että eheyden rikkoutuessa toipuminen tapahtuu nopeasti ja ilman sivuvaikutuksia.
- Kestävyys: IoT-järjestelmien täytyy sietää uudenlaisia tietoturvahyökkäyksiä, varsinkin fyysisiä, kuten luvussa 2.2 mainittiin. Käytettyjen algoritmien täytyy olla kryptografisesti kestäviä, jotta niiden murtaminen olisi mahdollisimman hankalaa ja aikaa vievää. On syytä huomioida, että kaikki kolme aiempaa haastetta saattavat olla ristiriidassa kestävyyden kanssa, joten on tärkeää luoda oikeanlainen kompromissi.

IoT-ympäristössä täytyy ottaa huomioon se, että samaa dataa käsitteleviä osapuolia voi olla useita. On mahdollista, että jotkin näistä osapuolista eivät syystä tai toisesta pysty suorittamaan datan eheystarkistuksia. Syynä saattaa olla esimerkiksi täydellisen reaaliaikaisuuden vaatimus, joka vähentää käytössä olevaa suoritusaikaikkunaa. Tällaisessa poikkeustapauksessa on välttämätöntä, että data on saatavilla myös ilman eheystarkistuksia. Jotta datan eheyden varmistamismenetelmä soveltuisi hyvin IoT-ympäristöön, sen täytyy mahdollistaa käsittely myös ilman eheystarkistuksia (Pöhls 2015). Käytännössä vaatimuksesta seuraa se, että sovelluskerroksen eheystarkistus soveltuu paremmin IoT-ympäristöön joustavuutensa ta-

kia.

Täydellisen reaaliaikaisuuden vaatimus on IoT-järjestelmissä kuitenkin harvinainen. Useimmiten on täysin hyväksyttävää, että eheystarkistukset aiheuttavat 200ms – 2s viiveen (Mössinger ym. 2016). Vertailukohtana Habib ym. (2005) esittävät reaaliaikaiseen median suoratoistoon soveltuvan datan eheyden varmistusmenetelmän, josta käy hyvin ilmi, miten erilaiset vaatimukset suoratoistolla on verrattuna IoT-ympäristön tavanomaisiin käyttötarkoituksiin verrattuna. Tutkimusraportista on hyötyä IoT-järjestelmien eheysvarmistuksen suunnittelussa, sillä se antaa perspektiiviä sille, millaisin perustein datan eheystarkistusmenetelmiä on kehitetty ennen IoT-järjestelmien olemassaoloa.

3.2 Päästä-päähän ulottuvan toteutuksen merkitys IoT-ympäristössä

Pöhls (2015) nostavat esille huomion siitä, että vaikka kuljetuskerroksella toteutettu datan eheystarkistus on tärkeä osa kokonaisuutta, se ei suojele datan eheyttä lähettämisen jälkeen. Usein IoT-järjestelmissä dataa käsitellään useassa eri solmussa vielä sen jälkeen, kun data on mitattu ja lähetetty laitteelta eteenpäin. Lähetysoperaation eheyden lisäksi on yhtä tärkeää varmistaa, että eheys säilyy myös kaikissa näissä jatko-operaatioissa, joita datalle tehdään. Pöhls (2015) ovat havainneet, että nykyisissä IoT-järjestelmissä tämä ongelma on otettu huomioon hyvin harvoin.

Ehdotettu ratkaisu, JSS (JSON Sensor Signatures), pyrkii kokonaisvaltaiseen, päästä päähän ulottuvaan eheyden varmistamiseen määrittämällä eheystarkistuksen algoritmin sekä datan rakenteen ja notaation. Datan notaatioksi on valittu JSON (Bray 2014), sillä se on erittäin suosittu Web-järjestelmissä, joihin IoT-järjestelmät useimmiten linkittyvät. Toinen varteenotettava vaihtoehto olisi ollut XML (Bray ym. 2008), mutta sen soveltuvuus ei ole yhtä hyvä, sillä XML-notaatioon kuuluu paljon ylimääräistä monimutkaisuutta, mikä haittaa käyttöä teknisesti rajoittuneessa IoT-ympäristössä. Jokaisen kommunikaatioviestin JSON-datarakenteeseen liitetään datasta laskettu allekirjoitus, jonka avulla sovelluskerroksella voidaan varmistaa datan eheys missä tahansa dataa käsittelevässä solmussa. Ratkaisun perusajatuksena on, että kaikki solmut varmistavat datan eheyden samalla kun data virtaa laitteilta

kohti erilaisia palveluita. Jos kuitenkin jokin solmu ei esimerkiksi vanhuutensa [1] vuoksi voi suorittaa eheystarkistusta, se pystyy käsittelemään datan täysin normaaliin tapaan myös ilman eheystarkistuksia. (Pöhls 2015)

Menetelmä ratkaisee monta IoT-ympäristöön liittyvää haastetta. Eritoten heterogeenisyyden tuomat ongelmat poistuvat, mikäli vain on mahdollista toteuttaa sovelluskerroksen eheystarkistukset riittävän moneen solmuun. Lisäksi ylläpitoa helpottaa se, että kaikissa solmuissa käytetään samaa algoritmiä datan allekirjoittamiseen. Kun varmistetaan, että data on säilynyt täysin samanlaisena alkuperäiseltä laitteelta putken viimeiselle palvelulle saakka, on luotu pohja koko järjestelmän luotettavuudelle.

3.3 ECC (Elliptic Curve Cryptography)

Datan eheystarkistuksessa käytettävät allekirjoitukset laskettiin alun perin RSA-algoritmeillä (Maletsky 2015). RSA-algoritmien haittapuolena on kuitenkin se, että avaimet ovat hyvin pitkiä ja vievät siten paljon muistia. Ratkaisuna pitkien avainten ongelmaan on ECC (Elliptic Curve Cryptography), jonka ansiosta vastaavat kryptografiset operaatiot pystytään toteuttamaan huomattavasti pienemmillä avaimien pituuksilla. Avaimien pituuden suhteen 256-bittinen ECC-avain on yhtä kryptografisesti kestävä kuin 3072-bittinen RSA-avain (Maletsky 2015). IoT-järjestelmissä on tärkeää käyttää ratkaisuja, jotka säästävät muistia, minkä seurauksena ECC-algoritmit ovat yleistyneet.

Viime vuosien aikana tutkijat ovat toteuttaneet useita ECC-algoritmeja hyödyntäviä kirjastoja, jotka on optimoitu sulautetuille ja rajoitetuille järjestelmille, kuten IoT-päätelaitteille. Liu ja Ning (2008) esittävät täysin konfiguroitavissa olevan kirjaston nimeltä TinyECC, joka voidaan optimoida kulloinkin käytössä oleville suoritusresursseille sopivaksi. Kirjasto sisältää lähes kaikki tiedossa olevat optimisaatiot ECC-operaatioille, joita käyttöönottamalla tai poistamalla käytöstä voidaan tehdä kompromisseja suoritusaikojen, RAM/ROM-vaatimusten ja energiankulutuksen välillä. TinyECC-kirjaston lähdekoodi on kirjoitettu nesC-ohjelmointikielellä (Gay ym. 2003) ja tietyissä kohdissa on käytetty assembly-koodia suorituksen tehostamiseksi.

1. Viitataan ohjelmistoihin, joiden ylläpito on lopetettu, engl. *legacy software*.

Szczechowiak ym. (2008) ovat kehittäneet vastaavan kirjaston nimeltä NanoECC, joka perustuu C- ja C++-kielillä toteutettuun MIRACL-kirjastoon (Scott 2003). Myös NanoECC:ssä on käytetty nesC-ohjelmointikieltä. NanoECC on optimoitu siten, että suoritusaika on ollut ensimmäisenä prioriteettina ja muistin käyttö toisena.

Cheneau (2014) on kehittänyt ecc-light-certificate -kirjaston, jota voidaan hyödyntää ECC-sertifikaattien säilömisessä siten, että ne vievät mahdollisimman vähän tilaa muistissa. Toteutus on tehty Contiki OS:lle, mutta sen luvataan toimivat millä tahansa alustalla, joka tukee C-kielen kääntäjää.

ECC-algoritmien alajoukko ECDSA (Elliptic Curve Digital Signature Algoritm) on erityisen käyttökelpoinen, kun halutaan varmistaa datan eheys IoT-ympäristössä. ECDSA-implementaatiot sisältävät kaksi eri toimenpidettä, allekirjoituksen luonnin ja allekirjoituksen varmentamisen (Yalçin 2016). Allekirjoitus luodaan käyttämällä salaista avainta, ja luodun allekirjoituksen oikeellisuuden voi varmentaa käyttämällä julkista avainta. Täten ECDSA-allekirjoitusta voidaan suoraan käyttää MAC-tarkistuksessa.

Huomionarvoista on, että ECC-kryptografia perustuu siihen, että tiettyjä matemaattisia operaatioita on vaikea suorittaa käänteisesti. On kuitenkin todistettu, että kvanttitietokone pystyy purkamaan ECC-algoritmin mekanismin polynomisessa ajassa (Shor 1997). Näin ollen kvanttitietokoneiden kehittymisen myötä on väistämätöntä, että ECC-algoritmit vanhenevat lähitulevaisuudessa. Kvanttilaskennan kestäviä kryptografisia algoritmejä tutkitaan, mutta merkittävää edistystä ei ole vielä saavutettu (Simonite 2016).

3.4 Datan eheyden varmistaminen osana IoT-ympäristön tietoturvaa

Datan eheyden varmistamisella on kaksi päätarkoitusta tietoturvan kannalta (Pöhls 2015):

1. Varmistetaan, että dataa ei ole modifioitu luvattomia keinoja käyttäen.
2. Varmistetaan, että data on peräisin sen alkuperäiseltä lähettäjältä eikä keltään muulta.

Mainitut varmistukset ovat erittäin olennaisia tietoturvan kokonaiskuvan kannalta. Monet IoT-ympäristössä tietoturvauhkia aiheuttavat hyökkäysmallit perustuvat juuri siihen, että hyökkääjä kykenee joko modifioimaan verkossa liikkuvaa dataa tai esittämään olevansa jokin luo-

tettu datan lähettäjä. Esimerkki tällaisesta hyökkäysmallista on man-in-the-middle -hyökkäys, jossa hyökkääjä kaappaa liikenteen kahden solmun välillä ja muokkaa datan sellaiseksi, että se aiheuttaa ei-toivottuja lopputuloksia. Hyvä datan eheyden varmistusmenetelmä estää kaikki edellämainitun kaltaiset hyökkäykset, kun muuttunutta dataa ei oteta lainkaan käsittelyyn.

Kun datan eheystarkistuksia käsitellään tietoturvan kontekstissa, on syytä mainita kryptografian merkitys. Sillä viitataan siihen, että data allekirjoitetaan kryptografisesti kestävää algoritmia käyttäen, jolloin hyökkääjä ei kykene luomaan omia MAC-allekirjoituksia luvattomasti muokatulle datalle. Jos data allekirjoitetaan jollakin helposti käännettävällä algoritmilla, niin hyökkääjän on helppo luoda oma funktio, joka muodostaa laillisen näköisen allekirjoituksen mille tahansa datalle. Jos taas data allekirjoitetaan kryptografisesti kestävällä menetelmällä, esimerkiksi 384-bittistä avainta käyttävällä ECDSA-algoritmillä, niin sen toistaminen on hyökkääjälle erittäin haastavaa. Eheyden säilymistä voidaan myös tukea kokonaisten viestien salaamisella, mutta salaus ei itsessään liity datan eheystarkistuksiin eikä datan eheyden säilyttäminen edellytä salausta (Pöhls 2015).

Alqassem ja Svetinovic (2014) nostavat datan eheyden yhdeksi viidestä IoT-tietoturvan kulmakivestä, jotka ovat:

- Pääsyn valvonta: Identifioinnin, autentikaation ja autorisaation täytyy toimia siten, että ainoastaan luvan saaneet entiteetit pystyvät kommunikoimaan IoT-systeemissä.
- Datan eheys: Protokollien avulla voidaan varmistaa, että datan täydellinen rakenne säilyy ja että kaikki data on kulkeutunut perille asti. Samalla täytyy kuitenkin sallia dynaamiset dataoperaatiot verkon joka solmussa.
- Kontekstuaalinen eheys: Datan keräyksen täytyy kunnioittaa henkilökohtaista yksityisyyttä kussakin kontekstissa. Lisäksi datan virtaukselle on asetettava sopivat rajoitteet, virtaa se sitten yhdessä kontekstissa tai usean eri kontekstin läpi.
- Tunkeutumisen havaitseminen: Järjestelmä havaitsee hyökkäyksen ja katkaisee hyökkääjältä pääsyn verkkoon. Havaitsemislogiikassa täytyy ottaa huomioon, että hyökkääjä saattaa olla myös verkossa autorisoitu entiteetti.
- Kiistämättömyys: Tietoliikenteessä hyödynnetään luotettavia kommunikaatiokanavia ja täten varmistetaan, että yksikään entiteetti ei pysty huijaamaan esimerkiksi piilotta-

malla oman identiteettinsä tai keskeyttämällä verkossa tapahtuvan operaation.

Listan perusteella voidaan havaita, että datan eheyden varmistamisella on suuri merkitys IoT-ympäristön tietoturvassa. Useasti mainitaan myös CIA-malli (Confidentiality, Integrity and Availability) eli luottamuksellisuus, eheys ja saavutettavuus (Kraijak ja Tuwanut 2015; Mahmoud ym. 2015). CIA-mallin idea vastaa aiempaa listausta, mutta se asettaa suuremman arvon myös sille, että kaikki tieto ja jokainen palvelu on kaikkien verkon entiteettien saavutettavissa. Tämä vaatimus vastaa hyvin IoT:in perusvisiota, jossa halutaan yhdistää mahdollisimman monta älykästä laitetta toisiinsa (Mahmoud ym. 2015).

Tarkoituksena ei ole, että jokainen edellä mainittu tietoturvavaatimus käsitellään erikseen ja ratkaistaan yksi kerrallaan. Hou ym. (2004) esittävät, kuinka pääsyn valvonta, datan eheyden varmistaminen ja kryptografisesti erittäin kestävä salausmekanismi voidaan yhdistää. Tällöin saadaan ratkaistua useita tietoturvan perusvaatimuksia kerralla, ja samalla varmistetaan eri osien saumaton yhteistoiminta. Kyseistä ratkaisua ei ole kehitetty yksinomaan IoT-ympäristöön, mutta tutkimuksen yhteydessä on kiinnitetty erityistä huomiota suorituskyky- ja muistivaatimuksiin, mikä puoltaa soveltuvuutta matalan suorituskyvyn ympäristöön.

Luvussa 2.2 mainittujen yleisten IoT-järjestelmiin liittyvien tieturvauhkien lisäksi datan eheystarkituksia vastaan voidaan hyökätä ajoitushyökkäyksellä (engl. *timing attack*). Ajoitushyökkäyksessä hyökkääjä lähettää massoittain erilaisia syötteitä järjestelmään, ja mittaa tarkan ajan, joka kuluu syötteen käsittelyyn. Suoritusaikojen perusteellisella analysoinnilla voidaan saada selville järjestelmän käyttämä salausavain. Perinteisesti ajoitushyökkäyksen käyttö on edellyttänyt suoraa pääsyä laitteistoon, mutta Brumley ja Boneh (2005) esittävät, kuinka ajoitushyökkäys voidaan suorittaa verkon yli. Hyökkäyksen tehokkuus demonstroidaan siten, että OpenSSL-kirjaston RSA-algoritmin salausavain saadaan selville noin kahdessa tunnissa. Tällainen verkon yli tapahtuva hyökkäys on erityisen vaarallinen IoT-verkoille, joiden valvonta on haastavaa.

4 Datan eheystarkistukset käytännön IoT-ympäristössä

IoT-ympäristössä on erityisen tärkeää testata tutkimuksen kohteena olevat konseptit käytännössä, sillä laitteet saattavat käyttäytyä oletetusta poikkeavalla tavalla. Tietoturvan tapauksessa erityisesti matala suoritinteho ja RAM-muistin määrä saattavat aiheuttaa sen, että teoriapohjalla perustellut tietoturvamenetelmät eivät ole käytännössä toteutuskelpoisia tai ovat epäkäytännöllisiä. Myös verkon latenssit tai huonolaatuiset yhteydet saattavat aiheuttaa sellaisia ongelmia, joita ei olisi ilman käytännön testausta havaittu.

Tämän tutkielman luvussa 3 selvitettiin, että IoT-ympäristössä datan eheys voidaan varmistaa kuljetuskerroksella tai sovelluskerroksella. Kirjallisuuslähteissä esitetään muutamia erityispiirteitä, jotka ovat ominaisia kunkin kerroksen eheystarkistuksille. Lähteissä ei kuitenkaan oteta vahvasti kantaa siihen, kumpi kerros soveltuu IoT-käyttötarkoituksiin paremmin, tai että onko aina ehdottomasti syytä sisällyttää molempien kerrosten tarkistukset. Tämän tutkielman konstruktiivisen osuuden eräs tavoite oli selvittää, mitä hyötyjä kunkin kerroksen eheystarkistukset tuovat kokonaisuudelle ja sitä kautta analysoida sopivuutta IoT-ympäristöön.

Eri kerrosten vertailun lisäksi konstruktiivisessa osuudessa haluttiin saada tieto siitä, mitä hyötyjä datan eheyden varmistamisesta on tietoturvan kasvattamisen lisäksi. Tämän vuoksi laadittiin erilaisia hypoteettisia testitilanteita, joissa datan eheys vaarantuu joko tietoturvahyökkäyksen aiheuttamana tai muusta syystä. Testitilanteet on kuvailtu luvussa 4.5.

Testausympäristö varustettiin palvelinkoneella ja kahdella Raspberry Pi 2 -tietokoneella, jotka simuloivat IoT-laitteita. Raspberry Pi -tietokoneissa on tiedostetusti enemmän suorituskykyä kuin tavanomaisessa IoT-päätelaitteessa, ja tämä on otettu huomioon testien laadinnassa ja analysoinnissa. Muilta osin Raspberry Pi -tietokoneet soveltuivat IoT-ympäristön testaukseen erinomaisesti. Testauksen aikana verkossa liikkui myös testaustilanteen ulkopuolista liikennettä, jotta olosuhteet olisivat mahdollisimman realistiset.

4.1 Testauksessa käytetyn mittausdatan rakenne

Laitteet lähettivät palvelimelle mittausdataa, jonka JSON-muotoinen perusrakenne on seuraava:

```
{
  "data":12.4,
  "measurementId":3,
  "nodeId":42,
}
```

Attribuutti `"data"` kuvaa itse mittauksen arvoa, `"measurementId"` on laitekohtainen juokseva numerointi mittaukselle ja `"nodeId"` kertoo laitteen tunnuksen. Testiympäristössä `"data"` ei vastannut mitään oikeaa mittausta, vaan se oli laitteiden generoima satunnainen arvo. Oikeassa tilanteessa se voisi olla esimerkiksi lämpötila, ilmankosteus, laitteen ohi ajaneiden autojen lukumäärä, ihmisen syke jne. Attribuutti `"measurementId"` kertoo, monesko mittausarvo on ko. laitteella menossa. Lukuarvo kasvaa jokaisen mittauksen jälkeen yhdellä ja se nollataan luvun 255 jälkeen. Se vastaa osaltaan myös datan eheydestä, sillä palvelin hylkää väärällä `"measurementId"`-arvolla saapuvat mittaukset. Tällä tavoin hankaloitetaan hyvin yksinkertaisella tavalla datan uudelleenlähetykseen perustuvien hyökkäysten hyödyntämistä.

4.2 Testauksessa käytetty datan eheyden varmistusmenetelmä

Laitteiden ja palvelimen välisessä kommunikaatiossa sovellettiin JSS (JSON Sensor Signatures) -formaattia, jonka on esittänyt Pöhls (2015). Se on sovelluskerroksella toimiva datan eheyden varmistamisen menetelmä, jossa datalle luodaan allekirjoitus valitulla algoritmillä, mutta data pidetään saatavilla myös vaihtoehtoisesti ilman eheystarkitustusta. Formaatilla on neljä päätavoitetta:

1. Allekirjoitetun datan pitää olla samalla tavalla saavutettavissa kuin jos se ei olisi allekirjoitettu.
2. Säilytetään JSON-notaation yksinkertaisuus, sillä se on todennäköisesti syy JSON-

notaation laajaan levinneisyyteen.

3. Allekirjoituksen täytyy tuottaa mahdollisimman vähän ylimääräistä metadataa kommunikaation rakenteeseen.
4. Allekirjoitus pystytään luomaan ja varmistamaan rajoitetun suorituskyvyn laitteissa.

Käytännössä kolme ensimmäistä tavoitetta saavutetaan siten, että allekirjoitetun datan JSON-formaatti on seuraavanlainen:

```
{
  "jss.protected":{"alg":"NIST192p"},
  "data":12.4,
  "measurementId":3,
  "nodeId":42,
  "signature":"K8_hgByDUaUX...kdyDgR"
}
```

Rivinvaihdot lisätty helpottamaan visuaalisen esityksen tulkintaa.

Tämän esimerkin JSON-objektissa `"data"`, `"measurementId"` ja `"nodeId"` ovat itse mittausdatan attribuutteja, kun taas `"jss.protected"` ja `"signature"` ovat JSS-formaatin vaatimia metatietoja, joissa määritellään allekirjoituksessa käytetty algoritmi ja tärkeimpänä datalle laskettu allekirjoitus. Merkittävää on, että kaikki mittausdata on säilytetty JSON-objektin "juuritasolla", jotta JSS-formaatin ensimmäinen tavoite täyttyy. Toinen merkittävä asia on se, että mittausdatan attribuutit on järjestetty aakkosjärjestykseen, jotta saavutetaan yksikäsitteinen esitys datalle. Datan muuttamista ennalta sovittuun yksikäsitteiseen muotoon kutsutaan myös kanonikalisoinniksi.

Testeissä allekirjoituksen algoritmiksi valittiin ECDSA (Elliptic Curve Digital Signature Algorithm) 192-bittisellä avaimella. Algoritmiä kutsutaan myös nimellä NIST192p, sillä se on NIST:in (National Institute of Standards and Technology) kehittämä. ECDSA itsessään on joukko erilaisia funktioita, jotka asettuvat aiemmin mainitun kattotermin ECC:n alle. Muita ECC-toteutuksia ovat esimerkiksi ECDH, ECIES ja ECMQV ja niitä kaikkia käytetään kryptografiassa.

Käytännössä ECDSA-algoritmin hyödyntäminen vaatii, että data allekirjoitetaan salaisella avaimella, joka ei saa paljastua. Allekirjoituksen oikeelllisuuden varmistus tehdään julkisella avaimella, jonka paljastumisesta kolmansille osapuolille ei ole haittaa. Toimiva kokonaisuus syntyy, kun palvelimelle tallennetaan laitekohtaiset julkiset avaimet, ja salaiset avaimet pidetään laitteiden muistissa. Julkisia avaimia voidaan jakaa kaikkien palveluiden kanssa, jotka ovat laitteiden lähettämän datan kanssa tekemisissä.

Itse allekirjoitusprosessi sisältää monta vaihetta:

1. (Tehdään ainoastaan kerran, ennen ensimmäistä lähetystä) Luodaan salainen allekirjoitusavain käyttäen määritettyä algoritmiä, esimerkiksi NIST192p, ja tallennetaan se muistiin.
2. Luodaan JSS-otsake `"jss.protected"`, jossa kerrotaan allekirjoituksessa käytetty algoritmi.
3. Enkoodataan koko luotu JSS-otsake base64url-enkoodauksella: `Impzcy5wc...5MnAifQ`.
4. Kanonikalisoidaan mittausdata (Järjestetään attribuutit aakkosjärjestykseen ja poistetaan kaikki välilyönnit sekä muut turhat merkit).
5. Enkoodataan mittausdata base64url-enkoodauksella: `ImRhdGEiO...JZCI6NDI`.
6. Liitetään base64-enkoodattu JSS-otsake base64-enkoodattuun mittausdataan ja lisätään väliin merkki '.': `Impzcy5wc...5MnAifQ.ImRhdGEiO...JZCI6NDI`.
7. Lasketaan SHA-256 -tiiviste liitetylle merkkijonolle.
8. Allekirjoitetaan syntynyt tiiviste valitulla algoritmillä, esimerkiksi NIST192p.
9. Enkoodataan syntynyt allekirjoitus base64url-enkoodauksella: `eHlnT0k0d...SW95QjhpbQ`.
10. Liitetään mittausdatan JSON-esityksen alkuun JSS-otsake ja loppuun allekirjoitus.

Allekirjoitusprosessin jälkeen data voidaan lähettää vastaanottajalle. Vastaanottajan prosessi allekirjoituksen varmentamiseksi on samankaltainen kuin allekirjoitusprosessi:

1. Vastaanotetusta JSON-objektista erotetaan JSS-otsake `"jss.protected"`.
2. JSS-otsakkeesta selvitetään käytetty algoritmi ja alustetaan kyseisen algoritmin tarvitsemat funktiot.

3. Enkoodataan JSS-otsake base64url-enkoodauksella: `Impzcy5wc...5MnAifQ`.
4. Erotetaan JSON-objektista mittausdata:
   `"data":12.4,"measurementId":3,"nodeId":42`
5. Enkoodataan mittausdata base64url-enkoodauksella: `ImRhdGEiO...JZCI6NDI`.
6. Erotetaan JSON-objektista allekirjoitus: `eHlnT0k0d...SW95QjhpbQ`.
7. Liitetään base64-enkoodattu JSS-otsake base64-enkoodattuun mittausdataan ja lisätään väliin merkki '.': `Impzcy5wc...5MnAifQ.ImRhdGEiO...JZCI6NDI`.
8. Lasketaan SHA-256 -tiiviste liitetylle merkkijonolle.
9. Ladataan lähettäjää vastaava julkinen avain muistista.
10. Varmennetaan tiiviste käyttäen lähettäjän julkista avainta ja JSS-otsakkeessa määritettyä algoritmiä, esimerkiksi NIST192p.

Jos viimeisessä kohdassa varmentaminen epäonnistuu, eli allekirjoitus ei vastannut dataa, niin koko vastaanotetun JSON-objektin käsittely lopetetaan ja data hylätään kelvottomana. Toisaalta onnistuneen varmentamisen jälkeen voidaan olla varma siitä, että

1. dataa ei ole peukaloitu missään kuljetuksen vaiheessa, ja että
2. data on peräisin juuri siltä osapuolelta, jolta sen on ilmoitettu olevan peräisin.

Yhtäkään viestiä ei ole syytä lähettää ilman allekirjoitusta. Esimerkiksi viestin perillemenon kuittausten allekirjoittaminen ja varmentaminen on yhtä tärkeää kuin mittausdatan allekirjoittaminen ja varmentaminen. Ilman kuittausten allekirjoitusta mahdollinen samassa verkossa toimiva hyökkääjä voisi tekaista kuittauksia ja lähettää niitä laitteille, jolloin dataa häviäisi.

4.3 Testatun palvelinohjelmiston arkkitehtuuri

Palvelimen ohjelmisto ohjelmoitiin Python-ohjelmointikielellä käyttäen kielen versiota 2.7. Pythonin valintaa puolsivat soveltuvuus socket-pohjaiselle ohjelmoinnille, helppo käyttöönotto, kieleen sisäänrakennettu JSON-parsinta ja se, että kielelle on olemassa valmiit kirjastot ECC-algoritmeille sekä Firebase-pilvitietokannalle, johon laitteiden lähettämä data tallennettiin testien aikana. Näiden ominaisuuksien avulla kokonaisuudesta saatiin tiivis ja varmatoiminen. Pythonista on olemassa myös uudempi versio 3.5, mutta sen yhteensopivuut-

ta kaikkien kirjastojen kanssa ei voitu etukäteen taata, joten valittiin varmasti yhteensopiva versio.

Laitteiden ja palvelimen välisessä liikenteessä käytettiin TCP-protokollaa. Liikennöinti toteutettiin hyödyntäen Pythonin `socket`-luokkaa ja tarvittaessa `OpenSSL` -luokan funktioita `SSL.Context()` sekä `SSL.Connection()`, joilla soketti saatiin tukemaan TLS-protokollaa. TLS-protokollan autentikoinnissa käytettiin itse allekirjoitettua (engl. *self-signed*) sertifikaattia, mikä oli testausympäristössä täysin riittävä ratkaisu.

Testausta varten palvelin ohjelmoitiin kuuntelemaan neljää eri TCP-porttia (11000, 11001, 11002 ja 11003), joista kukin noudatti erilaista konfiguraatiota:

- Portti 11000: TLS ei käytössä, sovelluskerroksen eheystarkistus ei käytössä. Tässä portissa ei siis ollut käytössä mitään datan eheystarkistuksia, jolloin se toimi vertailukohtana kaikille muille eheystarkistuksille.
- Portti 11001: TLS käytössä, sovelluskerroksen eheystarkistus ei käytössä. Tämän portin tarkoituksena oli havainnollistaa, miten TLS-protokolla soveltuu datan eheyden varmistamiseen sellaisenaan.
- Portti 11002: TLS ei käytössä, sovelluskerroksen eheystarkistus käytössä. Tämän portin tarkoituksema oli havainnollistaa, miten sovelluskerroksen eheystarkistus suoriutuu silloin, kun sitä tukemassa ei ole kuljetuskerroksen protokollaa.
- Portti 11003: TLS käytössä, sovelluskerroksen eheystarkistus käytössä. Tämän portin tarkoituksena oli havainnollistaa, millaisia tuloksia saavutetaan silloin, kun käytössä on kaksi merkittävintä datan eheyden varmistamisen menetelmää.

Portti	TLS	Sovelluskerroksen eheystarkistus (JSS)
11000	Ei	Ei
11001	Kyllä	Ei
11002	Ei	Kyllä
11003	Kyllä	Kyllä

Taulukko 1. Palvelinohjelmiston porttien konfiguraatio.

JSS-formaatin vaatimiin ECDSA-allekirjoituksiin palvelimella käytettiin python-ecdsa -kir-

jastoa (Warner 2010). Se sisältää tuen kaikille yleisimmille ECDSA-algoritmeille: prime192v1, secp224r1, prime256v1, secp384r1 ja secp521r1. Lisäksi kirjasto tukee Bitcoinien kryptauksessa käytettävää secp256k1-algoritmiä. Algoritmien nimessä oleva luku kertoo käytettävän avaimen pituuden bitteinä. Warner (2010) raportoi laskennan olevan noin 30-40 kertaa hitaampi kuin korkealaatuisen C++-implementaation, mikä on tyypillinen ero C++- ja Python-ohjelmien välillä.

Palvelimen vastaanottama ja hyväksymä data lähetettiin Firebase-pilvitietokantaan. Se on NoSQL-tietokanta, joka tallentaa datan suoraan JSON-muodossa, mikä helpottaa datan jatkokäsittelyä huomattavasti. Kun data tallennetaan JSON-muodossa, tietokantaan voidaan tallentaa JSS-allekirjoitus sellaisenaan osana tallennettavaa objektia. Tällä tavoin mikä tahansa dataa tietokannasta lukeva palvelu voi myös varmistaa datan eheyden sovelluskerroksen tarkistuksella. Riittää, että palvelulla on tiedossa datan allekirjoittajan (IoT-päätelaitteen) julkinen ECDSA-avain. Datan ei myöskään tarvitse kiertää enää Python-palvelimen kautta, vaan palvelut voivat suoraan käyttää Firebasen rajapintoja.

Tärkeä asia IoT-ympäristössä on myös se, että laitteelle kuitataan datan vastaanotto vasta sen jälkeen, kun ollaan varmoja, että data on onnistuneesti tallennettu tietokantaan. Jos palvelin kuittaisi datan vastaanotetuksi ennen, kuin se on tallennettu tietokantaan, niin tietokannan häiriö voisi aiheuttaa datan lopullisen katoamisen. Nyt laitteelle annetaan mahdollisuus odottaa palvelimelta kuittausta, ja jos kuittausta ei kuulu, data voidaan lähettää uudelleen. Toiminnallisuutta kutsutaan päästä-päähän kuittaamiseksi. Sama vaikutus saavutettaisiin sillä, että palvelin pitää puskurissaan kaiken laitteilta vastaanotetun datan, kunnes kyseinen data on kuitattu tallennetuksi tietokantaan. Tällöin puskurointikuorma siirrettäisiin laitteilta palvelimelle. Järjestelmän ominaisuudet, kuten laitteiden lukumäärä, mittausdatan sisältämien muuttujien lukumäärä ja datan lähetyksen tiheys, määrittävät sen, kumpi toteutus on missäkin tilanteessa parempi.

4.4 Testatun IoT-laitteiston ohjelmistoarkkitehtuuri

Myös Raspberry Pi -päätelaitteiden ohjelmointikieleksi valittiin Python 2.7. Suurin tekijä valinnassa oli se, että ECDSA-toiminnoille löytyi valmis toteutus palvelimen ohjelmistosta, jol-

loin olemassa olevaa ohjelmakoodia voitiin hyödyntää uudelleen. Toiseksi Python-ohjelmien selkeästi hitaampi suoritusnopeus verrattuna esimerkiksi C-ohjelmiin (joita IoT-laitteet usein ajavat) antoi mahdollisuuden päästä lähemmäksi todellisen IoT-ympäristön suoritusarvoja.

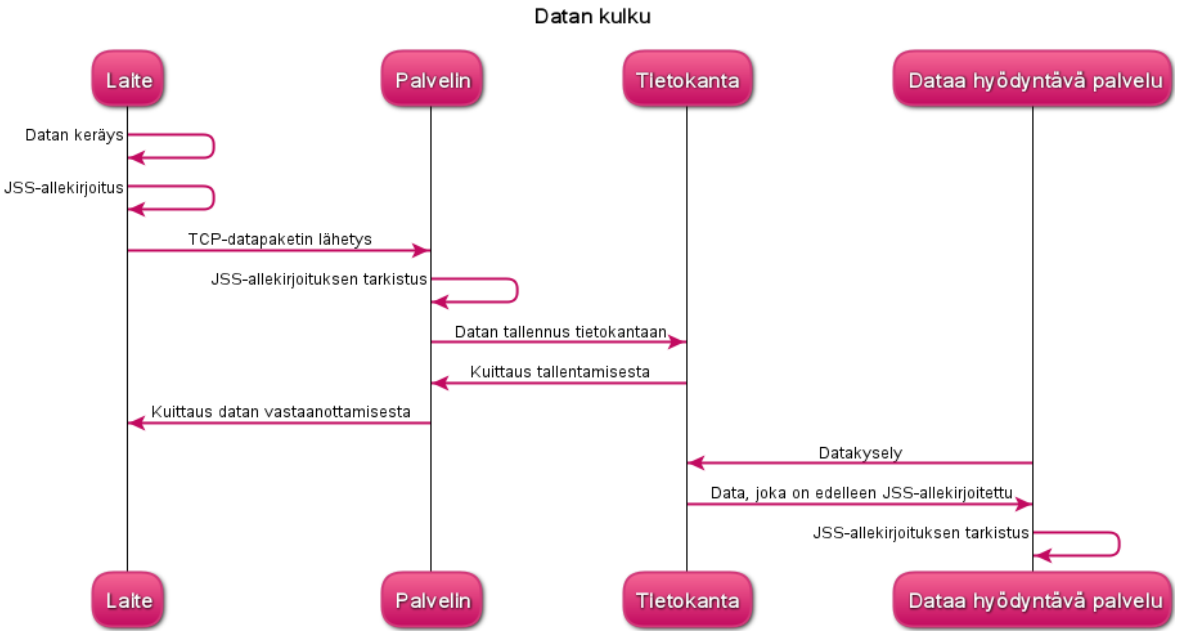
Molempiin Rasberry Pi -laitteisiin asennettiin 4 eri ohjelmaa, joiden toiminta vastasi edellä mainittuja palvelimen porttien konfiguraatioita. Kukin ohjelma generoi ja lähetti 2 sekunnin välein mittausdataa tiettyyn palvelimen porttiin. Täten voitiin helposti kontrolloida, mihin portteihin dataa lähetettiin milläkin hetkellä. Lähettämisen jälkeen laitteet jäivät odottamaan kuittausta datan onnistuneesta vastaanottamisesta. Jos kuittausta ei tullut 5 sekunnin sisällä, data lähetettiin uudelleen. Toteutus vaati jonkin verran puskurointikapasiteettia laitteelta, mutta kuvatun kaltaisessa tilanteessa kapasiteettivaatimus rajoittui muutamiin kymmeniin tavuihin. Puskuroinnin kannalta tärkeää oli huomioida, että puskurin täyttyminen ei aiheuttanut sivuvaikutuksia vaan se käsiteltiin järkevästi. Järkevä käsittely tarkoitti tässä tapauksessa sitä, että laite pyyhki vanhimman puskurissa olevan datan pois muistista.

Kuten palvelimella, myös laitteiden tapauksessa liikennöintiin käytettiin `socket`-luokkaa. Lisäksi TLS-suojausta tarvittaessa käytettiin Pythonin natiivia `ssl.wrap_socket()` -funktiota. JSS-formaatin vaatimiin ECDSA-allekirjoituksiin käytettiin niin ikään python-ecdsa -kirjastoa (Warner 2010).

Palvelimen, laitteiden ja muiden osapuolien roolit datan virtauksen suhteen käyvät ilmi sekvenssikaaviosta 2.

4.5 Testattujen tilanteiden kuvaukset

Datan eheystarkistuksien testaamista varten laadittiin neljä erilaista testiskenaariota, jossa datan eheys vaarantuu. Näissä skenaarioissa aiemmin esitellyt datan eheyden varmistusmenetelmät testattiin käytännössä tavoitteena selvittää, millaisia ongelmia ne pystyvät ratkaisemaan. Lopullisena päämääränä oli löytää konfiguraatio, jota käyttämällä kaikkien neljän testiskenaarion aiheuttamat vaaratilanteet pystyttäisiin ehkäisemään. Kaikissa tilantessa oletettiin, että hyökkääjä tietää palvelimen IP-osoitteen ja pystyy lähettämään sinne dataa palomuurin ohi, mutta ei ole saanut haltuunsa salaisia tietoja kuten ECDSA-salausavainta tai SSL-sertifikaattia. Neljä testattua tilannetta olivat seuraavanlaiset:



Kuvio 2. Datan kulku järjestelmässä, jossa eheys on varmistettu päästä päähän.

1. Hyökkääjä pyrkii aiheuttamaan järjestelmälle haittaa siten, että lähettää palvelimelle itse generoitua dataa jonkin muun laitteen nimissä (matkimishyökkäys, Adams (2011a)). Hyökkääjän tarkoituksena on se, että tietokantaan pääsee ylimääräistä dataa, jolloin ei voida tietää, mikä data on alun perin ollut aitoa.
2. Hyökkääjä on kyennyt murtautumaan verkkoon siten, että pystyy kaappaamaan laitteiden lähettämän liikenteen, muokkaamaan sitä ja lähettämään sen muokattuna palvelimelle (man-in-the-middle -hyökkäys, Desmedt (2011)). Palvelin ei suoraan voi nähdä hyökkääjän olemassaoloa, sillä data näyttäisi tulevan täysin laillisesta lähteestä.
3. Hyökkääjä on kyennyt murtautumaan verkkoon siten, että pystyy kaappaamaan laitteiden lähettämän liikenteen. Hyökkääjä toistaa kaappaamansa viestiliikenteen sillä tarkoituksella, että palvelin hyväksyisi lähetetyn datan toistamiseen (replay-hyökkäys, Adams (2011b)). Kuten ensimmäisessä testitapauksessa, myös tässä testitapauksessa tavoitteena on saada tietokantaan epäaitoa dataa. Replay-hyökkäys on eräs man-in-the-middle -hyökkäyksen variantti.
4. Laitteen ohjelmistossa on tarkoitukseton virhe, jonka takia tiettyjen ehtojen täyttyessä laitteelta lähtevä data on väärän mallista. Tilanteessa ei ole aktiivista hyökkääjää.

4.6 Tilanteiden läpivienti

Ensimmäisessä testitilanteessa hyökkääjällä ei ollut mahdollisuutta takaisinmallintaa eheystarkistuksia, mutta mittausdatan rakenne oli hyökkääjän tiedossa. Tapaus voisi seurata esimerkiksi siitä, että hyökkääjä on saanut käsiinsä laitteen ja palvelimen välistä TCP-kommunikaatiota kuvaavan dokumentin. Hyökkäystä varten toinen Raspberry Pi -laitteista muokattiin toimimaan "matkijalaitteena", jossa oli toimiva kommunikaatioprotokolla, muttei datan allekirjoitusta tai TLS-yhteyden mahdollisuutta, sillä hyökkääjä ei ollut saanut haltuunsa ECDSA-salausavainta tai SSL-sertifikaattia. Matkijalaitteelle valittiin yksi `"nodeId"`-arvo, joka kuvasi sen laitteen tunnusta, jota aiottiin matkia. Sen jälkeen matkijalaite lähetti palvelimelle lukuisia satunnaisia mittausdatoja myös satunnaisilla `"measurementId"`-arvoilla tarkoituksenaan, että ainakin osa päätyisi palvelimen käsittelyyn. JSS-eheystarkistuksia käyttäviin portteihin yritettiin myös liittää satunnaisesti generoitu ”allekirjoitus”-merkkijono mittausdatan mukana. Matkijalaitteen toiminnan aikana seurattiin palvelimen lokitietoja ja sitä, päätyikö dataa tietokantaan saakka.

Toinen testitilanne toteutettiin siten, että toiselta Raspberry Pi -laitteelta otettiin väliaikaisesti pois kaikki IoT-laitetta mallintavat toiminnot ja sen sijaan se valjastettiin toiminaan toisen Raspberry Pi:n ja palvelimen välissä ikään kuin liikenteen kaappaajana ja hyökkääjänä. Kaikki kommunikaatio asetettiin kulkemaan hyökkääjälaitteen kautta. Kun hyökkääjälaite vastaanotti dataa, jonka oli määrä mennä suoraan toiselta laitteelta palvelimelle, siitä muokattiin `"data"`-kentän arvoa. Muokkaamisen jälkeen data edelleenlähetettiin palvelimelle. Kuten aiemmin, toiminnan aikana seurattiin palvelimen lokitietoja ja sitä, päätyikö dataa tietokantaan saakka.

Kolmannessa testitilanteessa oli kyse siitä, että hyökkääjä ei edes yrittänyt kiertää eheystarkistuksia, vaan käytti olemassaolevia allekirjoituksia hyväkseen. Kun dataa ei muokattu millään tavalla, hyökkääjän osalta oli perusteltua olettaa, että sama allekirjoitus hyväksyttäisiin uudelleen. Koska TCP-protokollassa on sekvenssinumeroihin perustuva suojaus triviaaleja replay-hyökkäyksiä vastaan (Postel 1981), niin hyökkäys täytyi toteuttaa sovelluskerroksen viestien uudelleenlähetyksillä — pelkkä TCP-liikenteen kaappaus ja uudelleenlähetys ei olisi voinut toimia. Siispä uudelleenlähetystä varten käytettiin toisen testitilanteen asettelua, mutta hyökkääjälaitteen ohjelmistoa muutettiin siten, että viestien muokkaaminen

vaihdettiin viestien toistamiseksi. Toistettava data saatiin Wireshark-ohjelmalla kaappaamalla asianomaisen portin TCP-liikennettä noin 10 sekunnin ajan ja kopioimalla TCP-pakettien data-osuus. Laite asetettiin uudelleenlähettämään tätä kopioitua dataa. Uudelleenlähetyksen aikana seurattiin palvelimen lokitietoja ja sitä, päätyikö dataa tietokantaan saakka.

Neljännessä testitilanteessa laitteen normaalia ohjelmakoodia muokattiin siten, että satunnaisesti joko `"measurementId"` tai `"nodeId"` oli väärä, tai että jokin kenttä puuttui lähtevästä mittausdatasta kokonaan. Jo etukäteen oli todennäköistä, etteivät eheystarkistukset kykenisi havaitsemaan tämänkaltaisia virheitä vaan niiden tarkistus olisi toteutettava muualla palvelimen ohjelmistossa. Siitä huolimatta asia päätettiin varmistaa ja testata, jotta heräisi ajatuksia sen suhteen, kuinka ongelma olisi mahdollista ratkaista. Laite asetettiin lähettämään satunnaisesti virheellistä dataa ja palvelimen lokitietoja sekä tietokantaa seurattiin.

5 Tulokset

Tutkielman käytännön osuuden tavoitteena oli selvittää, kuinka hyvin tietyt datan eheyden varmistusmenetelmät toimivat. Päämääränä oli eristää olemassaolevat menetelmät ja sitä kautta saavuttaa yksiselitteinen kyllä/ei -tieto siitä, auttaako tietty datan eheyden varmistusmenetelmä tietynlaisessa tilanteessa. Yksiselitteisen tiedon lisäksi haluttiin analysoida menetelmien käyttäytymistä monitahoisesti ja raportoida kaikenlaiset huomionarvoiset seikat, joita testauksen aikana havaittiin. Näitä tietoja yhdistämällä saatiin selville, minkälaisia datan eheyden varmistusmenetelmiä tulisi vaatia kaikenlaiselta IoT-ympäristössä toimivalta kommunikaatiolta.

Toinen tavoite oli tuottaa toistettava malli, jota voitaisiin käyttää tulevaisuudessa, kun testataan uusia IoT-ympäristöön suunniteltuja datan eheyden varmistamismenetelmiä. Tämän tavoitteen suhteen on kuitenkin selvää, että testiskenaarioita tarvitaan lukumäärällisesti enemmän, mikäli halutaan suorittaa kokonaisvaltaista hyväksymistestausta jollekin menetelmälle. Tämä tutkielma voi kuitenkin toimia perusrunkona vastaavalle testaukselle.

5.1 Eri menetelmien käyttäytyminen testatuissa tilanteissa

Jokainen neljästä testitilanteesta ajettiin läpi vähintään neljä kertaa, yhdesti kuhunkin palvelimen eri konfiguraatioon (porttiin). Alalukuihin on eritelty se, pääsikö peukaloitu tai rikkinäinen data eheystarkistusten läpi kyseisessä testitilanteessa. Tämä ominaisuus kuvantaa sen, onko menetelmästä lopullisesti absoluuttista hyötyä tietyssä tilanteessa, mutta tutkimuksessa pidettiin tärkeänä huomioida myös muita kokonaisuuteen vaikuttavia seikkoja. Tämän vuoksi alalukuihin on myös kirjattu kaikki testitilanteissa tehdyt havainnot merkittävistä löydöksistä sivuhuomioihin.

5.1.1 Matkimishyökkäyksen analysointi

Seuraavassa on eritelty matkimishyökkäys-testitapauksen huomiot porttikohtaisesti:

- Portti 11000 (TLS ei käytössä, JSS ei käytössä): Kaikenlaisen autentikoinnin puuttues-

sa matkimisen tuloksena syntynyt data päätyi palvelimen käsittelyyn ja tietokantaan asti. Palvelin käytännössä "luuli", että datan on lähettänyt se osapuoli, jota hyökkääjä matki. Lokiin ei jäänyt mitään merkintää siitä, että jotakin epätavallista olisi tapahtunut.

- Portti 11001 (TLS käytössä, JSS ei käytössä): Hyökkäystä kokeiltiin sekä kokonaan ilman TLS-protokollaa että tekaistulla mutta sinänsä validilla SSL-sertifikaatilla. Kun palvelin odotti TLS-kommunikaatiota, mutta hyökkääjälaite käytti paljasta yhteyttä, niin TLS-kättely luonnollisesti epännostui eikä hyökkääjälaite saanut palvelimeen edes yhteyttä. Palvelimen lokiin jäi rivi "wrong SSL version number", mikä viittaa siihen, että hyökkääjälaitteen ohjelmiston sokettiabstraktio yritti ottaa yhteyttä suojatulla oletusprotokollalla, joka on SSLv2.3 tai SSLv3. Kun hyökkääjälaitteeseen toteutettiin TLS-protokolla tekaistulla SSL-sertifikaatilla, niin palvelin lokitti "key verify failed"ja katkaisi yhteydenottopyynnön niin kuin oli oletettavaa. Tietokantaan ei päätynyt kummassakaan tapauksessa dataa, eli eheyden varmistusmenetelmät toimivat erinomaisesti.
- Portti 11002 (TLS ei käytössä, JSS käytössä): Hyökkäystä kokeiltiin sekä kokonaan ilman JSS-allekirjoitusta että tekaistulla JSS-allekirjoituksella. Ilman JSS-allekirjoitusta palvelin havaitsi heti, että allekirjoitus puuttuu ja katkaisi yhteyden laitteeseen. Näin ollen laitteen lähettämä data ei päätynyt minkäänlaiseen käsittelyyn. Palvelimen lokissa oli viesti siitä, että tarvittava otsake "jss.protected" puuttui. Tekaistulla JSS-allekirjoituksella datan käsittely pääsi hieman pidemmälle, mutta pysähtyi kohtaan, jossa JSS-allekirjoitus tarkastetaan. Palvelin lokitti, että JSS-allekirjoitus ei ollut validi. Täten myöskään tässä tapauksessa matkittu data ei päässyt mittausdatan käsittelyvaiheeseen. Siispä tietokantaan asti ei päätynyt yhtään dataa, mikä on erinomainen tulos.
- Portti 11003 (TLS käytössä, JSS käytössä): Testitapaus päätyi täsmälleen samoihin tuloksiin kuin portin 11001 tapaus, sillä sovelluskerroksen käsittelyyn ei päästy missään vaiheessa. Kuljetuskerroksen eheystarkistus toimi jälleen erinomaisesti.

Toteutettu hyökkäys oli tässä testitapauksessa niin triviaali, että joko kuljetus- tai sovelluskerroksen tarkistus oli täysin riittävä eliminoimaan hyökkäysuhan. Kuitenkin ilman mitään eheystarkistuksia hyökkäys pääsi järjestelmän läpi, joten vähintään jomman kumman kerroksen eheystarkistus on välttämätön.

5.1.2 Man-in-the-middle -hyökkäyksen analysointi

Seuraavassa on eritelty man-in-the-middle -hyökkäystapauksen huomiot porttikohtaisesti:

- Portti 11000 (TLS ei käytössä, JSS ei käytössä): Hyökkääjälaite kaappasi palvelimelle matkalla olleen viestin, josta se kykeni lukemaan selkokielisen JSON-datan ja muokkaamaan sitä. Muokattu data lähetettiin palvelimen porttiin, ja palvelin luki datan aivan kuin se olisi ollut alkuperäisen laitteen lähettämää. Tässäkään tapauksessa palvelimen lokiin ei jäänyt mitään merkintää siitä, että jotakin epätavallista olisi tapahtunut.
- Portti 11001 (TLS käytössä, JSS ei käytössä): Koska TLS-protokolla salaa liikenteen, hyökkääjälaite ei enää kyennyt lukemaan viestistä selkokielistä JSON-dataa. Näin ollen JSON-viestiä ei pystytty muokkaamaan siten, että muokkauksen vaikutukset olisivat kontrolloitavissa. Viestin kaappamisen jälkeen kuitenkin yritettiin muokata satunnaisia merkkejä tavuvirrasta. Tällöin TLS-protokollaan sisäänrakennettu MAC-tarkistus havaitsi datan muuttuneen matkalla, ja palvelimen lokiin kirjautuikin rivi "MAC verify failed". Tietokantaan ei päätynyt dataa, vaan palvelin katkaisi yhteyden heti kun MAC-tarkistus palautti virheen. Siispä eheyden varmistus toimi erinomaisesti.
- Portti 11002 (TLS ei käytössä, JSS käytössä): Hyökkääjälaite kaappasi palvelimelle matkalla olleen viestin, josta se kykeni lukemaan selkokielisen JSON-datan ja muokkaamaan sitä. Muokattu data lähetettiin palvelimen porttiin. Palvelimelle toteutettu JSS-allekirjoituksen tarkistus kuitenkin havaitsi datan muuttuneen, ja määrityksen mukaisesti palvelin katkaisi yhteyden välittömästi. Palvelin lokitti, että JSS-allekirjoitus ei ollut validi, eikä täten tietokantaan päätynyt dataa. Testi uusittiin useita kertoja, jotta mahdollinen törmäyskestävyyteen liittyvä haavoittuvuus allekirjoituksen algoritmissa olisi saattanut tulla esille. Haavoittuvuutta ei löytynyt, joten eheyden varmistuksen kirjattiin toimineen erinomaisesti.
- Portti 11003 (TLS käytössä, JSS käytössä): Testitapaus päätyi täsmälleen samoihin tuloksiin kuin portin 11001 tapaus, sillä sovelluskerroksen käsittelyyn ei päästy missään vaiheessa. Kuljetuskerroksen eheystarkistus toimi jälleen erinomaisesti.

Tämä testitapaus ei missään nimessä ollut enää triviaali, mutta osoitti, että jomman kumman kerroksen eheystarkistus on riittävä tyypillistä man-in-the-middle -hyökkäystä vastaan. Huomion arvoista on, että hyökkääjän on mahdollista purkaa TLS-protokollan salaus, mikä-

li jokin TLS:n perusehdoista ei täyty [1]. Tämä kuitenkin pätee vain salaukseen eikä sisäänrakennettuihin MAC-tarkistuksiin, joten datan eheyden näkökulmasta kuljetuskerroksen ja sovelluskerroksen tarkistukset ovat tämän testin valossa yhtä varmat. Vähintään toisen kerroksen eheystarkistus kuitenkin vaaditaan, sillä kokonaan ilman eheystarkistuksia hyökkäys pääsi järjestelmän läpi.

5.1.3 Replay-hyökkäyksen analysointi

Seuraavassa on eritelty replay-hyökkäys -testitapauksen huomiot porttikohtaisesti:

- Portti 11000 (TLS ei käytössä, JSS ei käytössä): Hyökkääjälaite kaappasi palvelimelle matkalla olleen viestin, josta eristettiin payload-osuus Wiresharkin avulla. Tämä eristetty data lähetettiin palvelimelle useaan kertaan käyttäen jo muodostettua TCP-yhteyttä. Palvelin ei havainnut itse hyökkäystä, mutta datassa olevat `"measurementId"` -arvot (jotka nyt siis olivat samat kaikelle toistetulle datalle) estivät toistetun datan päätymisen palvelimen käsittelyyn ja tietokantaan. `"measurementId"` -arvon kasvattaminen yhdellä jokaisen lähetetyn TCP-paketin jälkeen olisi kuitenkin triviaalia, sillä kaapattu viesti oli salaamaton.
- Portti 11001 (TLS käytössä, JSS ei käytössä): Hyökkääjälaite kaappasi palvelimelle matkalla olleen viestin, josta eristettiin payload-osuus Wiresharkin avulla. Eristetty data, joka oli TLS:n vuoksi salattua, lähetettiin palvelimelle useaan kertaan käyttäen jo muodostettua TCP-yhteyttä. TLS:n eheystarkistukset eivät havainneet hyökkäystä, sillä vain sovelluskerroksen payload-osuutta oli peukaloitu ja muut osat olivat aitoja. Jälleen `"measurementId"` -arvojen ansiosta toistettu data ei päätynyt palvelimen käsittelyyn ja sitä kautta tietokantaan. Edellisen portin tapauksessa ehdotettu triviaali `"measurementId"` -arvon kasvatus ei ollut mahdollinen, sillä data oli salattua ja siten yhden arvon muuttaminen mahdotonta. Siispä TLS toimi epäsuorasti datan eheyden varmistamisessa.
- Portti 11002 (TLS ei käytössä, JSS käytössä): Hyökkääjälaite kaappasi palvelimelle matkalla olleen viestin, josta eristettiin payload-osuus Wiresharkin avulla. Tämä eristetty data lähetettiin palvelimelle useaan kertaan käyttäen jo muodostettua TCP-

1. Lisätietoja `http://stackoverflow.com/questions/14907581/`

yhteyttä. Jälleen `"measurementId"` -arvojen ansiosta toistettu data ei päätynyt palvelimen käsittelyyn ja sitä kautta tietokantaan. JSS-allekirjoitukset olivat kuitenkin valideja. Ensimmäisen portin tapauksessa ehdotettu triviaali `"measurementId"` -arvon kasvatus ei ollut mahdollinen, sillä arvon muuttaminen muutti myös JSS-allekirjoitusta, minkä jälkeen olemassaoleva allekirjoitus ei ollut enää validi. Siispä datan eheys on epäsuorasti varmistettu `"measurementId"` -arvon ja JSS-allekirjoituksen yhteistyön avulla.

- Portti 11003 (TLS käytössä, JSS käytössä): Hyökkääjälaite kaappasi palvelimelle matkalla olleen viestin, josta eristettiin payload-osuus Wiresharkin avulla. Eristetty data, joka oli TLS:n vuoksi salattua, lähetettiin palvelimelle useaan keraan käyttäen jo muodostettua TCP-yhteyttä. TLS:n eheystarkistukset tai JSS-allekirjoitusten tarkistukset eivät havainneet hyökkäystä. Jälleen `"measurementId"` -arvojen ansiosta toistettu data ei päätynyt palvelimen käsittelyyn ja sitä kautta tietokantaan. Kahden edellisen portin kohdalla mainituista syistä yksikään viesti ei päätynyt palvelimen lopulliseen käsittelyyn ja tietokantaan saakka. TLS-protokollan ja `"measurementId"` -arvojen ansiosta eheys säilyi. JSS-tarkistukset tarjosivat vielä toisen tason varmistuksen, mikäli TLS-protokollan salaus olisi murrettu, minkä todettiin edellisessä alaluvussa olevan mahdollista tietyillä ehdoilla.

Tässä testitapauksessa parhaimman suojauksen eheyden vaarantumista vastaan antoi protokollaan määritelty `"measurementId"` -arvo. Sitä voidaan pitää sovelluskerroksen eheystarkistuksena, ja sen merkitys osoittautui korvaamattomaksi. Testi osoitti, että merkityksettömältä tuntuva metadata on syytä hyödyntää eheystarkistuksissa aina, kun se on mahdollista. Tämän testitapauksen perusteella on syytä suositella molempien kerrosten eheystarkistuksia replay-hyökkäyksiä vastaan.

5.1.4 Datan virheellisen rakenteen analysointi

Seuraavassa on eritelty datan virheellisen rakenteen testitapauksessä nousseet huomiot porttikohtaisesti:

- Portti 11000 (TLS ei käytössä, JSS ei käytössä): Laite lähetti rakenteellisesti virheel-

listä dataa. Jos `"measurementId"`-arvo puuttui tai oli väärä, niin dataa ei otettu ollenkaan käsittelyyn. `"data"`-arvon tai `"nodeId"`-arvon puuttuminen ei tässä palvelimen toteutuksessa vaikuttanut käsittelyyn, vaan niiden puuttuessa data lähetettiin silti tietokantaan. Myöskään väärä `"nodeId"`-arvo ei vaikuttanut käsittelyyn millään tavalla, vaan virheellinen data päätyi tietokantaan.

- Portti 11001 (TLS käytössä, JSS ei käytössä): TLS-protokollan sisällyttäminen kommunikaatioon ei vaikuttanut rakenteellisesti virheellisen datan kulkuun millään tavalla.
- Portti 11002 (TLS ei käytössä, JSS käytössä): JSS-allekirjoitusten sisällyttäminen kommunikaatioon ei vaikuttanut rakenteellisesti virheellisen datan kulkuun millään tavalla, sillä allekirjoitus muodostettiin alun perinkin virheelliselle datalle.
- Portti 11003 (TLS käytössä, JSS käytössä): Myöskään TLS-protokollan ja JSS-allekirjoitusten yhdistäminen ei vaikuttanut rakenteellisesti virheellisen datan kulkuun millään tavalla.

Testeistä kävi ilmi, että tietoturvanäkökulmasta suunnitellut datan eheyden varmistusmenetelmät eivät sellaisenaan sovellu datan rakenteen analysointiin ja/tai korjaamiseen. Sovelluskerroksen joustavat eheystarkistukset voisivat kuitenkin potentiaalisesti sisältää tiedon "skeemasta"eli datan oletetusta rakenteesta, johon verrattaisiin vastaanotettua JSON-dataa. Mikäli vastaanotetusta JSON-datasta puuttuisi jokin vaadittu attribuutti, niin palvelin voisi käsitellä virheen esimerkiksi lähettämällä tietyntyyppisen virhekuittauksen laitteelle.

5.2 Menetelmien soveltuvuuden analysointi

Testien tulokset datan eheyden varmistusmenetelmien osalta on lueteltu taulukossa 2.

Testitilanne	Ei tarkistuksia	TLS	JSS	TLS ja JSS
Matkimishyökkäys	Ei suojannut	Suojasi	Suojasi	Suojasi
Man-in-the-middle -hyökkäys	Ei suojannus	Suojasi	Suojasi	Suojasi
Replay-hyökkäys	Ei suojannut	Suojasi epäsuorasti	Suojasi epäsuorasti	Suojasi epäsuorasti
Datan virheellinen rakenne	Ei suojannut	Ei suojannut	Ei suojannut	Ei suojannut

Taulukko 2. Datan eheyden varmistusmenetelmien testitulokset.

Tulosten perusteella havaittiin, että TLS ja JSS suojasivat toteutettujen testitapausten kohdalla täsmälleen samoilta vaaratilanteilta. Tämä ei johtunut menetelmien samankaltaisuudesta,

vaan siitä, että osa testitilanteista vaati yksinkertaisia ja osa monimutkaisempia datan eheyden suojaamiskeinoja. Monimutkaisimmat testitilanteet olivat sellaisia, joita varten kumpaakaan menetelmää ei ole suunniteltu. Joka tapauksessa pelkän "kyllä/ei-tyyppisen taulukoinnin perusteella kumpaakaan menetelmää ei voi suositella toisen ylitse.

On tärkeää analysoida myös, millainen potentiaali menetelmillä on laajentua suojaamaan myös tilanteilta, joilta ne eivät tällä hetkellä suojaa. Testeissä replay-hyökkäyksen kohdalla nousi esiin tärkeä huomio siitä, että juoksevan sekvenssinumeron rooli datan eheyden säilyttämisessä on suuri. Tämän huomion perusteella datan eheyden varmistamisen prosessiin olisi tärkeää lisätä sisäänrakennettu sekvenssinumeron käsittely, jolloin sovelluskerrokselle ei tarvitsisi liittää manuaalista mittauksen järjestysnumeron tarkastelua. Kuljetuskerroksen TCP-protokollassa on toki sisäänrakennettu sekvenssinumerointi, mutta se ei auta tapauksessa, jossa TCP-liikenne kaapataan ennen palvelimelle pääsyä. Myös TLS-protokollan sekvenssinumerointi kärsii samasta ongelmasta. Man-in-the-middle -hyökkäyksiltä täytyy testitulosten perusteella suojautua sovelluskerroksella, sillä sovelluskerrokseen ei vaikuta se, onko liikennettä peukaloitu alemmalla kerroksella. Tästä syystä voidaan ehdottaa, että JSS-proseduuriin lisätään sisäänrakennettu lähettäjäkohtainen sekvenssinumero, joka sisällytetään allekirjoitukseen ja täten estetään myös sen peukalointi matkalla tai myöhemmin sovelluskerroksella. Sisäänrakennetun sekvenssinumeroinnin avulla JSS suojaisi myös replay-hyökkäyksiltä suoraan, sen sijaan että suojaus tapahtuisi epäsuorasti manuaalisen metadatan käsittelyn avulla.

Toinen lisäys, joka JSS-proseduuriin voitaisiin testien perusteella tehdä, on JSON-muotoisen datan rakenteen tarkistaminen. Tarkistus voisi yksinkertaisuudessaan sisältää tiedon kaikista pakollisista kentistä ja niiden JSON-datatyypeistä. Tämä tieto sisällytettäisiin esimerkiksi siihen JSON-objektiin, joka nykyään sisältää tiedon käytetyistä algoritmeistä, eli attribuutin `"jss.protected"` alle. Tieto pakollisista kentistä parsittaisiin JSS-allekirjoituksen tarkistamisen jälkeen, ja virheellisen datan tapauksessa lähettäjälle voitaisiin lähettää tietyntyyppinen kuittaus siitä, että vastaanotettu data oli virheellistä.

Kokonaisuudessaan testien perusteella JSS-menetelmän potentiaali toimia kokonaisvaltaisena datan eheyden suojaajana on parempi kuin TLS-protokollan, mutta toisaalta haavoittuvassa IoT-ympäristössä ei myöskään ole mitään syytä poisjättää TLS-protokollan käyttöä. Kun

lisäksi TLS-protokolla vastaa kuljetuksen aikaisesta viestien salaamisesta, voidaan suositella käytettävän sekä TLS-protokollaa että JSS-allekirjoituksia datan eheyden varmistamiseksi. Ne eivät missään nimessä ole toisiaan poissulkevia menetelmiä, vaan tarjoavat kahden eri kerroksen limittäisen suojauksen vahvistaen toisiaan.

6 Yhteenveto

Tässä tutkielmassa suoritettiin katsaus IoT-järjestelmiin ja niiden tietoturvan tasoon. Esiteltiin IoT-tietoturvan tila, uhat ja haasteet sekä kehitettyjä ratkaisuja tietoturvaongelmiin. IoT-tietoturvaa analysoitiin ensin yleisesti ja sen jälkeen datan eheyden näkökulmasta. Tutkielman ylimmän tason tavoitteena oli saavuttaa käsitys kolmen konseptin yhteistoiminnasta: IoT, tietoturva ja datan eheys. Tarkennettuna tavoitteena oli tutkia, miten datan eheys vaikuttaa tietoturvaan ja millaiset datan eheyden varmistusmenetelmät soveltuvat IoT-ympäristöön.

Tutkielmassa tutustuttiin useisiin eri menetelmiin ja teknologioihin, joita hyödynnetään IoT-järjestelmissä erityisesti datan eheyden varmistamisen roolissa. Näitä ovat esimerkiksi MAC, RSA, ECC, ECDSA, TLS ja JSS. Termien takana olevien teknologioiden merkitystä kokonaisuuden kannalta avattiin teoriaosuudessa. Kokonaisuuden hahmottamiseen liittyi myös se, että tutkielmassa analysoitiin datan eheyden varmistamisen roolia tietoturvan näkökulmasta. Teoriaosuudessa saavutettujen tietojen perusteella vertailtiin eri menetelmien toimintaa. Vertailun avulla saatiin luotua joukko järkeviä testausskenaarioita, joissa eri menetelmien erot pyrittiin saamaan mahdollisimman hyvin esille.

Teoriapohjan avulla suoritettiin käytännön testejä, joissa tarkoituksella vaarannettiin datan eheys. Testejä varten rakennettiin räätälöidyt palvelin- ja laiteohjelmistot, joiden tarkoituksena oli tarjota testitilanteille mahdollisimman hyvät puitteet. Testien läpiviennit pyrittiin raportoimaan riittävän yksityiskohtaisesti ja selkeästi, jotta niiden toistaminen ja laajentaminen olisi mahdollista. Testien tulokset vastasivat osittain etukäteistoiveita: Testien avulla pystyttiin havaitsemaan parannusehdotuksia käytetyille menetelmille, mutta kovin suuria eroavuuksia ei löydetty menetelmien välillä. Selvempien eroavuuksien löytäminen olisi vaatinut useampia erilaisia testiskenaarioita.

Tutkielman pohjalta on mahdollista jatkaa datan eheyden varmistamismenetelmien tutkimusta siten, että suoritetaan samat testit muille tiedossa oleville menetelmille ja/tai lisätään testitapauksia oikean maailman ongelmien ja tietoturvahyökkäysten pohjalta. On myös syytä tutkia, kuinka eheyden varmistusmenetelmiin saataisiin mukaan datan oletettu rakenne ("skeema") ja sekvenssinumerointi. Lisäksi eräs ehdotus koskee sen tutkimista, kuinka laitteiden

fyysistä peukalointia voidaan käyttää hyödyksi eheyttä vastaan hyökätessä. IoT-laitteet sisältävät usein debug-väyliä ja vastaavia reittejä, joita hyökkääjä voi käyttää hyväkseen ja jotka täytyy siis suojata. Datan eheystarkistukset voisivat toimia hyvin, kun halutaan selvittää, lähettääkö jokin laite fyysisen peukaloinnin seurauksena epäkelpoa dataa. Tapaus oli aluksi tarkoitus testata tässä tutkielmassa, mutta se rajattiin aikataulusyistä pois.

Lähteet

Adams, Carlisle. 2011a. “Impersonation Attack”, toimittanut Henk C. A. van Tilborg ja Sushil Jajodia, 596–596. Encyclopedia of Cryptography and Security. Boston, MA: Springer US. ISBN: 978-1-4419-5906-5. doi:10.1007/978-1-4419-5906-5_80". http://dx.doi.org/10.1007/978-1-4419-5906-5_80.

———. 2011b. “Replay Attack”, toimittanut Henk C. A. van Tilborg ja Sushil Jajodia, 1042–1042. Encyclopedia of Cryptography and Security. Boston, MA: Springer US. ISBN: 978-1-4419-5906-5. doi:10.1007/978-1-4419-5906-5_92". http://dx.doi.org/10.1007/978-1-4419-5906-5_92.

Alqassem, I., ja D. Svetinovic. 2014. “A taxonomy of security and privacy requirements for the Internet of Things (IoT)”. Teoksessa *2014 IEEE International Conference on Industrial Engineering and Engineering Management,* 1244–1248. doi:10.1109/IEEM.2014.7058837.

Asplund, M., ja S. Nadjm-Tehrani. 2016. “Attitudes and Perceptions of IoT Security in Critical Societal Services”. *IEEE Access* 4 (): 2130–2138. ISSN: 2169-3536. doi:10.1109/ACCESS.2016.2560919.

Atzori, L., A. Iera ja G. Morabito. 2010. “The Internet of Things: A survey”. *Computer Networks* 54, numero 15 (): 2787–2805. ISSN: 1389-1286. doi:10.1016/j.comnet.2010.05.010.

Babar, S., A. Stango, N. Prasad, J. Sen ja R. Prasad. 2011. “Proposed embedded security framework for Internet of Things (IoT)”. Teoksessa *Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on,* 1–5. doi:10.1109/WIRELESSVITAE.2011.5940923.

Bray, Tim. 2014. *The JavaScript Object Notation (JSON) Data Interchange Format.* RFC 7159. Google, Inc. https://tools.ietf.org/html/rfc7159.

Bray, Tim, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler ja François Yergeau. 2008. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* Tekninen raportti. https://www.w3.org/TR/2008/REC-xml-20081126/.

Brumley, David, ja Dan Boneh. 2005. "Remote timing attacks are practical". Web Security, *Computer Networks* 48 (5): 701. ISSN: 1389-1286. doi:http://dx.doi.org/10.1016/j.comnet.2005.01.010". http://www.sciencedirect.com/science/article/pii/S1389128605000125.

Capossele, A., V. Cervo, G. De Cicco ja C. Petrioli. 2015. "Security as a CoAP resource: An optimized DTLS implementation for the IoT". Teoksessa *2015 IEEE International Conference on Communications (ICC),* 549–554. doi:10.1109/ICC.2015.7248379.

Chen, L., H. Ba, W. Heinzelman ja A. Cote. 2013. "RFID range extension with low-power wireless edge devices". Teoksessa *Computing, Networking and Communications (ICNC), 2013 International Conference on,* 524–528. doi:10.1109/ICCNC.2013.6504140.

Cheneau, Tony. 2014. *ecc-light-certificate: A simple ECC certificate library.* https://github.com/nist-emntg/ecc-light-certificate.

Desmedt, Yvo. 2011. "Man-in-the-Middle Attack", toimittanut Henk C. A. van Tilborg ja Sushil Jajodia, 759–759. Encyclopedia of Cryptography and Security. Boston, MA: Springer US. ISBN: 978-1-4419-5906-5. doi:10.1007/978-1-4419-5906-5_324". http://dx.doi.org/10.1007/978-1-4419-5906-5_324.

Gay, David, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer ja David Culler. 2003. "The nesC Language: A Holistic Approach to Networked Embedded Systems". *SIGPLAN Not.* 38, numero 5 (): 1–11. ISSN: 0362-1340. doi:10.1145/780822.781133. http://doi.acm.org/10.1145/780822.781133.

Goralski, W. 2009. *The Illustrated Network: How TCP/IP Works in a Modern Network.* 25–60. 2008046728. Elsevier/Morgan Kaufmann Publishers. ISBN: 9780123745415. https://books.google.fi/books?id=cChUlQEACAAJ.

Gou, Q., L. Yan, Y. Liu ja Y. Li. 2013. “Construction and Strategies in IoT Security System”. Teoksessa *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing,* 1129–1132. doi:10.1109/GreenCom-iThings-CPSCom.2013.195.

Habib, Ahsan, Dongyan Xu, Mikhail Atallah, Bharat Bhargava ja John Chuang. 2005. “A tree-based forward digest protocol to verify data integrity in distributed media streaming”. *IEEE Transactions on Knowledge and Data Engineering* 17, numero 7 (): 1010–1014. ISSN: 1041-4347. doi:10.1109/TKDE.2005.102.

Hernández-Ramos, J. L., D. G. Carrillo, R. Marín-López ja A. F. Skarmeta. 2015. “Dynamic security credentials PANA-based provisioning for IoT smart objects”. Teoksessa *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on,* 783–788. doi:10.1109/WF-IoT.2015.7389153.

Hou, Fangyong, Zhiying Wang, Yuhua Tang ja Zhen Liu. 2004. “Protecting integrity and confidentiality for data communication”. Teoksessa *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on,* 1:357–362. doi:10.1109/ISCC.2004.1358430.

Kraijak, S., ja P. Tuwanut. 2015. “A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends”. Teoksessa *11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015),* 1–6. doi:10.1049/cp.2015.0714.

Laplante, P. A., ja N. Laplante. 2016. “The Internet of Things in Healthcare: Potential Applications and Challenges”. *IT Professional* 18, numero 3 (): 2–4. ISSN: 1520-9202. doi:10.1109/MITP.2016.42.

Leo, M., F. Battisti, M. Carli ja A. Neri. 2014. “A federated architecture approach for Internet of Things security”. Teoksessa *Euro Med Telco Conference (EMTC), 2014,* 1–5. doi:10.1109/EMTC.2014.6996632.

Lincke, N., N. Kuntze ja C. Rudolph. 2015. “Distributed security management for the IoT”. Teoksessa *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM),* 1373–1376. doi:10.1109/INM.2015.7140499.

Liu, A., ja P. Ning. 2008. “TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks”. Teoksessa *Information Processing in Sensor Networks, 2008. IPSN ’08. International Conference on,* 245–256. doi:10.1109/IPSN.2008.47.

Liu, C., Y. Zhang ja H. Zhang. 2013. “A Novel Approach to IoT Security Based on Immunology”. Teoksessa *Computational Intelligence and Security (CIS), 2013 9th International Conference on,* 771–775. doi:10.1109/CIS.2013.168.

Ly, K., ja Y. Jin. 2016. “Security Challenges in CPS and IoT: From End-Node to the System”. Teoksessa *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI),* 63–68. doi:10.1109/ISVLSI.2016.109.

Mahmoud, R., T. Yousuf, F. Aloul ja I. Zualkernan. 2015. “Internet of things (IoT) security: Current status, challenges and prospective measures”. Teoksessa *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST),* 336–341. doi:10.1109/ICITST.2015.7412116.

Maletsky, Kerry. 2015. *White paper: RSA vs ECC Comparison for Embedded Systems.* Tekninen raportti. 1600 Technology Drive, San Jose, CA 95110 USA: Atmel Corporation. http://www.atmel.com/Images/Atmel-8951-CryptoAuth-RSA-ECC-Comparison-Embedded-Systems-WhitePaper.pdf.

Meulen, R. van der. 2015. *Gartner Says 6.4 Billion Connected "Things"Will Be in Use in 2016, Up 30 Percent From 2015.* http://www.gartner.com/newsroom/id/3165317.

Modadugu, N., ja E. Rescorla. 2004. “The Design and Implementation of Datagram TLS”. Teoksessa *ISOC Symposium on Network and Distributed Systems Security (NDSS).* ISBN: 1-891562-18-5, 1-891562-17-7.

Mössinger, M., B. Petschkuhn, J. Bauer, R. C. Staudemeyer, M. Wójcik ja H. C. Pöhls. 2016. "Towards quantifying the cost of a secure IoT: Overhead and energy consumption of ECC signatures on an ARM-based device". Teoksessa *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM),* 1–6. doi:10.1109/WoWMoM.2016.7523559.

Panwar, M., ja A. Kumar. 2015. "Security for IoT: An effective DTLS with public certificates". Teoksessa *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in,* 163–166. doi:10.1109/ICACEA.2015.7164688.

Postel, Jon. 1981. *Transmission Control Protocol.* RFC 793. University of Southern California. https://www.ietf.org/rfc/rfc793.txt.

Pöhls, H. C. 2015. "JSON Sensor Signatures (JSS): End-to-End Integrity Protection from Constrained Device to IoT Application". Teoksessa *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2015 9th International Conference on,* 306–312. doi:10.1109/IMIS.2015.48.

Rahman, R. A., ja B. Shah. 2016. "Security analysis of IoT protocols: A focus in CoAP". Teoksessa *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC),* 1–7. doi:10.1109/ICBDSC.2016.7460363.

Ray, S., S. Bhunia, Y. Jin ja M. Tehranipoor. 2016. "Security validation in IoT space". Teoksessa *2016 IEEE 34th VLSI Test Symposium (VTS),* 1–1. doi:10.1109/VTS.2016.7477288.

Razouk, W. 2014. "Zigbee Security within the Framework of IoT". Teoksessa *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications,* 265–265. doi:10.1109/SOCA.2014.57.

Riahi, A., Y. Challal, E. Natalizio, Z. Chtourou ja A. Bouabdallah. 2013. "A Systemic Approach for IoT Security". Teoksessa *2013 IEEE International Conference on Distributed Computing in Sensor Systems,* 351–355. doi:10.1109/DCOSS.2013.78.

Riahi, A., E. Natalizio, Y. Challal, N. Mitton ja A. Iera. 2014. “A systemic and cognitive approach for IoT security”. Teoksessa *Computing, Networking and Communications (ICNC), 2014 International Conference on,* 183–188. doi:10.1109/ICCNC.2014.6785328.

Sahraoui, S., ja A. Bilami. 2014. “Compressed and distributed host identity protocol for end-to-end security in the IoT”. Teoksessa *2014 International Conference on Next Generation Networks and Services (NGNS),* 295–301. doi:10.1109/NGNS.2014.6990267.

Sarigiannidis, P., E. Karapistoli ja A. A. Economides. 2015. “VisIoT: A threat visualisation tool for IoT systems security”. Teoksessa *2015 IEEE International Conference on Communication Workshop (ICCW),* 2633–2638. doi:10.1109/ICCW.2015.7247576.

Scott, Michael. 2003. “MIRACL–Multiprecision Integer and Rational Arithmetic C/C++ Library”. *Shamus Software Ltd, Dublin, Ireland.*

Sevis, K. N., ja E. Seker. 2016. “Survey on Data Integrity in Cloud”. Teoksessa *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud),* 167–171. doi:10.1109/CSCloud.2016.35.

Shah, S. H., ja I. Yaqoob. 2016. “A survey: Internet of Things (IOT) technologies, applications and challenges”. Teoksessa *2016 IEEE Smart Energy Grid Engineering (SEGE),* 381–385. doi:10.1109/SEGE.2016.7589556.

Shor, Peter W. 1997. “Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer”. *SIAM J.Sci.Statist.Comput.* 26:1484. doi:10.1137/S0097539795293172".

Simonite, Tom. 2016. “NSA Says It “Must Act Now” Against the Quantum Computing Threat”. Viitattu 3. marraskuuta 2016. https://www.technologyreview.com/s/600715/nsa-says-it-must-act-now-against-the-quantum-computing-threat/.

Szczechowiak, Piotr, Leonardo B. Oliveira, Michael Scott, Martin Collier ja Ricardo Dahab. 2008. “NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks”, toimittanut Roberto Verdone, 305–320. Wireless Sensor Networks: 5th European Conference, EWSN 2008, Bologna, Italy, January 30-February 1, 2008. Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-77690-1. doi:10.1007/978-3-540-77690-1_19". http://dx.doi.org/10.1007/978-3-540-77690-1_19.

Tellez, M., S. El-Tawab ja H. M. Heydari. 2016. “Improving the security of wireless sensor networks in an IoT environmental monitoring system”. Teoksessa *2016 IEEE Systems and Information Engineering Design Symposium (SIEDS),* 72–77. doi:10.1109/SIEDS.2016.7489330.

Urien, P. 2016. “Innovative DTLS/TLS security modules embedded in SIM cards for IoT trusted and secure services”. Teoksessa *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC),* 276–277. doi:10.1109/CCNC.2016.7444778.

Warner, Brian. 2010. *python-ecdsa.* https://github.com/warner/python-ecdsa.

Wurm, J., K. Hoang, O. Arias, A. R. Sadeghi ja Y. Jin. 2016. “Security analysis on consumer and industrial IoT devices”. Teoksessa *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC),* 519–524. doi:10.1109/ASPDAC.2016.7428064.

Xu, T., J. B. Wendt ja M. Potkonjak. 2014. “Security of IoT systems: Design challenges and opportunities”. Teoksessa *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD),* 417–423. doi:10.1109/ICCAD.2014.7001385.

Yalçin, T. 2016. “Compact ECDSA engine for IoT applications”. *Electronics Letters* 52 (15): 1310–1312. ISSN: 0013-5194. doi:10.1049/el.2016.0760.

Zhang, Z. K., M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen ja S. Shieh. 2014. “IoT Security: Ongoing Challenges and Research Opportunities”. Teoksessa *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications,* 230–234. ID: 1. doi:10.1109/SOCA.2014.58.