Samuel Nordström

Lohkoketjut SSL/TLS-salauksessa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

23.4.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Samuel Nordström Lohkoketjut SSL/TLS-salauksessa 28 sivua 23.4.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Ilpo Kuivanen

Insinöörityön tavoitteena oli hyödyntää lohkoketjuja selainten SSL/TLS-salauksessa (TLS-salaus). Työn tavoitteena oli parantaa TLS-salausta poistamalla TLS-salauksesta sen heikoin lenkki, Certificate Authorityt (CA).

Certificate Authorityt allekirjoittavat verkkosivujen sertifikaatteja, ja tämän järjestelmän toimiminen vaatii, että käyttäjän pitää luottaa, että CA:t hoitavat hommansa kunnolla. Käytännössä se tarkoittaa sitä, että CA:t allekirjoittavat sertifikaatteja vain verifioiduille verkkotunnusten omistajille ja lisäksi heillä pitää olla tietoturva kunnossa, etteivät hakkerit pääse salausavaimiin käsiksi.

Lohkoketjut ovat jaettuja tietokantoja, ja näiden avulla voisi olla mahdollista saada TLS-salaus toimimaan ilman Certificate Authorityja ja sitä kautta saada TLS-salauksesta turvallisempaa, jossa loppukäyttäjät saisivat itse hallita kaikkea siihen liittyvää. Tämän työn tarkoituksena onkin tutkia tällaisen teknisen toteutuksen mahdollisuutta ja toteuttaa MVP-versio (minimum viable product) tästä, jossa käyttäjä voi tallentaa lohkoketjuun verkkotunnuksensa ja sen sertifikaatin julkinen salausavain verifioiduille verkkotunnuksille. Näin selaimet voivat tarkistaa lohkoketjusta, onko verkkosivun antama sertifikaatti luotettava vai ei, eikä välikäsiä tarvita.

Sovelluksen kehityksessä käytettiin Ethereum-lohkoketjua, joka mahdollistaa Turing-täydellisten (Turing-complete) sovellusten rakentamisen lohkoketjuun ilman, että tarvitsee itse rakentaa lohkoketjua. Kielenä lohkoketjusovelluksessa käytettiin Ethereumin kehittämää Solidityä. Käyttöliittymässä käytettiin ReactJS-kirjastoa yhdessä web3-kirjaston kanssa, joka mahdollistaa helpon kommunikoinnin Ethereum-lohkoketjun kanssa.

Avainsanat	lohkoketju, ssl, tls, ethereum

Author Title Number of Pages Date	Samuel Nordström Using blockchains in SSL/TLS 28 pages 23 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Ilpo Kuivanen, Senior Lecturer

The purpose of this thesis is to leverage blockchains for SSL/TLS (TLS). The goal is to make TLS safer by making the weakest link of TLS – Certifcate Authorities (CA) – obsolete.

Certificate Authorities sign the certificates for web domains and in order for this system to work, end user must trust that CAs do their due diligence properly. In practice this means that CAs must only sign certificates for verified domain owners and their cyber security must be handled properly to prevent their private keys from leaking to hackers.

Blockchains are distributed databases and with them it could be possible to make CAs obsolete, enhance TLS's security and give end users the power to handle all their TLS private keys. In this thesis we inspect the possibility for a technical implementation like this. We will also create an MVP (minimum viable product) blockchain application, which will allow end users to save the domain and the domain certificate's public key to a blockchain from which browsers could then query if domain's provided certificate is to be trusted or not and therefore remove the need for third parties.

In the development of the application, Ethereum blockchain was used. It allows developers to create Turing-complete applications to their blockchain without the need to create own implementation of blockchains. Language for Ethereum-development was Ethereum's own programming language, Solidity. In front-end ReactJS library was used to create user interface in compliance with web3 library which allows easy communication with Ethereum blockchain.

Keywords	blockchain, ssl, tls, ethereum

Sisällys

Lyhenteet ja käsitteet

CA — Certificate Authority, eli verkkosivujen käyttämien sertifikaattien allekirjoittaja/varmentaja.

Dapp — Desentralisoitu applikaatio, eli ohjelma joka toimii ainakin osittain desentralisoidusti.

Desentralisaatio — Prosessi, jossa päätöksentekovaltaa jaetaan yksittäisiltä tahoilta useammille.

MItM — Man-In-The-Middle, eli hyökkäys jossa hyökkääjä asettuu uhrin ja tämän koneelta lähtevän ja tulevan verkkoliikenteen väliin ja pystyy tarkastelemaan tässä verkkoliikenteessä liikkuvaa tietoa.

SSL — Secure Sockets Layer, eli vanha versio TLS-salauksesta.

TLS — Transport Layer Security, eli Internet-sovellusten käyttämä salausprotokolla. Lyhennettä käytetään tässä työssä puhuttaessa SSL/TLS-salauksesta.

1 Johdanto

SSL/TLS-salausta käytetään nykyään lähes kaikilla moderneilla kilpailukykyisillä verkkosivuilla. Se mahdollistaa käyttäjän verkkoliikenteen salaamisen verkkosivun kanssa ja estää samanaikaisesti Man-In-The-Middle (MItM) -hyökkäykset, jolla kolmas osapuoli pystyy kuuntelemaan käyttäjän ja verkkosivun välistä liikennettä.

SSL/TLS-salauksessa on kuitenkin vielä yksi heikko lenkki, jolle loppukäyttäjät eivät voi mitään. Käyttääkseen TLS-salausta julkisella verkkosivullaan verkkosivun omistaja tarvitsee käytännössä aina Certificate Authorityn allekirjoittaman sertifikaatin, jonka avulla verkkosivun käyttäjä pystyy varmistumaan, että verkkosivun lähettämät salaustiedot todella ovat luotettavia ja kuuluvat kyseiselle verkkosivulle. Tähän ongelmaan ei ole vielä kehitetty muuta teknistä ratkaisua TLS-salauksen parissa.

Vuonna 2008 julkaistiin järjestelmän kuvauspaperi virtuaalisesta valuutasta nimeltään Bitcoin [1], jonka suurin keksintö ei suinkaan ollut virtuaalivaluutta vaan sen käyttämä teknologia, lohkoketjut. Lohkoketjuilla tarkoitetaan jaettua tietokantaa, jossa kaikki tieto on julkista ja jonne kerran tiedot laitettua sitä ei enää pysty muokkaamaan tai poistamaan muuta kuin kirjoittamalla ketju uudestaan halutusta kohdasta. Tämä prosessi on suunniteltu erilaisissa lohkoketjusovelluksissa todella vaikeaksi toteuttaa käytännössä, etenkin kun lohkoketjussa on useita ylläpitäjiä. Lisäksi useissa julkisissa lohkoketjuissa, kuten Bitcoin ja Ethereum [2], lohkoketjun ylläpito on suunniteltu siten, että siihen pystyy liittymään kuka tahansa verkon ylläpitäjäksi, jotka yrittävät ratkaista vaikeita laskutoimituksia ja oikean ratkaisun löytänyt palkitaan. Näillä palkkioilla saadaan motivoitua ylläpitäjiä pitämään verkkoa yllä.

Vaikka Bitcoin oli uranuurtaja lohkoketjujen saralla, tähän työhön valittiin lohkoketjuksi Ethereum, joka on valuutan sijasta erikoistunut dappien, eli desentralisoitujen applikaatioiden rungoksi. Desentralisoinnilla tarkoitetaan tässä tapauksessa sitä, ettei millään yksittäisellä taholla ole valtaa hallita lohkoketjussa olevaa koodia ja dataa, vaan sitä hallitsee yhdessä kaikki toisistaan riippumattomat ylläpitäjät.

Ethereumin avulla ohjelmoijan ei tarvitse itse toteuttaa ja ylläpitää lohkoketjua, vaan Ethereum tukee ns. smart contracteja, eli sovelluksia lohkoketjussa, jonne sovelluskehit-

täjät voivat julkaista omaa Turing täydellistä -koodia [3]. Se tarkoittaa, että sillä voidaan luoda mikä tahansa algoritmi, jos ei muistirajoitteita kuitenkaan oteta huomioon. Käytännössä se näkyy niin, että se tukee kaikkia ohjelmoinnin perusteita, kuten funktioita, if- ja for-lausekkeita, sekä muuttujia ja täten sillä pystyy toteuttamaan moninaisia ohjelmia.

Tavoitteena tässä työssä onkin toteuttaa Ethereumiin smart contract, johon verkkosivustojen omistajat voivat tallettaa oman julkisen salausavaimen ja verkkosivunsa. Tätä varten tarkoituksena on myös tehdä käyttöliittymä, josta käyttäjä voi kommunikoida smart contractin kanssa. Tästä smart contractista sitten selaimet ja muutkin käyttäjät voivat tarkistaa, kuuluuko kyseinen salausavain tälle sivulle.

2 SSL/TLS-salaus

2.1 Taustatietoa

TLS ja sen edeltäjä SSL on tarkoitettu verkkoliikenteen salaamiseen ja datan aitouden todentamiseen monenlaisissa verkkosovelluksissa, kuten verkkosivustoissa, faksissa ja sähköpostissa [4]. Tässä työssä keskitytään kuitenkin verkkosivustojen salaukseen.

Nykyään puhekielessä SSL-salauksesta puhuttaessa usein kuitenkin tarkoitetaan TLS-salausta. TLS-protokolla on pilkottu paloihin, niin kutsuttuun TLS Record -protokollaan [5]. Kaikki TLS:n yli lähetetyt paketit koostuvat tällaisista recordeista.

TLS recordeja on neljää erilaista: *change_cipher_spec*, *alert*, *handshake* ja *application_data.* Change_cipher_spec-recordia [6] käytetään ilmoittamaan vaihdosta salausstrategiassa, esim. salausavaimessa ja/tai algoritmissa. Alert-record [7] voidaan lähettää nimensä mukaisesti varoituksia ja erroreita. Tätä recordia käytetään myös yhteyden katkaisemiseen. Handshake-recordia [8] käytetään palvelimen ja käyttäjän tunnistautumiseen ja yhteisten salausalgoritmien ja -avainten sopimiseen. Application_data-recordia [9] käytetään sitten päätepisteiden välisessä varsinaisessa viestinnässä

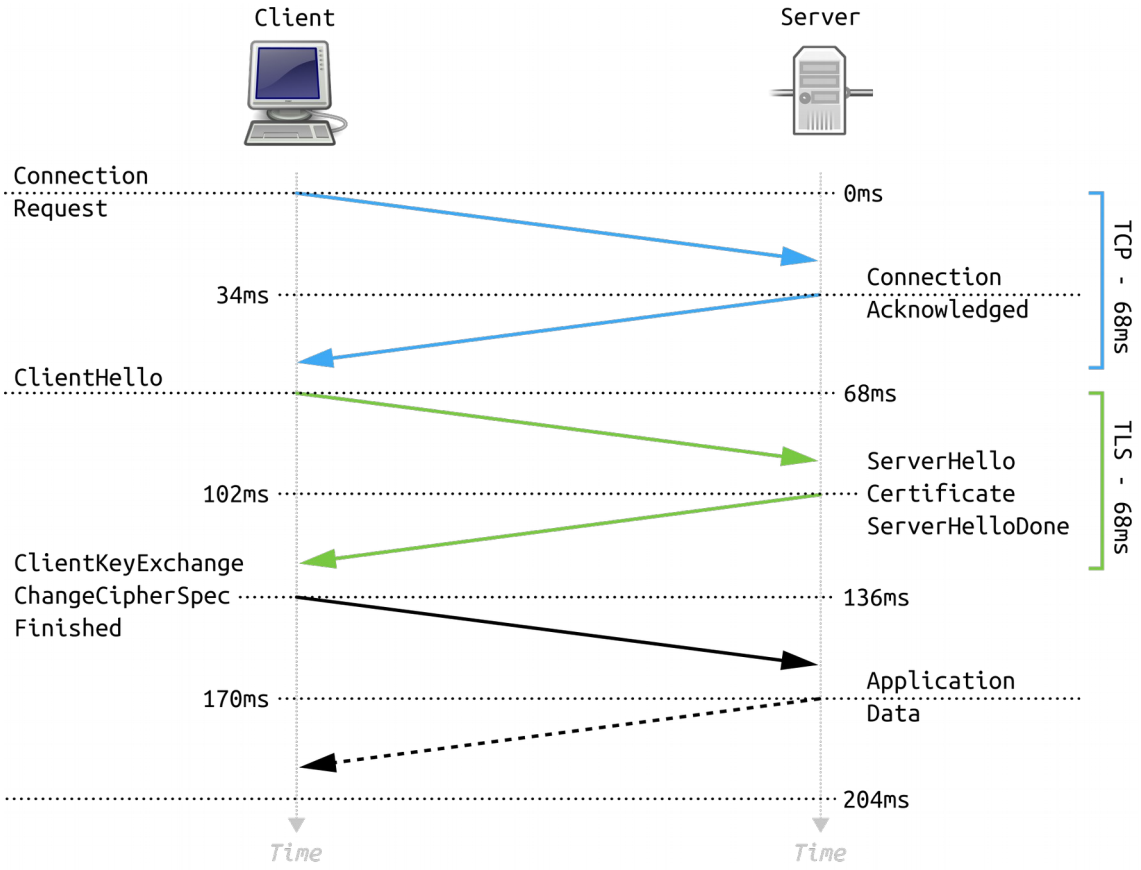
2.2 Man-In-The-Middle

MItM-hyökkäyksellä [10] tarkoitetaan hyökkäystä, jossa hyökkääjä asettuu kahden päätepisteen, esimerkiksi verkkosivun käyttäjän ja sen palvelimen väliin. Tällöin hyökkääjä pystyy kuuntelemaan yhteydessä liikkuvaa tietoja ja halutessaan myös muokkaamaan sitä. TLS-salaus on kehitetty salaamaan liikenne siten, että hyökkääjä ei pysty saamaan selvää, mitä tietoa käyttäjän ja palvelimen välissä liikkuu, eikä myöskään pysty huomaamattomasti muokkaamaan tätä tietoa. Usein MItM-hyökkäys tapahtuu langattoman verkkoyhteyden kautta.

2.3 TLS Handshake

Kiinnostavin vaihe TLS-protokollassa tämän työn kannalta on TLS Handshake. TLS Handshake -protokollassa palvelin ja käyttäjä tunnistautuvat toisilleen ja sopivat yhteisen salausavaimen ja molemmille sopivan salausalgoritmin. Tämän vaiheen aikana käyttäjä lähettää aluksi viestin palvelimelle, joka sisältää muun muassa listan käyttäjän tukemista SSL/TLS-salausalgoritmeista. Palvelin vastaa tälle parhaiten sopivalla algoritmilla ja lähettää samalla oman sertifikaattinsa, joka sisältää palvelimen julkisen avaimen. Käyttäjä verifioi, että sertifikaatti on aito. Tämä tarkoittaa selainten tapauksessa sitä, että sertifikaatin verkko- tai ip-osoite vastaa sivua, jolla käyttäjä asioi, eikä sertifikaatti ole vanhentunut. Lisäksi sertifikaatti on allekirjoitettu salaisella avaimella, jonka julkinen pari löytyy selaimen luotetuista allekirjoittajista, eli Certificate Authorityista.

Kun käyttäjä on verifioinut sertifikaatin, hän luo alustavan salausavaimen ja salaa sen palvelimen julkisella avaimella ja lähettää tämän viestin palvelimelle. Palvelin purkaa viestin omalla salaisella avaimellaan ja nyt molemmilla, palvelimella ja käyttäjällä on sama alustava salausavain käytössä, eikä MItM-hyökkäys toimi edes tässä vaiheessa, koska hyökkääjä ei pysty purkamaan sertifikaatin julkisella avaimella salattua käyttäjän lähettämää viestiä. Käyttäen käyttäjän luomaa alustavaa salausavainta, molemmat luovat varsinaisen salausavaimen, jota sitten käytetään käyttäjän ja palvelimen välisessä viestinnässä. Tämä kaikki tapahtuu ennen kuin mitään muuta dataa on lähetetty.



Kuva 1. TLS Handshake

Kuvassa 1 näkyy karkea kuvaus selaimen ja palvelimen välisestä TLS Handshakesta, joka on merkitty vihreillä nuolilla. Kuvassa näkyy lisäksi ylhäällä sinisellä TCP-yhteyden luonti ja alhaalla handshaken jälkeinen salattu kommunikointi selaimen ja palvelimen välillä.

2.4 Certificate Authorityt

Certificate Authorityt allekirjoittavat palvelinten sertifikaatteja niin, että käyttäjät voivat luottaa näihin sertifikaatteihin. Ne toimivat siten, että kaikista alimmalla tasolla on root CA, eli juuritason CA. Tämän CA:n julkinen avain on selainten tiedossa. Juuritason CA sitten luo yhden tai useamman välikäsi CA:n, joita kutsutaan intermediate CA:iksi. Nämä välikäsi-CA:t toimivat usein itsenäisesti juuritason CA:sta ja allekirjoittavat palvelinten sertifikaatteja. Tarkoitus välikäsissä on se, että juuritason CA:n salainen avain voidaan pitää turvassa pois verkosta, ja jos jokin välikäsi CA:ista vaikka hakkeroidaan,

niin tämä juuritason CA voi kumota välikäsi CA:n sertifikaatin. Näin selaimet eivät enää luota sen allekirjoittamiin sertifikaatteihin.

Ongelmana on, että mitä jos Certificate Authorityt allekirjoittavatkin sertifikaatteja henkilöille, jotka eivät omista sertifikaatin verkkotunnusta? Esimerkiksi tapaus jossa kiinalainen CA allekirjoitti hyökkääjälle sertifikaatin github.com-verkkosivulle, koska hyökkääjällä oli pääsy githubin hänelle myöntämälle subdomainille, esim. *<käyttäjätunnus>.github.com* [11]. Sitten on myös tapauksia, joissa hakkerit onnistuvat allekirjoittamaan itselleen sertifikaatteja sivuille, joita he eivät omista. Tästä hyvinä esimerkkeinä ovat hyökkäykset DigiNotaria [12] tai Comodoa [13] vastaan, jossa hyökkääjät onnistuivat luomaan itselleen valideja sertifikaatteja muun muassa Googlelle, Yahoolle ja Mozillalle ja näiden avulla hyökkääjät esittivät oikeita kohteita ja pystyivät keräämään verkkosivuvierailijoiden tietoja, jotka pitäisi olla salattuja, kuten kirjautumistunnuksia ja gmailin sähköposteja. Pahinta tässä on, ettei verkkosivuston omistajalla tai käyttäjällä ole paljoakaan tehtävissä estääkseen näitä hyökkäyksiä, koska kaikki valta on Certificate Authorityillä.

Tähän ongelmaan on kuitenkin yritetty kehittää erinäisiä ratkaisuja, jotka pienentäisivät Certificate Authorityjen väärinkäytön riskejä. Yksi näistä on *name constraints [14],* jolla annetaan CA:lle oikeudet allekirjoittaa sertifikaatteja vain tietyille verkko-osoitteille ja/tai kieltää allekirjoittamasta niitä joillekin verkko-osoitteille. Voidaan kieltää esimerkiksi CA allekirjoittamasta example.com-sivustolle sertifikaattia.

Toinen ratkaisu, jota verkkosivustojen omistajat voivat käyttää, on *public key pinning* [15], jossa ensimmäistä kertaa, kun käyttäjä ottaa verkkosivustoon yhteyttä, verkkosivusto lähettää käyttäjälle listan hyväksytyistä julkisista avaimista ja niiden voimassaoloajoista, jotka käyttäjän selain sitten tallentaa muistiin. Myöhemmin kun käyttäjä ottaa uudelleen yhteyttä selain tarkistaa, löytyykö verkkosivuston sertifikaatin julkinen avain tallennetuista julkisista avaimista. Eli se auttaa estämään MItM-hyökkäyksiä vasta sen jälkeen, kun käyttäjä on ensimmäisen kerran ottanut yhteyttä oikeaan verkkosivustoon ja saanut tietoonsa hyväksytyt julkiset avaimet. Pahimmassa tapauksessa, jos MItM-hyökkäys suoritetaan ensimmäinen yhteydenoton aikana, käyttäjälle annetaan vain hyökkääjän julkiset avaimet, eikä käyttäjä tämän jälkeen enää luota oikeisiin sivuston antamiin avaimiin.

Vielä yksi ratkaisuista on *certificate transparency* [16], jonka tarkoituksena on, että CA:t kirjoittaisivat kaikki allekirjoittamansa sertifikaatit julkiseen tietokantaan, eivätkä käyttäjät luottaisi sertifikaatteihin, joita ei kyseisestä tietokannasta löydy. Tämä ei kuitenkaan estäisi sertifikaatteja samalle sivustolle esiintymästä useaan kertaan, mutta antaisi mahdollisuuden verkkosivustojen omistajille tarkkailla, jos jossain on myönnetty sertifikaatti heidän sivulleen.

Näitä ratkaisuja, joilla pyritään minimoimaan Certificate Authorityjen riskejä, on vielä monia muitakin, mutta kaikissa on sama ongelma; yksikään ei ole täysin vedenpitävä. Ongelmia tulee muun muassa siinä, että kaikki eivät pidä kiinni samoista käytännöistä, esimerkiksi jokin selain ei välttämättä välitä verkkosivuston public key pinningistä tai tarkista, onko sertifikaatti listattu julkiseen tietokantaan certificate transparencyn mukaan. Ongelma on kuitenkin aina se, että tämä systeemi vaatii toimiakseen luottamista vähintään yhteen kolmanteen osapuoleen, joka voi halutessaan huijata tai sitä voidaan hyväksikäyttää. Tämä on se ongelma, johon tässä työssä pyritään löytämään ratkaisu.

2.5 Verkko-osoitteen validointi

Verkko-osoitteen validointiin on useita tapoja, ja se vaihtelee CA:sta riippuen, mutta usein se tehdään verkko-osoitteen DNS-tietojen, eli Domain Name Systemin avulla, joka muuntaa verkkotunnukset IP-osoitteiksi [17]. Esimerkiksi DigiCert [18] katsoo verkko-osoitteen DNS:n *whois*-tiedoista verkko-osoitteen haltijan sähköpostin ja lähettää sinne ohjeet verkko-osoitteen validointia varten.

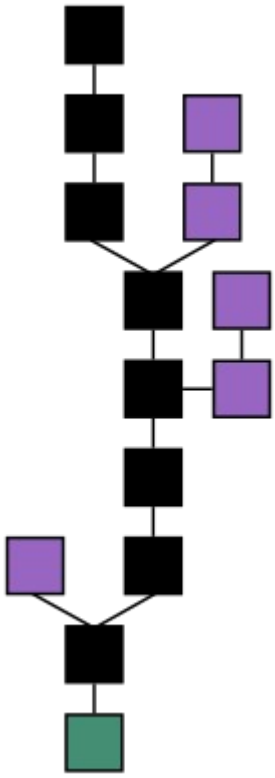
Toinen, tämän työn kannalta kiinnostavampi tapa validoida, on pyytää verkkosivuston omistajaa laittamaan sivullensa tiettyyn osoitteeseen pyydetyn haasteen. Esimerkiksi, jos käyttäjä omistaa verkko-osoitteen example.com, pyydetään häntä laittamaan example.com/challenge osoitteeseen viesti ”proof”. Tämän jälkeen, kun käyttäjä ilmoittaa suorittaneensa vaaditun haasteen, CA tarkastaa, löytyykö pyydetty haaste todella sivulta. Jos se löytyy, se tarkoittaa, että käyttäjällä on oikeudet muokata sivun sisältöä ja täten verkkosivu on validoitu.

3 Lohkoketjut

3.1 Taustatietoa

Bitcoinin luojan kehittämät lohkoketjut [20] ovat julkisia jaettuja tietokantoja, jonka voidaan ajatella koostuvan nimensä mukaisesti ketjusta lohkoja. Lohkoketjun ominaisuuksiin kuuluu, ettei sinne kerran jaettua dataa enää voi muokata tai sen muokkaaminen on todella hankalaa, sillä se vaatii enemmistön verkossa, ja koko ketju pitää kirjoittaa uusiksi halutusta pisteestä muokaten haluttuja kohtia. Lisäksi kaikki tieto ketjussa on julkista.

Lohkoketju on siis jaettu tietokanta, joka vaatii useita toisistaan riippumattomia ylläpitäjiä, jotta lohkoketjun hyödyt saadaan maksimoitua. Tämän esim. Bitcoin ja monet muutkin ovat tehneet siten, että käyttäjät saavat palkinnoksi lohkoketjun ylläpitämisestä itselleen jonkin palkinnon, Bitcoinin tapauksessa Bitcoineja. Lohkoketjuissa on lisäksi myös se hieno puoli, että ne ovat käytännössä 100 % ajasta toiminnassa eikä yksittäiset kaatuneet verkon ylläpitäjien järjestelmät, eli nodet vaikuta sen toimintaan.



Kuva 2. Lohkoketju

Kuvassa 2 näkyy, miten lohkoketjut yleensä rakentuvat. Alhaalla vihreällä on ketjun ensimmäinen lohko, niin sanottu *genesis*-lohko. Musta lohko tämän yläpuolella on lohkoketjun toinen lohko, joka - kuten kaikki genesiksen jälkeiset lohkot - sisältää kryptograafisen tiivisteen (hash) edellisestä lohkosta (genesis tässä tapauksessa), aikaleiman ja lohkon varsinaisen datan. Varsinainen data on Bitcoinin tapauksessa Bitcoin-siirtoja Bitcoin-osoitteesta toiseen.

Kolmanneksi alimpana on kaksi lohkoa vierekkäin, purppura ja musta. Purppura kuvaa ns. *orphan*-lohkoa, eli lohko, joka täytti kriteerit ollakseen lohkoketjun seuraava lohko, mutta enemmistö ei sitä kuitenkaan hyväksynyt, vaan he hyväksyivät oikealla olevan mustan lohkon. Orphan-lohkojen luojat saavat myös palkkion, mutta se palkkio on tietenkin käytössä vain lohkoissa, jotka tulevat tämän orphan-lohkon jälkeen. Toisin sanoen palkkio on arvoton, koska enemmistö käyttäjistä on muualla. Tämän takia nodet siirtyvät aina ketjuun, jossa on enemmistö, koska siellä saaduilla palkkioilla, eli Bitcoineilla on joitain arvoa.

Orphan-lohkot johtuvat yleensä siitä, että joku toinen verkon ylläpitäjistä on ratkaissut lohkon ensin. Tässä tapauksessa musta lohko on ratkaistu ensin, ja täten se on levinnyt nopeammin ympäri järjestelmän ja saanut hyväksynnän enemmistöltä. Lohkoille on siis olemassa monta validia ratkaisua. Yksi syy tähän on, että ylläpitäjät saavat valita, mitä Bitcoin-siirtoja lohkoon sisällytetään. Toinen syy selitetään seuraavassa luvussa, Proof of Work.

Ylempänä tapahtuu vielä kaksi kertaa samanlainen orphan-lohko. Tosin näissä tapauksissa osa nodeista on vielä lähtenyt jatkamaan tämän orphan-lohkon parissa, mutta kun nodet huomaavat, että pääketju eli musta ketju alkaa mennä heidän ketjustaan ohi, nodet vaihtavat siihen ketjuun, koska vaikka he saavatkin samat palkkiot omassa ketjussaan, niillä ei ole siellä mitään arvoa, koska enemmistö on muualla.

Lohkoketjuissa uusi lohko viittaa aina edelliseen lohkoon, edellisestä lohkosta tehdyn kryptograafisen tiivisteen (hash) [21] avulla. Näin on käytännössä mahdotonta muuttaa lohkon sisältöä jälkikäteen, koska muuten seuraavan lohkon viittaus edelliseen lohkoon on pielessä. Tällä taataan lohkoketjujen muuttumattomuus. Eli jos haluaa muuttaa jotain lohkoa, pitää kaikki sen jälkeen tehdyt lohkot laskea uudelleen. Näin saadaan tehtyä verkon lohkojen mielivaltainen muuttaminen käytännössä mahdottomaksi, koska se vaatisi että hyökkääjällä on enemmän laskentatehoa kuin rehellisillä ylläpitäjillä. Samal-

la tämä myös aiheuttaa sen, että mitä syvemmällä ketjussa lohko on, sitä vaikeampaa sen muuttaminen on. Tämän vuoksi usein esimerkiksi kauppaa käydessä, ennen kuin kauppatavara vaihtaa omistajaa, odotetaan muutaman lohkon ajan Bitcoin-siirtoja tehdessä, että voidaan varmistua, ettei kukaan enää pysty muuttamaan tehtyä siirtoa tekemättömäksi.

Lohkoihin viittaamiseen käytetyt tiivisteet ovat algoritmeja, joiden tarkoituksena on pakata niille annettu mielivaltaisen suuruinen syöte – tässä tapauksessa lohkon metadata (aikaleima yms.) ja Bitcoin-siirrot – ennalta määrättyyn kokoon. Sen tärkeimpiä ominaisuuksia ovat, että samasta syötteestä seuraa aina samanlainen lopputulos, ja se, että on käytännössä lähes mahdotonta löytää kaksi erilaista syötettä, jotka tuottaisivat saman lopputuloksen. Tällä saadaan taattua se, että vaikkakin teoriassa on mahdollista, niin käytännössä on kuitenkin lähes mahdotonta luoda kaksi lohkoa eri Bitcoin-siirroilla, mutta samalla tiivisteellä.

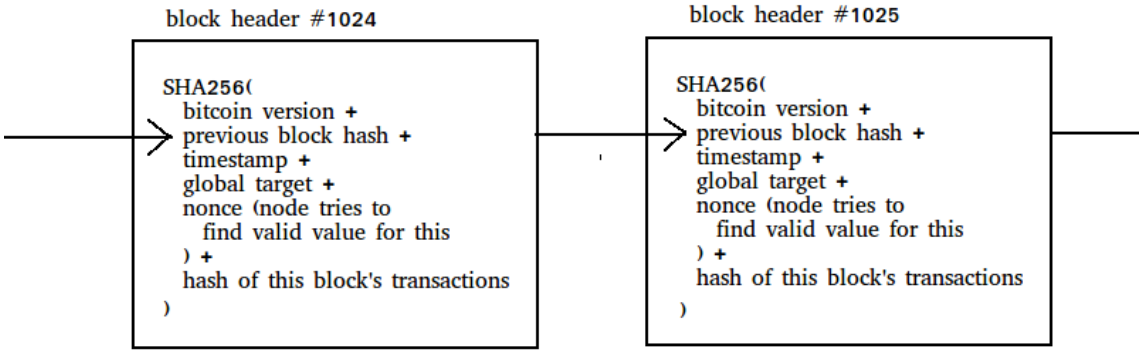
3.2 Osoitteet lohkoketjussa

Lohkoketjuissa kaikki toimii osoitteiden avulla. Bitcoinissa kaikilla käyttäjillä on oma julkinen ja salainen avain. Bitcoinissa käyttäjän tilin osoite on käyttäjän julkisesta avaimesta tehty tiiviste. Se näkyy Bitcoin-siirroissa ja se on se, mihin käyttäjät voivat lähettää Bitcoineja. Salaista avainta tarvitaan Bitcoin-siirtojen allekirjoitukseen. Näin kukaan ei voi väärentää Bitcoin-siirtoja, koska siirto ei ole validi, jos se on allekirjoitettu väärällä salaisella avaimella. Kuka tahansa, joka tietää salaisen avaimen, pystyy tekemään valideja transaktioita. Jos salaisen avaimen hukkaa, ei Bitcoineihin pääse enää mitenkään käsiksi.

Tämänkin vuoksi hyökkäys Bitcoin-verkkoa vastaan olisi harvoin kannattavaa, koska vaikka hyökkäys onnistuisi, ei hyökkääjä silti voisi siirtää Bitcoineja tileiltä, joiden salaista avainta hän ei tiedä. Pahinta mitä hyökkääjä voisi tehdä, on käytännössä omien siirtojen muokkaus, kuten poistaa oman siirron toiselle henkilölle, jolloin hyökkääjä ei menetäkään käyttämiään Bitcoineja. Mahdollinen hyöty tästä voisi olla, että hyökkääjä ostaisi jonkun kalliin esineen ja jälkikäteen poistaisi transaktion, jolloin hänellä olisi sekä esine että Bitcoinit.

3.3 Proof of Work

Proof of Work (PoW) [22][23] on Bitcoinin ja monien muidenkin lohkoketjujen tapa varmistaa, että lohkoketjuun tehdään vain sallittuja muutoksia, eikä esimerkiksi yritetä käyttää samoja bitcoineja useaan kertaan. Se toimii siten, että saadakseen uuden lohkon lohkoketjuun, noden, eli siis yhden järjestelmän ylläpitäjistä tulee ratkaista vaikea matemaattinen laskutoimitus. Bitcoinin tapauksessa tämä tapahtuu niin, että node laskee SHA-256-algoritmilla tiivisteen, johon tulee edellisen lohkon tiiviste, noden valitsemat validit transaktiot, eli käyttäjien Bitcoin-siirrot tililtä toiselle, aikaleima, globaali kohdetulos, Bitcoinin versio ja niin kutsuttu nonce, joka on numero, jota muuttamalla node yrittää saada lopputuloksen, joka on alle nykyisen globaalin kaikille nodeille saman kohdetuloksen. Koska SHA-256 on käytännössä sattumanvarainen algoritmi, on siis täysin sattumaa, kuka löytää ensimmäisenä oikean ratkaisun, mutta kuitenkin keskimäärin ne, kenellä on eniten laskentatehoa, löytävät eniten ratkaisuja ja täten luovat pisimmän ja virallisen lohkoketjun. Kuvassa 3 on havainnollistaminen tästä prosessista, jossa kuvataan lohkojen tunnistetiivisteiden luomista.



Kuva 3. Proof of Work

Kun node on löytänyt ratkaisun, se toimittaa ratkaisunsa muille nodeille, jotka varmistavat jokainen itsenäisesti, että lähetetty ratkaisu on oikein, eikä se sisällä epävalideja transaktioita, esimerkiksi ettei käyttäjä ole yrittänyt lähettää 10 Bitcoinia, vaikkei hänellä ole kuin 5. Kun ratkaisu on varmistettu, lisätään lohko ketjuun ja nodet alkavat työskennellä taas seuraavan lohkon parissa. Bitcoinin PoW on suunniteltu siten, että lohkoajat olisivat keskimäärin 10 minuuttia. Tämä saavutetaan 2 viikon välein tarkastelemalla edellisen kahden viikon keskimääräisiä lohkoaikoja ja laskemalla uusi globaali kohdetu-

los, jonka alle nodejen pitää tulos saada. Mitä pienempi kohdetulos, sitä vaikeampaa on oikea tulos löytää.

3.4 Proof of Stake

Proof of Stake (PoS) [24] on uudempi ratkaisu samaan ongelmaan kuin PoW, eli kuinka pitää kaikki nodet rehellisinä, etteivät ne yritä huijata verkkoa. Yksi suurimmista syistä, minkä takia PoS:ia on lähdetty kehittämään, on PoW:n turvallisuuden takaamiseksi vaadittu valtava laskentateho ja sitä kautta suuri sähkönkulutus. Bitcoinin on arvioitu kuluttavan vuodessa jopa noin 3 miljardin dollarin edestä sähköä [25]. Sen sijaan, että seuraavan lohkon luoja valittaisiin siten, kuka ratkaisee ensimmäisenä vaativan matemaattisen tehtävän, kuten PoW:ssa, PoS:ssa seuraavan lohkon luoja valitaan satunnaisesti nodejen panosta ja toteutustavasta riippuen mahdollisesti muita kriteereitä painottaen.

PoS:ssa nodet siis asettavat panokseksi valuuttaa, eli jos Bitcoin toimisi PoS:lla, niin nodet laittaisivat Bitcoinia panokseksi. Mitä enemmän panosta, sitä isompi todennäköisyys tulla valituksi seuraavan lohkon luojaksi. Usein tähän kuitenkin otetaan myös muita muuttujia, jotta varallisimmat eivät aina saisi luoda uutta lohkoa. Tällaisia muuttujia on muun muassa aika, jolloin node on viimeksi päässyt luomaan lohkon. Mitä kauemmin on mennyt aikaa, sitä isommalla todennäköisyydellä pääsee node luomaan uuden lohkon. Kun node sitten pääsee luomaan uuden lohkon, niin aika taas nollaantuu. Node saa päättää, mihin kohtaan ketjua se uuden lohkon luo.

PoS voidaan myös toteuttaa siten, että valitaan useita nodeja luomaan oma ehdotuksensa seuraavaksi lohkoksi. Tämäkin valinta tapahtuu satunnaisesti mahdollisesti joitain kriteerejä painottaen, kuten aiemmin on kuvattu. Kun nodet ovat luoneet omat ehdotuksena, järjestetään useita äänestyskierroksia, joihin kaikki ehdotuksen tehneet nodet saavat äänestää mielestään parasta lohkoa ja lopulta jäljelle jää yksi lohko, joka lisätään lohkoketjuun. Aiempiin järjestelmiin verrattuna tässä on myös se etu, että nodet pääsevät heti yhteisymmärrykseen, mikä on lohkoketjun pää, eikä orphan-lohkoja tule. Edelliseen PoS:iin verrattuna, missä valittiin vain yksi node lohkon luojaksi, tässä on etuna myös se, ettei tällä tavalla ketju ole enää riippuvainen yhdestä nodesta.

PoS:ssa asetettavan panoksen tarkoitus on se, että jos nodet äänestävät tai liittävät lohkonsa väärään ketjuun, eli ketjuun, jossa ei ole enemmistöä, nämä nodet menettävät toteutustavasta riippuen osan panoksestaan. Näin verkon huijaaminen saadaan tehtyä kannattamattomaksi.

Energiatehokkuuden lisäksi PoS:n on arvioitu nostavan lohkoketjun turvallisuutta tekemällä hyökkäykset yhä kalliimmiksi ja samalla kannattomattomammiksi. On myös arvioitu, että PoS lisäisi desentralisaatiota, koska kuka tahansa kenellä on rittävästi valuuttaa asettaakseen panoksen, pystyy liittymään nodeksi. PoS:ssa ei siis tarvita erillistä kalustoa, jossa on hyvä laskentateho, toisin kuin PoW:ssa. Käytännön vaikutuksista ei kuitenkaan vielä ole tarkkaa tietoa olemassa.

3.5 Vertailua tavalliseen tietokantaan

Milloin lohkoketjuja tulisi sitten käyttää tavallisten tietokantojen, kuten MongoDB ja MySQL sijasta? Tärkein ja käytännössä ainut syy on desentralisaation ja luottamuksen tarve. Desentralisaatio tässä tarkoittaa sitä, ettei kukaan yksittäinen taho pysty hallitsemaan tietokannassa olevaa dataa, ja koska lohkoketjuun laitettua dataa ei pysty kukaan enää muuttamaan, se tekee lohkoketjuista tällä hetkellä vertaansa vailla olevan alustan sovelluksille, joissa omistajuuden määrittäminen on tärkeää. Tällaisia ovat esimerkiksi virtuaalivaluutat tai tässä tapauksessa verkko-osoitteen omistajuus. Koska kaikki on julkista, niin kuka tahansa voi tarkistaa lohkoketjusta mitä haluaa, vaikkapa kuka omistaa verkko-osoitteen. Taasen perinteisissä tietokannoissa, jonne kuka tahansa, joka omistaa tietokantaan riittävät oikeudet, pystyy muokkaamaan tietokannan tietoja.

3.6 Ethereum

3.6.1 Taustatietoa

Ethereum [26], jonka idea laitettiin aluksi ilmoille vuonna 2013 ja julkaistiin vuonna 2015, on lohkoketju, joskin hyvin erilaisella filosofialla kuin Bitcoin. Siinä missä Bitcoinin tarkoitus on toimia valuuttana, Ethereumin tarkoitus on toimia alustana kehittäjille, minne kuka tahansa voi luoda oman lohkoketjusovelluksen, eli smart contractin. Ethereum, kuten myös Bitcoin, toimii PoW-periaatteella, mutta sen on tarkoitus lähiaikoina siirtyä

käyttämään PoS:ia [27]. Ethereumin keskimääräiset lohkoajat ovat noin 15 sekuntia, eli siirrot liikkuvat siellä monta kertaa nopeammin kuin Bitcoinissa.

Ethereumiin kehittäjät voivat kehittää smart contractien avulla esimerkiksi omia tokeneita, jotka toimivat valuutan tavoin, mutta niiden tarkoituksena on kuitenkin yleensä jokin muu kuin toimia pelkkänä valuuttana. Esimerkiksi Waltonchain (WTC) [28] on Ethereumiin rakennettu token, jonka tarkoituksena on yhdistää tuotteisiin kiinnitettävät RFID-sirut ja lohkoketjut, joiden avulla pystytään pienemmillä kuluilla seuraamaan tuotteen tuotantoprosessia ja logistiikkaa, eli esim. missä vaiheessa tuotantoprosessia tuote on tai missä hyllyllä tai kuljetusvaiheessa tuote tällä hetkellä sijaitsee. Samalla loppukäyttäjä, eli tuotteen ostaja pystyy varmentamaan RFID-sirun skannaamalla, että tuote on aito. Lohkoketjuja tässä hyödynnetään juurikin sen vuoksi, ettei kukaan pysty muokkaamaan RFID-sirun kautta lohkoketjuun tallennetuja tietoja jälkikäteen.

3.6.2 Smart contract

Smart contractit ovat siis Turing-täydellistä lohkoketjuun julkaistavaa koodia, jolla yleensä muokataan smart contractin tilaa, eli sinne tallennettuja muuttujia, jotka tallentuvat lohkoketjuun siten, että vain smart contract pystyy niitä muokkaamaan. Bitcoinissakin on tuki skriptaukselle, mutta se ei ole Turing-täydellinen, eli sen käyttömahdollisuudet ovat rajallisia, jonka vuoksi sitä ei tähän työhön valittu. Smart contractien ensimmäinen idea on julkaistu jo vuonna 1997 [29]. Tällöin smart contractien toimintaa Nick Szabo kuvaili seuraavanlaisesti tilanteessa, jossa auton omistaja ja velkoja tekevät sopimuksen auton omistajuudesta niin, että kun omistaja maksaa laskuja, auto pysyy vain omistajan hallussa, mutta jos omistajalta jää laskut maksamatta auton omistajuus siirtyy velkojalle, kunnes laskut on maksettu.

1. Lukko, jolla päästetään vain auton omistaja sopimukseen.

2. Takaovi, josta auton omistajuus voidaan siirtää velkojalle.

3a. Takaovi avataan vain jos laskuja jää maksamatta.

3b. Kun kaikki laskut on maksettu, takaovi suljetaan lopullisesti.

Tämä on siis smart contractien idea, eli luodaan sopimus, jossa on tietyt pelisäännöt, joista yksikään osapuoli ei enää hyväksyttyään sen voi perääntyä. Ethereumissa tämä

tapahtuu ohjelmallisesti, ns. ”code is law”, eli se mitä koodissa lukee, tapahtuu, eikä sitä voi kiertää. Eli jos auton omistajuus olisi edellisen esimerkin tapauksessa sähköisesti lohkoketjussa (vrt. kryptovaluutan omistajuus), kun molemmat osapuolet, auton omistaja ja velkoja ovat liittyneet sopimukseen, ei siitä enää kumpikaan voi poistua, jos koodi ei sitä salli. Sopimus sitten itsenäisesti tarkastelisi, onko omistaja maksanut laskuja vai ei ja tarpeen mukaan avaisi/sulkisi takaoven, jolla auton omistajuus siirretään velkojalle. Näin voidaan tehdä sitovia sopimuksia ilman kolmansia osapuolia.

Ethereumin smart contracteissa on kuitenkin pari melko rajoittavaa tekijää. Ensinnäkin kaikki smart contractissa suoritetut laskennalliset vaiheet, eli koodin eri osat maksavat tietyn summan etheriä, Ethereumin valuuttaa, riippuen siitä, kuinka raskasta vaihe on suorittaa. Tämä maksu on tehty estämään ikiluuppeja, joilla smart contractien tekijät voisivat tahallaan tai vahingossa tehdä palvelunestohyökkäyksen Ethereumin verkkoon saamalla nodet ikuiseen luuppiin. Lisäksi se estää käyttäjiä tekemästä todella raskaita smart contracteja, koska ne tulisivat kalliiksi sen suorittajille. Samalla tämä systeemi palkitsee nodet toiminnastaan, koska nämä maksut menevät aina seuraavan lohkon tekijälle. Toinen, ainakin tämän työn kannalta isompi rajoitus on se, että kaikkien smart contractien pitää olla täysin deterministisiä. Tämä tarkoittaa sitä, että smart contractin funktiota kutsuttaessa tämän hetkisen tilan ollessa x ja smart contractin syötteen ollessa y, seuraa aina sama lopputulos z, eli

$$f(x, y) = z$$

Tämä johtuu siitä, koska kaikkien eri nodejen pitää itsenäisesti tulla samaan lopputulokseen suorittaessaan smart contractin, ei siellä voi olla epädeterministisyyttä, muuten nodet eivät pääse sopuun, mikä on validi lohko ja mikä ei. Tämä deterministisyys taas johtaa siihen, ettei smart contracteissa voi käyttää mitään satunnaisuutta, joka tarkoittaa muun muassa sitä, ettei smart contract voi suoraan lähettää HTTP-kutsuja verkkosivuille tai hakea mitään muutakaan dataa lohkoketjun ulkopuolelta, koska ei ole takuita, että sieltä tulisi aina sama vastaus. Tähän on kuitenkin onneksi olemassa ratkaisu: oraakkelit.

3.6.3 Oraakkelit

Oraakkeleiden avulla smart contractit voivat siis kommunikoida ulkopuolisten palvelujen kuten verkkosivujen kanssa. Oraakkelit ovat kolmansien osapuolien ylläpitämiä pal-

veluita, jonka käytöstä ne yleensä perivät maksua. Oraakkelit toimivat siten, että ne luovat oman smart contractit lohkoketjuun, johon käyttäjät, jotka haluavat vaikkapa hakea joltain verkkosivulta tietoa, lähettävät pyynnön ”voisitko puolestani hakea tiedot verkkosivulta X”. Oraakkelin smart contract sitten lähettää tiedon omille palvelimilleen, jotka ovat täysin irralliset lohkoketjusta, eikä millään tavalla desentralisoituja, että täältä pitäisi hakea tietoa. Smart contracteissa voi siis ainakin Ethereumissa lähettää ns. eventtejä, jotka näkyvät kaikille, jotka kuuntelevat kyseistä eventtiä. Tähän eventtiin voi myös laittaa tietoja, kuten tässä tapauksessa verkko-osoite, josta tietoa halutaan. Tällaisen eventin oraakkelit siis nappaavat kiinni.

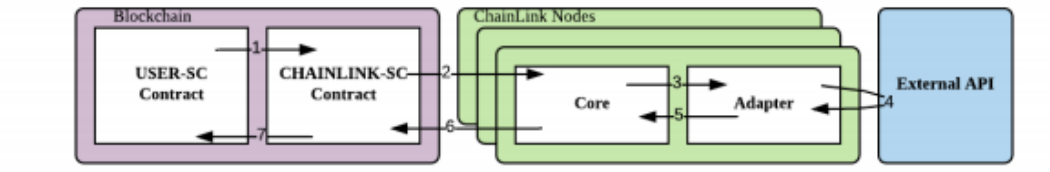
Kun palvelin on saanut vastauksen verkkosivulta, se kutsuu ennalta määritettyä funktiota käyttäjän smart contractista, johon se antaa tiedot kutsusta. Käyttäjän smart contractin osoite on annettu palvelimelle eventin yhteydessä, jossa annettiin myös verkko-osoite. Näin saadaan tietoa lohkoketjun ulkopuolelta, mutta tässä on yksi iso ongelma. Oraakkelit rikkovat lohkoketjusta saadun desentralisaation ja tekevät yhden haavoittuvaisen pisteen järjestelmään, jonka käyttäjän pitää vain luottaa toimivan oikein. Oraakkelihan voi esimerkiksi antaa ihan mitä vain tietoa se haluaa, minkä se olisi muka saanut käyttäjän haluamalta verkkosivulta. Tämä on iso ongelma, koska mitä hyötyä on desentralisoidusta applikaatiosta, jos siinä kuitenkin on yksi sentralisoitu heikko lenkki, joka voi halutessaan tai vahingossa tehdä tuhojaan. Oraakkelihan on saattanut esimerkiksi luoda itse sertifikaatin esim. google.com:lle ja antaa luomansa sertifikaatin julkisen avaimen ja verkko-osoitteen tässä työssä luotavalle smart contractille. Kun smart contract pyytää oraakkelia katsomaan, löytyykö Googlelta validointitiedosto, se vain palauttaa itseluomansa validointitiedoston suoraan kysymättä edes Googlelta ja näin saisi haltuunsa validin sertifikaatin Googlelle.

3.6.4 Oraakkeli ongelma

Tähän oraakkelien aiheuttamaan ongelmaan on kuitenkin yritetty kehittää ratkaisuita. Yksi pätevimmistä ratkaisuista on desentralisoitu oraakkeli, eli sen sijaan, että smart contract kysyisi vain yhdeltä oraakkelilta vastausta, se kysyykin useilta toisistaan riippumattomilta oraakkeleilta, että voisitko hakea tämän tiedon puolestani. Sitten riippuen kuinka varmoja halutaan vastauksesta olla, vaaditaan, että esimerkiksi yli 90 prosenttia oraakkeleista on samaa mieltä vastauksesta. Jos vaaditaan, että kaikki oraakkelit ovat samaa mieltä, se mahdollistaa helposti palvelunestohyökkäyksen smart contractia vastaan siten, että yksi oraakkeli antaa aina väärää vastausta. Tämäkin voidaan ratkaista

rahallisesti rankaisemalla oraakkeleita, jotka toimivat väärin, samaan tapaan kuin PoS-systeemissä ja täten tehdä huijausyrityksistä kalliita.

Desentralisoituihin oraakkeleihin on erikoistunut ainakin ChainLink [30], joka tarjoaa järjestelmän johon kuka tahansa voi liittyä oraakkeliksi ja saa tästä hyvästä palkkiota ja väärin toimimisesta sakotetaan. Lisäksi ChainLinkissä on järjestelmä, joka arvostelee oraakkeleiden laatua ja smart contract voi valita, kuinka laadukkaita ja kuinka paljon eri oraakkeleita se haluaa käyttää. Sitten smart contract tekee tarjouksen, johon ChainLinkissä toimivat oraakkelit voivat halutessaan tarttua, jos he täyttävät smart contractin asettamat kriteerit oraakkelille. Käyttäjä voi tässä vaiheessa muun muassa määrittää, että jos oraakkelin vastaus eroaa enemmistöstä liian paljon tai ollenkaan, se joutuu maksamaan X määrän sakkoa. Nämä tapahtumat on merkitty vaiheina 1 ja 2 kuvassa 4.



Kuva 4: ChainLinkin toimintaperiaate

Vaiheissa 3, 4 ja 5 oraakkelit sitten suorittavat käyttäjän smart contractin haluaman pyynnön. Core ja Adapter ovat ChainLinkin oraakkeliohjelman sisäisiä termejä. Vastauksen haettuaan oraakkelit palauttavat vastauksensa ChainLinkin omalle smart contractille tai käyttäjän halutessaan hänen omalle smart contractilleen. Tässä smart contractissa vastaukset yhdistetään yhdeksi käyttäjän haluamalla tavalla ja esimerkiksi vastaus voidaan merkitä epäluotettavaksi, jos yksikin oraakkelin vastaus on erilainen tai jos alle 80 % oraakkeleista on samaa mieltä vastauksesta. Vastauksista voidaan halutessaan laskea myös vaikka keskiarvo. Riippuu kuinka vahvaa turvaa ja millaista dataa käyttäjä hakee. Tämä on vaihe 6 kuvassa 4. Sen jälkeen vielä viimeisessä vaiheessa yhdistetty vastaus lähetetään alkuperäiseen käyttäjän smart contractiin.

ChainLink arvostelee oraakkeleiden hyvyyttä niiden käytettävyysajan perusteella, eli kuinka usein oraakkeli on onnistunut vastaamaan smart contractin pyyntöön. Lisäksi ChainLink arvostelee vastausten hyvyyttä vertailemalla oraakkelin vastausten poikkeamaa verrattuna muiden oraakkelien vastauksiin.

ChainLinkillä on myös suunnitelmissa tulevaisuudessa hyödyntää tietotekniikkalaitteistojen rautapohjaisten ratkaisujen tarjoamia turvallisuus ja datan aitouden varmentamismetodeja. Yksi tällainen on Intelin SGX [31], joka sallii ohjelmiston suorituksen ympäristössä, jossa ohjelman suoritukseen ei pysty mitenkään vaikuttamaan ohjelman ulkopuolelta, edes vaikka olisi kaikki käyttöoikeudet laitteeseen, kuten root-käyttäjällä Linux-järjestelmissä. Lisäksi muut ohjelmat eivät pääse mitenkään käsiksi SGX-ympäristössä pyörivän ohjelman dataan. SGX pystyy myös tarjoamaan vakuuden, että koodi on suoritettu SGX-ympäristössä. Tämä tarkoittaisi sitä, että ChainLinkin oraakkelit voisivat toimia täysin niin, ettei yksikään ihminen tai ohjelma pääse todistetusti sen toimintaan vaikuttamaan. SGX:ää on myös ajateltu mahdolliseksi sentralisoiduksi korvaajaksi lohkoketjuille, mutta myös käyttöön yhdessä niiden kanssa lohkoketjujen tarjoaman järjestelmävirheiden sietokyvyn, lisäturvallisuuden ja datan julkisuuden vuoksi. On kuitenkin mahdollista, että SGX sisältää haavoittuvaisuuksia, jotka mahdollistaisivat näiden turvatoimien kiertämisen, kuten tässä tapauksessa, jossa sieltä onnistuttiin hakemaan salausavaimia [32]. SGX:n tarkempi toiminta on kuitenkin tämän työn aihealueen ulkopuolella.

3.6.5 Dapp

Desentralisoitu applikaatio, eli lyhyesti dapp, tarkoittaa ohjelmistoa, joka ainakin osittain toimii desentralisoidusti. Tämän työn tapauksessa applikaation desentralisoitu osa on Ethereumin lohkoketjussa sijaitseva smart contract. Desentralisoiduiksi applikaatioksi voidaan luokitella myös peer-to-peer (p2p) -ohjelmistot, jotka toimivat useiden päätepisteiden avulla. Esimerkki tällaisesta ohjelmistosta voisi olla jokin torrenteihin erikoistunut ohjelmisto kuten µTorrent.

4 Ethereum ja TLS

4.1 Tavoite

Tavoitteena tässä työssä on siis tutkia teknistä toteutusta, jossa yhdistetään lohkoketjut ja TLS siten, että Certificate Authorityja ei enää tarvittaisi. Ensin pitäisi saada verkko-osoitteen omistajuus verifioitua. Se onnistuisi siten, että verkko-osoitteen haltija loisi aluksi verkkosivulleen itseallekirjoitetun sertifikaatin. Tämän jälkeen hän loisi tiedoston,

jonka nimeksi tulisi hänen Etherum-tilin osoite ja laittaisi sen ainoaksi sisällöksi ”proof”. Tämän juuri luodun tiedoston haltija allekirjoittaisi äsken luomallaan sertifikaatilla ja laittaisi tämän tiedoston jakoon omalle verkkosivustolleen osoitteeseen example.com/ethtls.

Tämän jälkeen haltija lähettäisi Ethereum-tililtä, jonka laittoi tiedoston nimeksi, smart contractille pyynnön verifioida tämä verkko-osoite (example.com) ja antaisi samalla itseluomansa sertifikaatin julkisen avaimen. Smart contract sitten katsoisi, miltä tililtä pyyntö tulee ja minkä verkko-osoitteen käyttäjä haluaa verifioida, ja pyytäisi oraakkelia lähettämään kutsun kyseiseen osoitteeseen.

Kun smart contract saa oraakkelilta vastauksen, se tarkistaisi käyttäjän antaman julkisen avaimen avulla, että vastaus on allekirjoitettu tätä avainta vastaavalla salaisella avaimella. Jos validointi epäonnistuu, se ilmoittaisi verkko-osoitteen haltijalle, että validointi epäonnistui. Muussa tapauksessa smart contract lisäisi verkko-osoitteen ja sen julkisen avaimen sekä sen voimassaoloajan vaikka puolen vuoden päähän smart contractin tietokantaan.

Nyt kun smart contract olisi onnistunut verifioimaan verkko-osoitteen ja lisäämään sen julkisen avaimen tietokantaansa, pitäisi vielä saada selaimet tukemaan näitä sertifikaatteja. Se onnistuisi siten, että sertifikaatin luonnin yhteydessä sertifikaatin extensions [33] -kenttään lisätään uusi extension, joka tarkoittaisi, että sertifikaatin aitoustodistus löytyy tästä smart contractista.

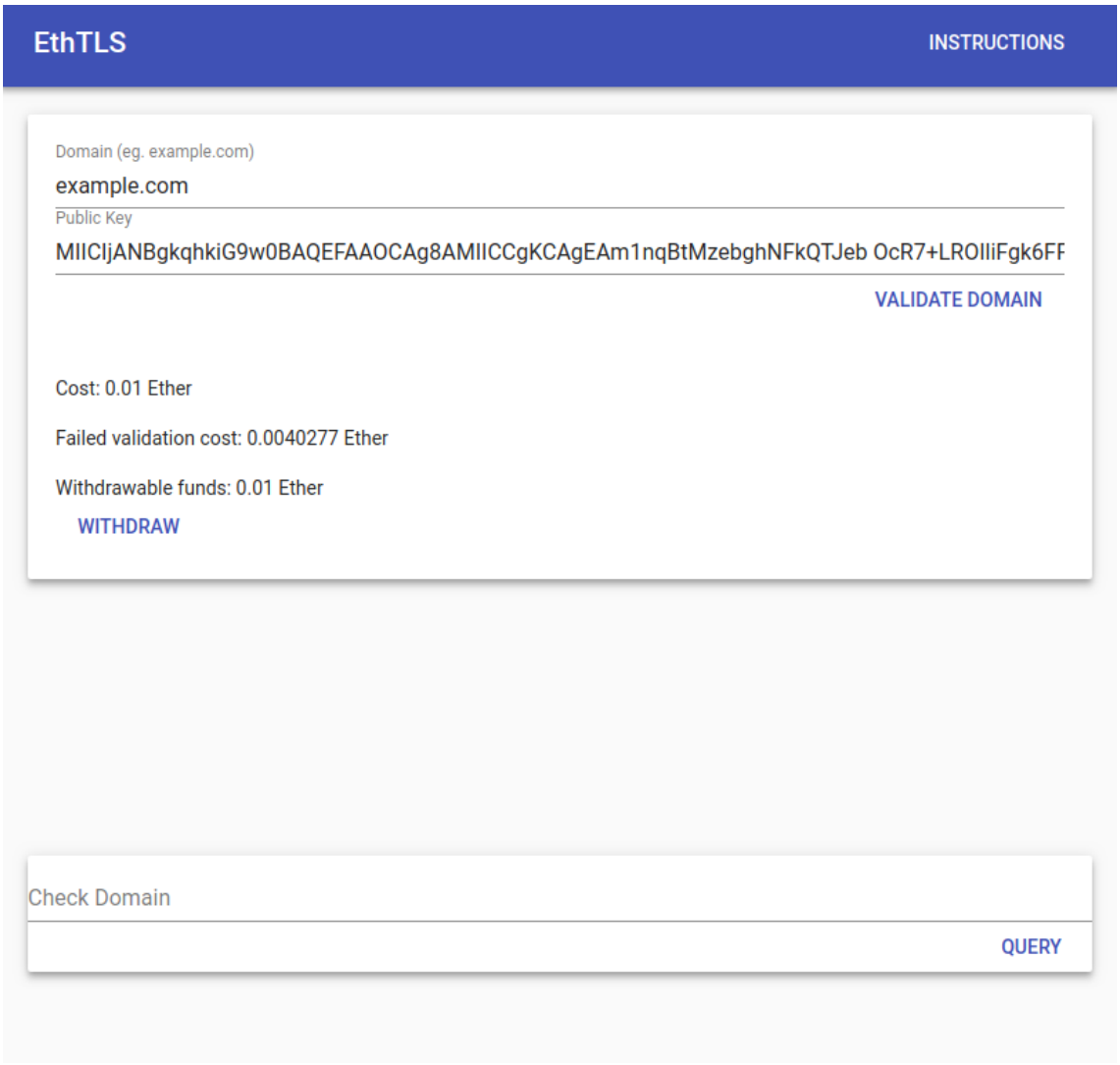
Sitten TLS Handshaken aikana, kun selain tarkistaa sertifikaatin aitouden, se katsoisi, että sertifikaatissa on kyseinen extension ja tietäisi lähettää kutsun Ethereumiin tehdylle smart contractille. Sieltä selain kysyisi kyseisen verkko-osoitteen tiedot, johon smart contract vastaisi sille annetulla julkisella avaimella ja sen voimassaoloajalla, jos ne löytyvät. Selain sen jälkeen tarkistaa, vastaako tämä lohkoketjusta saatu julkinen avain sertifikaatin julkista avainta. Jos avaimet vastaavat toisiaan ja sertifikaatti ei ole vanhentunut, sertifikaatti hyväksytään validiksi ja tallennetaan selaimen muistiin, jottei sitä tarvitse jokaisella käynnillä hakea uudestaan.

4.2 Totetutus

4.2.1 Desentralisoitu applikaatio

Desentralisoidun applikaation, eli dappin kehityksessä käytettiin Ethereumin smart contractissa kielenä Ethereumin kehittämää Solidityä. Dappin käyttöliittymä toteutettiin Reactilla. Lisäksi kehitysympäristössä oli mukana Truffle-viitekehys [34], jonka avulla saatiin lokaali Ethereum-testiverkko, johon smart contractit julkaistiin ja niiden kanssa kommunikoitiin. Tätä varten Truffle myös käänsi smart contractit Ethereumin virtuaalikoneen kielelle. Tässä Trufflen testiverkossa käyttäjä saa loputtomasti etheriä käyttöönsä. Eikä siellä ole lohkoaikoja ollenkaan. Sitä on siis täysin ilmainen käyttää, ja se helpottaa kehittäjän työtä paljon, joten jokainen Ethereum-kehittäjä tarvitsee sellaisen.

Smart contract käytti oraakkelinaan Oraclizea [35] sen helppouden takia. Oraclize perii pienen maksun n. 0.01 $ jokaisesta HTTP-hausta. Tämän maksun maksaa loppukäyttäjä, eli tässä tapauksessa henkilö, joka haluaa lisätä oman julkisen avaimensa smart contractiin.



Kuva 5: Käyttöliittymä

Smart contractia voi siis käyttää suoraan vaikka komentoriviltä Ethereum-clientilla, mutta helpompaa loppukäyttäjälle on toki käyttää käyttöliittymästä. Tätä varten tehtiin kuvassa 5 näkyvä simppeli käyttöliittymä. Eli käyttäjä laittaa verkko-osoitteensa tai halutessan vaikka ip-osoitteensa täsmälleen oikein ”Domain”-kohtaan. Alapuolelle käyttäjä laittaa julkisen avaimen, jolla sertifikaatti on allekirjoitettu.

Kun tiedot ovat oikein, käyttäjä painaa ”VALIDATE DOMAIN”-nappia ja odottaa vastausta smart contractilta. Jos jokin menee pieleen, käyttäjä voi ottaa lähettämänsä etherit takaisin ”WITHDRAW”-painikkeella. Muussa tapauksessa käyttäjä saa ilmoituksen, että verkkosivun validointi on onnistunut. Käyttäjä voi halutessaan käyttöliittymän

alareunasta hakea, löytyykö vaikka oman tai jonkun muun verkko-osoitteen tiedot lohkoketjusta. Oikeasta yläkulmasta käyttäjä voi tarkastella ohjeita.

Jos käyttäjä haluaa useammalle verkko-osoitteelle sertifikaatin, esim. foo.example.com ja example.com, pitää hänen lisätä molemmat sivut erikseen, koska järjestelmää ei ole ainakaan vielä suunniteltu toimimaan wildcard-sertifikaattien [36] kanssa. Wildcard-sertifikaateilla tarkoitetaan siis sitä, että sertifikaatti kattaa useita aliverkko-osoitteita eli vaikkapa *.example.com, joka sallisi muun muassa foo.example.com ja bar.example.com, muttei kuitenkaan foo.bar.example.com.

4.2.2 Itseallekirjoitetun sertifikaatin luonti

Käyttäjän pitää osata luoda oikeanlainen sertifikaatti, jotta sertifikaatti toimisi. Se onnistuu OpenSSL-kirjastoa [37] käyttäen seuraavasti:

1. Käyttäjä paikantaa koneeltaan tiedoston openssl.cnf. Tämä löytyy Debian-järjestelmissä usein sijainnista /etc/ssl/openssl.cnf.

2. Kyseisestä tiedostosta käyttäjä paikantaa kohdan [v3_ca], ja lisää siihen rivin, siten, että lopputulos on seuraava.

```
[ v3_ca ]
1.2.3.4.87073 = critical, ASN1:INTEGER:0
```

3. Tallennetaan tiedosto ja avataan komentorivi, johon kirjoitetaan seuraava komento.

```
openssl req -newkey rsa:2048 -nodes -keyout private.key -x509 -days 3000
-out domain.crt
```

4. Käyttäjä täyttää tiedot komentoriville auenneeseen lomakkeeseen. Tässä ainut tärkeä tieto, joka täytyy mennä oikein, on *Common Name.* Siihen käyttäjä laittaa oman verkko-osoitteensa täysin oikein, esim. example.com tai foo.example.com, muttei http://example.com.

5. Nyt käyttäjällä pitäisi olla sertifikaatti ja salainen avain.

Vaiheessa 2 saatiin OpenSSL käyttämään uutta extensionia, jolle annettiin tällä hetkellä sattumanvarainen Object Identifier (OID) [38]. Oikeasti kyseinen tai joku muu vapaa OID pitäisi saada standardoitua tähän käyttöön sekä saada selainten tuki kyseiselle extensionille, mutta tuo riittää havainnollistamistarkoituksiin. Extensionille määriteltiin arvoksi ASN.1-notaatiolla, [39] kokonaisluku, joka on suurudeltaan 0.

ASN.1-notaatiota käytetään datan luonteen määrittelyyn. Tarkoituksena tällä määrittelyllä oli, että extension, jonka OID on tässä työssä mielivaltaisesti valittu, 1.2.3.4.87073, hyväksyisi arvoikseen kokonaislukuja, jotka viittaisivat eri smart contracteihin ja/tai lohkoketjuihin. Tässä tapauksessa arvo 0 viittaisi siis tässä projektissa tehtyyn smart contractiin. Näin saataisiin tuki muillekin kuin vain yhdelle smart contractille, johon sertifikaatin voisi validoida. Tämän järjestelmän toimiminen tietenkin vaatisi, että kaikki arvot saataisiin standardisoitua.

Kohdassa 3 luotiin varsinainen sertifikaatti ja sitä vastaava salainen avain. Ensimmäisellä argumentilla, ”req”, määritettiin, että halutaan tehdä sertifikaatin allekirjoituspyyntö (CSR).

Seuraava komento ”-newkey rsa:2048” määritteli, että käyttäjä haluaa luoda uuden 2048-bittisen avainparin käyttäen RSA-salausta, jonka salaista avainta käytettäisiin sertifikaatin allekirjoituspyynnön allekirjoittamiseen. 2048 bittiä pitäisi taata riittävä turvallisuus vielä useiksi vuosiksi ja sitä on nopeampi käyttää kuin vaikkapa 4096-bittistä sertifikaattia. Käyttäjän tulee käyttää RSA-salausta; muuten validointi ei onnistu.

Tämän jälkeen määritettiin, että salaista avainta ei tule salata DES-salauksella komennolla ”-nodes”. Muuten avaimen käyttäminen vaatisi salasanan, joka ei oikein sovellu palvelinkäyttöön.

Lopuksi sertifikaatin allekirjoituspyyntö viimeisteltiin komennolla ”-keyout private.key”, eli tehtiin tiedosto nimeltään private, jonka pääte on .key. Tämä sisältää sertifikaatin käyttämän salaisen avaimen.

Tässä kohtaa, ennen kuin OpenSSL suorittaa komennon loppuun, eli vaiheen ”-x509 -days 3000 -out domain.crt”, näytetään käyttäjälle kohdan 4 lomake. Lomake on CSR:n standardilomake, johon täytetään käyttäjän organisaation tietoja, jotka lisätään sitten sertifikaattiin.

Kun käyttäjä on saanut lomakkeen täytettyä, jatketaan komennon suorittamista kertomalla OpenSSL:lle, että halutaan tehdä itseallekirjoitettu sertifikaatti käyttäen äskeisessä vaiheessa luotua sertifikaatin allekirjoituspyyntöä ja salaista avainta, komennolla ”-x509”. Käyttäjä määrittelee vielä avaimen voimassaoloajan ”-days 3000”, eli avain vanhenee 3000 päivän päästä. Tällä arvolla ei varsinaisesti ole väliä, koska tieto sertifikaatin voimassaolosta sijaitsee lohkoketjussa. X.509-standardi suosittelisi käyttämään arvoa ”99991231235959Z” tilanteissa, joissa avaimen vanhenemisesta ei ole varmaa tietoa [40], mutta kuitenkin selaintuen vuoksi, ei tätä arvoa tässä käytetty, koska on mahdollista, etteivät kaikki selaimet tue GeneralizedTime-aikoja, vaan käyttävät UTC-aikoja. Pääasia, että vanhenemisaika on kaukana tulevaisuudessa.

Lopuksi vielä komennolla ”-out domain.crt” määritetään, minkä nimisenä sertifikaatti halutaan saada.

4.2.3 Tiedoston allekirjoittaminen salaisella avaimella

Kun käyttäjällä on itseallekirjoitettu sertifikaatti ja sen salainen avain, pitäisi saada allekirjoitettua tiedosto, jonka smart contract tarkistaa. Näin voidaan varmistua oraakkelin antaman viestin tulleen henkilöltä, joka on validoimassa verkkosivuaan, eikä kukaan, esim. oraakkeli, ole voinut muuttaa viestiä. Tämä on vain lisävarmistus, joka tekee oraakkelien huijaamisesta hieman vaikeampaa. Näin voidaan olla varmoja viestin tulleen julkisen avaimen omistavalta henkilöltä.

1. Aluksi luodaan tiedosto, jonka nimeksi tulee käyttäjän Ethereum-tilin osoite, esimerkiksi ”0x627306090abab3a6e1400e9345bc60c78a8bef57” ja lisätään siihen teksti ”proof”. Laitetaan tämä osoite selkeyden vuoksi terminaalin environment muuttujaksi nimellä ”osoite”. Tekstin ”proof” täytyy olla täsmälleen tuossa muodossa, eikä siinä saa olla välilyöntejä ja rivinvaihtoja lopussa. Tiedoston pitää lisäksi käyttää UTF-8-koodausta ja rivinloput pitää olla koodattu Unix/Linux-tapaan. Muuten tiedostolle tulee väärä SHA-256-tiiviste, eikä validointi onnistu. Tämän tiedoston voisi laittaa myös verkosta saataville.

```
export osoite=0x627306090abab3a6e1400e9345bc60c78a8bef57
echo proof > $osoite-plaintext
```

2. Tässä vaiheessa käyttäjän olisi hyvä varmistaa, että tiedoston tiiviste on oikein. Tiivisteen tulisi vastata täysin alla olevaa tiivistettä; muuten validointi epäonnistuu.

```
sha256sum $osoite-plaintext
f6ed42a9d765eeb230a069bbc3d5dc346b2669594bb0b83cc6d14d5d967b8961
```

3. Oletetaan käyttäjän olevan samassa kansiossa, missä hänen private.key-tiedostonsa sijaitsee ja allekirjoitetaan todistus salaisella avaimella.

```
openssl dgst -sha256 -sign private.key -out $osoite $osoite-plaintext
```

4. Tiedostosta on nyt tehty SHA-256-tiiviste, ja se on allekirjoitettu, mutta se on binäärimuodossa, joka ei ole käytännöllinen muoto HTTP-kutsuissa, kun smart contract koittaa hakea tiedoston. Täten base-64 koodataan tiedosto.

```
openssl base64 -in $osoite > $osoite.temp && mv $osoite.temp $osoite
```

5. Käyttäjä voi vielä tässä vaiheessa varmistaa julkisen avaimensa avulla, että tiedosto on luotu oikein. Käyttäjän tulisi saada viesti ”Verified OK”, viesti saattaa vaihdella OpenSSL-versiosta riippuen.

```
openssl dgst -verify <(openssl x509 -in domain.crt -pubkey -noout) -sig-
nature <(openssl base64 -d -in $osoite) $osoite-plaintext
```

6. Nyt tiedosto on valmis jaettavaksi verkkosivulle ja $osoite-plaintext tiedoston voi poistaa. Tiedosto tulisi siis laittaa siten, että se on saatavilla osoitteesta <verkko-osoite>/ethtls/<Ethereum-osoite>.

4.2.4 Verkko-osoitteen validointi

Nyt kun validoitava palvelin on kunnossa ja sertifikaatti on tehty, käyttäjän on aika validoida sertifikaattinsa. Käyttäjän on helpointa tehdä se dappin kautta käyttäen graafista käyttöliittymää. Tämä vaatii, että käyttäjällä tulee olla Mist-selain, [41] jolla voi selata Ethereumiin rakennettuja dappeja. Mist on muuten kuin normaali selain, mutta Mistissä käyttäjä pystyy hallinnoimaan omia Ethereum-tilejään ja dappit pystyvät käyttämään näitä tilejä kommunikointiin smart contractien kanssa. Käyttäjä voi myös vaihtoehtoi-

sesti ladata lisäosan nykyiseen selaimeensa, kuten Metamask [42], joka mahdollistaa tavallisesta selaimesta kommunikoinnin Ethereumin lohkoketjun kanssa.

Ethereum-tilejä tukevalla selaimella käyttäjä menee dappin osoitteeseen, vaikkapa ethtls.com ja syöttää oman verkko-osoitteensa tismalleen samalla tavalla kuin se on sertifikaatissa. Sitten käyttäjä syöttää vielä sertifikaatin julkisen avaimen, jonka hän saa seuraavalla komennolla.

```
openssl x509 -in domain.crt -pubkey -noout
```

Vastauksen tulisi olla seuraavan näköinen, tosin eri arvolla

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuM44Rr3WcKvqI5NFjOa3
DrGYpQyglEk7Rm/Jtl76ZSU6tBvVHS05VkbKfbYVC3CY0mgcXq1CuMF1c3ZThRaO
+O949TQDZDqj8EAdNaYEMLCfxrXVRNUVyvZn0we0TrU288btMNb1qq65cgkmhUBk
iwV4O0/4sc6AKXZshqnLIM0EeJc9z6xqXyrYIDu3Qj5DDr5MZgvCA8ImZf9gQfvU
L2FUHHPpDZ5yCDBd2AVmm6SfxYJb/NxIkzKqzlVG4mLU4MEmxRMukJkzCEoQp536
G+NmM8mHbB7xs1HW0bxDuhAgQs+gBRjuR9hk3xQv20ZCZx+AcfQ0+iO+33IHCD+c
ZwIDAQAB
-----END PUBLIC KEY-----
```

Käyttäjä kopioi avaimen kokonaan, eli mukaan tulee myös -----BEGIN PUBLIC KEY----- ja -----END PUBLIC KEY-----. Käyttäjän tulee poistaa kaikki rivinvaihdot avaimesta siten, että siitä tulee yksi yhtenäinen pitkä merkkijono.

Kun käyttäjä on syöttänyt molemmat tiedot, verkko-osoitteen ja julkisen avaimen oikein, hän painaa validointi-painiketta ja käyttäjän Ethereum-tililtä otetaan 0.1 etheriä pantiksi. Jos validointi epäonnistuu, käyttäjä voi ottaa takaisin pantista jäljelle jääneet etherit, mitä smart contractin suorituksesta jäi. Jos validointi onnistuu, käyttäjältä jäljelle jääneet etherit siirtyvät smart contractin tekijälle.

Kun validointi onnistuu, on käyttäjällä sertifikaatti, jonka aitous on validoitu lohkoketjuun ilman sentralisoituja certificate authorityja. Vielä kuitenkin tarvitaan selainten tuki; muutenhan näillä sertifikaateilla ei ole mitään arvoa.

4.2.5 Selaintuki

Selaintuen tarkoitus olisi siis toimia niin, että selaimet katsoisivat TLS Handshaken aikana, että löytyykö sertifikaatista tässä työssä kehitetty extension 1.2.3.4.87073. Jos extension löytyy, se katsoisi sen arvon ja sillä perusteella tietäisi, mistä smart contractista hakea arvoja.

Miten selain sitten hakisi tietoja lohkoketjusta? Selain voisi pyytää tietoja joltain lohkoketjun nodelta HTTP-kutsulla, mutta tästä tulee sama ongelma kuin oraakkeleissa. Emme tiedä, puhuuko node totta lohkoketjun tilanteesta. Käytännössä tämä siis tarkoittaa sitä, että saadakseen varmasti oikean vastauksen käyttäjällä pitäisi olla koko lohkoketju itsellään tiedossa. Tämä taas ei ole käytännöllistä, koska esimerkiksi Ethereumin koko on tällä hetkellä 65 GB [43]. Tämän voisi ratkaista kehittämällä lohkoketjun, joka on vain ja ainoastaan tätä tarkoitusta varten tai sitten käyttämällä samanlaista ratkaisua kuin desentralisoiduissa oraakkeleissa.

Selaintuen saamiseksi kuitenkin pitäisi käydä pitkä prosessi, joka vaatisi paljon kehitystyötä, ja se muuttaisi hyvin pitkälti täysin nykyisen tavan tunnistaa aidot sertifikaatit. Lisäksi koko systeemi pitäisi saada standardisoitua, sekä varmistaa konseptin toimivuus ja turvallisuus, joten sen käytännön toteutus ei ole mahdollista tähän työhön.

4.2.6 Jatkokehitys

Koska validointiprosessi ei ole loppukäyttäjälle helppo, pitäisi koko prosessi ja samalla sertifikaatin uudistaminen automatisoida. Sitten käyttäjän tarvitsisi vain ladata esimerkiksi bash-tiedosto, jossa kysyttäisiin aluksi käyttäjältä tämän verkko-osoitteen nimeä, Ethereum-tilin tunnuksia ja skripti hoitaisi automaattisesti loput. Skripti voisi myös tarpeen mukaan tehdä käyttäjälle Ethereum-tunnukset ja ohjata käyttäjän sivulle, josta hän voi ostaa etheriä itselleen. Samalla bash-tiedostolla voitaisiin myös luoda cron-job [44], eli automaattisesti tiettyinä aikoina suoritettavaa koodia, joka hoitaisi sertifikaatin uusimisen. Näin tästä saataisiin todella helppo käyttökokemus.

Smart contractiin pitäisi myös tehdä kaiken varalta takaportti, josta smart contractin tekijä voisi käydä mitätöimässä validoidun sertifikaatin, jos joku onnistuisi saamaan sertifikaatin sivulle, jota ei omista, tai jos se pitäisi jostain muusta syystä mitätöidä. Mitätöinti voisi toimia myös niin, että siihen vaadittaisiin useampia ihmisiä, esimerkiksi annettai-

siin 10 luotetulle ihmiselle tai organisaatiolle oikeus mitätöidä sertifikaatti, mutta itse mitätöinti vaatisi, että vähintään 6 heistä hyväksyy sen. Näin kukaan ei voisi itsekseen käydä mitätöimässä sertifikaatteja.

Lisäksi, kuten aiemmin todettiin, Ethereum olisi ehkä hyvä korvata yksityisellä lohkoketjulla, joka on vain tätä tarkoitusta varten, jossa nodeina toimisi useat luotetut kolmannet osapuolet, kuten nykyiset Certificate Authorityt. He myös toimisivat oraakkeleina tässä lohkoketjussa. Näin saadaan hyödyt sekä luotetuista kolmansista osapuolista, että lohkoketjujen antamasta turvasta ja läpinäkyvyydestä, jossa verkon huijaamiseen vaadittaisiin Certificate Authorityjen enemmistö. Tämä ratkaisisi samalla Ethereumin tai jonkin muun julkisen lohkoketjun koko-ongelmat.

5 Pohdinta

Tämän työn tavoitteena oli kehittää uusi vaihtoehto sertifikaatteja allekirjoittaville Certificate Authorityille, niiden vallalle ja sen aiheuttamalle SSL/TLS-salauksen tietoturva-aukolle.

Tähän tavoitteeseen onnistuttiin kehittämään teknisesti pätevä ratkaisu, jolla Certificate Authorityt voisi korvata. Tätä varten luotiin Ethereumin lohkoketjuun smart contract, joka hallitsee verkko-osoitteiden validointia. Tietojen hakeminen Ethereumin lohkoketjusta kuitenkin osoittautui epäkäytännölliseksi sen suuren kokonsa vuoksi ja tähän ehdotettiin ratkaisuksi yksityistä lohkoketjua vain tätä tarkoitusta varten, jossa nodeina toimisi useita luotettuja kolmansia osapuolia. Toinen vaihtoehto oli käyttää useita nodeja haettaessa lohkoketjusta tietoja, jotta voidaan varmistua nodejen antamien tietojen pitävän paikkaansa.

Projektissa toteutettiin myös käyttöliittymä, jonka kautta käyttäjä voi helpommin kommunikoida tässä työssä kehitetyn smart contractin kanssa. Käyttöliittymä kuitenkin vaatisi vielä hiomista, jotta siitä saisi laadukkaamman näköisen ja sen käytettävyyttä tulisi parantaa.

Myös muut jatkokehitystarpeet huomioitiin, kuten se, että sertifikaatin validointiprosessi pitäisi automatisoida lähes täysin ja smart contractin testausta parantaa, sekä tehdä päivityksiä koodiin, kuten takaovi, josta mitätöidä sertifikaatteja.

Selaintukea tässä projektissa ei vielä saatu tehtyä, koska se vaatisi huomattavasti isompia resursseja ja käytännössä kokonaisen tiimin ajamaan asiaan. Tämä ei ollut tässä työssä mahdollista. Tämän työn tarkoitus oli enemmän tarkastella teknisen toteutuksen mahdollisuutta ja sen mahdollisia ongelmakohtia kuin tehdä täydellinen lopputuote.

Vaikka selaintuki saataisiinkin, se vaatisi silti sen, että nykyinen Certificate Authority-järjestelmä lakkautettaisiin täysin, koska jos edes yksi Certificate Authority on jäljellä, joka pystyy myöntämään sertifikaatteja mihin tahansa verkkosivulle, tämä ongelma ei poistu mihinkään. Tämän saavuttaminen on kuitenkin erittäin epätodennäköistä lähitulevaisuudessa, mutta toivottavasti vielä joskus nähdään Certificate Authorityjen poistuvan ja tilalle tulevan uuden paremman järjestelmän.

Lähteet

1 Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Verkkoaineisto. <http://article.gmane.org/gmane.comp.encryption.general/12588>. Päivitetty 31.10.2008. Luettu 31.3.2018.

2 Buterin ym. 2014. A Next-Generation Smart Contract and Decentralized Application Platform. Verkkoaineisto. <https://github.com/ethereum/wiki/wiki/White-Paper>. Päivitetty 4.4.2018. Luettu 31.3..2018.

3 Turing completeness. 2001. Turing Completeness. Verkkoaineisto. <https://en.wikipedia.org/wiki/Turing_completeness>. Päivitetty 15.4.2018. Luettu 17.4.2018.

4 Dierks, Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. Verkkoaineisto. <https://tools.ietf.org/html/rfc5246#section-1>. Päivitetty 8.2008. Luettu 31.3.2018.

5 Dierks, Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. Verkkoaineisto. <https://tools.ietf.org/html/rfc5246#section-6.2.1>. Päivitetty 8.2008. Luettu 31.3.2018.

6 Dierks, Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. Verkkoaineisto. <https://tools.ietf.org/html/rfc5246#section-7.1>. Päivitetty 8.2008. Luettu 31.3.2018.

7 Dierks, Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. Verkkoaineisto. <https://tools.ietf.org/html/rfc5246#section-7.2>. Päivitetty 8.2008. Luettu 31.3.2018.

8 Dierks, Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. Verkkoaineisto. <https://tools.ietf.org/html/rfc5246#section-7.3>. Päivitetty 8.2008. Luettu 31.3.2018.

9 Dierks, Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. Verkkoaineisto. <https://tools.ietf.org/html/rfc5246#section-7.4.9>. Päivitetty 8.2008. Luettu 31.3.2018.

10 Man-in-the-middle attack. 2002. Verkkoaineisto. <https://en.wikipedia.org/wiki/Man-in-the-middle_attack>. Päivitetty 28.3.2018. Luettu 14.4.2018.

11 Swati Khandelwal. 2016. Chinese Certificate Authority 'mistakenly' gave out SSL Certs for GitHub Domains. Verkkoaineisto. <https://thehackernews.com/2016/08/github-ssl-certificate.html>. Päivitetty 29.8.2016. Luettu 1.4.2018.

12 Dennis Fisher. 2011. Attackers Obtain Valid Cert for Google Domains, Mozilla Moves to Revoke It. Verkkoaineisto. <https://threatpost.com/attackers-obtain-valid-cert-google-domains-mozilla-moves-revoke-it-082911/75590/>. Päivitetty 29.8.2011. Luettu 1.4.2018.

13 Paul Roberts. 2011. Phony SSL Certificates issued for Google, Yahoo, Skype, Others.

Verkkoaineisto. <https://threatpost.com/phony-ssl-certificates-issued-google-yahoo-skype-others-032311/75061/>. Päivitetty 23.3.2011. Luettu 1.4.2018.

14 Cooper, NIST, Santesson, Microsoft, Farrell, Trinity College Dublin, Boeyen, Entrust, Housley, Vigil Security, Polk. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Verkkoaineisto. <https://tools.ietf.org/html/rfc5280#section-4.2.1.10>. Päivitetty 5.2008. Luettu 1.4.2018.

15 Evans, Palmer, Sleevi, Google. 2015. Public Key Pinning Extension for HTTP. Verkkoaineisto. <https://tools.ietf.org/html/rfc7469#section-1>. Päivitetty 4.2015. Luettu 1.4.2018.

16 Laurie, Langley, Kasper, Google. 2013. Certificate Transparency. Verkkoaineisto. <https://tools.ietf.org/html/rfc6962#section-1>. Päivitetty 6.2013. Luettu 1.4.2018.

17 Domain Name System. 2001. Verkkoaineisto. <https://en.wikipedia.org/wiki/Domain_Name_System>. Päivitetty 7.4.2018. Luettu 8.4.2018.

18 SSL Certificate Validation Process from DigiCert. 2018. Verkkoaineisto. <https://www.digicert.com/ssl-validation-process.htm>. Päivitetty 2018. Luettu 8.4.2018.

19 Barnes, Cisco, Hoffman-Andrews, EFF, McCarney, Let's Encrypt, Kasten, University of Michigan. Automatic Certificate Management Environment (ACME). 2018. Verkkoainesto. <https://ietf-wg-acme.github.io/acme/draft-ietf-acme-acme.html>. Päivitetty 27.3.2018. Luettu 8.4.2018.

20 Blockchain. 2014. Verkkoaineisto. <https://en.wikipedia.org/wiki/Blockchain>. Päivitetty 6.4.2018. Luettu 7.4.2018.

21 Rogaway, Shrimpton. 2004. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. Verkkoaineisto. <https://link.springer.com/content/pdf/10.1007%2F978-3-540-25937-4_24.pdf>. Päivitety 2004. Luettu 7.4.2018.

22 Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Verkkoaineisto. <https://bitcoin.org/bitcoin.pdf> kappale 4 "Proof-of-Work". Päivitetty 31.10.2008. Luettu 7.4.2018.

23 Kiran Vaidya. 2016. Decoding the enigma of Bitcoin Mining. Verkkoaineisto. <https://medium.com/all-things-ledger/decoding-the-enigma-of-bitcoin-mining-f8b2697bc4e2>. Päivitetty 15.12.2016. Luettu 7.4.2018.

24 Proof-of-stake. 2013. Verkkoaineisto. <https://en.wikipedia.org/wiki/Proof-of-stake>. Päivitetty 2.4.2018. Luettu 7.4.2018.

25 Bitcoin Energy Consumption Index. 2018. Verkkoaineisto. <https://digiconomist.net/bitcoin-energy-consumption>. Päivitetty 7.4.2018. Luettu 7.4.2018.

26 Ethereum. 2014. Verkkoaineisto. <https://en.wikipedia.org/wiki/Ethereum>. Päivitetty 7.4.2018. Luettu 8.4.2018.

27 Casper. 2018. Verkkoaineisto. <https://github.com/ethereum/casper/blob/master/IMPLEMENTATION.md>. Päivitetty 2.4.2018. Luettu 8.4.2018.

28 Waltonchain White Paper. 2018. Verkkoaineisto. <https://www.waltonchain.org/doc/Waltonchain-whitepaper_en_20180208.pdf>. Päivitetty 8.2.2018. Luettu 8.4.2018.

29 Nick Szabo. 1997. The Idea of Smart Contracts. Verkkoaineisto. <http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html>. Päivitetty 1997. Luettu 8.4.2018.

30 Ellis, Juels, Nazarov. 2017. ChainLink A Decentralized Oracle Network. Verkkoaineisto. <https://link.smartcontract.com/whitepaper>. Päivitetty 4.9.2017. Luettu 12.4.2018.

31 Anati, Gueron, Jhnson, Scarlata. 2013. Innovative Technology for CPU Based Attestation and Sealing. Verkkoaineisto. <https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing>. Päivitetty 14.8.2013. Luettu 12.4.2018.

32 Richard Chirgwin. 2017. Boffins show Intel's SGX can leak crypto keys. Verkkoaineisto. <https://www.theregister.co.uk/2017/03/07/eggheads_slip_a_note_under_intels_door_sgx_can_leak_crypto_keys/>. Päivitetty 7.3.2017. Luettu 12.4.2018.

33 Cooper, NIST, Santesson, Microsoft, Farrell, Trinity College Dublin, Boeyen, Entrust, Housley, Vigil Security, Polk. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Verkkoaineisto. <https://tools.ietf.org/html/rfc5280#section-4.2>. Päivitetty 5.2008. Luettu 1.4.2018.

34 Truffle. 2018. Verkkoaineisto. <http://truffleframework.com>. Päivitetty 2018. Luettu 8.4.2018.

35 Oraclize. 2018. Verkkoainesto . <http://docs.oraclize.it/>. Päivitetty 13.2.2018. Lutettu 9.4.2018

36 Saint-Andre, Cisco, Hodges, PayPal. 2011. Verkkoaineisto. <https://tools.ietf.org/html/rfc6125#section-6.4.3>. Päivitetty 3.2011. Luettu 9.4.2018.

37 OpenSSL. 2017. Verkkoaineisto. <https://www.openssl.org/>. Päivitetty 2017. Luettu 9.4.2018.

38 Mealling, Verisign. 2011. Verkkoaineisto. <https://tools.ietf.org/html/rfc3061>. Päivitetty 2.2001. Luettu 10.4.2018.

39 Introduction to ASN.1. 2018. Verkkoaineisto. <https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx>. Päivitetty 2018. Luettu 10.4.2018.

40 Cooper, NIST, Santesson, Microsoft, Farrell, Trinity College Dublin, Boeyen, Entrust, Housley, Vigil Security, Polk. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Verkkoaineisto. <https://tools.ietf.org/html/rfc5280#section-4.1.2.5>. Päivitetty 5.2008. Luettu 10.4.2018

41 Mist Browser. 2018. Verkkoainesto. <https://github.com/ethereum/mist>. Päivitetty 11.4.2018. Luettu 13.4.2018.

42 Metamask. Verkkoainesto. <https://metamask.io/>. Luettu 13.4.2018.

43 Ethereum ChainData Size. 2018. Verkkoaineisto. <https://etherscan.io/chart2/chaindatasizefast>. Päivitetty 16.4.2018. Luettu 17.4.2018.

44 Crontab – schedule periodic background work. 2018. Verkkoaineisto. <http://pubs.opengroup.org/onlinepubs/9699919799/utilities/crontab.html>. Päivitetty 2018. Luettu 13.4.2018.