TURUN YLIOPISTO

Lohkoketjupohjainen dronelaitteiden pääsynhallinta- ja autentikointijärjestelmä

Tietoliikenne- ja kyberturvallisuusteknologia
Tieto- ja viestintätekniikan tutkinto-ohjelma
Tietotekniikan laitos, Teknillinen tiedekunta
Diplomityö

Laatija:
Suvi Rannisto

Ohjaajat:
Petri Sainio
Jouni Isoaho
Tahir Mohammad

Kesäkuu 2023

Turun yliopiston laatujärjestelmän mukaisesti tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -järjestelmällä.

Diplomityö
Tietotekniikan laitos, Teknillinen tiedekunta
Turun yliopisto

Oppiaine: Tietoliikenne- ja kyberturvallisuusteknologia
Tutkinto-ohjelma: Tieto- ja viestintätekniikka
Tekijä: Suvi Rannisto
Otsikko: Dronelaitteiden pääsynhallinta- ja autentikointijärjestelmä
Sivumäärä: 71 sivua
Päivämäärä: Kesäkuu 2023

IoT-ala kasvaa kiihtyvällä vauhdilla ja dronet ovat yksi viime vuosien merkittävimmistä syntyneistä teknologioista sekä nopeimmin kasvavista ja kehittyvistä IoT-alueista. Koska dronet ovat etäohjattavia miehittämättömiä ilma-aluksia, erityisesti kommunikoinnin suojaaminen ja tiedon lähettäjän varmentaminen ovat kriittisiä, jotta hyökkääjä ei pääse kaappaamaan dronea kesken lennon omaan hallintaansa. Tästä syystä droneihin on pyritty kehittämään mahdollisimman tehokkaita autentikointijärjestelmiä, jotta hyökkääjä ei pääse käsiksi dronen tietoihin tai väärinkäyttämään sitä.

Tämän diplomityön aiheena on tutustua IoT-laitteissa ja erityisesti droneissa käytettäviin autentikointijärjestelmiin sekä parantaa jo olemassa olevaa autentikointijärjestelmää. Kirjallisuuskartoituksella selvitetään, mitä dronet yleisesti ottaen ovat ja mihin niitä voidaan käyttää. Samalla esitellään käyttäjien identiteetin- ja pääsynhallinta, lohkoketjujen toiminta ja ominaisuudet sekä tutustutaan paremmin aiheeseen liittyviin tutkimuksiin. Tutkimuksissa kehitellyt autentikointijärjestelmät hyödyntävät lohkoketjua osana autentikointiprosessia.

Tapaustutkimuksessa suunniteltiin, kehitettiin ja analysoitiin lohkoketjuun pohjautuva pääsynhallinta- ja autentikointijärjestelmä IoD-verkolle. Työn tuloksena syntyi älysopimus, joka hallinnoi sallittujen listaa droneista, jotka pääsevät rekisteröitymään IoD-verkon lohkoketjua hyödyntävään autentikointijärjestelmään. Älysopimus kehitettiin Solidity-kielellä, ja se testattiin simulaatioympäristössä käyttämällä dynaamista ja staattista analysointia. Lisäksi kehitetystä pääsynhallinta- ja autentikointijärjestelmästä tehtiin turvallisuusanalyysi, jossa osoitetaan, että ratkaisu saavuttaa luottamuksellisuuden, eheyden ja saatavuuden turvallisuustavoitteet.

Asiasanat: dronet, IoT, IoD, lohkoketjut, autentikointijärjestelmä, älysopimus, Solidity, identiteetin- ja pääsynhallinta, kyberturvallisuus.

Master of Science in Technology Thesis
Department of Computing, Faculty of Technology
University of Turku

Subject: Cyber Security
Programme: Master's Degree Programme in Information and Communication Technology
Author: Suvi Rannisto
Title: Access Control and Authentication System for Drones
Number of pages: 71 pages
Date: June 2023

The IoT industry is growing at an accelerating pace, and drones are one of the most significant emerging technologies of recent years and the fastest growing and developing IoT fields. Since drones are remotely controlled unmanned aerial vehicles, especially the protection of communication and the verification of the sender of the information are critical, so that an attacker cannot hijack the drone. For this reason, efforts have been made to develop effective authentication schemes for drones, so that an attacker cannot access the drone's data or misuse it.

The aim of this thesis is to familiarize with authentication schemes used in IoT devices and especially in drones and to improve an already existing authentication scheme. The literature review surveys what drones are in general and what they can be used for. In addition, identity and access management, the functionality and features of blockchains, and related work on IoT authentication schemes utilizing blockchain are introduced.

In the case study, a blockchain-based access control and authentication system for the IoD network was designed, developed, and analyzed. In this system, a smart contract manages the whitelist of drones that can register in the IoD network's authentication system that utilizes a blockchain. The smart contract was written in Solidity and tested in a simulation environment using dynamic and static analysis. In addition, a security analysis was performed on the developed access control and authentication system, which shows that the solution achieves the security goals of confidentiality, integrity, and availability.

Keywords: drones, IoT, IoD, blockchains, authentication system, smart contract, Solidity, identity and access management, cyber security.

Sisällysluettelo

Lyhenteet

AS:	*Authorization Server*, valtuutuspalvelin. Myöntää asiakasohjelmalle valtuutusavaimen, kun käyttäjän identiteetti on onnistuneesti todennettu.
BVLOS:	*Beyond Visual Line of Sight*. Dronen lennättäminen ilman näköyhteyttä. Dronen pilotti ei pysty havainnoimaan dronea omin silmin.
CA:	*Certificate Authority*, varmentaja. Taho, joka takaa varmenteen hakijan identiteetin ja myöntää sähköisiä varmenteita.
CIA:	*Confidentiality, Integrity, Accessibility*, luottamuksellisuus, eheys ja saatavuus. Tietoturvallisuudessa käytettävä CIA-malli, joka varmistaa tietojärjestelmien turvallisuuden.
EVLOS:	*Extended Visual Line of Sight*. Dronen lennättäminen näköyhteydellä. Dronen pilotti käyttää tarkkailijoita dronen lennättämisetäisyyden kasvattamiseksi.
EVM:	*Ethereum Virtual Machine*, Ethereum-virtuaalikone. Tietokone, joka käsittelee Ethereum-lohkoketjun tilaa ja mahdollistaa älysopimusten käytön lohkoketjussa.
GCS:	*Ground Control Station*, ohjausasema. Laitteisto, jolla pilotti voi kommunikoida dronen kanssa ja ohjata sitä.
IAM:	*Identity and Access Management*, identiteetin- ja pääsynhallinta. Sisältää käytäntöjä, prosesseja ja teknologioita käyttäjien identiteettien ja käyttöoikeuksien hallintaan.
IoD:	*Internet of Drones*, droneiden Internet. Järjestelmä, jossa droneja on yhdistettynä Internetiin tiedonsiirtoa, ohjausta ja seurantaa varten.
IoT:	*Internet of Things*, esineiden Internet. Järjestelmä, jossa laite on yhdistettynä Internetiin tiedonsiirtoa, ohjausta ja seurantaa varten.

OAuth:	*Open Authorization*. Protokolla, jossa käyttäjä voi valtuuttaa toisen palvelun pääsyn omiin tietoihinsa paljastamatta tunnistetietojaan.
P2P:	*Peer to Peer*, vertaisverkko. Verkko, jossa jokainen solmu voi toimia sekä palvelimena että asiakkaana verkon muille jäsenille.
TTP:	*Trusted Third Party*, luotettavaa kolmas osapuoli. Luotettava taho, joka helpottaa kahden eri osapuolen välistä vuorovaikutusta.
UAV:	*Unmanned Aerial Vehicle*, miehittämätön ilma-alus. Ilma-alus, jossa ei ole pilottia tai muita ihmisiä.
UID:	*Unique Identifier*, yksilöllinen tunniste. Numeerinen tai aakkosnumeerinen merkkijono, joka yksilöi jonkin kokonaisuuden järjestelmässä.
UML:	*Unified Modeling Language*. Ensisijaisesti tietojärjestelmien mallintamiseen käytettävä graafinen mallinnuskieli erilaisten kaavioiden tekemiseen.
VLOS:	*Visual Line of Sight*. Dronen lennättäminen näköyhteydellä. Dronen pilotti pystyy säilyttämään suoran näköyhteyden droneen.

Käsitteet

Array-lista:	Ohjelmointikielissä käytettävä listarakenne, johon voidaan tallentaa arvoja.
Assembly:	Ohjelmointikieli.
Bitcoin:	Hajautettu sähköinen valuutta, joka käyttää vertaisverkkoa ja lohkoketjua tapahtumien käsittelyyn ja tallentamiseen.
Droneparvi:	Ryhmä droneja, jotka tekevät yhteistyötä tehtävän suorittamiseksi.
Ethereum:	Hajautettu sähköinen valuutta, joka käyttää vertaisverkkoa ja lohkoketjua tapahtumien käsittelyyn ja tallentamiseen.
Ganache:	Paikalliseen ympäristöön luotu Ethereum-lohkoketju.
Genesis-lohko:	Ensimmäinen lohko lohkoketjussa.
JavaScript:	Ohjelmointikieli.
Kaasu:	Mittayksikkö ja maksu, joka kuvaa onnistuneen tapahtuman suorittamisen tai älysopimuksen toteuttamisen kustannuksia.
Konsensusalgoritmi:	Algoritmi, jolla verkon käyttäjät muodostavat yhteisymmärryksen tiedon oikeellisuudesta ja virheellisyydestä.
Lohko:	Lohkoketjussa oleva tietorakenne, johon on tallennettu tapahtumia.
Lohkoketju:	Hajautettu ja jaettu tietokanta.
Louhinta:	Oikeanlaisen tiivisteen laskeminen uudelle lohkolle lohkoketjussa.
Mapping-lista:	Ohjelmointikielissä käytettävä listarakenne, johon voidaan tallentaa arvoja.
Määrite:	Määrittää älysopimuksessa, mitä käyttäjältä vaaditaan ennen toiminnon toteuttamista.
Nonce-arvo:	Lohkoketjuissa käytettävä arvo, jota muutetaan tietynlaisen tiivisteen saamiseksi.

Näkyvyys-määre: Määrittää toimintojen saavutettavuustason.

Pop()-metodi: Poistaa viimeisen arvon array-listasta.

Rakenne: Ohjelmointikielessä käytettävä tietotyyppi, joka sisältää kokoelman tietoja tallentavia muuttujia.

Remix IDE: Työkalu älysopimusten kehittämiseen Ethereum-lohkoketjussa.

Sallittujen luettelo: Pääsylista, joka sallii tunnistettujen käyttäjien pääsyn suojattuun järjestelmään.

Solidity: Ohjelmointikieli.

Solidity Static Analysis: Remix IDE:ssä saatavilla oleva työkalu älysopimuksen analysointiin.

Tiiviste: Muunnettu merkkijono tiivistefunktioon syötetystä tiedosta.

Tiivistefunktio: Algoritmi, jolla voidaan muodostaa syötetystä tiedosta kiinteän mittainen tiiviste.

Token: Tietoa sisältävä koodi, joka mahdollistaa pääsyn suojattuun järjestelmään.

Toiminnon tilan käyttäytymisen määrite:

Määrittää älysopimuksen toimintojen käyttäytymisen ja niiden vuorovaikutuksen lohkoketjuun tallennetun tiedon kanssa

Truffle: Kehitysympäristö Ethereumille.

Vaatimus: Määrittää, mitä käyttäjältä vaaditaan ennen älysopimuksen toiminnon toteuttamista.

Vaikeus-muuttuja: Osoittaa, kuinka vaikeaa ja aikaa vievää on löytää konsensusalgoritmin määrittelemä tiiviste uutta lohkoa luotaessa.

Visual Studio Code: Koodieditori, jolla voidaan kehittää ja testata kehitettäviä ohjelmia.

Älysopimus: Lohkoketjussa oleva tietokoneohjelma sopimusten tekemiseen.

1 Johdanto

Uusia IoT-laitteita otetaan käyttöön yhä enemmän ja niiden määrän ennustetaan kasvavan vuoteen 2025 mennessä jo yli 19 miljardiin [1]. IoT-laitteista dronet eli miehittämättömät ilma-alukset ovat olleet yksi viime vuosien nopeimmin kasvavista ja kehittyvistä IoT-alueista niiden monipuolisen käyttötarkoituksensa ja liikkuvuutensa ansiosta. Droneja ohjataan yleensä etänä, joten niiden kommunikoinnin suojaaminen on kriittistä. Haasteita kommunikoinnin suojaamiseen kuitenkin tuovat kaikille IoT-laitteille tyypilliset rajalliset laskentaresurssit esimerkiksi prosessoinnin, muistin ja energianlähteen osalta. Näiden haasteiden vuoksi alasta kiinnostuneet tutkijat pyrkivät jatkuvasti kehittämään tehokkaampia autentikointijärjestelmiä IoT-laitteille. Erityisesti lohkoketjujen hyödyntäminen nykyaikaisissa autentikointijärjestelmissä on yleistynyt lohkoketjuteknologian tarjoaman turvallisuuden ja tehokkuuden myötä.

1.1 Tutkimusongelma ja tavoitteet

Droneiden käytön suurin uhka kyberturvallisuuden näkökulmasta on ohjauksen hallinnan menettäminen siten, että joku toinen pystyy antamaan käskyjä dronelle. Tällaiset tilanteet, joissa dronen alkuperäinen pilotti ei pysty enää toteuttamaan omaa operaatiotaan eikä tiedä mihin hänen droneaan käytetään, voivat aiheuttaa vakavia ongelmia. Näitä ongelmia voivat olla muun muassa tietoturvan tai yksityisyyden vaarantuminen. Pahimmassa tapauksessa uhat voivat kohdistua turvallisuuteen sekä johtaa merkittäviin taloudellisiin menetyksiin. Droneiden ja muiden IoT-laitteiden kommunikoinnin suojaamiseksi on kuitenkin kehitetty erilaisia autentikointijärjestelmiä. Tässä työssä tarkoitus oli perehtyä näihin IoT-laitteille kehitettyihin autentikointijärjestelmiin ja tutkia, voidaanko jo olemassa olevaa järjestelmää parantaa tai jopa kehittää uusi ratkaisu. Tuloksen tulisi olla turvallisempi ja tehokkaampi autentikointijärjestelmä kuin nyt käytössä olevat tai ehdotetut järjestelmät.

Tutkimusongelman perusteella tutkimuskysymyksiksi muodostuivat:

- Mitä erilaisia autentikointijärjestelmiä IoT-laitteille on kehitetty?
- Miten parantaa jo olemassa olevaa autentikointijärjestelmää tai kehittää uusi, joka olisi turvallisempi, mutta mahdollisimman vähän laskentaresursseja kuluttava?

Työn ensimmäisenä vaiheena oli perehtyä erilaisiin IoT-laitteissa käytössä oleviin autentikointijärjestelmiin tutkimalla olemassa olevia ratkaisuja. Tutkimusten pohjalta

seuraavassa työvaiheessa tehtiin päätös, lähdetäänkö parantamaan jo olemassa olevaa autentikointijärjestelmää vai kehittämään täysin uutta järjestelmää. Kummassakin tapauksessa tavoitteena oli turvallisempi ja tehokkaampi autentikointijärjestelmä. Tehdyn päätöksen myötä suunniteltiin ja kehitettiin tulevan autentikointijärjestelmän järjestelmäarkkitehtuuria sekä lopullisen ratkaisun toteutusta. Paremman autentikointijärjestelmän kehittämisessä tulisi huomioida erityisesti autentikointijärjestelmien raskaat prosessit droneiden rajallisissa laskentaresursseissa. Rajallisten laskentaresurssien vuoksi viimeisenä vaiheena oli hienosäätää kehiteltyä autentikointijärjestelmää mahdollisimman tehokkaaksi droneille.

1.2 Tutkimusmenetelmät ja -aineisto

Työ on soveltava tutkimus, jossa hyödynnettiin jo olemassa olevaa tietoa paremman autentikointijärjestelmän kehittämiseksi. Työn toteuttamisessa käytettiin kirjallisuuskartoitusta ja tapaustutkimusta. Kirjallisuuskartoituksella esitellään aiheita, mitä uuden järjestelmän kehittämisessä käsitellään. Aiheina ovat dronet, käyttäjien identiteetin- ja pääsynhallinta sekä lohkoketjut. Näiden lisäksi kartoitetaan aiempia tutkimuksia, joissa on kehitetty lohkoketjua hyödyntäviä autentikointijärjestelmiä IoT-laitteille.

Tapaustutkimuksessa puolestaan kehitettiin parempaa autentikointijärjestelmää droneille. Autentikointijärjestelmän kehittämiseen käytettiin erilaisia simulointitapoja. Ratkaisun analysointi ja testaus puolestaan toteutettiin dynaamisesti ja staattisesti järjestelmän toimivuuden, turvallisuuden sekä tehokkuuden varmistamiseksi.

Lähdeaineistona työssä käytettiin pääasiassa eri yliopistojen tutkimuksia, joiden lisäksi tietoa haettiin myös alan artikkeleista. Mielenkiintoisimmista julkaisuista pyrittiin poimimaan järjestelmän hyvät puolet sekä heikkoudet. Tarkoitus ei ollut vertailla eri julkaisuissa käsiteltyjä autentikointijärjestelmiä keskenään, vaan saada kattava tietopohja hyödynnettäväksi tulevassa järjestelmäkehityksessä.

1.3 Tulos

Työn tuloksena kehitettiin lohkoketjuun ja älysopimukseen pohjautuva pääsynhallinta- ja autentikointijärjestelmä. Järjestelmä suunniteltiin pääsynhallintaa tarvitseville autentikointiratkaisuille, jotka hyödyntävät jo valmiiksi älysopimusta tukevaa lohkoketjua droneiden rekisteröitymiseen ja autentikointiin. Kehitetty ratkaisu on tarkoitettu erityisesti IoD-järjestelmiin, jossa voi olla useita droneja ja droneparvia.

Uudessa pääsynhallinta- ja autentikointijärjestelmässä dronen tulee pyytää rekisteröitymislupaa IoD-verkon käyttämään autentikointijärjestelmään älysopimukselta, jotta se voi kommunikoida verkon muiden laitteiden kanssa. Kehitetyn ratkaisun älysopimus on suuntaa antava, sillä järjestelmäarkkitehtuurit sekä organisaatioiden tarpeet ovat aina yksilökohtaisia. Tämän vuoksi kaikkiin yksityiskohtiin ei oteta kantaa, vaan älysopimusta voidaan muokata omien tarpeiden mukaan.

1.4 Työn rakenne

Työn rakenne on jäsennelty seuraavasti. Ensimmäisessä luvussa esitellään tämän työn aihe ja pääkohdat. Pääkohdissa käsitellään muun muassa tutkimusongelmaa, työn tavoitteita sekä mitä erilaisia tutkimusmenetelmiä ja -aineistoja käytettiin. Luvun lopussa esitellään lyhyesti työn tuloksena kehitetty pääsynhallinta- ja autentikointijärjestelmä. Seuraavassa Dronet-luvussa käydään läpi minkälaisia ominaisuuksia ja käyttötarkoituksia droneilla sekä droneparvilla ja droneiden Internetillä on. Lisäksi käsitellään droneihin liittyviä kyberturvallisuusuhkia sekä puolustuskeinoja.

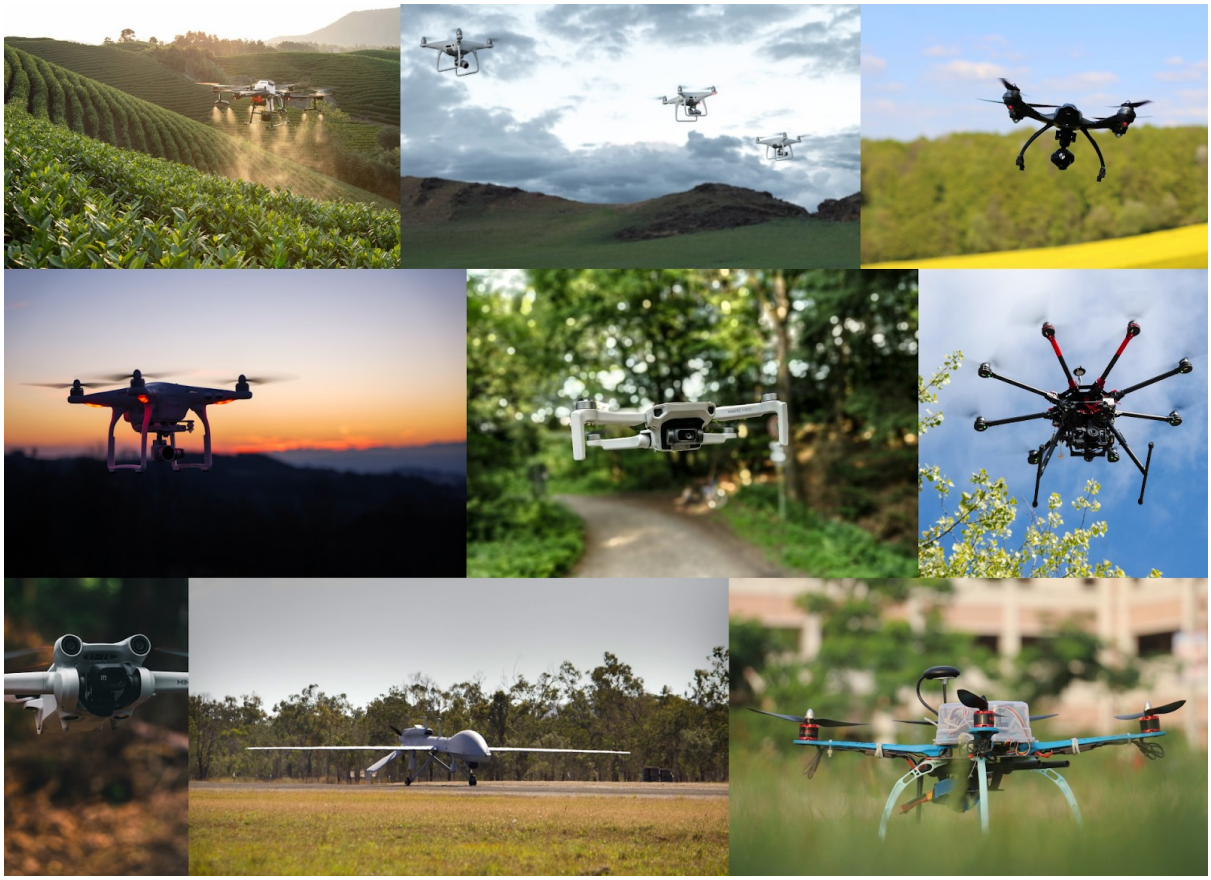
Käyttäjien identiteetin- ja pääsynhallinta -luvussa keskitytään IAM-kehykseen sekä pääkäyttäjien oikeuksien hallintaan. Näissä keskeisenä aiheena esitellään käyttäjien ja käyttäjäryhmien asianmukainen digitaalisten identiteettien hallinta osana organisaation tietoturvaa. Lohkoketjut-luvussa puolestaan tutustutaan lohkoketjuihin ja niiden toimintalogiikkaan sekä älysopimukseen. Lisäksi luku esittelee tämän työn aiheeseen liittyviä tutkimuksia, jotka ovat tarjonneet tietoa sekä ajatuksia tässä työssä kehitetyn järjestelmän toteuttamiseen. Tutkimukset keskittyvät lohkoketjua hyödyntäviin autentikointijärjestelmiin.

Luvussa Uusi kehitetty lohkoketjuun pohjautuva ratkaisu esitellään kehitetty pääsynhallinta- ja autentikointijärjestelmä sekä siihen kuuluva älysopimus. Uuden järjestelmän ja siihen kuuluvan älysopimuksen eri kehitysvaiheet esitellään ja se, miten juuri tällaiseen lohkoketjuun pohjautuvaan ratkaisuun päädyttiin. Lisäksi esitellään kehitetyn älysopimuksen rakenne, toteutus, testaustulokset sekä analysointi. Lopuksi vielä läpikäydään kehitetyn järjestelmän turvallisuusanalyysi. Sillä selvitetään, miten järjestelmä täyttää turvallisuustavoitteet. Kuudennessa eli viimeisessä luvussa kerrataan, mitä työn tapaustutkimuksessa suunniteltiin ja kehitettiin. Lisäksi pohditaan, saavutettiinko työn tavoitteet ja miten niissä onnistuttiin. Lopuksi vielä arvioidaan kehitetyn järjestelmän ominaisuuksia ja miten sitä voidaan muokata omaan käyttöön sopivaksi.

2 Dronet

Tässä luvussa esitellään, mitä drone ovat, mitä ominaisuuksia niihin kuuluu sekä mihin erilaisiin tarkoituksiin niitä voidaan käyttää. Lisäksi esitellään droneparvet ja droneiden Internet, jotka ovat kehittyneet droneiden käytön yleistymisen myötä. Luvun lopussa käsitellään vielä droneihin kohdistuvia uhkia ja niiltä suojautumista.

Dronella [2] tarkoitetaan yleensä mitä tahansa miehittämätöntä ilma-alusta (unmanned aerial vehicle, UAV). Ne eivät ole kuitenkaan vain ilmaan rajoitettuja, vaan niitä voidaan käyttää myös vedenalaiseen liikkumiseen (underwater drone, remote operative vehicle, ROV). Useimmiten droneja kuitenkin hyödynnetään monipuolisesti erilaisissa ilmassa tapahtuvissa operaatioissa. Näiden ilmassa operoivien droneiden ulkonäkö ja koko vaihtelevat suuresti, mikä on nähtävissä kuvassa 1. Pienimmät dronet ovat noin 3 senttimetrin kokoisia, kun taas suurimmat sotilaskäytössä olevat dronet voivat olla jopa 14 metrin pituisia ja siipiväliltään 40 metriä leveitä.



Kuva 1: Droneiden koko ja ulkonäkö vaihtelevat sen mukaan, mihin tarkoitukseen ne on suunniteltu.

Kuvat: Pexels, Pixabay: ArtHouse Studio, AzureEyes, ki-kieh, Lxz2208180358, matsi1612, Military_Material, nodos_pictures, ReneSobeckMedia, Sauvik_Ray.

Nykyään droneja käytetään yhä enemmän ryhmissä yksittäisen dronen lennättämisen sijaan. Tällöin kyseessä on droneparvi (drone swarm), jossa dronet tekevät yhteistyötä monimutkaisten tehtävien suorittamiseksi. Droneparvissa kommunikointi droneiden kesken sekä dronen ja ohjausaseman välillä on erittäin tärkeää. Kommunikaation tulee olla luotettavaa ja turvallista, jotta vastaanottava solmu pystyy luottamaan lähettävän solmun tietoihin tehtävää suorittaessaan.

Droneparvien lisäksi droneiden lisääntynyt käyttö on luonut niin kutsutun droneiden Internetin (Internet of Drones, IoD). Droneiden Internet eli IoD koordinoi droneiden pääsyä valvottuun ilmatilaan ja tarjoaa droneiden ja käyttäjien sekä näiden välisten yhteyksien hallintaa. Se koostuu erilaisista palveluista, joita ovat muun muassa infrastruktuuri- ja teollisuustarkastukset, valvonta- ja pelastusoperaatiot sekä tavarantoimitusjärjestelmät. [2]

IoD-järjestelmissä sekä droneparvien käytössä verkon arkkitehtuuri ja teknologiat määräytyvät droneiden käyttötarkoituksen ja -tilanteiden mukaan. Tällaisia määrääviä tekijöitä ovat esimerkiksi tarve katkeamattomalle tiedonsiirrolle, satelliittiyhteyden käyttö tai droneparven käyttö sisätiloissa. [2]

2.1 Dronen käyttömahdollisuudet ja ominaisuudet

Alun perin dronet olivat vain sotilaskäytössä, mutta nykyään niitä voidaan hyödyntää monissa erilaisissa siviilitehtävissä, kuten kuvassa 2 voidaan nähdä. Yksittäisten droneiden lisäksi myös droneparvien käyttömahdollisuuksia on kehitetty. Droneja ja droneparvia voidaan käyttää esimerkiksi seuraaviin tarkoituksiin [3, 4]:

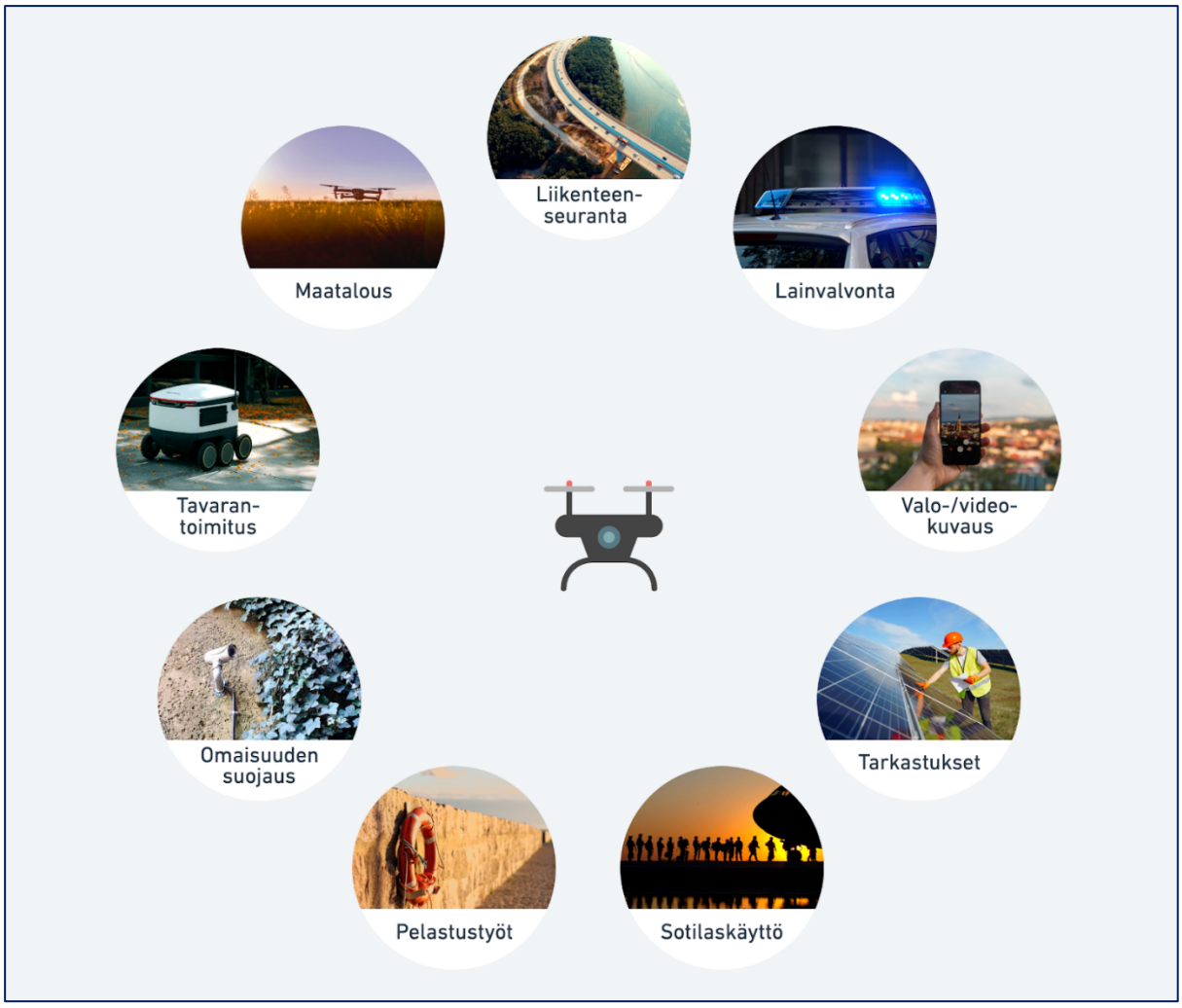
- **Tiedonkeruu**: Liikkuvuutensa ansiosta dronet ovat erinomaisia kohteiden seurantaan ja mittauksiin. Ne pystyvät helposti kuvantamaan laajoja ja vaikeasti saavutettavia alueita sekä tekemään erilaisia laskelmia ja 3D-malleja. Maataloudessa droneja käytetään esimerkiksi sadon seurantaan sekä pelto- ja metsälohkojen mittauksiin. Luonnonsuojelussa puolestaan droneja hyödynnetään kasvien ja villieläinten laskemiseen ja seurantaan. Lisäksi droneilla voidaan tarkkailla säätä ja liikennettä.
- **Valvonta**: Droneja voidaan käyttää erilaisissa valvontatehtävissä sekä analyysien tekemiseen. Ne ovat käteviä ihmisille vaikeasti saavutettavien ulkoisten rakenteiden analysointiin ja kartoittamiseen. Tällaisia kohteita ovat esimerkiksi korkeat rakennukset, sillat ja erilaiset laitokset maalla ja merellä. Lisäksi droneja käytetään

paljon teollisuudessa esimerkiksi aurinko- ja tuulipuistojen sekä muiden energia- ja teollisuuslaitosten valvontaan ja tarkastamiseen. Näiden ohella droneja käytetään myös pelastus- ja ensiaputehtävissä. Katastrofi- ja kriisialueilla niitä voidaan hyödyntää esimerkiksi etsintätehtävissä ja tulipalojen kartoittamisessa.

- **Valokuvaus ja videokuvaus**: Droneja käytetään usein valokuvaamiseen ja videokuvaukseen. Niitä voidaan hyödyntää muun muassa kiinteistöjen ja tuotteiden kuvaamiseen sekä maisema- ja panoraamakuvaukseen. Medialle drone puolestaan tarjoaa hyvän ja monipuolisen työkalun perinteisten kuvausvälineiden lisäksi. Sillä voidaan esimerkiksi kuvata urheilutapahtumia ja konsertteja ylhäältä päin tarjoten katsojille entistä rikkaampaa ja monitasoisempaa katselukokemusta.

- **Viihdekäyttö**: Useilla droneilla voidaan luoda näyttäviä valoesityksiä muodostamalla erilaisia kuvioita ja elävää esitystä. Droneiden etuna on niiden hiljaisuus ja roskattomuus verrattuna esimerkiksi ilotulitteisiin. Terveydelliset haitat ovat myös pienimpiä kuin ilotulitteiden ja laseresitysten.

- **Tavarantoimitus**: Dronet mahdollistavat kirjeiden ja tavaran toimituksen ilmateitse. Tämä käyttötapa ei kuitenkaan ole kovin kehittynyt, sillä se vaatii useiden eri näkökulmien huomioon ottamista. Droneja voidaan kuitenkin käyttää samoihin tarkoituksiin kuin robotteja, jotka kuljettavat tavaraa maasta käsin paikasta toiseen. Tällaisia kuljetuksia käytetään esimerkiksi isoissa tehtaissa ja terveydenhuollossa.



Kuva 2: Droneja voidaan käyttää moniin eri tarkoituksiin, kuten tiedonkeruuseen ja valvontaan.

Kuvat: Pexels: Daniel Frese, Denniz Futalan, Gustavo Fring, JESHOOTS.com, Kindel Media, Pixabay, Pok Rie, Vlad Bagacian.

Dronen käyttötarkoitus määrittelee, mitä toimintoja siihen kuuluu. Vakituisten komponenttien lisäksi droneen kuuluu lisätoimintoja, jotka auttavat sitä suoriutumaan sille tarkoitetusta tehtävästä. Dronen vakituisiin komponentteihin kuuluvat elektroninen nopeudensäädin (Electronic Speed Control, ESC), lennon ohjausyksikkö (Flight Controller Unit, FCU), prosessori ja muisti sekä Wi-Fi/RF-lähetin ja -vastaanotin. Lisätoimintoihin puolestaan kuuluvat esimerkiksi seuraavat:

- **Kamera**: Drone voidaan varustaa erilaisilla kameroilla ja se kuuluukin lähes poikkeuksetta vakiovarusteena kaikkiin droneihin. Vakiovarusteena oleva kamera on yleensä tarkoitettu perinteiseen valo- ja videokuvaukseen sekä suoratoistolähetysten tekemiseen. Perinteisen kameran lisäksi dronet voidaan varustaa erikoiskameroilla kuten infrapuna- tai lämpökameralla.

- **Tiedon tallennus**: Droneissa voidaan tallentaa tietoa, kuten kameran syötettä, eri tavoin. Kevyt ja edullinen ratkaisu tiedon tallentamiselle on vähän tilaa vievä SD-kortti. Kalliimpi vaihtoehto on sisäinen massamuisti. Massamuistin etuna on kuitenkin sen luotettavuus ja huomattavasti suurempi tallennuskapasiteetti verrattuna SD-korttiin. Droneen tallennuksen lisäksi jotkin dronet mahdollistavat pilvitallennuksen.

- **Tekoäly**: Kalliimmat dronet, jotka ovat yleensä tarkoitettu sotilas- tai viranomaiskäyttöön, voivat sisältää tekoälyn ominaisuuksia. Nämä ominaisuudet mahdollistavat esimerkiksi kohteen seurannan ja tunnistuksen, navigoinnin ja tiedon analysoinnin. [5]

- **Lisätty todellisuus (Augmented Reality, AR)**: Joissakin droneissa on mahdollisuus asettaa tietokonegrafiikan avulla virtuaalisia kohteita kameran syötteeseen. Myös AR-lasit voidaan yhdistää joihinkin droneihin, jolloin virtuaaliset kohteet näytetään lasien näkymässä kameran syötteen sijaan. Dronessa lisättyä todellisuutta voidaan käyttää esimerkiksi dronen sijainnin ja suunnan näyttämiseen. Lisäksi sillä pystytään ohjaamaan ja hallitsemaan dronea. [6, 7]

- **GPS/GNSS**: Useimmissa nykyaikaisissa droneissa on GPS- tai GNSS-navigointisysteemi, jonka avulla ne pystyvät liikkumaan tarkasti ja luotettavasti. Ilman GPS/GNSS-navigointia olevat dronet puolestaan käyttävät erilaisia antureita, joiden signaaleista muodostetaan 2D- tai 3D-kartta hyödynnettäväksi lennon aikana.

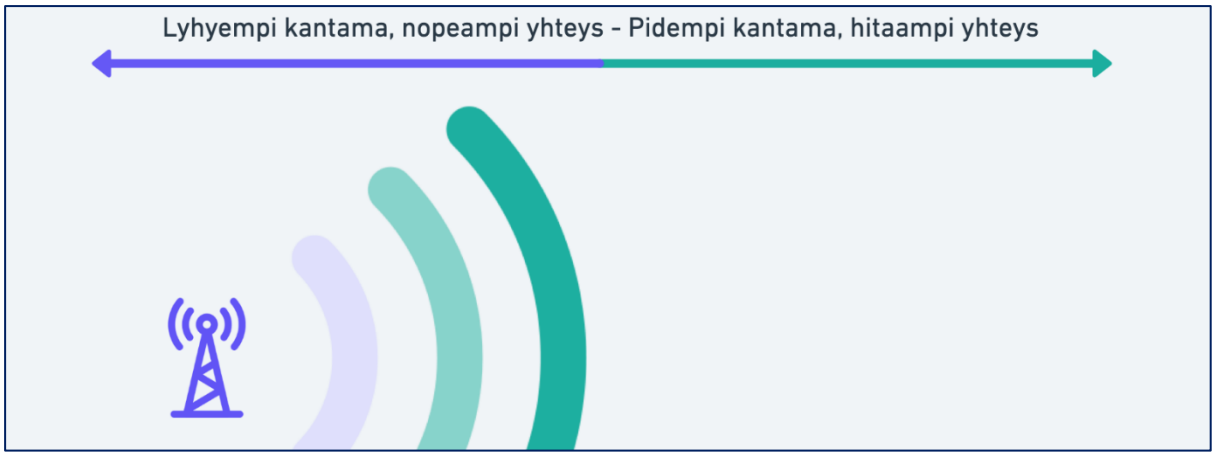
- **Anturit**: Droneissa on useita antureita, jotka ovat välttämättömiä dronen turvalliseen lennättämiseen. Dronen lennon hallintaan käytetään lennon ohjausyksikköä, johon lentämiseen tarvittavat anturit on yhdistetty. Kiihtyvyysmittarit yhdessä gyroskooppien ja kallistusantureiden kanssa mahdollistavat, että lennon ohjausyksikkö pystyy seuraamaan ja kontrolloimaan dronen asentoa. Dronen vakauttamiseen voidaan käyttää myös magneettisia antureita. Elektronisen nopeudensäätimen avulla lennon ohjausyksikkö pystyy puolestaan ohjaamaan ja säätämään dronen nopeutta ja suuntaa. Lentoreittien ja -suuntien hallintaan käytetään inertiamittausyksiköitä (inertial measurement unit, IMU) yhdessä navigointijärjestelmän kanssa. Korkeusmittari puolestaan mittaa lentokorkeutta ja sitä voidaan käyttää pitämään drone tietyllä korkeudella. Nykyään useimmissa droneissa on myös törmäyksen esto -toiminto. Tässä toiminnossa anturit havainnoivat etäisyyden esteeseen ja ohjaavat tarvittaessa dronen pysähtymään tai muuttamaan kurssia törmäysreitiltä. Dronen turvalliseen

lennättämiseen vaadittavien antureiden lisäksi, drone voidaan varustaa erilaisilla lisäantureilla. Lisäanturit auttavat varsinaisen operaation suorittamisessa. Tällaisia antureita ovat muun muassa ultraääni, LiDAR-etäisyysanturi ja kemialliset anturit. [3]

2.2 Dronen ohjaaminen ja lennätysetäisyydet

Dronen ohjaamiseen voidaan käyttää monenlaisia kommunikaatiomenetelmiä. Langaton yhteys on yleisin droneissa käytettävä viestintämenetelmä, joka mahdollistaa kommunikoinnin ohjausaseman (Ground Control Station, GCS) tai muun toimintakeskuksen kanssa. Komento- ja ohjaustietojen lisäksi langatonta viestintää käytetään telemetriatietojen ja hyötytiedon lähettämiseen. Erilaisia droneissa käytettäviä langattomia yhteyksiä ovat muun muassa LTE-, 4G- ja 5G-yhteydet sekä satelliittiyhteys. Yhteydellä pilotti pystyy antamaan komentoja dronelle signaalin lähettimen ja vastaanottimen avulla. Kun pilotti ohjaa dronea, lähetin lukee ohjaimen syötettä ja lähettää sitä signaaleilla langattomasti dronen vastaanottimeen. Vastaanotin puolestaan välittää saamansa tiedot eteenpäin dronen lennon ohjausyksikölle, joka ohjaa dronen liikettä.

Miljardit laitteet, kuten myös useimmat dronet, käyttävät langattomia radiotaajuusaaltoja. Radiotaajuuksien käytön heikkoutena on kuitenkin niiden herkkyys, jolloin signaalia voidaan helposti häiritä tarkoituksella tai tahattomasti. Lisäksi kuva 3 havainnollistaa signaalien kantaman vaikutuksen tiedon lähetysnopeuteen. Näin ollen lyhyemmällä kantamalla saavutetaan nopeampi yhteys, kun taas hitaammalla yhteydellä saavutetaan pidempi kantama. Käytännössä tämä tarkoittaa, että dronelle tarkoitetun operaation suorittamisessa on huomioitava signaalin lähetin-vastaanotinparin ominaisuudet sekä dronen operaation tarvitsema lentokantama.



Kuva 3: Signaalin kantama vaikuttaa tiedon lähetysnopeuteen.

Dronea voidaan lennättää monilla eri lentokantamilla. Lentokantamat voidaan määritellä etäisyyden ja kommunikointiyhteyden tehokkuuden lisäksi näköyhteyden perusteella. Näitä dronen näköyhteyteen perustuvia lennätysetäisyyksiä on kolme, joita ovat VLOS (Visual Line of Sight), EVLOS (Extended Visual Line of Sight) ja BVLOS (Beyond Visual Line of Sight) [8]. Etäisyydeltään lyhyin dronen lennätysetäisyys on VLOS, jossa drone on aina pilotin havaittavissa. VLOS-operaatiot ovat näin rajoittuneita dronen etäisyyteen pilotista, joka on hyvissä olosuhteissa tyypillisesti noin 500 metriä. 500 metriä on myös monissa asetuksissa määritelty suurin sallittu etäisyys pilotista tai tarkkailijasta. Tämä tietenkin rajoittaa merkittävästi, mihin dronea voidaan käyttää.

Huomattavasti pidemmän etäisyyden dronen operaatiolle mahdollistaa EVLOS. EVLOS-operaatiossa pidennetään dronen lennätysetäisyyttä käyttämällä pilotin lisäksi yhtä tai useampaa tarkkailijaa, jonka näköpiirissä drone on koko operaation ajan. EVLOS-operaatiot eivät kuitenkaan ole täysin varteenotettava vaihtoehto VLOS-operaatioille niiden epäkäytännöllisyyden ja logistiikan vuoksi. Koska dronen havainnointi on useiden ihmisten varassa suurilla etäisyyksillä, EVLOS vaatii erityistä koordinointikykyä ja ketjuttamista pilotilta sekä tarkkailijoilta.

Dronen todellinen hyötykäyttö esimerkiksi kaupallisessa tarkoituksessa edellyttää dronelentojen siirtymisen näkökentän ulkopuolelle. BVLOS tarkoittaa dronen käyttöä etänä niin, ettei pilotti pysty havainnoimaan dronea paljain silmin. Näkökentän ulkopuolella toimiminen mahdollistaa droneiden käytön pidempiin ja monimutkaisempiin operaatioihin. Näin lentoja voidaan laajentaa rajoitetuista tehtävistä täysin autonomisiin ja täten kustannustehokkaisiin lentoihin. BVLOS-operaatioita käyttävät erityisesti droneparvet ja armeijan dronet. Tällaisissa operaatioissa droneja hallitaan joko radio-ohjauksella, satelliittiohjauksella tai autonomisesti. Radio-ohjausta käytettäessä dronen ohjaamiseen voidaan hyödyntää useita ohjausasemia sekä lisäksi autonomisuutta. Autonomisessa ohjauksessa oleva drone on esiohjelmoitu toteuttamaan ohjelmistolla ohjailtavaa lentosuunnitelmaa. Tuona aikana drone välttämättä lähetä ja vastaanota mitään signaaleja. [9]

2.3 Dronen kyberturvallisuus

Yleistynyt dronen käyttö yksityishenkilöiden keskuudessa sekä kaupallisessa tarkoituksessa on herättänyt viranomaisten sekä turvallisuusasiantuntijoiden huolen droneiden turvallisesta lennättämisestä. Lennättämisessä huolta herättävät drone-kaappaukset sekä dronen käyttäminen erilaisiin fyysisiin ja virtuaalisiin hyökkäyksiin. Viranomaistahon ja

turvallisuusasiantuntijoiden huolenaiheiden lisäksi ihmiset ovat entistä enemmän huolissaan yksityisyydestään omissa kiinteistöissään droneiden käytön yleistyessä.

Droneihin kohdistuvista kyberhyökkäyksistä yleisimmät liittyvät dronen kaappauksen lisäksi lennon häiritsemiseen tai tiedon manipulointiin ja varastamiseen. Tällaisia hyökkäyksiä ovat esimerkiksi palvelunestohyökkäys (Denial-of-Service, DoS), mies välissä -hyökkäys (Man-in-the-Middle, MITM), GPS-sijainnin väärentäminen (GPS Spoofing) sekä identiteetin väärentäminen. Lisäksi hyökkäyksissä voidaan käyttää vuotaneita tunnistautumistietoja. Osa droneihin liittyvistä hyökkäyksistä kohdistuu tiettyyn merkkiin, malliin tai ohjelmistoon eikä samoja hyökkäyksiä pystytä täten kohdentamaan kaikkiin droneihin. [10, 11]

Kuten edellä jo mainittiin, yksi yleisimmistä kyberhyökkäyksistä droneen on dronen kaappaaminen pilotin ohjauksesta. Dronen kaappauksessa hyökkääjä voi käyttää radiotaajuutta, Wi-Fi-yhteyttä tai muuta palvelua. Näitä käyttämällä drone ja ohjausasema voidaan havaita ja näin saada tärkeää tietoa esimerkiksi dronen Wi-Fi-verkosta ja MAC-osoitteesta. Tämän jälkeen hyökkääjä voi katkaista dronen ja pilotin hallitseman ohjausaseman välisen yhteyden todennuksen purku -hyökkäyksellä (deauthentication attack). Todennuksen purku -hyökkäyksen seurauksena hyökkääjällä on mahdollisuus ottaa drone omaan hallintaansa ja ohjata drone esimerkiksi luokseen. Tällöin hyökkääjällä on mahdollisuus fyysisesti muokata dronea ja saada itselleen sen keräämää tietoa. [12]

Vaikka drone on usein hyökkäyksen kohde, sitä voidaan käyttää myös hyökkäyksen toteuttamiseen. Tällaisia hyökkäyksiä ovat esimerkiksi tunkeutuminen suojatuille alueille, IoT-laitteiden korruptointi sekä tunkeutuminen heikosti suojattuihin Wi-Fi-verkkoihin ja järjestelmiin. Lisäksi hyökkäys voi kohdistua tietojen keräämiseen, tutkimiseen tai varastamiseen.

Kyberhyökkäysten ehkäisemiseen droneissa ja IoD-verkoissa on saatavilla monia erilaisia turvallisuusjärjestelmiä. Ohjelmallisiin turvallisuusjärjestelmiin kuuluvat muun muassa palomuuri, virustorjunta, pääsyoikeuden todentaminen ja tunkeilijan havaitsemisjärjestelmä (Intrusion Detection System, IDS). Laitepohjaisia puolustuskeinoja ovat kryptografiset palvelut ja GPS-sijainnin väärentämistä estävät mekanismit (anti GPS spoofing). Fyysisen suojan puolestaan antaa esimerkiksi luvattoman käsittelyn suojaus ja havainnointi (Anti-Tamper). Muita tärkeitä turvallisuustoimenpiteitä ovat päivitysten ja laitteiston ajantasaisuus, pääsynhallinta ja identiteetin tunnistus, päästä päähän -salaus sekä uhkien hallinta (Threat Management).

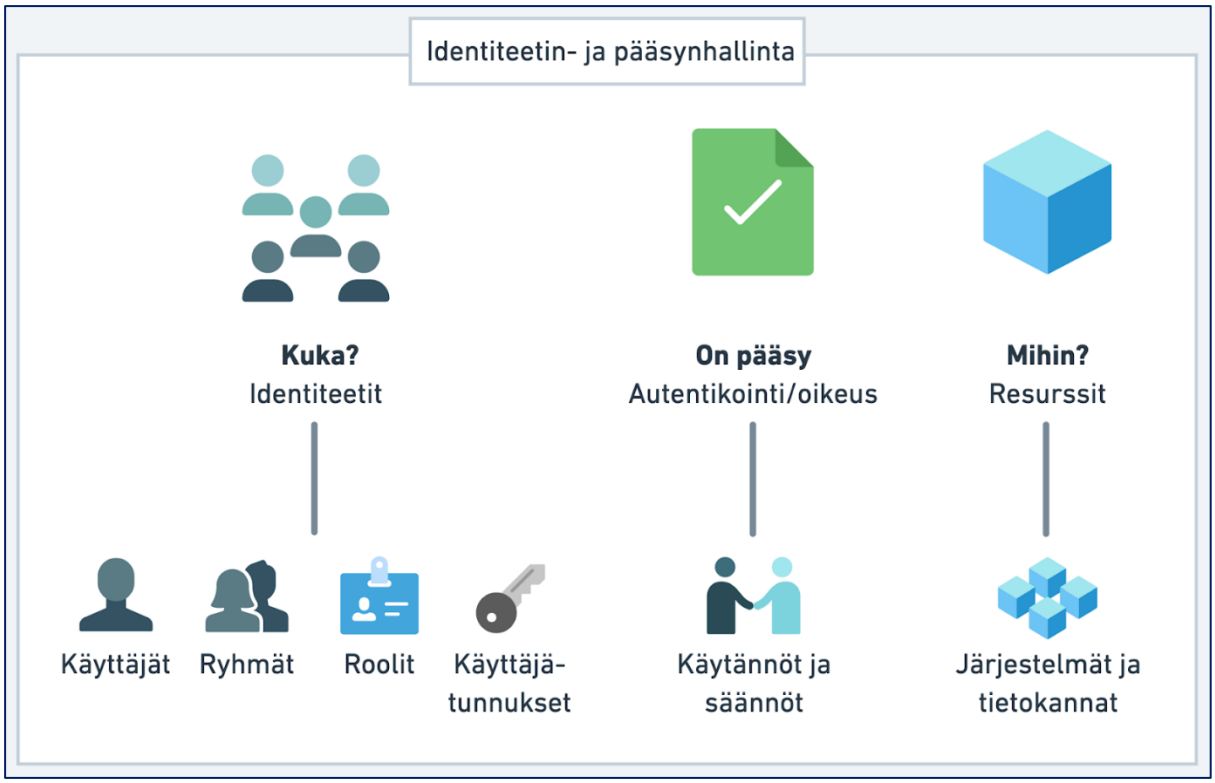
3 Käyttäjien identiteetin- ja pääsynhallinta

Tässä luvussa esitellään identiteetin- ja pääsynhallintakehys. Tämä kehys on merkittävä osa organisaation tietoturvallisuutta tarjoten ensimmäisen suojan useita kyberuhkia vastaan. Lisäksi käsitellään organisaatiolle erittäin kriittinen pääkäyttäjien oikeuksien hallinta. Pääkäyttäjätilit ovat usein hyökkääjien tietojenkalastelun kohteena tilien laajojen järjestelmäoikeuksien vuoksi.

Yhä useammat henkilöt sekä laitteet ovat jonkin tietojärjestelmän käyttäjiä. Tämä tarkoittaa, että meillä on digitaalinen identiteetti, joka mahdollistaa muun muassa verkkopankissa asioimisen. Kuvassa 4 havainnollistettu identiteetin- ja pääsynhallinta kehys (Identity and Access Management, IAM) [13, 14] on luotu helpottamaan digitaalisten henkilöllisyyksien hallintaa sekä käyttäjien ja ylläpitäjien työtä. Se sisältää käytäntöjä, teknologioita ja prosesseja, joilla pystytään todentamaan käyttäjän identiteetti ja antamaan sille kuuluvat oikeudet. Identiteetin todentamisella estetään asiattomien pääsy kriittisiin järjestelmiin ja tiloihin sekä tiedon jakaminen ulkopuolisille tahoille. Näistä johtuen identiteetin- ja pääsynhallinta on merkittävä osa organisaation kyberturvallisuutta. Organisaatiokohtaisesti onkin erittäin tärkeää miettiä, minkälainen IAM-kehys otetaan käyttöön. Kehyksen suunnittelussa tulee huomioida organisaation tarve, rakenne ja sovellukset. Lisäksi tulee suunnitella, miten profiileja hallitaan ja kenellä on oikeus muokata toisten käyttäjien oikeuksia.

Identiteetin- ja pääsynhallintajärjestelmiin kuuluvat esimerkiksi kertakirjautumisjärjestelmät, monivaiheinen tunnistautuminen ja pääkäyttäjien oikeuksien hallinta (Privileged Access Management, PAM). Nämä mahdollistavat myös identiteettitietojen tallentamisen sekä tässä tapauksessa profiilitietojen tiedonhallinnan (Data Governance). Tiedonhallinnassa tietoa koskevat käytännöt määritetään niin, että ne eivät olisi vääriä tai puutteellisia.

Jos asianmukaista identiteetin- ja pääsynhallintaa ei ole käytössä, organisaatio on erityisen altis erilaisille kyberhyökkäyksille kuten tietojenkalasteluille, kiristysohjelmille ja tietovuodoille. Verizon-yrityksen tekemien tutkimusten mukaan jopa yli 80% tietomurroista johtuu nimenomaan heikoista tai huonosti hallinnoiduista käyttäjätunnuksista [15]. Nykyään kuitenkin IAM-järjestelmät tarjoavat hyvät automatisoidut ratkaisut, jolloin esimerkiksi entisen työntekijän valtuudet pystytään helposti poistamaan. Lisäksi se auttaa organisaatiota noudattamaan muuttuvia yksityisyys- ja tietosuojalainsäädäntöjä.



Kuva 4: Identiteetin- ja pääsynhallinta hallinnoi sähköisiä identiteettejä, joille on myönnetty valtuudet tiettyihin resursseihin.

3.1 Identiteetinhallinta

Digitaalinen identiteetti on avain, jolla käyttäjä pääsee suojattuihin järjestelmiin ja resursseihin asianmukaisin oikeuksin. Identiteetit sisältävät tietoja siitä kenelle henkilölle identiteetti kuuluu, mikä hänen käyttäjäroolinsa on, miten häneen voi olla yhteydessä ja mikä hänen roolinsa on organisaatiossa. Identiteetti perustuu identiteetin määritteisiin, jotka sovitaan organisaation tarpeiden ja rakenteen mukaan. Määritteissä otetaan huomioon muun muassa henkilöiden työtehtävät, turvaluokitus ja rooli organisaatiossa.

Identiteetinhallinnassa on huomioitava, että identiteetit voivat muuttua ajan kuluessa. Esimerkiksi työntekijän työnkuva voi muuttua hänen siirtyessään työyhteisössä toiseen työtehtävään tai vaihtaessa toimipistettä. Identiteetinhallinta mahdollistaakin muutosten seuraamisen, jotka vaikuttavat käyttäjän identiteettiin organisaatiossa.

Identiteetinhallinnalla organisaatio pystyy näin ollen hallitsemaan niiden käyttäjien tai käyttäjäryhmien identiteettejä, ketkä saattavat vaatia pääsyä organisaation käyttämiin järjestelmiin tai resursseihin. Oikeus tehdä muutoksia näihin identiteetteihin on organisaation sisällä vain harvoilla valituilla henkilöillä. Yleensä tällaiset henkilöt ovat organisaation tietotekniikkaosaston johtaja tai henkilöstöhallinnon edustajat.

3.2 Pääsynhallinta

Pääsynhallinta suojaa järjestelmiä luvattomalta käytöltä ja takaa oikeudet omaavien käyttäjien pääsyn kriittisiin järjestelmiin helposti ja turvallisesti. Ennalta määritellyt pääsynhallinnan säännöt vaikuttavat päätökseen valtuuttaa käyttäjä tietyin oikeuksin järjestelmään, kuten verkkoon tai palvelimeen. Käyttäjän oikeudet järjestelmässä voivat koskea sovellusten käyttöä tai tietojen katselua ja muokkausta. Pääsynhallinnassa tunnistautumiseen voidaan käyttää esimerkiksi kertakirjautumista tai korttia.

Pääsynhallinnassa käyttäjän identiteetti varmistetaan, kun henkilö pyytää pääsyä suojattuun resurssiin ja tunnistautuu. Identiteetin varmistaminen tapahtuu todentamalla käyttäjän identiteetti organisaation käytäntöjen mukaisesti. Kun käyttäjän identiteetti on todennettu, pääsynhallinta tekee myönteisen päätöksen päästää käyttäjä järjestelmään. Tällöin pääsynhallinta myöntää käyttäjälle oikeudet, jotka hänelle on annettu identiteetin hallinnan kautta. Päätös pääsyn hyväksymisestä sekä käyttöoikeuksien myöntämisestä on aina tarkalleen kyllä tai ei.

Identiteetin todentamista ei pidä sekoittaa valtuutukseen. Valtuutuksella tarkoitetaan, että identiteetillä on valtuutus käyttää tiettyä järjestelmää tai resurssia. Vaikka identiteetti on todennettu, se ei automaattisesti tarkoita sitä, että käyttäjällä olisi valtuutus käyttää kaikkia organisaation järjestelmiä ja resursseja. Valtuutukset määräytyvät aina identiteetin määritysten mukaan.

3.3 Pääkäyttäjien oikeuksien hallinta

Pääkäyttäjien oikeuksien hallinta tarkoittaa pääkäyttäjien tilien hallintaa. Nämä tilit ovat yleensä järjestelmän ylläpitoa ja hätätilanteita varten. Lisäksi pääkäyttäjätileillä voidaan hallita tavallisia käyttäjätilejä ja niihin liittyviä oikeuksia.

Pääkäyttäjien oikeuksien hallinta on kriittinen osa organisaation tietoturvaa, sillä merkittävä osa tietomurroista liittyy pääkäyttäjäoikeuksien varkauksiin ja väärinkäyttöihin. On yleistä, että organisaatioissa jaetaan pääkäyttäjätilien tunnuksia ja salasanoja useille eri ihmisille sen sijaan, että luotaisiin vähemmän oikeuksia sisältäviä tavallisia käyttäjätilejä. Kun laajat oikeudet antavat pääkäyttäjätilit ovat kaikkien käytettävissä, organisaation on vaikea toteuttaa korkeatasoista tietoturvaa.

Koska pääkäyttäjätileillä on erityisoikeuksia järjestelmän ominaisuuksiin sekä luottamukselliseen tietoon, hyökkääjät ovat erityisen kiinnostuneita saamaan niiden tunnukset ja salasanat haltuunsa. Pääkäyttäjätilin avulla hyökkääjä pystyy hankkimaan lisää oikeuksia itselleen sekä liikkumaan organisaation laitteissa, järjestelmissä ja verkossa. Lisäksi hyökkääjä pystyy muuttamaan, poistamaan tai varastamaan tietoa. Hän voi myös pyrkiä tekemään muutoksia käyttäjätilien tunnuksiin tai järjestelmien asetuksiin. Yleensä hyökkääjällä on myös tapana luoda takaovia järjestelmiin, jotta hän pystyy palaamaan, vaikka hän menettäisi pääkäyttäjätilin hallinnan.

Paras tapa puolustautua pääkäyttäjätilien varkauksilta ja väärinkäytöiltä on ottaa käyttöön pääkäyttäjien oikeuksien hallinta sekä pitää pääkäyttäjätilien tunnukset ja salasanat turvassa. Tämän lisäksi tilien käyttöoikeudet tulee myöntää ainoastaan niitä tarvitseville. Mitään tilitietoja ei tulisi koskaan jakaa käyttäjien kesken, koska yhteisiä tilejä käyttämällä ei pystytä jäljittämään kuka on kirjautunut ja tehnyt mitäkin muutoksia. Jokaisella työntekijällä on myös hyvä olla perustiedot siitä, miten hyökkääjät kalastelevat tietoja. Näin työntekijä osaa käyttäytyä vastuullisesti Internetissä ja pystyy mahdollisesti myös havaitsemaan, jos tilin hallinta on vaarantunut. Kun organisaatiossa noudatetaan tietoturvaohjeistuksia ja turvallisia toimintatapoja, organisaation on helpompaa saavuttaa haluttu tietoturvan taso toimintansa varmistamiseksi.

4 Lohkoketjut

Tässä luvussa käsitellään lohkoketjuja (blockchain). Lohkoketjut ovat useimmille tuttuja kryptovaluutoista, mutta niihin voidaan tallentaa myös muuta digitaalista tietoa. Tiedon suojaamiseksi lohkoketjuilla on erilaisia tapoja estää hyökkäyksiä sekä virheellisiä tapahtumia. Näistä suojautumistavoista esitellään tiiviste, konsensusalgoritmi sekä vertaisverkko. Lopuksi käsitellään älysopimuksia, jotka ovat lohkoketjuissa toimivia sovelluksia.

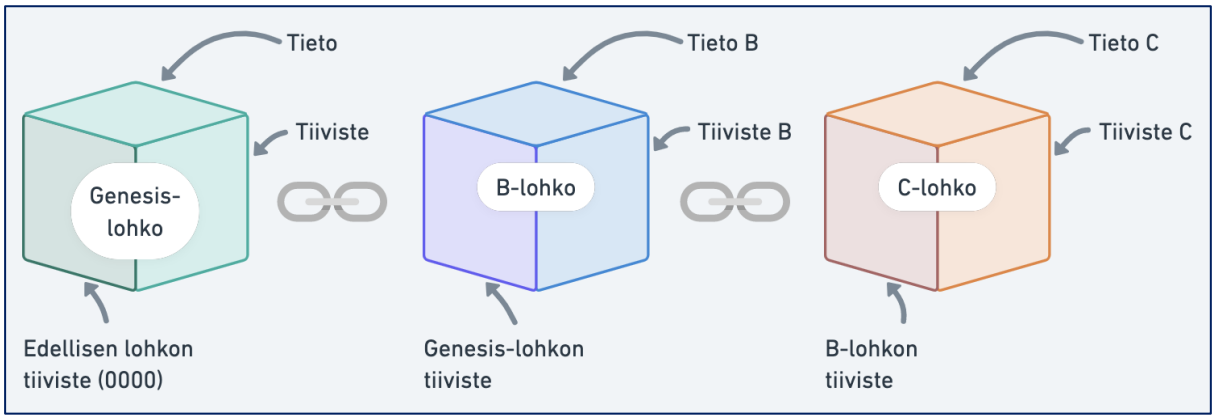
Lohkoketjun konseptin kehitys alkoi jo 1990-luvulla. Myöhemmin vuonna 2008 siihen tehtiin merkittäviä parannuksia ja uudistettu lohkoketjuteknologia otettiin käyttöön Bitcoin-kryptovaluutassa. Bitcoinin suuren suosion myötä lohkoketjujen käyttö erilaisten digitaalisten tietojen tallentamiseen yleistyi nopeasti. Nykyään sitä käytetäänkin monilla eri aloilla esimerkiksi markkinoiden seuraamiseen sekä maksuliikenteen, terveystietojen ja toimitusketjujen hallitsemiseen.

Useiden hyvien ominaisuuksiensa vuoksi monet kokevat lohkoketjujen olevan yksi ihmiskunnan suurimmista teknologisista saavutuksista Internetin keksimisen ohella. Suosiostaan ja monista hyvistä puolistaan huolimatta lohkoketjuissa on kuitenkin myös omat heikkoutensa. Näitä ovat esimerkiksi sen monimutkaisuus ja hitaus tiedon lisääntyessä. Näistä heikkouksista johtuen sanotaan myös, että lohkoketjujen mahdollisuuksia liioitellaan ja tulevaisuudessa siirrytään vieläkin tehokkaampiin teknologioihin. Yksi tällainen teknologia olisi esimerkiksi tietokantojen jakaminen osiin eli sirpalointi (sharding). Eriävistä mielipiteistä huolimatta voidaan kuitenkin olla yhtä mieltä siitä, että lohkoketjut tarjoavat ihmisille paljon mahdollisuuksia sekä ominaisuuksia.

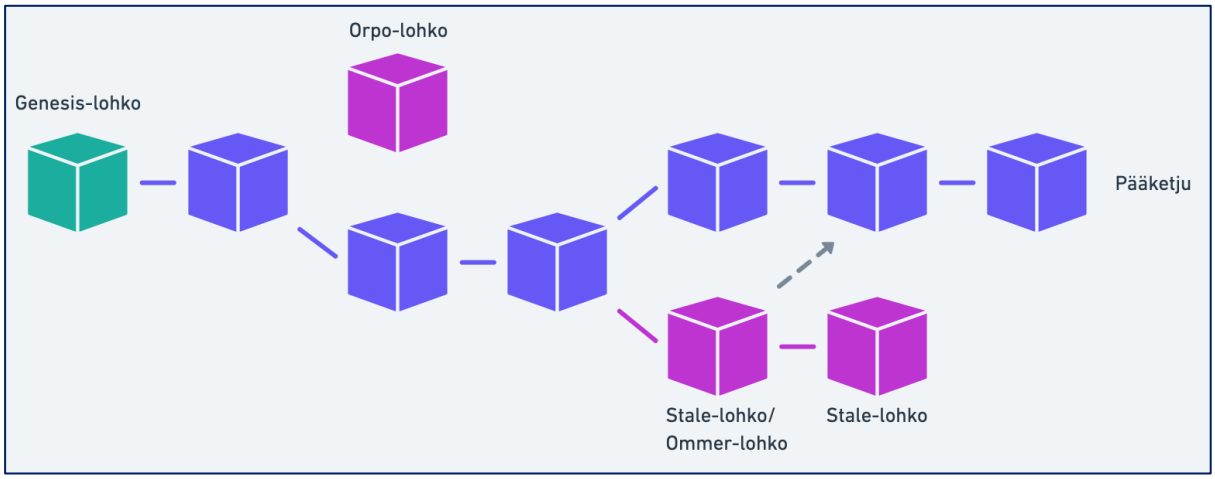
4.1 Lohkoketjun rakenne ja toiminta

Lohkoketju [16, 17, 18, 19] on nimensä mukaisesti ketju lohkoja (block). Jokaiseen lohkoon tallennetaan haluttua tietoa järjestelmän määrittelyjen mukaisesti. Varsinaisen tiedon lisäksi lohko sisältää muun muassa kryptografisen tiivisteen sekä edellisen lohkon tiivisteen, kuten kuvassa 5 esitetään. Viittaus edellisen lohkon tiivisteeseen saa aikaan lohkoketjujen ketjumaisen rakenteen. Tällöin lohkoista muodostuu kronologisessa järjestyksessä oleva ketju, joka on kryptografisesti suojattu tietojen väärentämiseltä. Kuvassa 6 esitetään lohkoketjun rakennetta.

Lohkoketjun ensimmäistä lohkoa kutsutaan genesis-lohkoksi. Se luodaan automaattisesti heti uuden lohkoketjun käyttöönoton yhteydessä, sillä se ei voi viitata edelliseen lohkoon. Genesis-lohkon luonnin jälkeen lohkoketjun osallistujat voivat normaalisti luoda uusia lohkoja tapahtumista. Tapahtumat syntyvät, kun käyttäjät tekevät muutoksia ja vaihtavat tietoja keskenään. Kaikki tapahtumat ja lohkoketjun tiedot ovat julkisia ja niitä voi seurata kuka tahansa. Julkisesta luonteestaan huolimatta lohkoketjun tilin omistajan oikeaa henkilöllisyyttä on kuitenkin vaikea saada selville.



Kuva 5: Lohkoketjun lohkossa on viittaus aina sitä edeltävään lohkoon. Poikkeuksena on ketjun ensimmäinen lohko, jota kutsutaan genesis-lohkoksi.



Kuva 6: Lohkoketjussa lohkoilla on nimitykset, jotka riippuvat niiden sijoittumisesta ketjuun. Orpo-lohkolla ei ole tunnettua vanhempaa eli edellistä lohkoa. Stale-lohko sijaitsee ketjussa, joka ei ole pääketju. Ommer-lohko, toiselta nimeltään eno- tai täti-lohko, ei myöskään sijaitse pääketjussa. [20]

4.1.1 Tiiviste ja konsensusalgoritmi

Kun lohkoketjun verkossa on riittävästi tapahtumia, lohkoketjun louhijat alkavat louhia tapahtumien tiedoista tiivistettä uuden lohkon luomiseksi. Tiiviste (hash) toimii lohkon tunnisteena ja on aina yksilöllinen, kuten sormenjälki. Se muodostetaan lohkon tiedoista

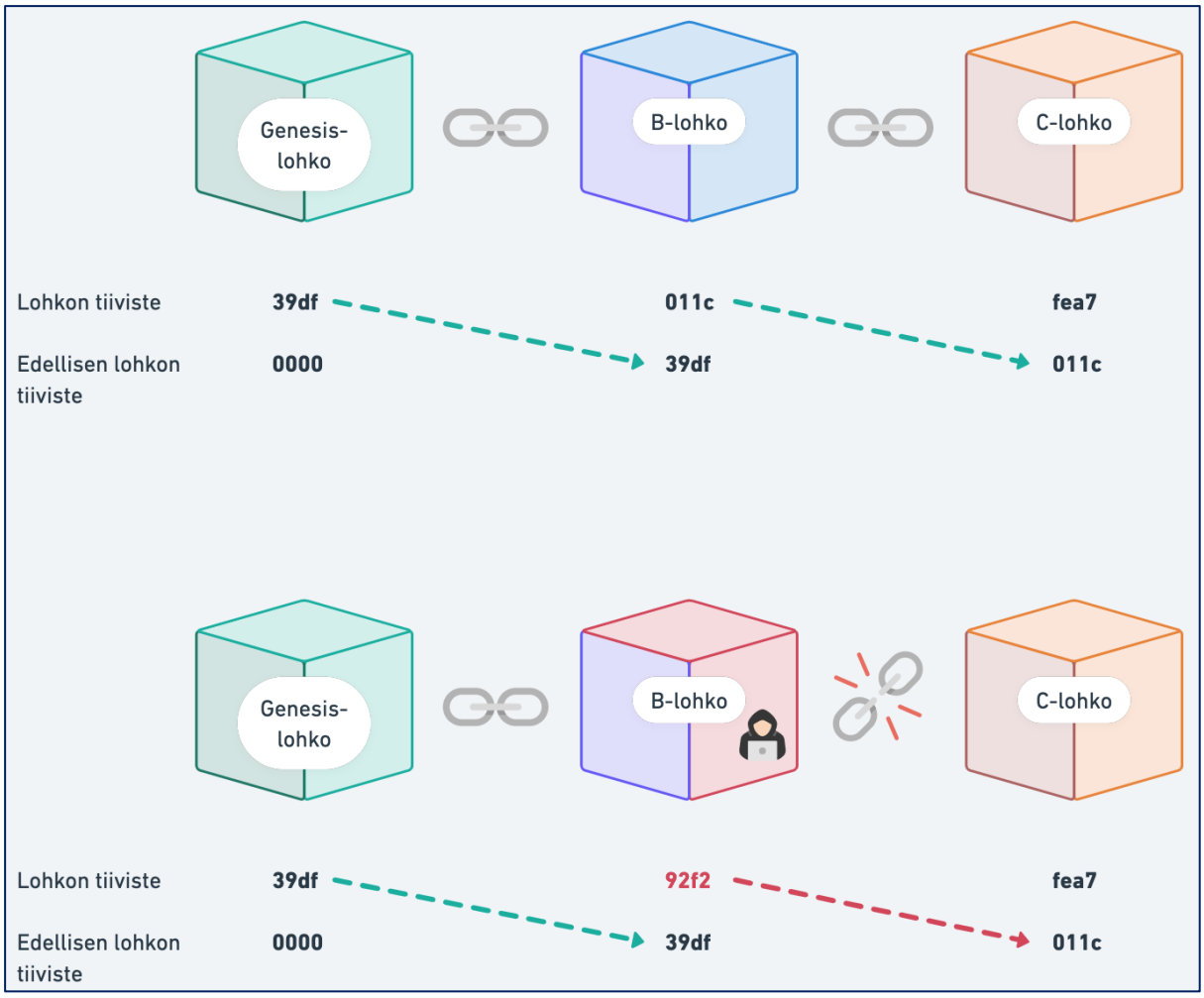
käyttämällä esimerkiksi kryptografista tiivistefunktiota SHA-256 tai Keccak-256, jotka havainnollistetaan kuvassa 7. Tiiviste sisältää näennäisesti satunnaisia kirjaimia ja numeroita, mutta jos sama tieto ajetaan uudelleen tiivistefunktiosta, tuloksena on samanlainen merkkijono. Jos kuitenkin tietoa muutetaan yhdenkin merkin verran, tiiviste muuttuu erilaiseksi.

Tiivistettä ei voi purkaa eikä alkuperäistä tietoa ole mahdollista saada selville, sillä tiivistefunktiot toimivat vain yhteen suuntaan. Näin ollen tiivistefunktioita ei tule sekoittaa salausalgoritmeihin, joissa tiedon pystyy salaamaan ja salauksen purkamaan takaisin alkuperäiseen muotoon. Tiivisteiden käyttö muualla kuin lohkoketjuissa on hyödyllistä, kun halutaan todistaa tieto tiedetyksi ennen jotakin hetkeä, mutta sitä ei haluta vielä paljastaa. Tällöin vakuutena voidaan antaa tiiviste. Kun tieto myöhemmin halutaan paljastaa, tiedon saaja voi ajaa tiedon tiivistefunktion läpi. Tiivisteiden ollessa samanlaiset voidaan varmistaa tiedon antajan väite oikeaksi.

Tieto	**lohkoketju**
SHA-256	**75ad5d4200242d5b36327be3eced090963faeb7efc348677cf3e1202d0043c84**
Keccak-256	**9e990cc613630c07b4d82ac3e0389acab1f57ba76f0998065c6a3266e5b3c5d8**

Kuva 7: Tieto voidaan ajaa kryptografisen tiivistefunktion läpi, jolloin syntyy tiiviste. Tiivisteestä on mahdotonta saada selville alkuperäinen syötetty tieto.

Tiivisteiden käyttäminen lohkoissa tekee lohkoon tallennettujen tietojen muuttamisesta ja väärentämisestä vaikeaa, sillä jokainen muutos tiedoissa muuttaa lohkon tiivistettä. Tämä ilmiö on havainnollistettu kuvassa 8. Koska muutokset muuttavat lohkoa, sitä seuraavan lohkon viittaus muutettuun lohkoon ei pidä enää paikkaansa. Täten ketju on rikkoutunut ja muutos ei ole pätevä, ellei seuraavan lohkon väärää viittausta päivitetä vastaamaan muutetun lohkon uutta tiivistettä. Lohkon tietojen muuttaminen tai väärentäminen luo näin ollen ketjureaktion, jossa muutettua lohkoa seuraavien lohkojen viittaukset edeltävään lohkoon tulee päivittää ja lohkojen tiivisteet laskea uudelleen.



Kuva 8: Lohkoketjussa seuraavan lohkon tulee aina viitata sitä edeltävän lohkon tiivisteeseen. Jos kuitenkin edellistä lohkoa on manipuloitu, ei seuraavan lohkon viittaus enää pidä paikkaansa, sillä edellisen lohkon tiiviste on muuttunut.

Uuden tiivisteen laskeminen tiivistefunktiolla ei ole vaikeaa. Jotta lohkon väärentäminen tai uuden lohkon lisääminen ei olisi helppoa, lohkoketjut käyttävät konsensusalgoritmia. Näitä ovat esimerkiksi työntodiste (Proof of Work) ja varantotodiste (Proof of Stake). Konsensusalgoritmilla määritellään yhteisymmärrys lohkoketjun verkon osallistujien kesken siitä, mikä tieto on oikeaa ja mikä virheellistä. Tähän määrittelyyn käytetään koneellista laskentaa, jolla hidastetaan lohkoketjun prosesseja. Prosesseja hidastamalla pystytään vähentämään virheitä ja hyväksymättömiä muutoksia sekä estämään erilaisia hyökkäyksiä. Lisäksi sillä estetään kryptovaluuttalohkoketjuissa kaksinkertainen kulutus (double spending) eli käyttäjä ei kuluta rahaa, jota hänellä ei enää ole käytössä.

Konsensusalgoritmit edellyttävät tietokoneilta sitä enemmän aikaa, laskentatehoa, muistia ja energiaa, mitä vaativammat kriteerit tiivisteelle on asetettu. Tiivisteen muoto määritellään vaikeus-muuttujalla (difficulty), joka määrää nollien lukumäärän tiivisteen alkuosassa. Mitä

enemmän nollia vaaditaan, sitä enemmän tietokoneelta kuluu aikaa laskea kriteeriä vastaava tiiviste. Vaikeus-muuttuja näin ollen määrittelee sen, kuinka vaikeaa on löytää oikeanlainen tiiviste, jota uuden lohkon luonnissa vaaditaan.

Tiivisteen laskentaprosessia konsensusalgoritmin määrittelemän kriteerin mukaan kutsutaan louhinnaksi. Louhinnassa lohkon nonce-arvoa (number only used once) muutetaan erilaisen tiivisteen saamiseksi. Tätä numeroarvoa muutetaan niin kauan, kunnes tiivistefunktio antaa kriteerejä vastaavan tiivisteen. Kun kriteerejä vastaava tiiviste on louhittu ja lohko lisätty lohkoketjuun, louhija saa palkkion vastineeksi omien laskentaresurssiensa käyttämisestä. Palkkion määrittelee konsensusalgoritmi. Tällä kannustetaan lohkoketjun osallistujia ylläpitämään lohkoketjua konsensusalgoritmin määrittelemien sääntöjen mukaan.

4.1.2 Vertaisverkko

Rakenteensa ja konsensusalgoritmin lisäksi lohkoketjut suojaavat itseään hajauttamisella. Hajautetussa arkkitehtuurissa tiedot tallennetaan useille eri palvelimelle yhden sijaan, jolloin yhden palvelimen toimintahäiriö ei vielä vaaranna tietoja. Lohkoketjuissa tietokannat ovat hajautettuja ja ne ovat kaikille avoimia. Tällaisia tietokantoja kutsutaan vertaisverkoksi (Peer to Peer, P2P), jossa tietokoneet voivat toimia sekä palvelimena että asiakkaana verkon muille osallistujille toisin kuin normaalissa verkossa.

Lohkoketjun vertaisverkkoon kuka tahansa voi liittyä sekä luoda, jakaa ja ylläpitää tietokantoja. Jokainen, joka liittyy tähän verkkoon, saa lohkoketjun tietokannasta tarkan kopion. Tällä osallistuja voi tarkistaa tietojen oikeellisuuden. Vertaisverkon varmennus on yksi merkittävistä lohkoketjun turvallisuusominaisuuksista, mikä varmistaa tiedon luotettavuuden. Kun louhija on onnistuneesti laskenut konsensusalgoritmin kriteerit täyttävän tiivisteen, louhija jakaa lohkon tiedot oikealla nonce-arvolla verkon muille osallistujille. Osallistujista varmentajat laskevat louhijan antamista tiedoista tiivisteen varmistaakseen ehdotetun lohkon oikeellisuuden. Jos yli 50% varmentajista saa louhijan kanssa samanlaisen konsensusalgoritmin kriteerejä vastaavan tiivisteen, uusi lohko hyväksytään ja lisätään lohkoketjuun. Tämän vertaisverkon todentamisen ansiosta välikäsiä ja luotettavia kolmansia osapuolia ei tarvita varmentajiksi. Näin ollen kaikki vertaisverkon jäsenet voivat luottaa tietoon ja muihin osallistujiin ilman, että he tuntevat toisiaan.

4.2 Älysopimus

Älysopimus (smart contract) on lohkoketjussa oleva tietokoneohjelma, jolla pystytään valvomaan sopimuksien tekemistä digitaalisesti ilman kolmansia osapuolia. Se on itseään toteuttava sopimus, jossa se automaattisesti käynnistää tietyn toiminnon. Toiminto toteutetaan, kun sille tehty pyyntö täyttää älysopimukseen asetetut ehdot. Älysopimus valvoo näin ollen ennalta sovittujen ehtojen täyttymisen ja varmistaa osapuolten saavan sen, mitä sopimuksessa on sovittu. Perinteisiin sopimuksiin verrattuna älysopimuksen tavoitteena on tarjota turvallinen ja luotettava sopimus, joka vähentää prosesseja, aikaviiveitä sekä lisäkustannuksia.

Älysopimuksen voi ohjelmoida kuka tahansa. Se voi käsittää esimerkiksi kahden osapuolen välisen kryptovaluuttojen vaihdon tai minkä tahansa digitaalisen omaisuuden hallinnan. Sitä voidaan käyttää myös muihin tarkoituksiin, kuten autentikointijärjestelmiin. Älysopimukset ja niiden toimintalogiikka sekä niihin liittyvät tapahtumat ovat aina julkisia. Lisäksi älysopimuksen tapahtumat ovat peruuttamattomia eikä oletusarvoisesti älysopimuksia voida poistaa. Kuvassa 9 on nähtävillä esimerkki, miltä yksinkertainen älysopimus voi näyttää.

Älysopimukset eivät ole saatavilla kaikissa lohkoketjuissa. Esimerkiksi Bitcoin ei tue älysopimusten käyttöä. Ethereum puolestaan tukee niitä, sillä se on Turing-täydellinen (Turing-complete) [21]. Turing-täydellinen tulee Alan Turingin luomasta konseptista Turingin kone, joka pystyy laskemaan mitä tahansa, kunhan saatavilla on rajaton määrä muistia [22]. Käytännössä tämä tarkoittaa Turing-täydellisyyden mahdollistavan edistyneiden ohjelmien sekä ohjelmointikielien kehittämisen. Tällaisia Ethereumin kehittämiä ohjelmointikieliä ovat esimerkiksi Solidity- ja Serpent-kielet, joilla voidaan suorittaa silmukoita ja ehtolauseita sekä tallentaa sovelluksen paikallista tilaa funktiokutsujen välillä. Näiden kielien avulla Ethereum pystyy tukemaan muiden kuin kryptovaluuttojen käyttäjien sovelluksia, kuten älysopimuksia.

Ethereumissa älysopimus on eräänlainen Ethereum-tili, joka sijaitsee tietyssä osoitteessa Ethereum-lohkossa. Tiliä ei kuitenkaan hallitse käyttäjä, vaan se otetaan käyttöön lohkoketjussa. Käyttöönotettaessa Ethereum kääntää älysopimuksen ensin tavukoodiksi. Tämän jälkeen tavukoodi ladataan Ethereum-virtuaalikoneeseen (Ethereum virtual machine, EVM) ja suoritetaan [23]. Käyttöönoton jälkeen eri käyttäjätilit voivat olla vuorovaikutuksessa älysopimuksen kanssa ja lähettää sille tapahtumia. Nämä tapahtumat suorittavat älysopimuksessa määriteltyjä toimintoja. [24]

Älysopimuksen käyttöönotto ja tapahtumien lähettäminen lohkoketjussa kuluttavat lohkoketjun laskennallisia resursseja. Täten älysopimuksen käyttäminen maksaa. Ethereum-lohkoketjussa tapahtumien kustannuksia kutsutaan kaasuksi (gas). Kaasun hinta määräytyy käyttäjän valmiudesta maksaa digitaalista valuuttaa yhtä kaasuyksikköä kohden. Maksullisuudesta johtuen älysopimuksen sisällön optimointi on kehitysvaiheessa tärkeää. Optimoinnissa muokataan kehitettyä älysopimusta toimintojen laskennallisten resurssien kulutuksen vähentämiseksi. Optimointi ei kuitenkaan saa vaarantaa älysopimuksen turvallisuutta.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.11;

contract HelloContract {

    string message;

    constructor() {
        message = "Hello!";
    }

    function setMessage(string memory newMessage) external {
        message = newMessage;
    }

    function getMessage() external view returns(string memory _message){
        return message;
    }
}
```

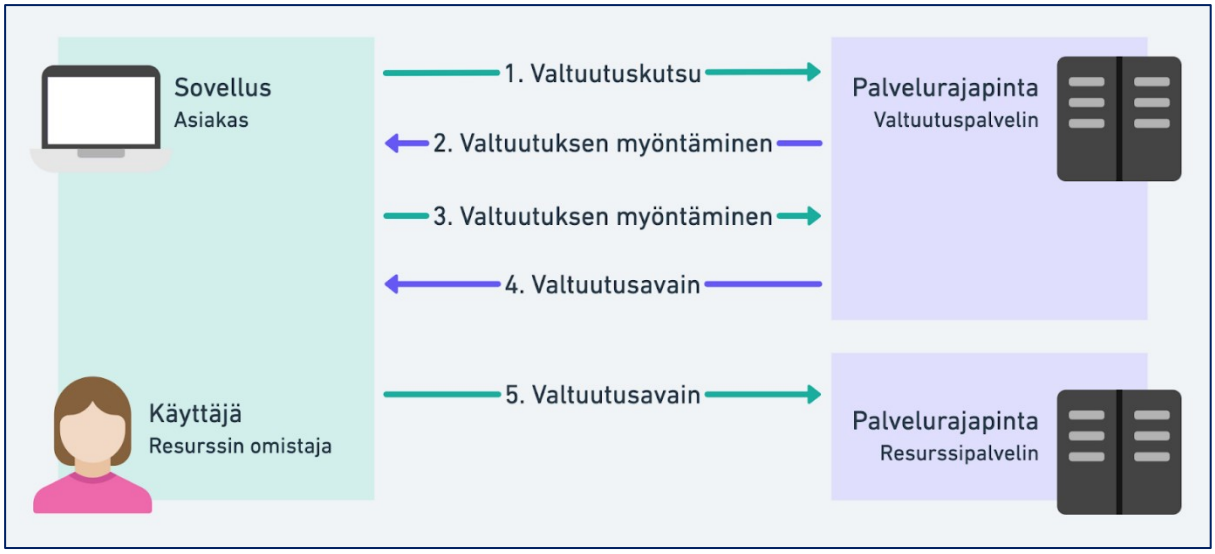
Kuva 9: Esimerkki älysopimuksesta, jossa käyttäjä voi kirjoittaa viestin ja palauttaa sen.

4.3 Lohkoketjupohjaisia autentikointiratkaisuja IoT-verkoille

Erilaisten lohkoketjupohjaisten autentikointijärjestelmien kehittäminen on nostanut suosiotaan lohkoketjujen tarjoaman hyvän turvallisuuden, luotettavuuden ja maailmanlaajuisen saavutettavuuden vuoksi. Lohkoketjujen hyödyntäminen IoT-laitteiden autentikoinnissa on herättänyt kiinnostusta, koska sen on todettu olevan hyvä vaihtoehto OAuth-protokollan (Open Authorization) käytölle. OAuth on yleisesti pääsynhallinnassa ja autentikoinnissa käytössä oleva protokolla. Se on kuitenkin hyvin raskas laskentaresurssirajoitteisille IoT-laitteille.

OAuth-protokollassa todennetaan tokeneiden eli valtuutusavainten avulla verkkosivuston tai sovelluksen käyttäjän henkilöllisyys hänen ja kolmannen osapuolen välillä. Tällöin käyttäjä

pystyy valtuuttamaan kolmannen osapuolen pääsyn omiin yksityisiin tietoihinsa ilman oman käyttäjätunnuksensa ja salasanansa luovuttamista [25]. Tätä mekanismia käyttävät esimerkiksi Google, Facebook ja Twitter, jotta käyttäjät voivat jakaa tiliään koskevia tietoja muiden verkkosivustojen ja sovellusten kanssa. OAuth käyttää todentamiseen kolmatta osapuolta eli valtuutuspalvelinta (Authorization Server, AS). Tällaisen kolmannen osapuolen käyttö lisää vaihdettujen viestien määrää osapuolten välillä, jolloin sen käyttö on yleensä laskentaresursseja kuluttavaa. Koska lohkoketjussa ei käytetä kolmatta osapuolta, useat tutkimukset ovat todenneet lohkoketjua hyödyntävien autentikointijärjestelmien olevan tehokkuudessaan parempi vaihtoehto OAuth-protokollan käytölle. Tällöin viestien määrä puolin ja toisin vähenee, mikä puolestaan pienentää laskentaresurssien kulutusta. Viestien vaihto OAuth-protokollassa on esitetty kuvassa 10.



Kuva 10: Viestinvaihto, kun järjestelmä käyttää OAuth-järjestelmää.

Lohkoketjua voidaan hyödyntää OAuth-protokollan korvaamiseen monin eri tavoin. Al Ahmed et al. [26] ehdottaa tutkimuksessaan kevyttä autentikointiprotokollaa IoT-verkkoihin hyödyntäen lohkoketjua. Lohkoketjun hyödyntäminen tuo autentikointiprotokollaan hajautetun ympäristön, joka on huomattavasti parempi kuin jo pitkään IoT-tietoturvamalleissa käytetyt keskitetyt avaintenvaihtopalvelimet. Keskitettyjen avaintenvaihtopalvelimien heikkoutena on niiden haavoittuvaisuus erilaisille häiriöille ja hyökkäyksille. Huolimatta hajautetusta ympäristöstä lohkoketjujen käytössä on myös haasteensa, sillä niitä hyödyntävien autentikointien tiedetään olevan laskennallisesti vaativia. Tutkimuksessa tämä on kuitenkin ratkaistu Authentication-Chains-protokollalla, joka on kehitetty olemaan kevyt IoT-laitteille. Siinä IoT-laitteiden identiteetin todentamiseen käytettävä lohkoketjun konsensusalgoritmi on mukautettu IoT-laitteiden rajallisiin laskentaresursseihin. Protokollassa IoT-laitteet eli solmut

järjestetään klustereiksi ja niille luodaan omat autentikointilohkoketjut. Nämä klustereiden autentikointilohkoketjut puolestaan yhdistetään vielä toisella lohkoketjulla.

Toisenlaisen ratkaisun OAuth-protokollalle ehdottaa Javed et al. tutkimus [27], jossa ehdotetaan Bitcoin-lohkoketjuun ja hyperelliptiseen käyrään pohjautuvaa autentikointia. Kyseisessä autentikointijärjestelmässä lohkoketjua käytetään varmentajana (Certificate Authority, CA) laitteiden välisessä kommunikoinnissa IoD-verkossa. Täten varsinaista varmenteen myöntäjää tai luotettavaa kolmatta osapuolta (Trusted Third Party, TTP) ei tarvita. Autentikointijärjestelmässä lohkoketjun tapahtumat ovat sertifikaatteja, joilla lähettävä drone tunnistautuu vastaanottavalle dronelle tai ohjausasemalle ja toisin päin. Lohkoketjun hyödyntäminen varmentajana sekä salausmenetelmässä elliptisen käyrän korvaaminen hyperelliptisellä käyrällä vähentävät huomattavasti dronen laskentaresurssien käyttöä tarjoten kuitenkin vahvan suojauksen kommunikoinnille. Tutkimuksen tekijöiden mukaan kehitetty vahva suojaus kestää hyvin yleisiä droneiden kommunikointiin ja autentikointiin kohdistuvia hyökkäyksiä, kuten toisto- (replay), mies välissä - ja palvelunestohyökkäyksiä. Tämän lisäksi järjestelmä on kustannustehokas laskennan ja viestinnän suhteen. Myös suorituskyky ja turvallisuus ovat paremmat kuin muissa vuoteen 2022 mennessä kehitetyissä kilpailevissa autentikointiratkaisuissa.

Lohkoketjun käyttöä IoD-verkon autentikoinnissa puoltaa myös Akram et al. tutkimus [28], jossa hyödynnetään lohkoketjun tukemaa älysopimusta. Älysopimukseen tallennetaan ennalta droneiden osoitteet ja tunnistautumistiedot, joita käytetään autentikoinnin yhteydessä. Dronen autentikoituessa sen tunnistautumistietoja verrataan koko autentikointiprosessin ajan lohkoketjussa jo oleviin tietoihin, jotta epäilyttävät solmut pystytään tunnistamaan. Mikäli tiedot eivät vastaa lohkoketjussa aiemmin tallennettuja tietoja, kyseinen solmu merkitään haitalliseksi ja poistetaan järjestelmästä. Solmu poistetaan verkosta myös siinä tapauksessa, että se autentikoituu jo jonkun muun dronen käytössä olevilla tunnistautumistiedoilla. Lisäksi autentikointijärjestelmä pitää kirjaa, miten solmujen lähettämiä paketteja lähetetään ja vastaanotetaan. Jos haitallinen solmu kaappaa paketteja tai haitallisia paketteja lähetetään paljon kohdesolmulle, verkko tunnistaa epätavanomaisen käytöksen ja poistaa haitallisen solmun. Näillä eri tavoilla järjestelmä pystyy suojautumaan muun muassa Sybil- ja mies välissä -hyökkäyksiltä. Kaksisuuntaisen todennuksensa lisäksi järjestelmästä tekee turvallisen sen käyttämä yksityinen lohkoketju, jossa hyödynnetään kustannustehokasta valtuustodistetta (Proof of Authority, PoA) konsensusalgoritmina käyttäjien yhteisymmärryksen saavuttamiseksi.

Edellä mainituissa tutkimuksissa IoT-laitteiden autentikointijärjestelmiin on kehitetty hyvin erityyppisiä lohkoketjuun pohjautuvia ratkaisuja. Näiden tutkimusten tarkoituksena on ollut vastata perinteisten autentikointijärjestelmien käytön haasteisiin IoT-laitteissa. Merkittävänä haasteena ovat olleet perinteisten autentikointijärjestelmien raskaat prosessit, jotka kuluttavat IoT-laitteiden rajallisia laskentaresursseja. Muun muassa droneissa nämä raskaat laskennat kuluttavat dronen rajallisten laskentaresurssien ohella myös enemmän virtaa ja täten lyhentävät dronen lentoaikaa. Lohkoketjujen hyödyntämisen on kuitenkin todettu tarjoavan varteenotettavan vaihtoehdon keventää IoT-laitteiden autentikointiprosessia sekä samalla tehdä järjestelmistä entistä turvallisempia verrattuna perinteisiin autentikointijärjestelmiin.

Vaikka edellä mainituissa tutkimuksissa on kehitelty hyviä vaihtoehtoja perinteisille autentikointijärjestelmille ja keskitetyille avaintenvaihtopalvelimille, on niissä myös prosesseja, joiden kulkua ei tarkalleen kuvata. Esimerkiksi Javed et al. [27] ja Akram et al. [28] tutkimuksissa ei oteta kantaa siihen, mitkä dronet saavat rekisteröityä autentikointijärjestelmään. Puutteellisen pääsynhallinnan vuoksi haitallinen drone pystyy teoriassa rekisteröitymään järjestelmään samalla tavalla kuin ei-haitalliset dronet. Tällöin edellä mainittujen tutkimusten tavat tunnistaa haitallinen drone eivät päde. Kun haitallinen drone pääsee rekisteröitymään autentikointijärjestelmään, sen ei välttämättä tarvitse kaapata kahden muun dronen välistä kommunikaatiota tai tekeytyä toiseksi droneksi toteuttaakseen hyökkäystään. Edellä mainittujen seikkojen vuoksi, kyseisissä tutkimuksissa tai niiden selonteossa olisi vielä parannettavaa. Näin ollen ne osaltaan antoivat ajatuksia tässä työssä kehitetylle pääsynhallinta- ja autentikointijärjestelmälle.

5 Uusi kehitetty lohkoketjuun pohjautuva ratkaisu

Tämä luku kuvaa tässä työssä kehitettyä lohkoketjuun ja älysopimukseen pohjautuvaa pääsynhallinta- ja autentikointijärjestelmää. Luvussa käsitellään kehitetyn ratkaisun taustoja, arkkitehtuuri, toiminnallisuudet sekä suunnittelun ja kehittelyn vaiheet. Lisäksi käydään läpi kehitetyn älysopimuksen optimointi, testaus ja analysointi.

5.1 Järjestelmän kehittämisen taustat ja konsepti

Järjestelmän kehittäminen alkoi erilaisiin autentikointijärjestelmiin tutustumisella. Aiheen laajuuden vuoksi IoT-laitteille kehitettyjen autentikointijärjestelmien aihe päätettiin rajata lohkoketjua hyödyntäviin ratkaisuihin. Lohkoketjut tarjoavat tunnetusti monia hyviä ominaisuuksia ja sen vuoksi niiden hyödyntäminen autentikoinnissa on yleistynyt nopeasti. Varsinkin IoT-laitteissa lohkoketjujen hyödyntäminen autentikointijärjestelmässä ratkaisee useita IoT-laitteiden autentikointiin liittyviä haasteita.

Uuden järjestelmän kehittäminen alkoi eri tutkimusten innoittamana. Varsinkin Javed et al. [27] ja Akram et al. [28] tutkimukset herättivät ajatuksia näiden autentikointijärjestelmien pääsynhallinnasta. Javed et al. tutkimuksessa dronen tulee lähettää omat salausavaimensa julkista lohkoketjua käyttävälle autentikointijärjestelmälle rekisteröitymistä varten. Rekisteröitymisen jälkeen drone käyttää lohkoketjun tapahtumia sertifikaatteina autentikoinnissa. Akram et al. tutkimuksessa puolestaan rekisteröityminen tapahtuu tallentamalla dronen tunnistautumistiedot älysopimukseen. Näin autentikoituvan dronen tunnistautumistietoja pystytään vertaamaan todentamisvaiheessa älysopimuksessa jo oleviin tietoihin. Kummassakaan tutkimuksissa ei kuitenkaan oteta kantaa siihen, mitkä dronet ovat sallittuja rekisteröitymään näihin autentikointijärjestelmiin ja täten kommunikoimaan verkon muiden laitteiden kanssa. Tämä on potentiaalinen haavoittuvuus. Teoriassa kenellä tahansa on tällöin mahdollisuus rekisteröidä oma drone autentikointijärjestelmään, kunhan rekisteröitymiseen vaadittavat yksityiskohdat ovat tiedossa. Tämän jälkeen millä tahansa dronella on kyky autentikoitua verkon muille laitteille ja lähettää omaa tietoa sekä vastaanottaa muiden tietoa.

Koska ongelmana on, mitkä dronet voivat rekisteröityä autentikointijärjestelmälle, järjestelmä tarvitsee pääsynhallinnan. Kyseisen ongelman ratkaisemiseksi tässä työssä kehitettävän uuden järjestelmän tavoitteeksi tuli parantaa jo olemassa olevaa autentikointijärjestelmää tarjoamalla siihen pääsynhallinta. Erilaisten tutkimusten ja artikkeleiden läpikäynnin jälkeen,

parhaimmaksi ratkaisuksi pääsynhallintaan todettiin lohkoketjussa toimiva älysopimus. Älysopimuksen ja lohkoketjun käyttöön päädyttiin niiden tarjoaman tehokkuuden ja turvallisuuden vuoksi. Lisäksi ratkaisu ei juurikaan lisää dronen laskentaresurssien käyttöä, sillä pääsynhallinta käyttää samaa teknologiaa ja sovellusta kuin autentikointijärjestelmä.

Keskeinen konsepti kehitetyssä pääsynhallinta- ja autentikointijärjestelmässä on älysopimuksen käyttö. Älysopimus huolehtii IoD-verkon autentikointijärjestelmän pääsynhallinnasta hallinnoimalla droneiden sallittujen luetteloa (whitelist). Tällä tavoin varmistetaan, että ainoastaan sallitut dronet saavat rekisteröityä autentikointijärjestelmään. Dronen lisäämisestä älysopimukseen ja hyväksymisestä autentikointijärjestelmään päättävät järjestelmänvalvojat. He ovat luotettavia tahoja, joita älysopimuksen omistaja on lisännyt älysopimukseen. Järjestelmänvalvojien käyttäminen on merkittävä osa kehitettyä järjestelmää, sillä sen käyttö pienentää riskiä, joka usein hajautetussa ympäristössä on. Lohkoketjun hajautetussa ympäristössä kuka tahansa voisi lisätä oikeanlaiset tiedot omaavan dronen älysopimukseen. Näin ollen ilman järjestelmänvalvojia autentikointijärjestelmästä puuttuisi edelleen riittävän turvallinen pääsynhallinta.

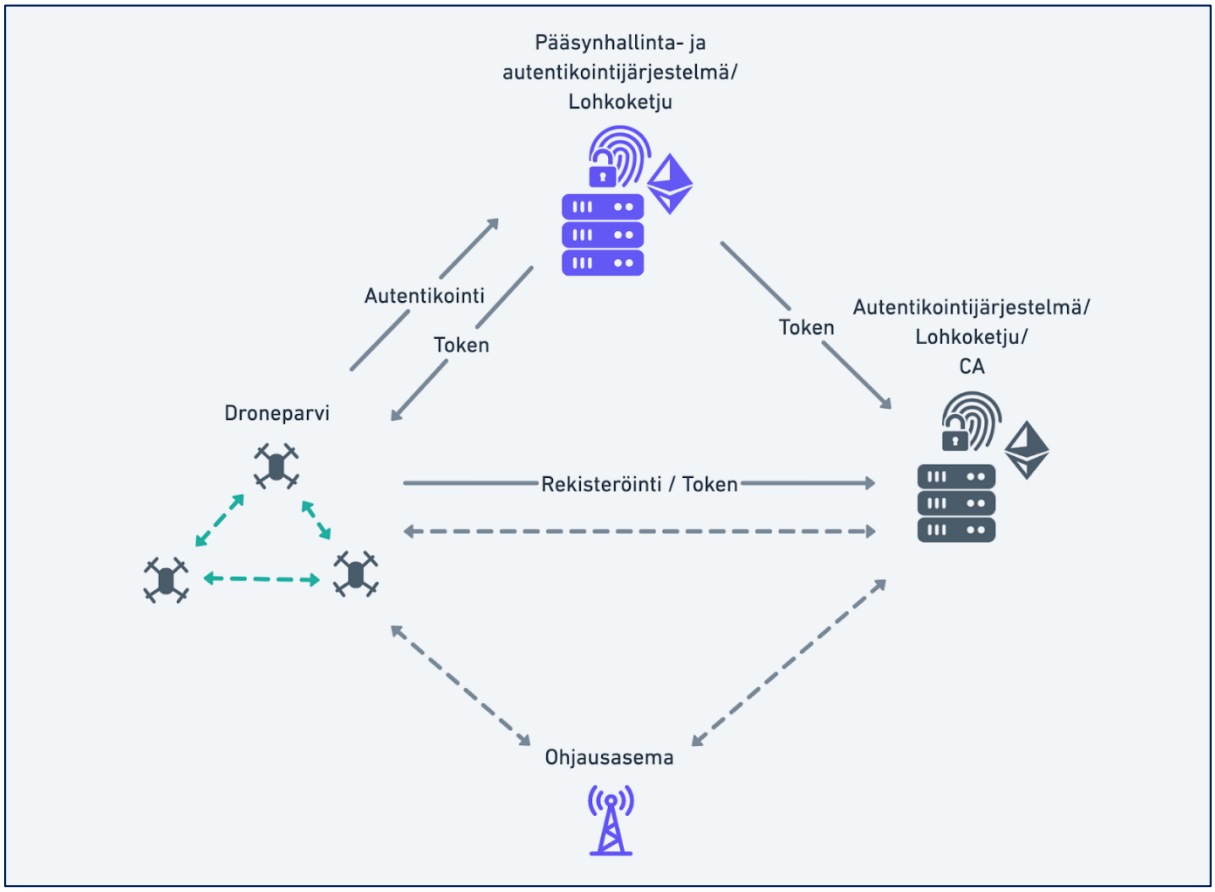
Kehitetyssä järjestelmässä ei oteta kantaa tekniikasta, millä tavalla järjestelmänvalvojaa pyydetään lisäämään drone älysopimukseen. Järjestelmää voidaan käyttää esimerkiksi organisaatiossa, jossa on suuri määrä droneja. Tällöin droneiden lisääminen älysopimukseen voidaan hoitaa sisäisesti järjestelmänvalvojatilillä. Jos puolestaan dronen omistaja on organisaation asiakas, hän voi ottaa yhteyttä organisaatioon, jonka järjestelmänvalvoja lisää dronen tiedot älysopimukseen. Dronen lisääminen älysopimukseen voi tapahtua joko manuaalisesti suoraan järjestelmänvalvojantilillä tai automaattisesti käyttämällä sovellusta. Dronen lisäämisen ohella ei oteta kantaa myöskään dronen poistamiseen älysopimuksesta. Tämä prosessi voi tapahtua automatisoidusti tietyn ajan kuluttua tai dronen tiedot voidaan poistaa manuaalisesti tietyn operaation jälkeen.

5.2 Verkkomalli

Kehitetyn ratkaisun käyttämä verkkomalli jakautuu kolmeen osaan: dronet, pääsynhallinta ja autentikointi. Ensimmäinen osa kattaa verkkoon aikovat ja verkossa olevat dronet, jotka voivat kommunikoida 5G-mobiiliyhteydellä tai Wi-Fi-yhteydellä. Verkko muodostuu, kun dronet ovat toiminnassa ja ne aloittavat tehtävän suorittamisen joko yksin tai parvessa. Tehtävän suorittamiseksi verkon dronet voidaan varustaa erilaisilla ominaisuuksilla. Lisäksi dronet ohjelmoidaan käyttötarkoituksensa mukaan sekä operoivatko ne parvessa. Turvallisen

toiminnan varmistamiseksi parvissa lentävät dronet ohjelmoidaan tunnistamaan naapuriensa vyöhyketunnus, sijainti, nopeus ja korkeus.

Toinen osa verkkomallista käsittää pääsynhallinnan, jolla droneiden pääsyä verkon autentikointijärjestelmään hallinnoidaan. Saatuaan pyytämänsä rekisteröitymisluvan älysopimukselta drone ottaa yhteyttä autentikointijärjestelmään rekisteröityäkseen siihen. Verkkomallin kolmas osa sisältää lohkoketjuun pohjautuvan autentikoinnin. Autentikointi tapahtuu dronen ja ohjausaseman välillä sekä parvessa myös droneiden välillä. Verkkoon voidaan myös lisätä droneiden laskentataakkaa helpottavia sumulaskentaa käyttäviä laitteita. Jos sumulaskentaa käyttävänä laitteena toimii toinen drone, samanlainen autentikointi tapahtuu dronen ja sen välillä. Verkkomalli on esitetty kuvassa 11.

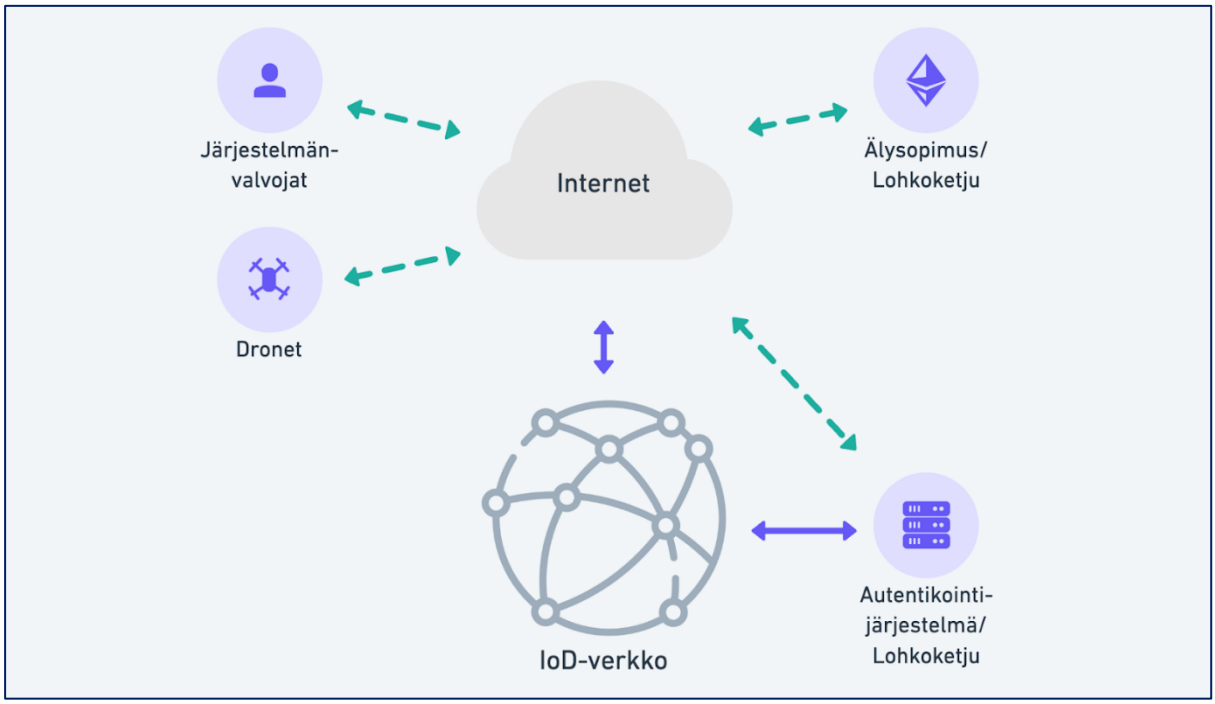


Kuva 11: Verkkomalli, johon kuuluvat dronet, pääsynhallinta- ja autentikointijärjestelmä sekä droneiden ja ohjausaseman autentikoinnista vastaava autentikointijärjestelmä.

5.3 Järjestelmäarkkitehtuuri

Järjestelmän arkkitehtuuri koostuu kolmesta pääkomponentista: järjestelmänvalvojat, dronet ja IoD-verkon autentikointijärjestelmä. Kaikilla pääkomponenteilla on julkiset lohkoketjutilit, joilla ne pystyvät olemaan osa verkon autentikointia ja pääsynhallintaa Internetin välityksellä.

Järjestelmänvalvojilla ja droneilla on yksilölliset Ethereum-osoitteet ja rajapinta älysopimukseen Ethereum-lompakkosovelluksen kautta. Myös IoD-verkon autentikointijärjestelmällä on yksilöllinen Ethereum-osoite. Lisäksi sillä on rajapinta älysopimukseen Ethereum-asiakasohjelman kautta. Kehitetyn ratkaisun arkkitehtuuri esitetään kuvassa 12.



Kuva 12: Kehitetyn ratkaisun järjestelmäarkkitehtuuri.

Kuvassa 13 esitetään järjestelmäarkkitehtuurin käyttäjien, laitteiden ja järjestelmien keskinäinen vuorovaikutus. Osapuolten roolit pääsynhallinnassa ja autentikoinnissa ovat seuraavat:

- **Järjestelmänvalvojat**: IoD-verkon autentikointijärjestelmän pääsynhallinta mahdollistaa luvan rekisteröityä kyseiseen autentikointijärjestelmään. Tästä pääsynhallinnasta vastaavat järjestelmänvalvojat, jotka ylläpitävät älysopimuksessa droneiden sallittujen luetteloa lisäämällä ja poistamalla droneiden tietoja. Järjestelmän ensimmäinen järjestelmänvalvoja on älysopimuksen omistaja. Vain älysopimuksen omistajalla on oikeus lisätä uusia järjestelmänvalvojia älysopimukseen ja poistaa heitä siitä.

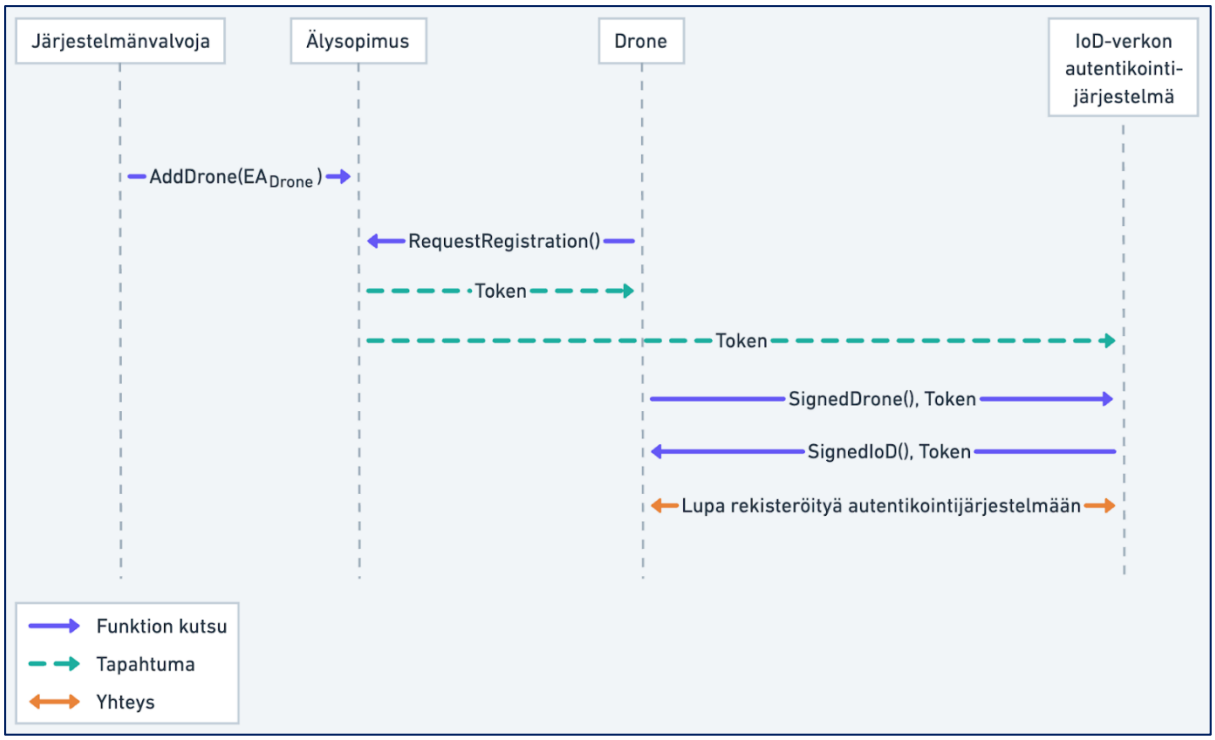
- **Dronet**: Dronet ovat osa verkkoa ja niille on asetettu jokin tehtävä. Kun uusi drone haluaa rekisteröityä autentikointijärjestelmään, tulee sen pyytää rekisteröitymislupaa älysopimukselta. Pääsynhallintaprosessissa älysopimus tarkistaa löytyykö dronen

Ethereum-osoite droneiden sallittujen luettelosta. Jos osoite löytyy, dronella on lupa rekisteröityä autentikointijärjestelmään. Onnistuneen pääsynhallintaprosessin jälkeen älysopimus myöntää dronelle voimassa olevan tokenin, jonka drone allekirjoittaa ja lähettää yksityisen verkon kautta autentikointijärjestelmälle rekisteröitymistä varten.

- **Älysopimus**: Älysopimus hallinnoi autentikointijärjestelmän pääsynhallintaa ylläpitämällä droneiden ja järjestelmänvalvojien sallittujen luetteloita. Älysopimusta käyttävät järjestelmänvalvojat, dronet sekä autentikointijärjestelmä. Kun sallittu drone pyytää rekisteröitymistä autentikointijärjestelmään, älysopimus lähettää dronelle ja autentikointijärjestelmälle requestAccepted-viestin voimassa olevalla tokenilla. Tokenin avulla kyseinen drone pääsee rekisteröitymään autentikointijärjestelmään. Rekisteröitymisen lisäksi token hallitsee TTL-arvolla aikaa, kuinka kauan dronella on lupa olla verkossa. Jos dronen tietoja ei löydy älysopimuksesta eikä näin ollen dronella ole lupaa rekisteröityä autentikointijärjestelmään, lähetetään requestRejected-viesti.

- **Autentikointijärjestelmä**: IoD-verkon autentikointijärjestelmän tehtävänä on huolehtia solmujen välisestä autentikoinnista verkossa. Jotta uusi drone voi rekisteröityä autentikointijärjestelmään, drone lähettää sille älysopimukselta saamansa tokenin yksityisen verkon kautta. Tokenin saatuaan autentikointijärjestelmä tarkistaa sen oikeellisuuden ja vertaa sitä älysopimukselta saamaansa tokeniin. Jos dronen tiedot täsmäävät tokenissa oleviin tietoihin, token on voimassa ja molempien tokeneiden tiedot vastaavat toisiaan, autentikointijärjestelmä lähettää dronelle vastauksena requestAccepted-viestin. Tällöin drone pääsee rekisteröitymään autentikointijärjestelmään. Jos jokin tiedoista ei täsmää tai token ei ole enää voimassa, lähetetään vastauksena requestRejected eikä dronella ole lupaa rekisteröityä järjestelmään.

- **Käyttäjä**: Käyttäjä voi edustaa yksittäistä pilottia, asiakasta tai isompaa organisaatiota. Käyttäjä pyytää järjestelmänvalvojaa lisäämään oman dronensa älysopimukseen, mikäli hänellä itsellään ei ole järjestelmänvalvojan oikeuksia.



Kuva 13: Sekvenssikaavio järjestelmäarkkitehtuurin käyttäjien ja laitteiden vuorovaikutuksesta.

5.4 Simulaatioympäristön kehittäminen

Kehitetyn pääsynhallinta- ja autentikointijärjestelmän pääkomponentin eli älysopimuksen suunnittelu alkoi järjestelmäarkkitehtuurin simulaatioympäristön kehittämisellä. Sen tarkoituksena oli simuloida järjestelmän osallistujien keskinäisiä vuorovaikutuksia ja näin auttaa hahmottamaan, mitä toiminnallisuuksia varsinaiseen älysopimukseen tarvitaan. Simulaatioympäristöön kehitettiin käyttäjät, järjestelmänvalvojat, dronet, älysopimus sekä autentikointijärjestelmä. Ohjelmointi kielenä käytettiin JavaScript-kieltä ja kehitysympäristönä toimi Visual Studio Code -koodieditori.

Simulaatioympäristön kehittäminen aloitettiin järjestelmänvalvojista. Alusta lähtien tehtiin päätös, että älysopimuksen omistaja on automaattisesti älysopimuksen ensimmäinen järjestelmänvalvoja. Lisäksi vain hänellä on oikeus lisätä ja poistaa järjestelmänvalvojia sallittujen luettelosta. Sen sijaan käyttäjien ja droneiden toteutuksissa simulaatioympäristössä oli alkuun epäselvyyttä, kumman tahon tulisi pyytää rekisteröitymislupaa älysopimukselta. Käyttäjältä voitaisiin vaatia tai antaa tunnistautumistiedot, joilla käyttäjä pyytäisi rekisteröitymislupaa dronelleen. Tällainen toteutus on esimerkiksi joissakin IoT-laitteiden ensiaktivoinneissa. Kehittelyn ja testauksen edetessä kuitenkin todettiin, ettei kyseinen logiikka ole ihanteellinen tähän ratkaisuun. Sen sijaan dronen tulee pyytää

rekisteröitymislupaa älysopimukselta. Tämän päätöksen jälkeen simulaatioympäristöön lisättiin vielä lohkoketjuun liittyvät toiminnot. Näitä olivat älysopimus ja siihen kuuluvan tokenin toteutus. Simulaatioympäristön älysopimukseen toteutettiin useampia erilaisia toimintoja ja muuttujia, mutta varsinaisen älysopimuksen kehittelyvaiheessa osa toiminnoista todettiin tarpeettomiksi tai niitä muutettiin. Esimerkiksi useat simulaation toiminnoissa käytetyistä parametreistä olivat lopulta tarpeettomia, sillä simulaatiossa ei pystytty huomioimaan todellista älysopimuksen ja lohkoketjun välistä vuorovaikutusta.

Vaikka simulaatioympäristön toteuttaminen jossain määrin johdatteli harhaan, se lopulta kuitenkin auttoi selventämään kokonaislogiikkaa, jota varsinaisen älysopimuksen kehittämiseen tarvittiin. Varsinainen älysopimus yhdessä lohkoketjun kanssa toimii huomattavasti erilaisella logiikalla kuin JavaScript-kielellä toteutettu simulaatio. Näin ollen oli odotettavaa, ettei simulaatioympäristöön toteutettu älysopimus tulisi olemaan täysin samanlainen kuin lopullisen älysopimuksen toteutus.

5.5 Älysopimuksen toteutus

Lopullisen älysopimuksen kehittäminen alkoi, kun järjestelmäarkkitehtuurin pääpiirteiden simuloinnilla saatiin vaadittavat tiedot älysopimukseen tarvittavista toiminnoista. Älysopimuksen kehittämiseen valittiin Solidity-ohjelmointikieli, koska se on yksi yleisimmin käytetty kieli älysopimuksissa. Lisäksi se on yhteensopiva monien erityyppisten lohkoketjujen kanssa. Kehitysympäristöinä käytettiin Remix IDE:tä ja Visual Studio Codea. Remix IDE on ilmainen ja vapaasti selaimella käytettävä simulaatioympäristö, jossa voi kehittää ja testata älysopimuksia.

5.5.1 Solidity-kielen erikoispiirteet

Solidity-kielestä ei ollut aikaisempaa ohjelmointikokemusta. Sen omaksumista kuitenkin edesauttoivat samat toimintaperiaatteet, joita useat eri ohjelmointikielet noudattavat. Huolimatta samankaltaisista toimintaperiaatteista, jokaisella kiellä on myös yksilöllisiä piirteitä. Nämä piirteet tekevät kielestä hyvän tietyssä ympäristössä tai käyttötarkoituksessa. Solidity-kielen yksilöllisten piirteiden ymmärtäminen oli haastavampaa ja niiden hyödyntäminen älysopimukseen tehtiin vaiheittain eri kehittämiskierroksilla.

Älysopimuksen ja Solidity-kielellä kehittämisen yksi suurimmista oppimisprosesseista oli omaksua älysopimuksen vuorovaikutus lohkoketjun kanssa. Esimerkiksi Solidity-kielessä

muuttujan tyyppinä voi olla address, joka yleensä viittaa Ethereum-osoitteeseen. Lisäksi käyttäjätilin osoite, josta pyyntö tehdään älysopimukseen, voidaan selvittää käyttämällä älysopimuksessa msg.sender-toimintoa.

Toinen erilainen ominaisuus Solidity-kielessä on mapping-lista perinteisen array-listan ohella. Yleensä eri ohjelmointikielissä lista, kuten array-lista, on tietyn pituinen. Listassa arvojen tulostus tai tiedon noutaminen tehdään silmukassa, joka pyörii enintään niin monta kertaa kuin, mikä on listan pituus. Mapping-listaa ei kuitenkaan voi käydä läpi silmukassa, koska sillä ei ole pituutta. Mapping-lista sisältää heti alustuksen yhteydessä kaikki mahdolliset avaimet, joiden arvot on alustettu oletusarvolla. Tämän rakenteen vuoksi tiedon noutaminen mapping-listasta on huomattavasti tehokkaampaa kuin array-listasta, koska tieto pystytään noutamaan heti, kunhan arvon avain on tiedossa. Tehokkuutensa vuoksi mapping-listoja hyödynnetään kehitetyssä älysopimuksessa sallittujen luetteloihin, joihin tallennetaan järjestelmänvalvojien ja droneiden tiedot.

Mapping-listojen käytöstä huolimatta älysopimuksessa hyödynnetään myös kahta array-listaa, jotka toimivat mapping-listojen arvojen tulostamisen apuna. Näihin listoihin tallennetaan mapping-listojen avainten eli järjestelmänvalvojien ja droneiden Ethereum-osoitteet. Kun kaikki mapping-listan arvot halutaan tulostaa, tarvitaan jokaisen arvon avain. Avaimet noudetaan silmukassa array-listasta ja avaimia vastaavat arvot tallennetaan uuteen väliaikaiseen array-listaan. Tämän jälkeen mapping-listan arvoja sisältävä väliaikainen lista voidaan tulostaa.

Huolimatta array-listan tutusta rakenteesta listan toiminta Solidity-kielessä poikkeaa hieman muista ohjelmointikielistä. Solidity-kielessä arvon poistaminen listasta tapahtuu delete-käskyllä. Käsky ei kuitenkaan poista arvoa array-listasta, vaan ainoastaan asettaa sen oletusarvoksi. Tästä syystä myös array-listan pituus ei muutu, vaan pysyy samana. Delete-käskyn erilainen toiminta tuli tästä syystä huomioida dronen ja järjestelmänvalvojan poistamisen yhteydessä. Ratkaisuna ongelmaan käytettiin array-listaan kuuluvaa pop()-toimintoa. Poistettava arvo etsitään for-silmukassa ja sen tilalle siirretään array-listan viimeinen arvo. Tämän jälkeen viimeinen arvo poistetaan pop()-toiminnolla. Ratkaisu muuttaa array-listaan tallennettujen Ethereum-osoitteiden järjestystä, mutta tässä tapauksessa arvojen järjestyksellä ei ole merkitystä.

Listojen ohella uudenlainen Solidity-kieleen liittyvä toimintalogiikka oli vaatimusten käyttö. Älysopimuksissa yleisenä käytäntönä on, että useimmille toiminnoille määritellään vaatimukset. Jos vaatimus ei täyty, pyyntö peruutetaan ja toimintoa ei suoriteta. Vaatimuksia voidaan toteuttaa Solidity-kielessä kahdella eri tavalla. Vaatimus voi olla ehto (require), joka lisätään toiminnon sisälle tai määrite (modifier), jota kutsutaan toiminnon ulkopuolella. Ehto ja määrite ovat periaatteessa täysin samanlaiset toiminnot. Kuitenkin jos samaa vaatimusta käytetään useammassa toiminnossa, on järkevämpää kutsua määritettä kuin kirjoittaa sama ehto moneen kertaan. Kehitetyssä älysopimuksessa hyödynnetään molempia tapoja. Määritteet luotiin tunnistamaan omistaja ja järjestelmänvalvoja. Ehtoina puolestaan toteutettiin duplikaatin lisäämisen sekä olemattoman tiedon poistamisen estäminen järjestelmänvalvojien ja droneiden sallittujen luettelossa.

Älysopimukseen toteutettiin myös rakenteita (struct), jotka määrittelevät tietotyypin ja sisältävät kokoelman tietoja tallentavia muuttujia. Rakenteita luotiin kolme, jotka olivat Admin, Drone ja Token. Lopulliset rakenteet muotoituivat vähitellen älysopimuksen kehittelyn edetessä. Ne muuttuivat yksinkertaisemmaksi, kun osa kehittelyvaiheen tiedoista todettiin tarpeettomiksi. Varsinkin myöhemmin toteutetun kaasuyksikkötestauksen seurauksena, toimintojen logiikan karsiminen yksinkertaisti huomattavasti rakenteita.

Edellä mainittujen rakenteiden ja toimintojen lisäksi myös näkyvyysmääreiden käyttö Solidity-kielessä poikkeaa jonkin verran monista muista ohjelmointikielistä. Yleensä ohjelmointikielissä on esimerkiksi luokat ja paketit. Näissä näkyvyysmääreiden tehtävänä on määritellä, mistä luokkien toimintoja voidaan kutsua. Älysopimuksissa puolestaan ei ole täysin samanlaisia määreitä ja rakennetta, vaan näkyvyysmääreet määrittelevät älysopimuksen toimintojen sisäisen ja ulkoisen näkyvyyden. Solidity-kielessä public-määreellä ilmaistaan, että älysopimuksen toimintoa voidaan kutsua kaikkialta. External-määreellä puolestaan ilmaistaan, että toimintoa voidaan kutsua ainoastaan älysopimuksen ulkopuolelta. External-määreen vastakohtana käytetään private-määrettä. Tällöin private-määreen omaavaa toimintoa voivat kutsua vain älysopimuksen sisällä olevat muut toiminnot. Edellä mainittujen määreiden lisäksi on vielä internal-määre, joka laajentaa private-määrettä. Näin ollen internal-määreen omaavaa toimintoa voivat kutsua myös kyseisestä älysopimuksesta johdetut toiset älysopimukset.

Näkyvyysmääreiden oikea käyttö on yleisesti tärkeä osa ohjelman turvallisuutta, jolla minimoidaan luvattomien muutosten tekeminen jostain muualta päin ohjelmaa. Niillä on kuitenkin myös toinen merkittävä vaikutus älysopimuksissa. Näkyvyysmääreet nimittäin vaikuttavat kaasun kulutukseen eli siihen, miten paljon laskentaresursseja toiminto vie lohkoketjussa. Näkyvyysmääreiden tärkeyden ymmärtäminen tästä näkökulmasta opittiin älysopimuksen optimoinnin ja kaasuyksikkötestauksen yhteydessä. Ensimmäisten testaustulosten analysoinnin jälkeen kaikkien toimintojen määritteet tarkistettiin, jotta älysopimus olisi mahdollisimman tehokas.

Viimeisenä Solidity-kielen vieraampana ominaisuutena olivat toiminnon tilan käyttäytymisen määritteet (state mutability modifier). Näitä määritteitä ovat pure, view ja payable. Pure-toiminnot eivät näytä lohkoketjun tietoja eivätkä muuta lohkoketjun tilaa, mutta niitä voidaan käyttää jonkin muun muuttujan arvon tulostamiseen. View-toiminnot eivät myöskään voi muuttaa lohkoketjun tilaa, mutta ne voivat näyttää lohkoketjun tietoja. Payable-toiminnot puolestaan tarjoavat maksuominaisuuden älysopimukseen eli älysopimus pystyy ottamaan vastaan kryptovaluuttaa. Kehitetyssä älysopimuksessa ei käytetä payable- tai pure-toimintoja. Muutama toiminto sen sijaan on määritelty view-toiminnoksi ja loput toiminnot eivät sisällä lainkaan toiminnon tilan käyttäytymisen määritettä. [29]

5.5.2 Älysopimuksen rakenne

Kehitetty älysopimus sisältää kuusi tilamuuttujaa (state variable), kolme rakennetta, kaksi määritettä sekä yhdeksän toimintoa. Edellä mainittu älysopimuksen rakenne on esitetty kuvan 14 UML-kaaviossa. Älysopimuksen järjestelmänvalvojien ja droneiden tiedot tallennetaan kahteen tilamuuttujaan mapping-listana. Mapping-listan avaimet puolestaan tallennetaan kahteen muuhun tilamuuttujaan array-listana. Viimeiset kaksi tilamuuttujaa tallentavat tiedot älysopimuksen omistajasta ja tokenin yksilöllisestä tunnisteesta. Tilamuuttujat on esitetty kuvassa 15 ja ne ovat seuraavat:

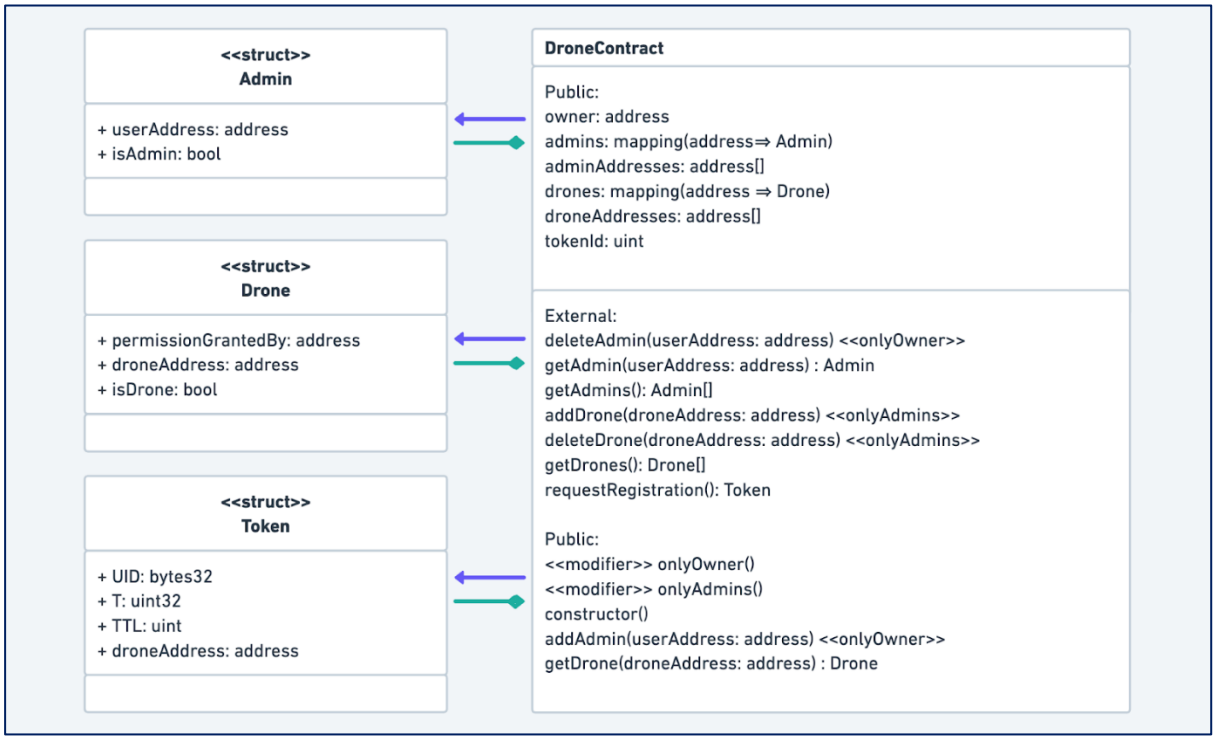
- **owner**: Owner-tilamuuttuja tallentaa älysopimuksen luojan Ethereum-osoitteen. Tätä muuttujaa käytetään onlyOwner-määritteessä.
- **admins**: Admins-mapping-lista sisältää sallittujen järjestelmänvalvojien Admin-rakenteet eli lista edustaa järjestelmänvalvojien sallittujen luetteloa. Tätä luetteloa käytetään onlyAdmins-määritteessä.

- **adminAddresses**: AdminAddresses-array-listaan tallennetaan admins-mapping-listan avainten nimet eli järjestelmänvalvojien Ethereum-osoitteet. Listaa käytetään apuna, kun halutaan tulostaa järjestelmänvalvojien tietoja.

- **drones**: Mapping-lista drones sisältää sallittujen droneiden Drone-rakenteet eli lista edustaa droneiden sallittujen luetteloa. RequestRegistration()-toiminto käyttää tätä selvittääkseen, onko rekisteröitymislupaa pyytävällä dronella lupa rekisteröityä IoD-verkon autentikointijärjestelmään.

- **droneAddresses**: DroneAddresses-array-listaan tallennetaan drones-mapping-listan avainten nimet eli droneiden Ethereum-osoitteet. Tätä käytetään apuna, kun halutaan tulostaa sallittujen droneiden tietoja.

- **tokenId**: TokenId-muuttuja tekee jokaisesta requestAccepted-viestin tokenista yksilöllisen. Kun token luodaan, tokenId:tä kasvatetaan aina yhdellä yksiköllä.



Kuva 14: UML-kaavio kehitetystä älysopimuksesta.

```
address owner;

mapping(address => Admin) admins;
address[] adminAddresses;

mapping(address => Drone) drones;
address[] droneAddresses;

uint tokenId;
```

Kuva 15: Älysopimuksen kuusi tilamuuttujaa.

Kuuden tilamuuttujan ohella älysopimuksessa käytetään kolmea rakennetta, jotka ovat Admin, Drone ja Token. Admin- ja Drone-rakenteet ovat rakenteeltaan lähes samanlaiset, mutta Drone-rakenne sisältää lisäksi tiedon sen lisänneestä järjestelmänvalvojasta. Token-rakenne on puolestaan luotu tallentamaan tokenin sisältämät tiedot. Rakenteet, joita älysopimuksessa käytetään näkyvät kuvassa 16 ja ne ovat seuraavat:

- **Admin**: Admin sisältää muuttujat userAddress ja isAdmin. Muuttuja userAddress käsittää järjestelmänvalvojan Ethereum-osoitteen. Muuttuja isAdmin puolestaan sisältää tiedon, onko haettava järjestelmänvalvoja sallittujen luettelossa. Se on oletusarvoisesti aina epätosi (false), mutta jos järjestelmänvalvoja on sallittujen luettelossa, muuttuja on tosi (true). IsAdmin-muuttujan arvon tarkistamisella estetään duplikaatin syntyminen, mikäli älysopimuksessa jo olevaa järjestelmänvalvojaa yritetään lisätä uudelleen älysopimukseen. Lisäksi isAdmin-muuttujalla pystytään estämään olemattoman järjestelmänvalvojan poistaminen älysopimuksesta.

- **Drone**: Dronella on muuttujina permissionGrantedBy, droneAddress ja isDrone. Muuttuja permissionGrantedBy sisältää dronen älysopimukseen lisänneen järjestelmänvalvojan Ethereum-osoitteen. DroneAddress-muuttuja puolestaan käsittää dronen Ethereum-osoitteen. Lisäksi isDrone-muuttujan avulla varmistetaan, ettei duplikaatteja synny droneiden sallittujen luetteloon tai olematonta dronea yritetä poistaa älysopimuksesta.

- **Token**: Tokenin toteutuksessa on neljä muuttujaa, jotka ovat UID, T, TTL ja droneAddress. UID on yksilöllinen tunniste, johon on käytetty tiivistefunktiota. T on aikaleima ja TTL tarkoittaa ilmoitettua tokenin voimassaolon kestoa. Kun tokenin voimassaoloaika on umpeutunut, tulee dronen pyytää uutta rekisteröitymislupaa. Muuttuja droneAddress on tokenin saaneen dronen Ethereum-osoite.

```
struct Admin {
    address userAddress;
    bool isAdmin;
}

struct Drone {
    address permissionGrantedBy;
    address droneAddress;
    bool isDrone;
}

struct Token {
    bytes32 UID;
    uint T;
    uint32 TTL;
    address droneAddress;
}
```

Kuva 16: Älysopimuksen kolme rakennetta.

Kehitetyssä älysopimuksessa on käytössä kaksi määritettä. Määritteet sisältävät tarkistuksen, onko älysopimukseen pyynnön tehneellä käyttäjätilillä oikeus tehdä kyseistä pyyntöä. Määritteet on esitetty kuvassa 17 ja ne ovat seuraavat:

- **onlyOwner**: OnlyOwner-määritettä käytetään järjestelmänvalvojien hallintaan. Määritteen sisältämän toiminnon saa tehdä ainoastaan älysopimuksen omistaja.
- **onlyAdmins**: OnlyAdmins-määrite on käytössä droneiden sallittujen luettelon ylläpidossa. Määritteen sisältämän toiminnon saavat tehdä ainoastaan älysopimuksen järjestelmänvalvojat.

```
modifier onlyOwner() {
    require(
        msg.sender == owner,
        "Only owner can call this."
    );
    _;
}

modifier onlyAdmins() {
    require(
        admins[msg.sender].isAdmin,
        "Only admins can call this."
    );
    _;
}
```

Kuva 17: Älysopimuksen kaksi määritettä.

Kehitetty älysopimus sisältää seuraavat yhdeksän toimintoa, jotka on esitetty myös kuvissa 18 ja 19:

- **addAdmin(address userAddress) public onlyOwner**: Toiminto lisää järjestelmänvalvojan älysopimukseen. Se saa parametrina tulevan järjestelmänvalvojan Ethereum-osoitteen. Järjestelmänvalvojan lisäämisen voi tehdä ainoastaan älysopimuksen omistaja.

- **deleteAdmin(address userAddress) external onlyOwner**: Toiminto poistaa järjestelmänvalvojan älysopimuksesta. Parametrina käytetään poistettavan järjestelmänvalvojan Ethereum-osoitetta. Poistamisen voi tehdä ainoastaan älysopimuksen omistaja.

- **getAdmin(address userAddress) public view returns(Admin memory)**: Toiminto palauttaa älysopimuksessa olevan järjestelmänvalvojan tiedot. Parametrina annetaan halutun järjestelmänvalvojan Ethereum-osoite. Kuka tahansa voi kutsua tätä toimintoa.

- **getAdmins() external view returns(Admin[] memory)**: Toiminto palauttaa listan kaikista kyselyn hetkellä aktiivisena olevista järjestelmänvalvojista. Kuka tahansa voi kutsua tätä toimintoa.

- **addDrone(address droneAddress) public onlyAdmins**: Toiminto lisää dronen älysopimukseen. Se tallentaa parametrina saadun dronen Ethereum-osoitteen sekä dronen lisänneen järjestelmänvalvojan Ethereum-osoitteen. Dronen voi lisätä ainoastaan järjestelmänvalvoja.

- **deleteDrone(address droneAddress) external onlyAdmins**: Toiminto poistaa dronen älysopimuksesta käyttämällä parametrina saatua dronen Ethereum-osoitetta. Dronen voi poistaa ainoastaan järjestelmänvalvoja.

- **getDrone(address droneAddress) public view returns(Drone memory)**: Toiminto palauttaa älysopimuksessa olevan dronen tiedot. Parametrina annetaan halutun dronen Ethereum-osoite. RequestRegistration()-toiminto käyttää tätä selvittäessään, tuleeko rekisteröitymislupaa hakevan dronen pyyntö hyväksyä. Kuka tahansa voi kutsua tätä toimintoa.

- **getDrones() external view returns (Drone[] memory)**: Toiminto palauttaa kyselyn hetkellä aktiivisina olevien droneiden tiedot. Kuka tahansa voi kutsua tätä toimintoa.
- **requestRegistration() external returns (Token memory)**: Toiminto palauttaa joko requestAccepted- tai requestRejected-viestin. Toiminto tarkistaa, onko rekisteröitymislupaa pyytävä drone sallittujen luettelossa. Jos drone löytyy luettelosta, se antaa dronelle voimassa olevan tokenin.

```
function addAdmin(address userAddress) public onlyOwner {
    require(!admins[userAddress].isAdmin, "Admin already exists.");

    Admin storage newAdmin = admins[userAddress];
    newAdmin.userAddress = userAddress;
    newAdmin.isAdmin = true;

    adminAddresses.push(userAddress);
}

function deleteAdmin(address userAddress) external onlyOwner {
    require(admins[userAddress].isAdmin, "Admin doesn't exist.");

    delete admins[userAddress];

    for (uint i = 0; i < adminAddresses.length;) {
        if (adminAddresses[i] == userAddress) {
            adminAddresses[i] = adminAddresses[adminAddresses.length - 1];
            adminAddresses.pop();
            break;
        }

        unchecked {
            i++;
        }
    }
}

function getAdmin(address userAddress) external view returns(Admin memory) {
    Admin memory admin = admins[userAddress];
    return admin;
}

function getAdmins() external view returns(Admin[] memory) {
    Admin[] memory returnAdmins = new Admin[](adminAddresses.length);

    for (uint i = 0; i < adminAddresses.length;) {
        returnAdmins[i] = admins[adminAddresses[i]];

        unchecked {
            i++;
        }
    }
    return returnAdmins;
}
```

Kuva 18: Älysopimuksessa voidaan lisätä, poistaa ja hakea järjestelmänvalvojia.

```
function addDrone(address droneAddress) external onlyAdmins {
    require(!drones[droneAddress].isDrone, "Drone already exists.");

    Drone storage newDrone = drones[droneAddress];
    newDrone.permissionGrantedBy = msg.sender;
    newDrone.droneAddress = droneAddress;
    newDrone.isDrone = true;

    droneAddresses.push(droneAddress);
}

function deleteDrone(address droneAddress) external onlyAdmins {
    require(drones[droneAddress].isDrone, "Drone doesn't exist.");

    delete drones[droneAddress];

    for (uint i = 0; i < droneAddresses.length;) {
        if (droneAddresses[i] == droneAddress) {
            droneAddresses[i] = droneAddresses[droneAddresses.length - 1];
            droneAddresses.pop();
            break;
        }

        unchecked {
            i++;
        }
    }
}

function getDrone(address droneAddress) public view returns(Drone memory) {
    Drone memory drone = drones[droneAddress];
    return drone;
}

function getDrones() external view returns (Drone[] memory) {
    Drone[] memory allDrones = new Drone[](droneAddresses.length);
    for (uint i = 0; i < droneAddresses.length;) {
        address droneAddress = droneAddresses[i];
        allDrones[i] = drones[droneAddress];

        unchecked {
            i++;
        }
    }
    return allDrones;
}
```

Kuva 19: Älysopimuksessa voidaan lisätä, poistaa ja hakea droneja.

5.6 Älysopimuksen testaus

Kehitetyn pääsynhallinta- ja autentikointijärjestelmän dynaaminen testaus keskittyi älysopimuksen on-chain-toimintojen testaamiseen. Automatisoituun yksikkötestaukseen

käytettiin Remix IDE:tä ja Solidity-kieltä. Kokeilumielessä yksikkötestaukset toteutettiin myös paikallisessa ympäristössä käyttämällä Ganache- ja Truffle-npm-kirjastoja. Kirjastot luovat Ethereum-simulaatioympäristön kehittämistä ja testausta varten. Funktiot paikalliseen yksikkötestaukseen luotiin JavaScript-kielellä.

Automatisoitujen testien lisäksi älysopimus testattiin myös manuaalisesti. Manuaalinen testaus suoritettiin Remix IDE:ssä hyödyntäen sen tarjoamia testitilejä. Älysopimuksen luojan eli omistajan Ethereum-osoitteena käytettiin osoitetta 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4. Toisen järjestelmänvalvojan osoitteena puolestaan käytettiin 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2. Sallitun dronen osoitteena oli käytössä 0x617F2E2fD72FD9D5503197092aC168c91465E7f2.

5.6.1 Järjestelmänvalvojien lisääminen ja poistaminen

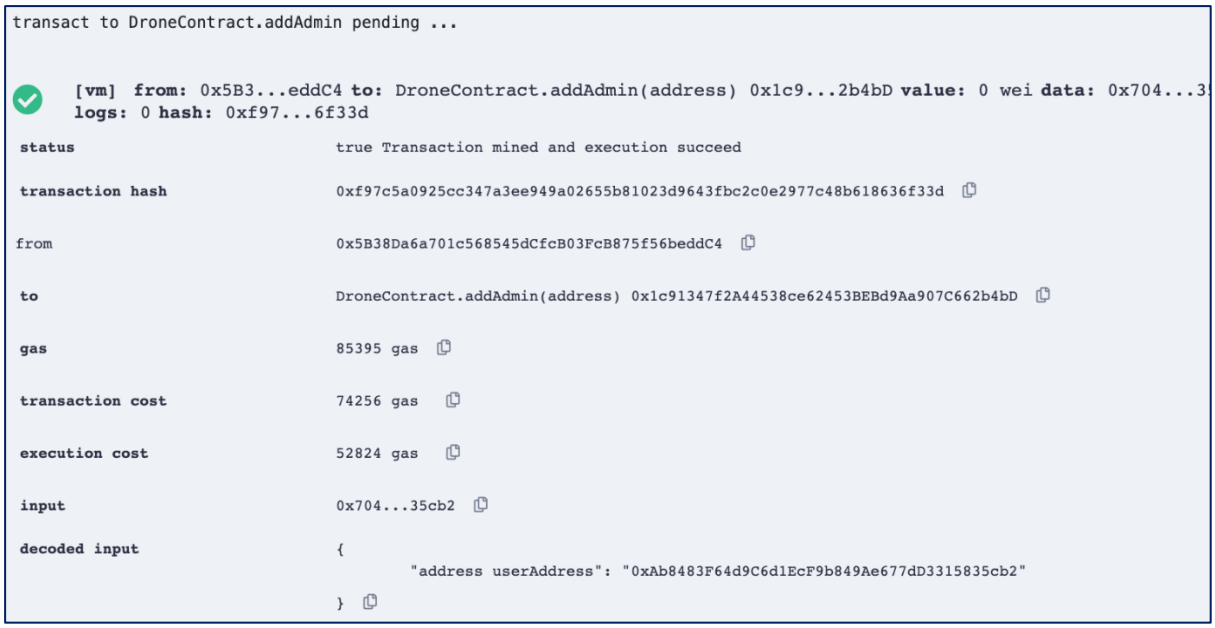
Älysopimuksen ensimmäiseksi järjestelmänvalvojaksi tulee lisätä älysopimuksen luoja eli omistaja. Lisäämisen tulee tapahtua älysopimuksen käyttöönoton yhteydessä eli toiminto tulee tehdä älysopimuksen konstruktorissa. Näin ollen omistajan lisäämistä järjestelmänvalvojien sallittujen luetteloon ei tarvitse tehdä erikseen älysopimuksen käyttöönoton jälkeen.

Onnistunut automaattinen järjestelmänvalvojan lisäys voidaan tarkistaa heti älysopimuksen käyttöönoton jälkeen palauttamalla kaikki tiedot järjestelmänvalvojien sallittujen luettelosta. Luettelossa kuuluisi tällöin olla älysopimuksen omistajan tiedot. Kuva 20 osoittaa, että heti älysopimuksen luonnin jälkeen järjestelmänvalvojia on yksi ja luettelossa oleva Ethereum-osoite vastaa älysopimuksen omistajan Ethereum-osoitetta.

Kuva 20: Älysopimuksen omistaja on onnistuneesti lisätty järjestelmänvalvojien sallittujen luetteloon heti älysopimuksen käyttöönoton yhteydessä.

Uuden järjestelmänvalvojan lisääminen tulee sallia ainoastaan älysopimuksen omistajalta. Näin ollen uuden järjestelmänvalvojan lisäämisen tehneen pyynnön tilin tulee vastata älysopimuksen omistajan Ethereum-osoitetta. Kuva 21 osoittaa pyynnön hylkäämisen, jos pyynnön on tehnyt joku muu kuin älysopimuksen omistaja. Jos pyynnön tekee omistaja, uuden järjestelmänvalvojan lisäys onnistuu. Onnistunut lisäys on esitetty kuvissa 22 ja 23.

Kuva 21: Järjestelmänvalvojan lisäys estetään, jos pyynnön tehnyt ei ole älysopimuksen omistaja.



Kuva 22: Uuden järjestelmänvalvojan lisäys onnistuu, kun pyynnön tehnyt on älysopimuksen omistaja.

Kuva 23: Uuden järjestelmänvalvojan Ethereum-osoite on onnistuneesti lisätty järjestelmänvalvojien sallittujen luetteloon.

Järjestelmänvalvojan lisäämisessä huomioidaan mahdollisen duplikaatin muodostuminen. Tämän tarkoituksena on estää turhan toiminnon toteuttaminen ja täten ylimääräisen kaasun kulutus. Duplikaatin syntymisen estämiseksi käytetään isAdmin-muuttujaa. Kun uutta järjestelmänvalvojaa ollaan lisäämässä, tulee kyseisen muuttujan arvo tarkistaa. Tarkistamiseen tarvitaan tulevan järjestelmänvalvojan Ethereum-osoite. Osoitteella tulee hakea sitä vastaava Admin-rakenne järjestelmänvalvojien sallittujen luettelosta. Jos haetun Admin-rakenteen isAdmin-muuttujan arvo on tosi, kyseessä on duplikaatti. Täten uuden järjestelmänvalvojan lisäämistä ei tule suorittaa. Onnistunut duplikaatin syntymisen estäminen esitetään kuvassa 24.

Kuva 24: Älysopimuksen omistaja yrittää lisätä uudelleen saman järjestelmänvalvojan, jonka hän on jo lisännyt aiemmin kuvassa 22.

Järjestelmänvalvojan lisäämisen ohella järjestelmänvalvoja voidaan myös poistaa älysopimuksesta. Poistaminen tulee sallia ainoastaan älysopimuksen omistajalta. Jos poistamista yrittää joku muu, pyyntö tulee hylätä. Pyynnön hylkäys on esitetty kuvassa 25. Järjestelmänvalvojan poistamisessa omistajan tulee antaa poistettavan järjestelmänvalvojan Ethereum-osoite. Osoitteella älysopimuksen tulee etsiä kyseinen järjestelmänvalvoja sallittujen luettelosta ja poistaa hänet siitä. Onnistunut järjestelmänvalvojan poistaminen esitetään kuvassa 27. Kuvissa 26 ja 28 puolestaan näytetään tilanne ennen ja jälkeen järjestelmänvalvojan poistamista.

Jos poistettavaa järjestelmänvalvojaa ei löydy sallittujen luettelosta, poistopyyntö tulee jättää suorittamatta. Kun järjestelmänvalvojaa ollaan poistamassa, tulee jo aiemmin mainitun isAdmin-muuttujan arvo tarkistaa. Tarkistamiseen tarvitaan poistettavan järjestelmänvalvojan Ethereum-osoite. Osoitteella tulee hakea järjestelmänvalvojien sallittujen luettelosta osoitetta vastaava Admin-rakenne. Jos haetun Admin-rakenteen isAdmin-muuttujan arvo on epätosi, poistettavaa järjestelmänvalvojaa ei ole olemassa. Täten järjestelmänvalvojan poistamista ei suoriteta. Olemattoman järjestelmänvalvojan poistamisen estäminen esitetään kuvassa 29.

Kuva 25: Muun kuin älysopimuksen omistajan pyyntö poistaa järjestelmänvalvoja hylätään.

Kuva 26: Järjestelmän valvojat ennen järjestelmänvalvojan 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB poistoa.

Kuva 27: Älysopimuksen omistajan pyyntö poistaa järjestelmänvalvoja onnistuu.

Kuva 28: Älysopimuksessa olevat järjestelmänvalvojat järjestelmänvalvojan 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB poiston jälkeen.

Kuva 29: Pyyntö poistaa järjestelmänvalvoja, jota ei löydy älysopimuksesta, hylätään.

5.6.2 Droneiden lisääminen ja poistaminen

Droneiden sallittujen luettelon tulee olla tyhjä, kun älysopimus otetaan käyttöön. Tämä voidaan tarkistaa heti käyttöönoton jälkeen palauttamalla kaikki tiedot droneiden sallittujen luettelosta. Kuva 30 osoittaa, että sallittujen luettelo on tyhjä heti älysopimuksen luonnin jälkeen.

Dronen lisäämisen ja poistamisen tulee seurata samaa logiikkaa kuin järjestelmänvalvojan lisääminen ja poistaminen, sillä erolla, että droneiden sallittujen luetteloa voivat hallita omistajan lisäksi myös järjestelmänvalvojat. Jos joku muu kuin järjestelmänvalvoja tekee pyynnön lisätä tai poistaa drone älysopimuksesta, pyyntö tulee hylätä. Pyynnön hylkääminen on esitetty kuvassa 31.

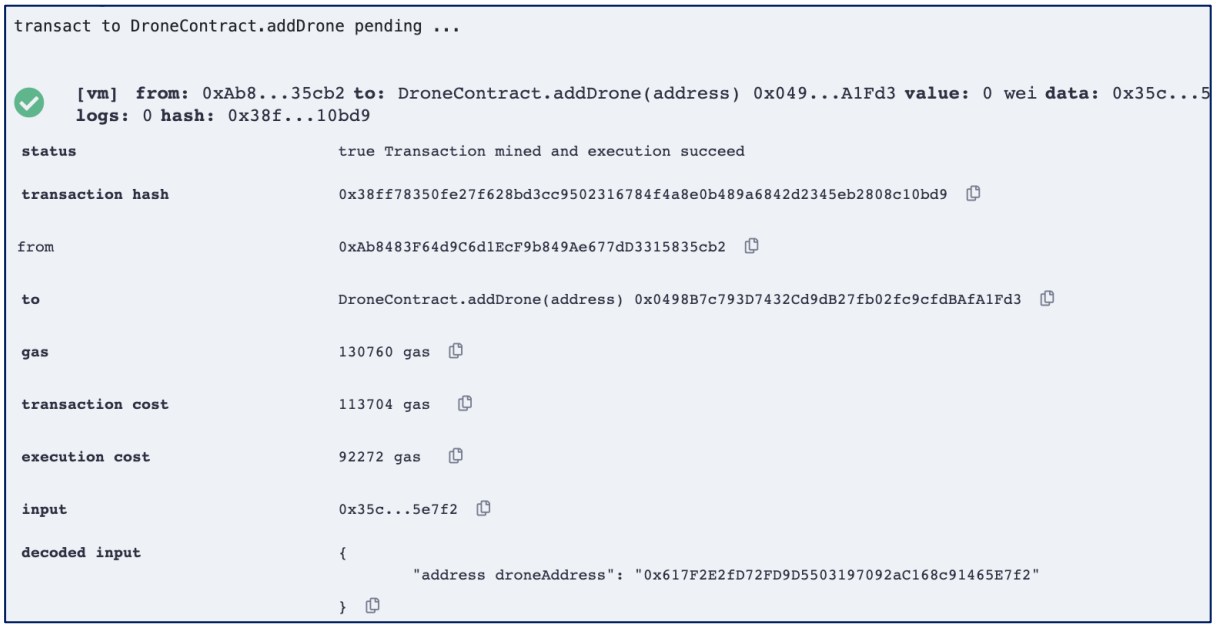
Kuva 30: Heti älysopimuksen käyttöönoton jälkeen droneiden sallittujen luettelo on tyhjä.

Kuva 31: Muun kuin järjestelmänvalvojan pyyntö lisätä drone älysopimukseen hylätään.

Dronea lisättäessä älysopimukseen järjestelmänvalvojan tulee antaa dronen Ethereum-osoite. Tämän jälkeen älysopimuksen tulee automaattisesti tallentaa pyynnön tehneen järjestelmänvalvojan Ethereum-osoite uuden Drone-rakenteen permissionGrantedBy-muuttujaan. Lisäksi annettu dronen Ethereum-osoite tulee tallentaa Drone-rakenteen droneAddress-muuttujaan ja asettaa isDrone-muuttuja todeksi. IsDrone-muuttuja vastaa

isAdmin-muuttujaa järjestelmänvalvojien Admin-rakenteessa. Onnistuneen dronen lisääminen sallittujen luetteloon Drone-rakenteena esitetään kuvassa 32.

Mikäli lisättävä drone on jo olemassa älysopimuksessa, kyseisellä Drone-rakenteella tulisi olla isDrone-muuttujan arvona tosi. Tällöin pyyntö lisätä kyseinen drone älysopimukseen tulee hylätä duplikaatin ja toiminnon toteuttamisen estämiseksi. Onnistunut duplikaatin estäminen on esitetty kuvassa 33.



Kuva 32: Järjestelmänvalvojan pyyntö lisätä drone älysopimukseen onnistuu.

Kuva 33: Järjestelmänvalvoja yrittää lisätä dronen, joka on jo älysopimuksessa.

Dronen poistaminen älysopimuksesta tulee sallia ainoastaan järjestelmänvalvojalta, mikä on nähtävissä kuvassa 34. Poistettaessa dronea sallittujen luettelosta järjestelmänvalvojan tulee antaa poistettavan dronen Ethereum-osoite älysopimukselle. Osoitteella älysopimuksen tulee etsiä kyseinen drone sallittujen luettelosta ja poistaa se sieltä. Kuva 36 osoittaa, että halutun dronen poistaminen älysopimuksesta onnistuu. Kuvissa 35 ja 37 puolestaan esitetään tilanne ennen ja jälkeen dronen poistamista.

Mikäli poistettavaa dronea ei löydy sallittujen luettelosta, pyyntö poistaa drone tulee jättää suorittamatta. Dronen olemassaolo tulee tarkistaa käyttämällä poistettavan dronen Ethereum-osoitetta ja sitä vastaavan Drone-rakenteen isDrone-muuttujaa. Jos isDrone-muuttujan arvo on epätosi, poistettavaa dronea ei ole olemassa. Täten dronen poistamista ei suoriteta. Olemattoman dronen poistamisen estäminen esitetään kuvassa 38.

Kuva 34: Dronen poistaminen älysopimuksesta estetään, jos pyynnön tehnyt ei ole järjestelmänvalvoja.

Kuva 35: Älysopimuksessa olevat dronet ennen dronen 0x17F6AD8Ef982297579C203069C1DbfFE4348c372 poistoa.

Kuva 36: Järjestelmänvalvojan pyyntö poistaa drone älysopimuksesta onnistuu.

Kuva 37: Älysopimuksessa olevat dronet dronen 0x17F6AD8Ef982297579C203069C1DbfFE4348c372 poiston jälkeen.

Kuva 38: Järjestelmänvalvojan pyyntö poistaa drone, jota ei ole älysopimuksessa, hylätään.

5.6.3 Tokenin pyytäminen ja myöntäminen

Dronen tulee pyytää rekisteröitymislupaa IoD-verkon autentikointijärjestelmään älysopimukselta. Älysopimuksen tulee tällöin tarkistaa, onko pyynnön tehneen dronen tiedot droneiden sallittujen luettelossa. Tarkistamiseen tulee käyttää rekisteröitymislupaa pyytävän dronen Ethereum-osoitetta sekä osoitetta vastaavan Drone-rakenteen isDrone-muuttujaa. Jos isDrone-arvo on epätosi, ei dronella ole rekisteröitymislupaa. Täten dronen tulee saada vastauksena requestRejected-viesti. Tämä viesti voidaan nähdä kuvassa 39. Jos puolestaan dronen tiedot ovat sallittujen luettelossa eli isDrone arvo on tosi, älysopimuksen tulee palauttaa tälle requestAccepted-viesti voimassa olevalla tokenilla. Kyseinen viesti on esitetty kuvassa 40.

Kun rekisteröitymislupaa hakeneen dronen pyyntö on tarkistettu ja hyväksytty, älysopimuksen tulee antaa dronelle voimassa oleva token. Älysopimuksen tulee asettaa tokeniin yksilöllinen tunniste, aikaleima pyynnön ajankohdasta, määritelty voimassaoloaika ja pyynnön tehneen dronen Ethereum-osoite. Yksilöllisen tunnisteen tulisi sisältää tokenin yksilöllinen tunniste, dronen Ethereum-osoite sekä aikaleima, jotka kaikki on ajettu Keccak-256-tiivistenfunktion läpi.

Kuva 39: Ei-sallitun dronen pyyntö rekisteröityä IoD-verkon autentikointijärjestelmälle hylätään.

Kuva 40: Sallitun dronen pyyntö rekisteröityä IoD-verkon autentikointijärjestelmälle onnistuu, jolloin se saa voimassa olevan tokenin.

5.7 Älysopimuksen optimointi

Yksikkötestauksen jälkeen keskityttiin älysopimuksen optimointiin, joka on tärkeä osa älysopimuksen tehokkuuden kehittämisprosessia. Kehitetyssä älysopimuksessa optimointi

suoritettiin manuaalisesti. Saatavilla on kuitenkin myös työkaluja, jotka automaattisesti etsivät älysopimuksesta optimointia vaativia kohtia. Kehitetyssä älysopimuksessa optimointiin käytettiin Remix IDE:tä, jolla on helppo suorittaa toimintojen kaasunkulutuksen testausta. Ennen optimoinnin aloittamista älysopimuksen senhetkiset käyttöönottokustannukset tarkistettiin Solidity-kielen omilta sivuilta [30]. Tarkistuksessa käyttöönottokustannukset todettiin äärettömiksi eli älysopimuksessa oli korjauksia ja optimointia vaativia kohtia. Vaadittavien korjausten jälkeen käyttöönottokustannuksiksi tarkentui 1 102 555 kaasuyksikköä. Kustannukset saatiin kuitenkin laskemaan 782 169 kaasuyksikköön älysopimukseen tehtyjen optimointien ansiosta.

Älysopimuksessa oli useita kohtia, joita voitiin optimoida. Ensimmäisenä älysopimuksesta optimoitiin monimutkaiset toiminnot ja rakenteet. Lopullisen älysopimuksen rakennetta voitiin huomattavasti yksinkertaistaa tarpeettomien simulaatioympäristössä käytettyjen muuttujien poistamisella. Rakenteen yksinkertaistaminen puolestaan vähensi laskentaresurssien käyttöä eli kaasun kulutusta.

Älysopimuksen rakenteen korjaamisen jälkeen siirryttiin tutkimaan Solidity-kielen tarjoamia optimointitekniikoita. Solidity-kielen optimointitekniikoista on saatavilla paljon tietoa, miten koodia voidaan optimoida mahdollisimman vähän kaasua kuluttavaksi. Optimoinnin jälkeen kaikkien toimintojen toimivuus varmistettiin vielä kertaalleen suorittamalla yksikkötestaukset uudelleen. Kohdat, joita älysopimuksessa optimoitiin olivat seuraavat [31, 32]:

- **Muuttujien käyttö ja alustus**: Yleensä on suositeltavaa käyttää muuttujia, jonka jälkeen muuttuja sijoitetaan haluttuun kohtaan koodissa. Solidity-kielessä kuitenkin muuttujien käyttö ja varsinkin niiden turha alustus kuluttavat ylimääräistä kaasua. Tämän vuoksi muuttujien käyttöä vähennettiin ja yksittäiset arvot sijoitettiin suoraan haluttuun paikkaan koodia.

- **Uint-tyyppi**: Erilaisia uint-tietotyyppejä on monia ja ne eroavat toisistaan kyvyllään tallentaa eri suuruisia arvoja. Mitä suurempia lukuja uint-muuttuja voi tallentaa, sitä enemmän se yleensä kuluttaa laskentaresursseja. Tämän vuoksi on hyvä määritellä käytettävät uint-muuttujat. Jos uint-tyyppiä ei ole määritetty erikseen, tarkoitetaan uint256-tyyppiä, joka pystyy tallentamaan 2^256-1 suuruisen arvon. Näin suurelle arvolle ei usein kuitenkaan ole tarvetta, joten muuttuja kannattaa määritellä pienemmäksi. Kehitetyssä älysopimuksessa muutettiin tokenin TTL-muuttujan eli tokenin voimassaoloajan uint-tyyppi uint32-tyypiksi. Suurin arvo uint32-tyypille on

2^32 - 1 eli 4 294 967 295, mikä tarkoittaa millisekunneista tunneiksi muutettuna 1193,046 tuntia eli 49 päivää. Yleensä tokeneiden voimassaoloaika on lyhyempi kuin 49 päivää, joten uint32-tyyppi todettiin sopivaksi TTL-muuttujan tyypiksi. Mikäli kuitenkin tokenin voimassaoloaika halutaan asettaa pidemmäksi, voidaan uint-tyyppi määrittää suuremmaksi. Muuttujan arvo voidaan myös tulkita suuremmassa aikayksikössä, kuten sekunneissa.

- **Public, private, external ja internal**: Toiminnallisuuksien näkyvyysmääreet kuluttavat eri määrän kaasua, sillä ne käsittelevät tietoa eri tavoin. Esimerkiksi public-määre kuluttaa enemmän kaasua kuin external-määre johtuen kalliimmasta tiedontallennustavasta. Kehitetyssä älysopimuksessa toiminnallisuuksien näkyvyysmääreet määriteltiin public-määrettä tarkemmiksi niiden näkyvyysvaatimusten mukaan. Näin näkyvyysmääreiden käytön tehokkuus saatiin optimaaliseksi.

- **Unchecked-lohko**: Muuttujan arvon kasvattaminen tai vähentäminen yhdellä yksiköllä tehdään yleensä käyttämällä muotoa n++ tai n--. Solidity-kielessä tämä muoto kuitenkin sisältää arvon muuttamisen lisäksi turvatarkistuksen, joka kuluttaa ylimääräistä kaasua. Turvatarkistus tehdään puskurin ylivuodon ja alivuodon ehkäisemiseksi, koska ne aiheuttavat usein tietoturvaongelmia. Ylivuodossa syöte ei mahdu sille osoitettuun muistialueeseen, kun vastaavasti alivuodossa syöte on liian pieni. Jos turvatarkistuksen haluaa poistaa kaasun kulutuksen vähentämiseksi, voidaan käyttää unchecked-lohkoa arvon kasvattamisen ympärillä. Tällöin tulee kuitenkin olla varma, ettei ylivuotoa ja alivuotoa tapahdu. Kehitetyssä älysopimuksessa käytetään arvon kasvattamista yhdellä yksiköllä esimerkiksi tokenId-muuttujassa. TokenId-muuttujan arvon kasvattamisessa ylivuotoa ei todennäköisesti tapahdu, joten arvon kasvattamisen ympärille lisättiin unchecked-lohko. Älysopimukseen voidaan kuitenkin haluttaessa lisätä if-lause varmistamaan, ettei ylivuotoa varmasti tapahdu. Lause tarkistaa onko tokenId suurempi tai yhtä suuri kuin 2^256-1. Jos näin on, tokenId alustetaan takaisin 0-arvoksi, josta sitä taas kasvatetaan. Tällainen yksinkertainen if-lause kuluttaa edelleen noin puolet vähemmän kaasua kuin ratkaisu, joka ei käytä unchecked-lohkoa.

- **Virheviestit**: Hyvänä käytäntönä on käyttää virheviestiä ilmoittamaan, miksi älysopimus ei voi toteuttaa käyttäjätilin tekemää pyyntöä. Virheviestit kuitenkin vievät

tilaa, joten on suositeltavaa pitää ne mahdollisimman lyhyinä käyttäen enimmillään 32 tavun mittaisia viestejä. Näin minimoidaan tarve tallentaa merkkijono kahteen eri muistipaikkaan. Mitä useampaan muistipaikkaan viestin sisältö joudutaan tallentamaan, sitä kalliimmaksi älysopimuksen käyttöönotto ja suorittaminen tulevat. Tämän vuoksi kehitetyn älysopimuksen virheviestit käytiin läpi ja optimoitiin mahdollisimman lyhyiksi kuitenkaan viestin sisällön tarkoitusta muuttamatta.

- **Useiden ehtojen ja ehtolauseiden käyttö**: Solidity-kielessä on edullisempaa käyttää useaa ehtoa erikseen kuin käyttää yhtä, johon lisätään AND-operaattorilla ehtoja. Sama pätee myös if-lauseiden käyttöön. Älysopimuksen kehitysprosessissa käytettiin alun perin erikseen ehtoa tarkistamaan, onko annettu parametri Ethereum-osoite. Tämän lisäksi samassa lauseessa saattoi olla vielä toinen ehto yhdistettynä AND-operaattorilla. Kaasun kulutuksen vähentämiseksi edellä mainitut ehdot toteutettiin erillisinä lauseina. Myöhemmin testaukseen käytetty Ethereum-osoitteen tarkistus kuitenkin poistettiin tarpeettomana, sillä EVM tarkistaa osoitteen muodon aina automaattisesti.

- **Toimintojen käyttö**: Mitä enemmän älysopimus sisältää toimintoja ja ylipäätänsä koodia, sitä enemmän se kuluttaa kaasua. Kehitetyssä älysopimuksessa käytettiin alun perin omia toimintoja jo olemassa olevan järjestelmänvalvojan ja dronen tarkistamiseen. Toiminnot kuitenkin sisälsivät vain yhden rivin koodia, joten toiminnot poistettiin ja niiden koodi lisättiin suoraan toimintoihin, jotka niitä kutsuivat.

Älysopimuksen koodia voidaan optimoida usealla eri tavalla. Kehitetyssä älysopimuksessa ei kuitenkaan voitu hyödyntää kaikkia näitä tapoja, koska sellaisia kehityskohtia ei ollut käytössä tai niitä ei ollut tarpeellista ottaa käyttöön. Yksi näistä optimointitavoista on Assembly-kielen käyttö [33]. Joitakin kohtia Solidity-kielessä voidaan korvata Assembly-kielellä, kuten for-silmukan toiminnallisuus. Lisäksi on toiminnallisuuksia, joihin on aina käytettävä Assembly-kieltä, sillä niitä ei voida toteuttaa Solidity-kielellä. Assembly-kielen käyttäminen on tehokasta ja vähemmän kaasua kuluttavaa, koska se muistuttaa läheisemmin EVM:n käyttämää kieltä.

5.8 Älysopimuksen analysointi

Älysopimusten käytön yleistyminen myötä niihin on kohdistettu yhä enemmän tietoturvahyökkäyksiä, jotka ovat saattaneet johtaa muun muassa taloudellisiin menetyksiin. Tästä johtuen on tärkeää analysoida älysopimus mahdollisten haavoittuvuuksien löytämiseksi ennen sen käyttöönottoa. Turvallisuuden tarkistaminen voi olla haastavaa ja aikaa vievää, mutta siihen on kehitetty työkaluja, joiden avulla tätä prosessia voidaan automatisoida ja yksinkertaistaa.

Tietoturvahaavoittuvuuksien haun lisäksi älysopimuksien analysointityökalut analysoivat koodin rakennetta, jotta se olisi toteutettu hyvien käytäntöjen mukaisesti. Tällaisia työkaluja Solidity-älysopimuksille ovat muun muassa Manticore, Slither ja Oyente. Älysopimusten analysointityökaluihin ei kuitenkaan pidä tukeutua ja luottaa liikaa, sillä vähäisemmän kehittelykiinnostuksen vuoksi nämä työkalut eivät vielä täysin kykene analysoimaan edistyksellisiä älysopimuksia. Täten analysointityökalut eivät osaa tunnistaa ja havaita kaikkia haavoittuvuuksia tai ne antavat aiheettomia virheilmoituksia.

Kehitetyn älysopimuksen ensimmäinen analysointi tehtiin Remix IDE:ssä saatavilla olevalla Solidity Static Analysis -lisäosalla. Analyysin ensimmäiset varoitukset koskivat manipuloitavissa olevan block.timestampin käyttöä. Manipuloitavuutensa vuoksi block.timestampia ei näin ollen pitäisi käyttää ainoana satunnaisuuden lähteenä. Kehitetyssä älysopimuksessa block.timestampiä käytettiin kuitenkin vain kertomaan, milloin token on luotu. Tokenin aikaleiman paikkaansa pitävyys tarkistetaan IoD-verkon autentikointijärjestelmässä. Autentikointijärjestelmä tarkistaa, ettei aikaleima viittaa tulevaisuuteen tai liian pitkälle menneisyyteen. Lisäksi vanhan tokenin mahdollinen manipulointi voidaan tunnistaa tokeneiden ristiintarkistuksella.

Seuraavat varoitukset koskivat ääretöntä kaasun kulutusta kehitetyn älysopimuksen delete- ja get-toiminnoissa. Kyseiset toiminnot käyttävät for-silmukkaa listojen läpikäymiseen. Nämä listat voivat olla minkä pituisia tahansa, joten listojen läpikäymiseen kuluvaa kaasun määrää ei pystytä arvioimaan eli kulutuksen arvio on ääretön. Jos kulutusta ei pystytä arvioimaan ja toiminnon kaasun tarve on suurempi kuin lohkokaasun raja, toimintoa ei voida suorittaa. Kehitetyssä älysopimuksessa ei kuitenkaan haluta määritellä järjestelmänvalvojien ja droneiden listojen pituutta, joten varoitukset voitiin jättää huomiotta.

Muutama analyysin antama varoitus koski toimintojen tilan käyttäytymisen määritteitä. Varoituksen sisältönä oli, että toiminnon pitäisi mahdollisesti olla constant/view/pure. Varoituksen kohteena olevat toiminnot kuitenkin muokkaavat kehitetyssä älysopimuksessa lohkoketjun tilaa. Lisäksi uudemmat Solidity-versiot eivät enää käytä constant-määritettä, joten kyseiset varoitukset voitiin turvallisesti jättää huomiotta.

Analyysin antamat loput varoitukset koskivat liian samanlaisia muuttujien nimiä, ehtojen käyttöä sekä arvon poistamista listasta. Kehitetyssä älysopimuksessa käytetyt muuttujien nimet ovat loogisia, joten huolimatta niiden samankaltaisuuksista, sekaannuksen riski on pieni. Varoitukset ehtojen käytöstä sisälsivät viestin, että ehdon tulee voida olla myös epätosi. Kehitetyssä älysopimuksessa ehtojen arvot voivat kuitenkin olla myös epätoseja. Arvon poistamiseen liittyvä varoitus puolestaan piti sisällään, että delete-käskyn käyttäminen jättää tyhjän tilan listaan. Mapping-listassa rakenteen poistaminen delete-käskyllä asettaa arvot takaisin oletusarvoiksi eli delete-käsky ei jätä väitettyä tyhjää tilaa listaan. Edellä mainituista syistä johtuen kaikki yllä olevat varoitukset voitiin jättää huomiotta.

Solidity Static Analysis -lisäosan ohella älysopimuksen analysointi tehtiin Slither-työkalulla, joka tekee staattisen koodianalyysin. Slitherin analyysissä käytetään löydöistä värejä punainen, keltainen ja vihreä, vakavuuden mukaan. Punaiset ovat kriittisiä varoituksia ja vihreät ovat kehotuksia, jotka olisi hyvä tehdä.

Slitherin käyttöönotossa koodista tuli kommentoida pois kaikki pop()-toiminnot ja unchecked-avainsanat, jotta se osasi tulkita älysopimusta. Siitä huolimatta, että Slither saatiin toimimaan muutosten jälkeen, se ei osannut vielä antaa tarkkaa analyysia vaan suositteli kokeilemaan kokeellista ABI-enkooderia yksityiskohtaisempaa analysointia varten. Kyseinen tuloste on esitetty kuvassa 41. Kokeellisen ABI-enkooderin käyttöönoton myötä tarkempia huomautuksia tuli muutamia. Usea huomautuksista liittyi _-merkin käyttöön parametrien nimissä. Lisäksi yksi huomautus liittyi numeroiden lukumäärään uint-muuttujan arvossa. Nämä huomautukset on esitetty kuvassa 42. Korjaukset liittyen parametrien nimiin tehtiin poistamalla _-merkki parametrien nimistä. Uint-huomautus suurista luvuista ei puolestaan aiheuttanut korjaustoimenpiteitä, sillä se koski TTL-muuttujan millisekunneissa ilmoitettua aikaa. Huomautuksessa kyse oli, että suuret luvut ovat vaikeita lukea ja niitä voidaan tulkita väärin. Näin ollen Slither suosittelee muuttamaan ne helpommin ymmärrettävään muotoon. Kuvassa 43 Slither-analyysiin jäi vielä kaksi huomatusta korjausten jälkeen. Punaisia eli vakavia varoituksia ei ole yhtään.

```
Compilation warnings/errors on tests/DroneContract.sol:
tests/DroneContract.sol:83:67: Error: This type is only supported in the new experimental ABI encoder.
Use "pragma experimental ABIEncoderV2;" to enable the feature.
    function getAdmin(address _userAddress) external view returns(Admin memory) {
                                                                  ^---^
tests/DroneContract.sol:88:48: Error: This type is only supported in the new experimental ABI encoder.
Use "pragma experimental ABIEncoderV2;" to enable the feature.
    function getAdmins() external view returns(Admin[] memory) {
                                               ^-----^
tests/DroneContract.sol:130:66: Error: This type is only supported in the new experimental ABI encoder.
 Use "pragma experimental ABIEncoderV2;" to enable the feature.
    function getDrone(address _droneAddress) public view returns(Drone memory) {
                                                                 ^---^
tests/DroneContract.sol:135:49: Error: This type is only supported in the new experimental ABI encoder.
 Use "pragma experimental ABIEncoderV2;" to enable the feature.
    function getDrones() external view returns (Drone[] memory) {
                                                ^-----^
tests/DroneContract.sol:148:53: Error: This type is only supported in the new experimental ABI encoder.
 Use "pragma experimental ABIEncoderV2;" to enable the feature.
    function requestRegistration() external returns(Token memory) {
                                                    ^---^
```

Kuva 41: Slither suositteli ottamaan käyttöön kokeellisen ABI-enkooderin, jotta älysopimuksesta voidaan saada parempia analysointituloksia.

```
INFO:Detectors:
solc-0.4.25 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter DroneContract.addAdmin(address)._userAddress (tests/DroneContract.sol#55) is not in mixedCase
Parameter DroneContract.deleteAdmin(address)._userAddress (tests/DroneContract.sol#65) is not in mixedCase
Parameter DroneContract.getAdmin(address)._userAddress (tests/DroneContract.sol#83) is not in mixedCase
Parameter DroneContract.addDrone(address)._droneAddress (tests/DroneContract.sol#101) is not in mixedCase
Parameter DroneContract.deleteDrone(address)._droneAddress (tests/DroneContract.sol#112) is not in mixedCase
Parameter DroneContract.getDrone(address)._droneAddress (tests/DroneContract.sol#130) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conv
entions
INFO:Detectors:
DroneContract.requestRegistration() (tests/DroneContract.sol#148-163) uses literals with too many digits:
        - token = Token(keccak256()(abi.encode(msg.sender,block.timestamp,tokenId)),block.timestamp,3600000,
msg.sender) (tests/DroneContract.sol#155-160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:tests/DroneContract.sol analyzed (1 contracts with 85 detectors), 8 result(s) found
```

Kuva 42: Slitherin antamat huomautukset ennen korjauksia.

```
INFO:Detectors:
solc-0.4.25 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
DroneContract.requestRegistration() (tests/DroneContract.sol#148-163) uses literals with too many digits:
        - token = Token(keccak256()(abi.encode(msg.sender,block.timestamp,tokenId)),block.timestamp,3600000,
msg.sender) (tests/DroneContract.sol#155-160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:tests/DroneContract.sol analyzed (1 contracts with 85 detectors), 2 result(s) found
```

Kuva 43: Slitherin antamat huomautukset korjausten jälkeen.

5.9 Turvallisuusanalyysi

Tässä turvallisuusanalyysissä käsitellään tekniikoita, joilla pystytään varmistamaan luottamuksellisuuden, eheyden sekä saatavuuden (confidentiality, integrity, accessibility, CIA, CIA triad) turvallisuustavoitteet kehitetyssä järjestelmässä. Luottamuksellisuus tarkoittaa tietojen suojaamista luvattomalta käytöltä. Eheydellä pyritään siihen, että tiedot ovat luotettavia, niistä ei puutu mitään ja luvaton käyttäjä ei pysty vahingossakaan

muuttamaan niitä. Saatavuudella puolestaan tarkoitetaan sitä, että resurssit ovat niitä tarvitsevien saatavilla.

Luottamuksellisuus saavutetaan kehitetyssä pääsynhallinta- ja autentikointijärjestelmässä estämällä luvattomien droneiden autentikoituminen muiden IoD-verkossa olevien laitteiden kanssa. Kehitetty älysopimus sallii ainoastaan älysopimuksen omistajan lisätä järjestelmänvalvojia. Tällä tavoin varmistetaan, että ainoastaan luotettavat henkilöt päättävät, mitkä dronet ovat oikeutettuja kommunikoimaan älysopimuksen hallinnoimassa verkossa muiden laitteiden kanssa. Lisäksi älysopimuksen dronelle myöntämä token todennetaan oikeaksi IoD-verkon autentikointijärjestelmän puolesta ristiin tarkistamisella. Näin voidaan varmistua siitä, että juuri kyseisellä dronella on oikeus rekisteröityä järjestelmään ja kommunikoida verkossa olevien laitteiden kanssa. Edellä mainituilla tavoilla varmistetaan, että verkossa olevien laitteiden käsittelemät tiedot pysyvät luottamuksellisena eikä luvaton drone pääse esimerkiksi muuttamaan tietoja, tai syöttämään väärää tai haitallista tietoa.

Koska järjestelmän yhtenä tehtävänä on rajata henkilöitä, ketkä voivat sallia droneiden rekisteröityä IoD-verkon autentikointijärjestelmään, automaattisesti järjestelmän saavutettavuus kärsii. Kehitetyssä älysopimuksessa kuitenkin, saavutettavuus on riittävän hyvä, ja sitä voidaan parantaa lisäämällä enemmän järjestelmänvalvojia. Huomioitavaa kuitenkin on, että mitä enemmän on järjestelmänvalvojia sitä enemmän se heikentää luottamuksellisuutta, koska yhä useampi pystyy hallitsemaan droneiden sallittujen luetteloa.

Järjestelmän eheys ja tiedon muuttumattomuus on erittäin tärkeää missä tahansa tietojärjestelmässä. Kehitetyssä järjestelmässä eheys saavutetaan muun muassa pääsynhallinnalla. Pääsynhallinnassa järjestelmänvalvojat varmistavat IoD-verkkoon rekisteröityvän dronen olevan luotettava. Näin estetään luvattoman dronen pääsy autentikoitumaan verkon muille laitteille ja mahdollisesti muokkaamaan verkon laitteiden käsittelemiä tietoja.

Eheys saavutetaan myös lohkoketjuteknologian avulla. Lohkoketjun lohkojen tietoja on lähes mahdotonta muuttaa. Tämän mahdollistavat lohkoketjun hajautetut tietokannat ja vertaisverkon varmennus. Lohkoketjussa tieto on hajautettuna ympäri verkkoa. Näin hajautettu ympäristö lisää luotettavuutta, sillä siinä ei ole yksittäistä pistettä, jonka toimintahäiriö keskeyttäisi koko järjestelmän toiminnan. Lohkoketjun vertaisverkon varmennuksessa puolestaan verkon käyttäjät tekevät yhteistyötä lohkoketjun suojaamiseksi. Jokainen lohkoketjun toimija pystyy katsomaan, tuottamaan ja ylläpitämään lohkoketjun

tietokantoja. Näin jokaisella on järjestelmän älysopimuksesta aina täydellinen kopio, jonka aitous ja tietojen oikeellisuus varmistetaan lohkoketjun muiden käyttäjien kanssa. Kaikki tapahtumat todennetaan ja tämän jälkeen tallennetaan lohkoketjuun turvallisesti. Tämän myötä järjestelmä pystyy suojautumaan tietojen manipuloinnilta. Lisäksi lohkoketjun läpinäkyvyys lisää käyttäjien vastuullisuutta ja turvallisuutta, koska käyttäjien tallentamat tiedot ja muut muutokset ovat aina jäljitettävissä. Lohkoketju pystyy saavuttamaan eheyden kehitetyssä järjestelmässä myös muuttumattomalla älysopimuksella. Kun älysopimus otetaan käyttöön lohkoketjussa, sitä ei voi enää päivittää. Tällä tavoin kukaan ei voi muokata älysopimusta saadakseen siitä omiin tarkoituksiinsa sopivan.

6 Yhteenveto ja johtopäätös

Tässä työssä ensimmäisenä tavoitteena oli tutkia IoT-laitteiden autentikointijärjestelmiä. Erilaisia IoT-laitteille kehitettyjä autentikointiratkaisuja on runsaasti, joten aiheen rajaaminen tietyn kriteerin mukaan oli asianmukaista. Useiden tutkimusten läpikäymisen jälkeen aihe rajautui lohkoketjua käyttäviin autentikointijärjestelmiin. Lohkoketjuteknologian hyödyntämiseen päädyttiin, sillä se on tehokas ja sisältää monia turvallisuusominaisuuksia. Lohkoketju tarjoaa muun muassa hajautetun ympäristön, joka ei ole yhtä haavoittuvainen erilaisia hyökkäyksiä ja häiriöitä vastaan kuin perinteisissä autentikointiratkaisuissa käytetyt keskitetyt järjestelmät. Lisäksi lohkoketjuun pohjautuvat autentikointijärjestelmät eivät tarvitse kolmatta osapuolta todentamiseen toisin kuin useat muut autentikointiratkaisut.

Aiheen rajauksen jälkeen tähän työhön valikoitui kolme tutkimusta, jotka esiteltiin. Näiden tutkimusten toteuttajat olivat Al Ahmed et al. [26], Javed et al. [27] sekä Akram et al. [28]. Tutkimukset valikoituivat niiden erilaisten tapojensa vuoksi hyödyntää lohkoketjua autentikoinnissa. Nämä tutkimukset antoivat myös eniten ajatuksia tässä työssä kehitetylle pääsynhallinta- ja autentikointijärjestelmälle. Al Ahmed et al. tutkimuksessa hyödynnetään Authentication-Chains-protokollaa sekä IoT-laitteiden klusterointia autentikointilohkoketjujen luomiseksi. Javed et al. tutkimuksen autentikointijärjestelmä perustuu lohkoketjun tapahtumien käyttämiseen varmentajana eli CA:na. Akram et al. tutkimuksessa puolestaan käytetään älysopimusta osana autentikointiprosessia. Sekä Akram et al. tutkimuksessa että Javed et al. tutkimuksessa ajatuksia herätti, etteivät ne ota kantaa siihen, mitkä dronet saavat rekisteröityä autentikointijärjestelmään.

Työn toisena tavoitteena oli parantaa jo olemassa olevaa autentikointijärjestelmää tai kehittää uusi järjestelmä droneille. Aiemmin mainittujen tutkimusten kartoittamisen perusteella päädyttiin kehittämään uusi jo olemassa olevaa autentikointijärjestelmää parantava ratkaisu. Ratkaisu suunniteltiin tarjoamaan pääsynhallintaa autentikointijärjestelmälle, joka käyttää lohkoketjua droneiden rekisteröitymiseen ja autentikointiin.

Tässä työssä kehitetyn pääsynhallinta- ja autentikointijärjestelmän tarkoituksena on estää luvattomien droneiden rekisteröityminen IoD-verkon autentikointijärjestelmään ja täten autentikoituminen verkon muille laitteille. Järjestelmä hyödyntää lohkoketjussa toimivaa älysopimusta, joka sisältää pääsynhallintaa varten luodut droneiden ja järjestelmänvalvojien sallittujen luettelot. Pääsynhallinta tapahtuu järjestelmässä, kun älysopimuksen omistaja antaa

luotettaville tahoille eli järjestelmänvalvojille valtuudet ylläpitää sallittujen droneiden luetteloa organisaation määrittelemillä kriteereillä. Kun drone pyytää rekisteröitymislupaa älysopimukselta, älysopimus tarkistaa kyseisen dronen luvan droneiden sallittujen luettelosta. Autentikoituminen puolestaan tapahtuu, kun drone on todennettu sallituksi. Tällöin älysopimus lähettää voimassa olevan tokenin dronelle ja IoD-verkon autentikointijärjestelmälle. Tämän jälkeen drone lähettää älysopimukselta saamansa tokenin vielä uudelleen autentikointijärjestelmälle. Järjestelmän tehtävänä on suorittaa ristiin tarkistus dronen rekisteröitymistä varten.

Järjestelmän ja älysopimuksen toimintojen kehittämisen jälkeen keskityttiin älysopimuksen optimointiin ja testaukseen. Optimoinnissa älysopimuksen koodia muokattiin huomattavasti tehokkaammaksi ja vähemmän laskentaresursseja kuluttavaksi kuin mitä se oli ennen optimointia. Optimoinnin jälkeen siirryttiin lopullisiin testauksiin, joissa kehitettyä järjestelmää testattiin käyttämällä dynaamista ja staattista analysointia. Analysointi keskittyi järjestelmän pääkonseptin eli älysopimuksen ja sen on-chain-toimintojen testaukseen. Dynaamisilla yksikkötesteillä varmistettiin ja osoitettiin älysopimuksen toimintojen toimivan tarkoituksenmukaisesti. Staattista analysointia puolestaan käytettiin koodin tarkistamiseen sekä etsimään haavoittuvuuksia älysopimuksen toteutuksesta. Analyysien tuloksena toteutuksesta löydettiin muutamia huomionarvoisia kohtia, mutta ei vakavia varoituksia. Löydetyt huomautukset analysoitiin ja tulosten perusteella ne voitiin turvallisesti ohittaa ilman toimenpiteitä.

Dynaamisten ja staattisten analysointien jälkeen älysopimukselle tehtiin vielä turvallisuusanalyysi. Turvallisuusanalyysillä käytiin läpi, miten kehitetty pääsynhallinta- ja autentikointijärjestelmä saavuttaa turvallisuustavoitteet. Turvallisuustavoitteet saavutetaan esimerkiksi rajaamalla henkilöitä, joilla on lupa käsitellä droneiden sallittujen listaa sekä hyödyntämällä lohkoketjua, joka käyttää monia turvallisuusominaisuuksia. Näistä turvallisuusominaisuuksista merkittävimmät ovat hajautettu ympäristö, vertaisverkon varmennus sekä lohkoketjun tarjoama läpinäkyvyys. Lisäksi lohkoketjussa toimivat älysopimukset ovat muuttumattomia eli niiden toteutusta ei voi muuttaa käyttöönoton jälkeen. Edellä mainituilla toteutuksilla järjestelmä pystyy suojautumaan tietojen muuttamiselta ja estämään luvattomien droneiden pääsyn IoD-verkkoon.

Kehitetyssä pääsynhallinta- ja autentikointijärjestelmässä lohkoketjun ja älysopimuksen käyttöön päädyttiin parannettavan autentikointijärjestelmän hyödyntämän lohkoketjun vuoksi.

On tehokasta, kun samaa teknologiaa ja sovellusta pystytään hyödyntämään myös pääsynhallintaan. Tällöin laitteen ei tarvitse sisältää uutta ylimääräistä laskentaresursseja kuluttavaa sovellusta. Näin ollen tässä työssä kehitetty pääsynhallinta- ja autentikointijärjestelmä on tehokas tapa lisätä pääsynhallinta autentikointijärjestelmään, joka käyttää älysopimusta tukevaa lohkoketjua. Tavalliset pääsynhallintajärjestelmät edellyttävät yleensä omat sovelluksensa, minkä lisäksi ne käyttävät kolmatta osapuolta identiteetin todentamiseen. Kehitetty järjestelmä puolestaan hyödyntää dronessa jo käytössä olevaa lohkoketjusovellusta. Tällöin drone ei tarvitse toista sovellusta eikä todentamiseen tarvita kolmatta osapuolta. Täten laskentaresurssien tarve ja kulutus ovat pienemmät. Myös rahallisesti älysopimuksen käyttö on kilpailukykyinen tavallisiin pääsynhallintajärjestelmiin verrattuna. Kustannukset eivät kuitenkaan ole kiinteät. Niihin vaikuttavat käytössä olevan lohkoketjun kryptovaluutan kurssi ja lohkoketjutapahtumien määrä.

Edellä mainitut hyödyt pätevät erityisesti silloin, kun IoD-verkon autentikointijärjestelmä käyttää älysopimusta tukevaa lohkoketjua. Muille autentikointijärjestelmille kehitetty järjestelmä ei tuo yhtä merkittävää etua tehokkuudessa. Muilla autentikointijärjestelmillä tarkoitetaan ratkaisuja, jotka eivät käytä lohkoketjua laisinkaan tai eivät käytä älysopimusta tukevaa lohkoketjua droneiden rekisteröitymiseen ja autentikoitumiseen. Myöskään yksityistä lohkoketjua käyttävä autentikointijärjestelmä ei välttämättä hyödy merkittävästi kehitetystä järjestelmästä, sillä sen lohkoketju sisältää jo jonkin asteisen pääsynhallinnan. Tämä kuitenkin riippuu täysin yksityisen lohkoketjun käyttölaajuudesta. Yleisesti ottaen lohkoketjuun pohjautuvat pääsynhallintajärjestelmät ovat kuitenkin ominaisuuksiltaan ja turvallisuudeltaan parempia kuin perinteiset pääsynhallintajärjestelmät.

Tässä työssä kehitetyssä pääsynhallinta- ja autentikointijärjestelmässä ajatuksena oli luoda älysopimus, joka antaa pohjan, miten sitä voidaan käyttää pääsynhallintatyökaluna IoD-verkossa. Organisaatio pystyy helposti muokkaamaan älysopimuksesta sopivan version omiin tarkoituksiinsa. Esimerkiksi tokeniin voidaan lisätä tietoja, tallentaa tokenit erilliseen listaan tai lisätä toiminto tokenin voimassaoloajan muuttamiseksi.

Tavoitteen mukaisesti työssä onnistuttiin kehittämään ratkaisu, joka parantaa jo olemassa olevaa autentikointijärjestelmää. Kehitetty järjestelmä toimii useille eri IoD-järjestelmille, mutta se on sopivin organisaatioille, joilla on käytössään useita droneja. Tällainen organisaatio voi olla esimerkiksi verkkoyhteyden tarjoaja tai valvontaan, seurantaan tai tavarantoimitukseen keskittynyt taho.

Lähteet

[1] L. S. Vailshery, “IOT connected devices worldwide 2019-2030”, *Statista*, 22-11-2022. [Internet]. Saatavilla: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/. [Viitattu: 29-04-2023].

[2] A. Abdelmaboud, “The internet of drones: Requirements, taxonomy, recent advances, and challenges of research trends”, *MDPI*, 25-08-2021. [Internet]. Saatavilla: https://www.mdpi.com/1424-8220/21/17/5718. [Viitattu: 29-04-2023].

[3] E. Balestrieri, P. Daponte, L. De Vito ja F. Lamonaca, “Sensors and measurements for unmanned systems: An overview”, *Sensors (Basel, Switzerland)*, 22-02-2021. [Internet]. Saatavilla: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7926674/. [Viitattu: 29-04-2023].

[4] M. Quaritsch, E. Stojanovski, C. Bettstetter ja G. Friedrich, “Collaborative microdrones: Applications and research challenges”, *ResearchGate*, 01-2008. [Internet]. Saatavilla: https://www.researchgate.net/publication/220987129_Collaborative_microdrones_Applications_and_research_challenges. [Viitattu: 29-04-2023].

[5] V. S. Bisen, “How AI based Drone Works: Artificial Intelligence Drone Use Cases”, *Medium*, 16-08-2022. [Internet]. Saatavilla: https://medium.com/vsinghbisen/how-ai-based-drone-works-artificial-intelligence-drone-use-cases-7f3d44b8abe3. [Viitattu: 29-04-2023].

[6] K. Konstantoudakis, K. Christaki, D. Tsiakmakis, D. Sainidis, G. Albanis, A. Dimou, and P. Daras, “Drone control in AR: An intuitive system for single-handed gesture control, drone tracking, and contextualized camera feed visualization in augmented reality”, *MDPI*, 10-02-2022. [Internet]. Saatavilla: https://www.mdpi.com/2504-446X/6/2/43. [Viitattu: 29-04-2023].

[7] T. Richmond, “How do drones and augmented reality work together?”, *TechTarget: IoT Agenda*, 03-01-2019. [Internet]. Saatavilla: https://www.techtarget.com/iotagenda/answer/How-do-drones-and-augmented-reality-work-together. [Viitattu: 29-04-2023].

[8] L. Davies, R. Bolam, Y. Vagapov ja A. Anuchin, “Review of unmanned aircraft system technologies to enable beyond visual line of sight (BVLOS) operations”, *Glyndŵr University*, 03-10-2018. [Internet]. Saatavilla: https://glyndwr.repository.guildhe.ac.uk/id/eprint/17376/1/GURO_381_PID5580125.pdf. [Viitattu: 29-04-2023].

[9] E. Politi, I. Panagiotopoulos, I. Varlamis ja G. Dimitrakopoulos, “A survey of UAS Technologies to enable beyond visual line of sight (BVLOS) operations”, *ResearchGate*, 01-2021. [Internet]. Saatavilla: https://www.researchgate.net/publication/351404253_A_Survey_of_UAS_Technologies_to_Enable_Beyond_Visual_Line_Of_Sight_BVLOS_Operations#pf2. [Viitattu: 06-05-2023].

[10] J.-P. Yaacoub, H. Noura, O. Salman, and A. Chehab, “Security analysis of drones systems: Attacks, Limitations, and recommendations”, *Internet of Things*, 09-2020. [Internet]. Saatavilla: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7206421/. [Viitattu: 29-04-2023].

[11] K.-Y. Tsao, T. Girdler ja V. G. Vassilakis, “A survey of cyber security threats and solutions for UAV Communications and flying ad-hoc networks”, *ScienceDirect*, 23-05-2022. [Internet]. Saatavilla: https://www.sciencedirect.com/science/article/pii/S1570870522000853. [Viitattu: 29-04-2023].

[12] A. Khan, “Hacking the Drones”, *OWASP Foundation*. [Internet]. Saatavilla: https://owasp.org/www-chapter-london/assets/slides/OWASP201604_Drones.pdf. [Viitattu: 29-07-2023].

[13] I. Azhar Mohammed, “Systematic review of Identity Access Management in Information Security.”, *ResearchGate*, 07-2017. [Internet]. Saatavilla: https://www.researchgate.net/publication/353887659_SYSTEMATIC_REVIEW_OF_IDENTITY_ACCESS_MANAGEMENT_IN_INFORMATION_SECURITY. [Viitattu: 08-05-2023].

[14] H. Kettani, P. Renee Carnley, “Identity and access management for the Internet of Things”, *ResearchGate*, 12-2019. [Internet]. Saatavilla: https://www.researchgate.net/publication/338228158_Identity_and_Access_Management_for_the_Internet_of_Things. [Viitattu: 08-05-2023].

[15] “2022 data breach investigations report”, *Verizon Business*, 2022. [Internet]. Saatavilla: https://www.verizon.com/business/resources/reports/dbir/. [Viitattu: 08-05-2023].

[16] “How does Bitcoin work?”, *bitcoin.org*. [Internet]. Saatavilla: https://bitcoin.org/en/how-it-works. [Viitattu: 30-04-2023].

[17] C. Smith (@corwintines), “Mining”, *ethereum.org*, 12-04-2023. [Internet]. Saatavilla: https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/mining/. [Viitattu: 30-04-2023].

[18] Z. Zheng, S. Xie, H. Dai ja X. Chen, “An overview of blockchain technology architecture consensus and future trends”, 06-2017. [Internet]. Saatavilla: https://www.researchgate.net/publication/318131748_An_Overview_of_Blockchain_Technology_Architecture_Consensus_and_Future_Trends. [Viitattu: 30-04-2023].

[19] B. Koteska, E. Karafiloski ja A. Mishev, “Blockchain Implementation Quality Challenges: A literature review”, *ResearchGate*, 09-2017. [Internet]. Saatavilla: https://www.researchgate.net/publication/320127088_Blockchain_Implementation_Quality_Challenges_A_Literature_Review#pf2. [Viitattu: 08-05-2023].

[20] S. Bansal, “The story of those that didn't make it - stales, uncles and ...orphans?”, *Medium*, 25-05-2020. [Internet]. Saatavilla: https://medium.com/tech-iiitg/the-story-of-those-that-didnt-make-it-stales-uncles-and-orphans-cea80de7c549. [Viitattu: 30-04-2023].

[21] M. Jansen, F. Hdhili, R. Gouiaa ja Z. Qasem, “Do smart contract languages need to be Turing complete?”, *ResearchGate*, 03-2019. [Internet]. Saatavilla: https://www.researchgate.net/publication/332072371_Do_Smart_Contract_Languages_Need_to_be_Turing_Complete. [Viitattu: 01-05-2023].

[22] A. Reprintsev, “Turing completeness”, *ResearchGate*, 04-2018. [Online]. Available: https://www.researchgate.net/publication/324334114_Turing_Completeness. [Viitattu: 01-05-2023].

[23] B. Peh, “Solidity bytecode and opcode basics”, *Medium*, 15-09-2017. [Internet]. Saatavilla: https://medium.com/@blockchain101/solidity-bytecode-and-opcode-basics-672e9b1a88c2. [Viitattu: 01-05-2023].

[24] P. Wackerow (@wackerow), “Introduction to smart contracts”, *ethereum.org*, 02-09-2022. [Internet]. Saatavilla: https://ethereum.org/en/developers/docs/smart-contracts/#:~:text=A%20%22smart%20contract%22%20is%20simply,be%20the%20target%20of%20transactions. [Viitattu: 01-05 -2023].

[25] D. Hardt, “The OAuth 2.0 authorization framework”, *IETF Datatracker*, 13-10-2012. [Internet]. Saatavilla: https://datatracker.ietf.org/doc/html/rfc6749. [Viitattu: 01-05-2023].

[26] M. T. Al Ahmed, F. Hashim, S. Jahari Hashim ja A. Abdullah, “Authentication-chains: Blockchain-inspired lightweight authentication protocol for IoT networks”, *MDPI*, 08-02-2023. [Internet]. Saatavilla: https://www.mdpi.com/2079-9292/12/4/867. [Viitattu: 05-04-2023].

[27] S. Javed, M. Asghar Khan, A. Muhammad Abdullah, A. Alsirhani, A. Alomari, F. Noor ja I. Ullah, “An efficient authentication scheme using blockchain as a certificate authority for the Internet of Drones”, *MDPI*, 20-09-2022. [Internet]. Saatavilla: https://www.mdpi.com/2504-446X/6/10/264. [Viitattu: 12-01-2023].

[28] J. Akram, A. Akram, R. H. Jhaveri, M. Alazab ja H. Chi, “BC-IoDT: Blockchain-based framework for authentication in Internet of Drone Things”, *arXiv.org*, 21-10-2022. [Internet]. Saatavilla: https://arxiv.org/pdf/2210.11745.pdf. [Viitattu 05-04-2023].

[29] A. Adediran, “Function state mutability in Solidity”, *Medium*, 28-02-2022. [Internet]. Saatavilla: https://medium.com/coinmonks/function-state-mutability-in-solidity-acb850eedccc. [Viitattu: 01-05-2023].

[30] “Solidity assembly”, *Solidity 0.5.3 documentation*. [Internet]. Saatavilla: https://docs.soliditylang.org/en/v0.5.3/assembly.html. [Viitattu: 02-05-2023].

[31] W. Shahda, “Gas optimization in solidity part I: Variables”, *Medium*, 08-05-2019. [Internet]. Saatavilla: https://medium.com/coinmonks/gas-optimization-in-solidity-part-i-variables-9d5775e43dde. [Viitattu: 11-03-2023].

[32] H. Pandey, “How we can optimize our solidity smart contracts?”, *Medium*, 12-12-2022. [Internet]. Saatavilla: https://medium.com/coinmonks/how-we-can-optimize-our-solidity-smart-contracts-b966e3e32fff. [Viitattu: 07-05-2023].

[33] “Try Solidity here”, *soliditylang.org*. [Internet]. Saatavilla: https://soliditylang.org/. [Viitattu: 02-05-2023].