Kanditutkielma

Tietojenkäsittelytieteen kandiohjelma

Satunnaisuus lohkoketjusovelluksissa

Lauri Tahvanainen

12.5.2022

Matemaattis-luonnontieteellinen tiedekunta

Helsingin yliopisto

Yhteystiedot

PL 68 (Pietari Kalmin katu 5)
00014 Helsingin yliopisto

Sähkopostiosoite: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen kandiohjelma

Tekijä — Författare — Author

Lauri Tahvanainen

Työn nimi — Arbetets titel — Title

Satunnaisuus lohkoketjusovelluksissa

Ohjaajat — Handledare — Supervisors

Prof. K. Heljanko, FM. A. Vainio

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kanditutkielma	12.5.2022	28 sivua

Tiivistelmä — Referat — Abstract

Lohkoketjusovellukset tarvitsevat satunnaisuutta. Lohkoketjun deterministisyyden ja lupavapauden takia lohkoketjusovelluksessa käytettävän satunnaisuuden täytyy olla manipuloimatonta, ennustamatonta sekä julkisesti todennettavaa. Tutkielmassa kartoitetaan olemassaolevia menetelmiä, joilla lohkoketjusovellus saa hyödynnettäväkseen kyseiset vaatimukset täyttävää satunnaisuutta. Menetelmät esitellään tiivisti, käydään läpi niiden oletuksia, heikkouksia sekä yksittäiselle käyttäjälle koituvia kommunikointikustannuksia. Pääpaino on menetelmissä, joissa sovelluksen käyttäjät tuottavat itse sovelluksen tarvitseman satunnaisuuden. Menetelmiä vertaillaan ja analysoidaan niiden soveltuvuutta erilaisiin lohkoketjusovelluksiin. Yksi menetelmä ei sovi kaikkiin käyttötarkoituksiin, vaan menetelmä on valittava käyttötarkoituksen mukaisesti. Tutkielman tuloksia voi käyttää esimerkiksi satunnaisuutta vaativan lohkoketjusovelluksen suunnittelussa.

ACM Computing Classification System (CCS)
Networks → Network types → Overlay and other logical network structures → Peer-to-peer networks
Mathematics of computing → Probability and statistics → Probabilistic reasoning algorithms → Random number generation

Avainsanat — Nyckelord — Keywords

blockchain, randomness

Säilytyspaikka — Förvaringsställe — Where deposited

Helsingin yliopiston kirjasto

Muita tietoja — övriga uppgifter — Additional information

Sisällys

1 Johdanto

Lohkoketjusovelluksilla on kymmeniä tuhansia päivittäisiä käyttäjiä [31]. Sovellukset pyrkivät tarjoamaan erilaisia ratkaisuja aina terveystietojen hallinnasta[†] hajautettuihin rahoitusmarkkinoihin[‡]. Lohkoketjuteknologia mahdollistaa sellaisten sovellusten toteuttamisen, joissa sovelluksen käyttäminen on avointa ja käyttäjät voivat käyttää sovelluksia anonyymisti eikä käyttäjien tarvitse silti luottaa toisiinsa eikä mihinkään kolmanteen osapuoleen.

Lohkoketjun deterministisyys, avoimuus ja käyttäjien anonyymiteetti luo kuitenkin haasteen lohkoketjusovelluksille, jotka vaativat oikein toimiakseen satunnaisuutta. Tälläisia sovelluksia ovat esimerkiksi hajautetut lotto-arvonnat[§], jotka tarvitsevat satunnaislukuja arvonnan suorittamiseen, sekä hajautetut tuomioistuimet, jotka tarvitsevat satunnaisuutta tuomareiden valitsemiseen [19].

Haitallisilla toimijoilla on intressi ennustaa tai manipuloida lohkoketjusovelluksessa käytettävää satunnaisuutta. Lohkoketjuteknologia mahdollistaa taloudellisen arvon vaivattoman siirtämisen järjestelmän käyttäjien välillä ja haitallisia toimijoita voi lohkoketjusovelluksissa motivoida satunnaisuuden ennustamiseen tai manipulointiin kymmenien, jopa satojen, miljoonien arvoiset palkkiot. Toisaalta arvon siirtämisen mahdollisuuden myötä lohkoketjuympäristössä satunnaisuutta tuottaviin menetelmiin voidaan lisätä taloudellisia ja peliteoreettisia kannustimia haitallisten toimijoiden torjumiseksi.

Satunnaisuuden tuottamista hajautetussa ympäristössä, jossa käyttäjät eivät luota toisiinta on tutkittu jo 1980-luvulla [6]. Vaikka suurin osa tutkielmassa esiteltävistä menetelmistä on suunniteltu nimenomaan lohkoketjuympäristöön, niin esittelen tutkielmassa myös satunnaisuuden tuottamisen protokollia jotka eivät vaadi toimiakseen lohkoketjua, mutta ne voidaan toteuttaa myös lohkoketjujen kontekstissa. Protokollat pyrkivät vastaamaan kysymykseen: Kuinka käyttäjät, jotka eivät luota toisiinsa voivat tuottaa satunnaisuutta? Hajautettua satunnaisuutta tuottavia protokollia hyödynnetään lohkoketjusovellusten lisäksi osana lohkoketjun toimintaa. Esimerkiksi tietyissä konsensusalgoritmeissa [15, 17], joiden avulla lohkoketjun osallistujat saavuttavat yhteisymmärryksen lohkoketjun nykyisestä tilasta, hyödynnetään hajautetusti tuotettua satunnaisuutta.

[†]https://medibloc.com/en/
[‡]https://ethereum.org/en/defi/
[§]https://pooltogether.com

Tässä tutkielmassa kokoan yhteen olemassaolevia menetelmiä, jotka tarjoavat ratkaisun kysymykseen: Kuinka lohkoketjusovellus saa hyödynnettäväkseen satunnaisuutta? Luvussa 2 määrittelen tarvittavat käsitteet ja vaatimukset satunnaisuudelle. Luku 3 esittelee menetelmiä, joilla lohkoketjusovellukseen saadaan satunnaisuutta sekä analysoi miten hyvin nämä menetelmät täyttävät satunnaisuudelle asetetut vaatimukset. Tutkielman pääpaino on protokollissa, joiden avulla sovelluksen käyttäjät tuottavat itse sovelluksen vaatiman satunnaisuuden. Luvussa käydään läpi eri menetelmien oletuksia, kommunikointikustannuksia, esitetään hyökkäyksiä, joita menetelmiä vastaan voidaan kohdistaa sekä esitetään huomioita menetelmien toteuttamisesta lohkoketjusovelluksessa. Luvussa 4 vertaillaan eri menetelmiä ja tarkastellaan niiden sopivuutta erilaisiin lohkoketjusovelluksiin. Luku 5 toimii yhteenvetona.

2 Määritelmät

Luvussa esitetään tutkielman kannalta keskeiset määritelmät. Määrittely aloitetaan tarvittavista kryptografisista primitiiveistä, joita tarvitaan lohkoketjun määritelmässä. Lohkoketjusovellus määritellään tarkemmin, minkä jälkeen käydään läpi lohkoketjusovelluksessa hyödynnettävään satunnaisuuteen kohdistettavat vaatimukset.

2.1 Kryptografiset primitiivit

Seuraavat kryptografiset primitiivit ovat oleellisia lohkoketjun rakennuspaloja ja niitä käytetään lohkoketjusovelluksissa.

Tiivistefunktio on funktio h, joka liittää mielivaltaisen pituiset syötteet saman pituisiin maalijoukon alkioihin, tiivisteisiin. Binäärimuodossa:

$$H : \{0,1\}^* \longrightarrow \{0,1\}^l, \text{ missä } l \text{ on tiivisteen pituus} \tag{2.1}$$

Jotta tiivistefunktio olisi *kryptografinen tiivistefunktio*, sen tulee täyttää kolme ominaisuutta [18]:

1. Funktio on vahvasti törmäyskestävä (Collision Resistant), eli on laskennallisesti kannattamatonta löytää kaksi syötettä, jotka tuottavat saman tiivisteen.
2. Funktio on alkukuvakestävä (Pre-image resistance), eli on laskennallisesti kannattamatonta löytää tiivistettä vastaava syöte.
3. Funktio on heikosti törmäyskestävä, eli jos annettuna on syöte, on laskennallisesti kannattamatonta löytää toinen syöte, jonka tiiviste on sama.

Tutkielmassa tiivistefunktioiden oletetaan olevan kryptografisia, ja käytetään vain termiä tiivistefunktio. Esimerkiksi Ethereum-lohkoketjussa on käytössä Keccak-256 tiivistefunktio [33].

Lohkoketjussa käytetään *Julkisen avaimen salausta* transaktioiden allekirjoittamiseen. Asymmetrisessä avainparissa on salainen avain (SK), sekä salaisesta avaimesta johdettu julkinen

avain (PK). Avainpari toimii lohkoketjussa osallistujan ainoana tunnisteena, mikä mahdollistaa avoimen ja anonyymin osallistumisen. Esimerkiksi Ethereum-ketjussa on käytössä elliptisiin käyriin nojaava ECDSA-algoritmi (Elliptic Curve Digital Signature Algorithm) [33].

2.2 Lohkoketju

Lohkoketju on hajautettu ja muuttumaton tilikirja [3]. Perinteisesti tilikirjalla tarkoitetaan tilien kokoelmaa, johon on merkitty kaikki tilien olemassaolon aikana tapahtuneet transaktiot. Uutta tietoa on mahdollista lisätä vain tilikirjan loppuun lisäämällä uusia transaktioita.

Lohkoketjussa uusia transaktioita voi lisätä vain luomalla lohkoketjun jatkeeksi uuden lohkon. Yksinkertaistaen, lohko koostuu transaktioista, sekä edellisen lohkon tiivisteestä. Edellisen lohkon tiiviste liittää uuteen lohkoon aikaisemmat lohkot muodostaen lohkoketjun. Lohkoketjun käyttäjät lähettävät tiliään koskevia transaktioita allekirjoittamalla transaktion salaisella avaimella, josta tilin osoite on johdettu. Uusi transaktio julkaistaan odottamaan, että ketjun päivittämisestä vastaavat konsensusmekanismia suorittavat käyttäjät lisäävät transaktion lohkoketjuun.

Lohkoketjussa on siis kaksi joukkoa käyttäjiä. Käyttäjien joukko, jotka lähettävät uusia transaktioita, sekä joukko, jotka suorittavat konsensusmekanismia lisätäkseen lohkoketjuun uusia transaktioita lohkoissa. Lohkoketju voi olla lupavapaa (permissionless) tai luvallinen (permissioned) [3]. Lupavapaassa lohkoketjussa kuka tahansa yksittäinen internetyhteyden omaava käyttäjä voi kuulua vapaasti kumpaan tahansa mainittuun joukkoon. Tämän lisäksi kenen tahansa on mahdollista tarkastella kaikkia lohkoketjussa aikaisemmin tapahtuneita transaktioita. Luvallisessa lohkoketjussa transaktioiden lähettämistä, konsensusmekanismiin osallistumista tai transaktioiden tarkastelua on rajattu tietylle joukolle käyttäjistä. Tutkielmassa oletetaan, että lohkoketjusovellusta suoritetaan lohkoketjussa, jossa kenellä tahansa on vähintään oikeus uusien transaktioiden lähettämiseen sekä aikaisempien transaktioiden tarkastelemiseen. Konsensusmekanismiin osallistumisessa voi olla rajoitteita, mutta osallistuminen ei ole täysin suljettu ennaltamäärätylle joukolle.

Konsensusmekanismia suorittamalla lohkoketjun käyttäjät saavuttavat yhteisymmärryksen siitä mikä on ketjun nykyinen validi tila. Uusien transaktioiden lisääminen lohkoissa tapahtuu suorittamalla konsensusmekanismia. Tutkielmassa huomioidaan kaksi merkittävintä konsensusmekanismia. Työtodistus (Proof of Work, PoW)-konsensusmekanismia,

käyttää kirjoitushetkellä esimerkiksi Bitcoin [21], sekä Ethereum [12] lohkoketjut. Uudempi ja yhä yleistyvä mekanismi on osakkuustodistus(Proof of Stake, PoS), jonka esimerkkinä toimii esimerkiksi Algorand-lohkoketjun konsensusmekanismi [15].

PoW-konsensuksessa jokainen konsensukseen osallistuja, joita kutsutaan kaivajiksi (Miner) kilpailee uuden lohkon lisäämisestä. Esitetään yksinkertaistetusti Bitcoin-valkopaperissa [21] esitellyn työtodistuksen toiminta.

Kaivaja valitsee uusista transaktioista joukon uutta lohkoa varten ja tarkistaa transaktioiden oikeellisuuden. Transaktiot, edellisen lohkon tiiviste, aikaleima sekä osallistujan valitsema satunnainen arvo (Nonce) syötetään tiivistefunktiolle. Aikaisempien lohkojen perusteella on sovittu vaikeutta kuvaava luku v. Jos saadun tiivisteen hexadesimaalinen arvo alkaa pienemmällä määrällä nollia kuin v on lohko hyväksyttävä. Tällöin kaivaja julkaisee uuden lohkon verkolle. Muut kaivajat voivat vielä vahvistaa lohkon oikeellisuuden suorittamalla yllä mainitun transaktioiden validoinnin ja tiivisteen laskemisen uudestaan löytäjän parametreilla ja varmistamalla, että saatu tiiviste vastaa löytäjän ilmoittamaa. Kun uusi lohko on validoitu niin kaivajat aloittavat prosessin alusta syötteenään uusi transaktioiden joukko, validoidun lohkon tiiviste, aikaleima, sekä nonce. Kaivajat yrittävät aina lisätä uutta lohkoa pisimmän validin ketjun jatkeeksi.

Jos tiiviste ei ala halutulla määrällä nollia niin kaivaja vaihtaa nonce-arvoa ja laskee tiivisteen uudestaan. Sopivan tiivisteen löytämiseen ei ole tiivisteen alkukuvakestävyyden myötä muuta keinoa kuin yrittää satunnaisia nonce-arvoja. Arvoa v muuttamalla voidaan vaikuttaa siihen kuinka kauan sopivan nonce-arvon löytämisessä kestää keskimäärin tietyllä laskentateholla ja arvo vaihtelee niin, että uusi lohko löydettäisiin keskimäärin aina samassa ajassa. Lohkoketjussa aikaisemmin olevien lohkojen transaktioiden muuttaminen vaatisi, että laskentatyö tehdään uudestaan muutetun sekä kaikkien sitä seuraavien lohkojen kohdalla. Konsensusmekanismi olettaa, että lohkojen luomiseen kohdistuvasta laskentatehosta yli 50% on rehellisten kaivajien hallussa, jolloin lohkoketjun muuttumattomuus on taattu, sillä rehelliset kaivajat kaivavat aina pisintä ketjua.

Löytäjä saa palkkiona uuden lohkon luomisesta vastaluotua valuuttaa, kuten myös transaktiokuluja. Transaktiokulu on transaktion lähettäjän määräämä arvo, joka lisätään lähetettävän arvon päälle, joten kaivajat valitsevat lohkoon transaktioita joissa on suurimmat transaktiokulut.

Tutkielman kannalta kiinnostavaa PoW-konsensusmekanismissa on se, että nonce-arvo, kuten myös siitä johdettu lohkon tiiviste, ei ole kenenkään ennustettavissa. PoW-lohkoketjujen lohkojen tiivistettä onkin mahdollista käyttää satunnaisuuden lähteenä. Tiivisteen käyttöä

satunnaisuuden lähteenä ja siihen liittyviä ongelmia käsitellään luvussa 3.

Osakkuustodistusta hyödyntävässä konsensusmekanismissa lohkon tuottaja valitaan käyttäjien tilien saldojen perusteella [22]. Lohkon tuottajaksi valittu käyttäjä tuottaa lohkon ja muut käyttäjät äänestävät lohkon oikeellisuudesta. Asynkronisessa verkossa konsensuksen taattu saavuttaminen deterministisellä konsensusprotokollalla on mahdotonta, jos edes yksi osallistuja toimii virheellisesti [14]. Tämän takia osakkuustodistuksessa valinta tehdään usein satunnaisesti, jolloin käytettävä konsensusprotokolla on epädeterministinen. Paino, joka määrää käyttäjän todennäköisyyden tulla valituksi lohkon tuottajaksi määräytyy käyttäjän tilin saldon mukaan. Koska lohkoketjun valuutalla on arvoa, on mekanismissa hyökkääjien omattava mittava määrä taloudellista arvoa, jotta he voivat luoda haluamiaan lohkoja. Usein oletus on, että 2/3 ketjun arvosta on rehellisten käyttäjien tileillä.

PoS-ketjujen konsensusalgoritmit tarvitsevat hajautetusti tuotettua satunnaisuutta lohkon luojan arpomiseen. Joitain tutkielmassa esiteltäviä protokollia voi siten käyttää myös konsensusmekanismin osana. Satunnaisuutta hyödyntävästä PoS-konsensusmekanismista toimii esimerkkinä Algorand-lohkoketjun konsensusmekanismi [15]. PoS-ketjun lohkon tiivistettä ei voi pitää PoW-ketjun lohkon tiivisteen tapaan satunnaisena sillä lohkon luo yksi validaattori, joka voi vaikuttaa tiivisteeseen valitsemalla lohkoon transaktioita, jotka tuottavat suotuisan tiivisteen.

2.3 Lohkoketjusovellus

Lohkoketjun voi nähdä myös tilakoneena. Jokainen transaktio edistää tilakonetta edellisten lohkojen määrittämän tilan pohjalta. Lohkoketjuun voidaan pelkkien tilien saldojen sijaan tallentaa kokonaisia ohjelmia. Näin lohkoketju toimii Turing-täydellisenä laskenta-alustana ja lohkoketju muodostaa virtuaalikoneen, jolla kuka tahansa voi suorittaa mitä tahansa muuttumatonta laskentaa lupavapaasti.

Tämän idean toteutti ensimmäisenä lohkoketju Ethereum [12]. Ethereum-lohkoketjussa ohjelmia suoritetaan lohkoketjussa pinopohjaisella Ethereum virtuaalikoneella (Ethereum Virtual Machine, EVM). EVM suorittaa tavukoodia, mutta ohjelmia voi kirjoittaa korkean tason ohjelmointikielellä, esimerkiksi Solidity:llä*. Lohkoketjuun tallennettuja ja suoritettavia ohjelmia kutsutaan älysopimuksiksi (Smart Contract) [12]. Älysopimuksella on

*https://docs.soliditylang.org

osoite ja rajapintoja joita osallistujat kutsuvat transaktiossa edistääkseen älysopimuksen laskentaa, sekä pysyvää muistia, mikä on käytettävissä eri transaktioiden välillä. Transaktion lähettäjä maksaa laskennan edistämisestä transaktiokuluna lohkoketjun virtuaalivaluuttaa laskennan määrän mukaan. Yhden transaktion aikana suoritettavalle laskennan määrälle on myös yläraja. Kirjoittamishetkellä Ethereum-ketjussa sopimuksena määritellyn valuutan, ERC-20 tokenin, siirtäminen osoitteesta toiseen vaati kuluina noin 5.28$ edestä Ethereum-valuuttaa [13]. Erityisen kallista on sopimuksen pysyvään muistiin tallentaminen.

Tutkielmassa *Lohkoketjusovelluksella* tarkoitetaan sovellusta, jonka ydinlogiikan suoritus tapahtuu lohkoketjussa. Käyttäjät voivat suorittaa ohjelman vaatimaa laskentaa lohkoketjun älysopimuksissa (On-chain) lähettämällä transaktoita, tai lohkoketjun ulkopuolella (Off-chain) omalla laitteellaan. Off-chain laskenta on käyttäjälle transaktiokulujen takia moninkertaisesti halvempaa kuin on-chain laskenta. Sovelluksen käyttäjien välinen viestintä tapahtuu lohkoketjun kautta. Osallistujat eivät siis lähetä suoraan toisilleen viestejä.

Satunnaisuutta tuottavaan protokollaan, joka suorittaa laskentaa lohkoketjussa, kohdistuu vaatimus mahdollisimman pienistä kommunikointikustannuksista On-chain laskennan vaatimien transaktiokulujen myötä. Älysopimuksia tukevissa lohkoketjuissa on viime vuosina tapahtunut merkittävää kehitystä ja jo nyt toiset lohkoketjut mahdollistavat Ethereum-ketjua moninkertaisesti pienemmät kulut. Esimerkiksi kirjoitushetkellä Algorand-lohkoketjun transaktiokulu oli 0.00091$ arvosta Algorand-virtuaalivaluuttaa [1]. Älysopimuslaskennan suuren taloudellisen kustannuksen ongelma ei ole kuitenkaan täysin poistunut erityisesti sovelluksissa, joissa käyttäjien on lähetettävä tuhansia transaktioita tai suoritettava pitkäkestoista laskentaa.

2.4 Lohkoketjusovelluksen käyttäjät

Satunnaislukuja tarvitsevaan lohkoketjusovellukseen oletetaan osallistuvan N käyttäjää, $N > 1$, joista jokaisella on käytössään lohkoketjussa käytetty julkisen avaimen salauksen avainpari, jonka mahdollistamilla allekirjoituksilla käyttäjä osoittaa identiteettinsä. Julkista avainta voidaan kutsua tutkielmassa lohkoketjuympäristössä myös nimellä *tili* tai *osoite*. Täten esimerkiksi sillä, että käyttäjä lähettää transaktion osoitteestaan tarkoitetaan, että käyttäjä lähettää transaktion allekirjoittaen sen yksityisellä avaimellaan, jota vastaa muille näkyvä julkinen osoite. Yksittäinen käyttäjä voi generoida itselleen haluamansa määrän osoitteita ja näyttäytyä täten muille osallistujille monina erillisinä käyttä-

jinä.

Lohkoketjusovelluksen lupavapauden myötä sovellusta voi käyttää myös hyökkääjät, jotka haluavat hyväksikäyttää sovelluksen haavoittuvaisuuksia. Hyökkääjän laskentatehon oletetaan olevan korkeintaan polynomisesti suurempi kuin rehellisten käyttäjien. Hyökkääjä ei siis kykene murtamaan vahvoiksi oletettuja kryptograafisia työkaluja, kuten esimerkiksi julkisen avaimen salausta. Hyökkääjä ei myöskään kykene estämään, viivyttämään tai väärentämään toisten käyttäjien lähettämiä transaktioita. Käyttäjät voivat myös kohdata teknisiä ongelmia, jotka estävät käyttäjää lähettämästä transaktioita. Rehellinen osallistuja lähettää aina protokollan vaatimia transaktioita.

Lohkoketjussa osallistujat jakavat lohkoketjun muodostaman ajan. Lohkoketju pyrkii tuottamaan uusia lohkoja keskimäärin vakioajassa. Järjestelmän aikayksikkönä voidaan käyttää lohkon numeroa. Yhteisen ajan myötä lohkoketjusovellukseen on mahdollista määritellä aikaraja esimerkiksi satunnaisen syötteen syöttämiselle.

2.5 Satunnaisuuden vaatimukset

Älysopimus voi saada satunnaisuutensa kolmannelta osapuolelta tai sovellus voi toteuttaa satunnaisuutta tuottavan protokollan. Esimerkiksi lottoarvonnan osallistujat voivat tuottaa itse voittajan määräävän satunnaisluvun. Protokolla voidaan toteuttaa myös yksittäisenä sovelluksena, jonka satunnaisuutta muut älysopimukset käyttävät. Tällöin protokolla toimii satunnaisuuden majakkana (Randomness Beacon).

Lohkoketjusovellusta varten satunnaislukuja tuottaviin menetelmiin kohdistetaan seuraavat vaatimukset, jotka menetelmien on vähintään täytettävä, jotta niiden tuottamaa satunnaisuutta voidaan käyttää lohkoketjusovelluksessa turvallisesti. Vaatimukset perustuvat aikaisemman tutkimuksen esittämiin vaatimuksiin [10, 27, 32].

1. Yksittäisen osallistujan ei ole mahdollista estää satunnaisluvun tuottamista.
2. Tuotettavia satunnaislukuja ei voi ennustaa.
3. On julkisesti todennettavissa, että menetelmän tuottamat satunnaisluvut on tuotettu menetelmää noudattaen.
4. Tuotettava satunnaisluku ei ole kenenkään manipuloitavissa.

Tutkielmassa oletetaan, että jokaisella osallistujalla on käytössään satunnaisuuden lähde, jota toiset osallistujat eivät voi ennustaa tai manipuloida. Rehellinen osallistuja käyttää aina tätä lähdettä, kun tältä kysytään satunnaista syötettä. Lähde voi olla esimerkiksi pseudosatunnainen generaattori (Pseudorandom Generator [16]), joka tuottaa tietyllä syötteellä aina saman arvon. Tutkielmassa oletetaan, että tämä lähde näyttää muille osallistujille täysin satunnaiselta. Tutkielma ei siis keskity satunnaislukugeneraattoreiden toimintaan, vaan menetelmiin joilla olemassaolevaa satunnaisuutta saadaan turvallisesti lohkoketjusovellukseen.

Vaatimusten lisäksi seuraavilla menetelmien ominaisuuksilla on merkitystä menetelmän käytettävyyden ja käytännön toteutuksen kannalta:

1. Mikä määrä t sovelluksen käyttäjistä $t \leq N$ voi olla haitallisia, niin että käytetty menetelmä kuitenkin tuottaa vaatimusten 1, 2, 3 ja 4 mukaisen luvun?
2. Mikä on yksittäisen käyttäjän kommunikointikompleksisuus? Esim $O(N)$.
3. Kestääkö menetelmä Sybil-hyökkäyksiä?

Kommunikointikompleksisuudella tarkoitetaan tutkielman tapauksessa sitä kuinka monta transaktiota yksittäinen osallistuja joutuu lähettämään osallistuakseen yhden satunnaisluvun tuottamiseen lohkoketjusovellukselle.

Sybil-hyökkäyksellä [11] tarkoitetaan hyökkäystä, jossa yksi käyttäjä esiintyy järjestelmässä monella identiteetillä. Yhden käyttäjän luomat identiteetit näyttäytyvät muille käyttäjille erillisinä käyttäjinä. Lohkoketjusovellus on lupavapaa ja käyttäjän identifioi vain osoite, joita yksittäinen käyttäjä voi generoida haluamansa määrän. Jos lohkoketjusovellukseen, joka toteuttaa esimerkiksi hajautetun satunnaisuutta tuottavan protokollan, voi liittyä vain transaktiokulun hinnalla, voi sovellus olla altis Sybil-hyökkäykselle. Jos sovellus ei ole altis Sybil-hyökkäykselle sanotaan, että sovellus on Sybil resistentti.

3 Satunnaisuuden tuottaminen

Luvussa käydään läpi eri menetelmiä, joilla lohkoketjusovellus saa käyttöönsä satunnaisuutta. Menetelmä ja sen ominaisuudet kuvataan tiivistetysti. Menetelmistä esitetään käytännön toteutukseen liittyviä huomioita ja hyökkäyksiä, joita menetelmiin voidaan kohdistaa.

3.1 Ulkoiset menetelmät

Lohkoketjusovelluksen ei tarvitse tuottaa itse satunnaisuuttaan. Esitetään kaksi yleistä tapaa, jolla älysopimukset voivat saada ulkoisesti käyttöönsä satunnaisuutta.

Yksinkertaisin lohkoketjusovelluksien käyttämä satunnaisuuden lähde on sovitun lohkon tiiviste. Esimerkiksi lottoarvonta voitaisiin suorittaa niin, että älysopimukseen on määritelty, että voitto jaetaan lohkonumeroltaan tietyn tulevan lohkon tiivisteen perusteella. Tiivistettä käytetään niin PoW- kuin PoS-ketjuissa, vaikkakin PoS-ketjussa tiiviste on helposti validaattorin manipuloitavissa. PoW-ketjun lohkon tiiviste täyttää vaatimukset 1-3, mutta kaivajat voivat vaikuttaa arvoon, joten luku on manipuloitavissa eikä täytä vaatimusta 4. Jos kaivajan saama hyöty epäedullisen tiivisteen omaavan lohkon julkaisematta jättämisestä on suurempi kuin lohkon julkaisemisesta saatava palkinto, niin kaivajalla on intressi jättää lohko julkaisematta. Tämä aiheuttaa vinouman satunnaislukujen jakaumaan kaivajalle edulliseen suuntaan. Esimerkiksi Bitcoin-ketjun tapauksessa hyökkääjä ei tarvitse yhden lohkon tiivisteen jakauman merkittävään manipuloimiseen enemmistöä tai edes merkittävää taloudellista panostusta suhteessa ketjun kokoon [25].

Toinen suosittu ulkoinen lähde on Chainlink VRF [9], joka käyttää satunnaisuuden tuottamiseen *Todennettavaa Satunnaisfunktiota (Verifiable Random Function, VRF)*. Määritellään VRF tutkielmalle riittävällä abstraktiotasolla. "VRF on pohjimmiltaan julkisen avaimen versio avainta käyttävästä kryptografisesta tiivisteestä"[23]. Vain salaisen avaimen omistaja voi laskea tiivisteen h syötteellä x, mutta kuka tahansa vastaavan julkisen avaimen omaava voi todentaa, että tiiviste on laskettu käyttäen salaista avainta.

$$h = H_{SK}(x) \tag{3.1}$$

Salaista avainta käyttäen tuotetaan myös todiste, jolla oikeellisuus voidaan todentaa kuitenkaan paljastamatta itse avainta.

Chainlink VRF:ää hyödyntävä sovellus käyttää Chainlink verkon satunnaislukuoraakkelia, joka tarjoaa sovellukselle vaatimukset 1-4 täyttävän satunnaisluvun käyttäen todennettavaa satunnaisfunktiota. Älysopimuksen pitää pyytää satunnaislukua, minkä jälkeen Chainlink verkon solmu lähettää älysopimukseen satunnaisen arvon sisältävän transaktion. Tämän satunnaisluvun saaminen kuitenkin maksaa Chainlink virtuaalivaluuttaa, eikä ulkopuolisesta palvelusta maksaminen ole välttämättä käytännöllistä sovellukselle.

Lohkoketjusovellus voi vaatia käytettävän menetelmää, jossa satunnaisluvun tuottaminen onnistuu itse sovelluksen käyttäjiltä. Seuraavaksi esitetään protokollia, joissa sovelluksen käyttäjät vastaavat satunnaisuuden tuottamisesta. Aloitetaan yksinkertaisella protokollalla, joka ei täytä vaatimuksia, mutta jonka pohjalle moni protokolla rakentuu.

3.2 Kommitointi ja paljastaminen

Kommitoinnin ja paljastamisen protokollassa satunnaisluku tuotetaan kahdessa osassa. Ensimmäisellä kierroksella jokainen osallistuja kommitoi ensin kaikkien nähtäväksi lohkoketjuun satunnaisluvun tiivisteen. Toisella kierroksella osallistujat paljastavat yksi kerrallaan kommitoidunsa satunnaisluvun, jonka tiiviste vastaa ensin kommitoitua tiivistettä. Paljastetuista luvuista lasketaan XOR tai tiiviste, minkä seurauksena saadaan satunnaisluku. Olettaen, että vähintään yksi osallistuja on rehellinen, tämä protokolla täyttää vaatimukset 2 ja 3. Ongelmana protokollassa on se, että viimeinen paljastaja näkee tuotettavan satunnaisluvun ja voi täten olla julkaisematta syötettään jos tuotettu luku ei ole suosiollinen. Tällöin hyökkääjä voi estää satunnaisuuden tuottamisen ja pakottaa protokollan aloittamaan alusta manipuloiden satunnaisuuden jakaumaa. Vaatimuksilla 1 ja 4 on siis yhteys, sillä pakottamalla protokollan aloittamisen alusta voi manipuloida protokollan tuottamia lukuja.

3.3 Todennettava Viivefunktio

Todennettava Viivefunktio (Verifiable Delay Function, VDF) on funktio, jonka evaluointi vaatii määritellyn määrän peräkkäisiä laskutoimituksia, mutta on tehokkaasti julkisesti vahvistettavissa, että tietty syöte tuottaa tietyn tuloksen [7]. VDF:n evaluointi ei nopeudu

rinnakkaistamalla evaluointia, ja se mahdollistaa teoreettisen alarajan asettamisen sille, kuinka kauan evaluoinnissa kuluu aikaa. Lohkon tiivistettä sekä osallistujien kommitointia käyttävien protokollien heikkouksia voidaan parantaa käyttämällä VDF:ää [7].

Tiivisteen käyttämisen tapauksessa älysopimuksessa määritetään, että satunnaisuuden lähteenä käytetään tietyn tulevaisuuden lohkon tiivistettä. Tiivistettä ei käytetä suoraan, vaan satunnainen arvo saadaan syöttämällä tiiviste VDF:lle. VDF:n evaluointi suoritetaan off-chain laskentana ja tulos julkaistaan lohkoketjuun. Kun VDF:n aiheuttama viive valitaan sopivasti, ei lohkon kaivaja ehdi simuloimaan satunnaista arvoa ja tekemään päätöstä lohkon julkaisemisesta ennen kuin rehelliset kaivajat ovat lisänneet uusia lohkoja lohkoketjuun.

Käytettäessä tiivistettä VDF:n kanssa on kommunikointikustannus vakio. Satunnaisluvun saamiseksi sopimuksen kanssa pitää vuorovaikuttaa vain kaksi kertaa. Kerran tulevan lohkon numeron päättämiseksi ja toisen kerran VDF:stä saadun tuloksen julkaisemiseksi ketjuun. Satunnaisluku on satunnainen riippumatta osallistujista ja luvun tuottaminen on julkisesti vahvistettavissa VDF:n ominaisuuksien mukaisesti. Tiiviste VDF:n kanssa on myös Sybil resistentti, sillä satunnainen luku ei riipu osallistujien syötteestä ja sen julkaisemiseksi tarvitaan vain yksi rehellinen osallistuja.

Parannellussa kommitoinnin protokollassa kaikki osallistujat syöttävät sopimukseen heti satunnaisen arvonsa ennen sovittua aikarajaa. Arvojen XOR tai tiiviste syötetään VDF:lle. Kuka tahansa osallistujista voi julkaista VDF:n tuottaman satunnaisen arvon. VDF:n viive aiheuttaa sen, että hyökkääjä ei voi simuloida satunnaista arvoa ennen syötteen julkaisemisen aikarajaa. Käyttäjän syötteen julkaisematta jättäminen ennen aikarajaa ei siis ole manipulointia, sillä oletuksen mukaan käyttäjä ei ole voinut tietää syöttämänsä arvon vaikutusta lopulliseen satunnaiseen arvoon. Protokollaan voidaan kuitenkin lisätä vaatimus vakuustalletuksesta, jonka käyttäjä menettää jos tämä ei julkaise syötettään ennen aikarajaa.

Protokollan kommunikointikustannus on myös vakio. Protokolla vaatii osallistujilta yhden transaktion satunnaisuuden kommitointia varten ja yhdeltä osallistujalta transaktion VDF:n tuottaman arvon julkaisemiseksi. Arvo on satunnainen eikä sen tuottamista voi estää kunhan vähintään yksi osallistuja on rehellinen. Protokolla on myös Sybil resistentti sillä yhden rehellisen osallistujan kommitointi riittää tekemään arvosta satunnaisen, jolloin kommitoimalla monilla eri identiteeteillä ei ole vaikutusta satunnaiseen arvoon.

VDF:n käytännön toteutusta vaikeuttaa käyttäjien laitteistoerot. VDF:n viive pitää valita oikein, jotta tehokaskaan hyökkääjä ei saa evaluoitua funktiota liian nopeasti, mutta

samalla pienemmän laskentatehon omaavien käyttäjien pitäisi pystyä evaluoimaan funktio käytännöllisessä ajassa. Hyökkääjä voi saada VDF:n evaluointiin etua laitteistotason optimoinnilla.

3.4 Julkinen todennettava salaisuuksien jakaminen

Salaisuuksien jakaminen (Secret Sharing, SS) on tekniikka, jolla salaisuuden s jakaja voi jakaa n osallistujalle salaisuuden niin, että mikä tahansa t osallistujan joukko voi rekonstruktuoida salaisuuden kun toisaalta mikä tahansa joukko kooltaan pienempi kuin t ei saa minkäänlaista tietoa salaisuudesta [28]. *Julkisesti todennettavassa salaisuuden jakamisessa (PVSS)* on julkisesti todennettavissa, että jakaja on jakanut validit salaisuuden osat [30]. Todentamisen voi suorittaa myös ulkopuoliset, joille ei ole jaettu salaisuuksien osia. Todennettavassa salaisuuden jakamisessa (VSS) osien todentamisen voi suorittaa vain salaisuuden osan vastaanottaja.

Yksi tapa estää kommitoinnin ja paljastamisen protokollan viimeisen paljastajan manipulointi on käyttää VSS:ää, mikä poistaa hyökkääjältä mahdollisuuden olla julkaisematta syötettään. Esimerkkinä VSS käytöstä kuvataan tiivistetysti RandShare-protokolla, joka täyttää kriteerit 1-4 kun hyökkääjiä on enintään t, mutta esimerkiksi kokonaiskommunikaatiokompleksisuus on $O(n^3)$ [32]. Yksittäisen osallistujan kommunikaatiokompleksisuus on $O(n^2)$. Lohkoketjusovelluksen tapauksessa protokollan on hyvä käyttää PVSS metodia, jotta kuka tahansa voi varmentaa protokollan oikeellisuuden. RandShare toimiikin vain yksinkertaisena esimerkkinä VSS:n käytöstä.

Protokollassa tehdään oletus, että $N = 3t + 1$ osallistujasta enintään t osallistujaa on epärehellisiä. Jokainen osallistuja jakaa ensin oman salaisen syötteensä N osaan VSS metodia käyttäen ja lähettää jokaiselle osallistujalle yhden osan. Protokollassa on määritelty, että salaisuuden rekonstruktuointi vaatii $t + 1$ osaa. Kun kaikilla osallistujilla on muiden osallistujien osat ja ne on vahvistettu, ilmoitetaan valideista osista muille osallistujille. Validien salaisuuksien osat julkaistaan kaikille osallistujille vasta kun on tiedossa, että valideja salaisuuksia on vähintään $t + 1$. Osallistuja voi rekonstruktuoida toisen osallistujan salaisuuden jos hän on vastaanottanut vähintään t osaa muilta osallistujilta. Siis epärehellinen osallistuja ei voi estää salaisuutensa julkaisemista, sillä oletuksen mukaan rehellisiä osallistujia on $2t+1$, mikä riittää siihen, että rehellisten osallistujien julkaistessa osansa, on rehellisillä osallistujilla vähintään $t + 1$ osaa jokaisesta salaisuudesta ja kaikki salaisuudet julkaistaan.

PVSS:ään pohjautuvia protokollia satunnaisuuden tuottamiseen on tutkittu laajasti viime vuosina [4, 5, 27, 32] . Usein protokollia ei ole tarkoitettu toteutetuksi lohkoketjusovelluksessa, vaan tutkimusta motivoi konsensusmekanismien johtajan valinnan tai julkisen satunnaisuuden majakan kehittäminen. Viimeisin protokolla OptRand saavuttaa $O(N^2)$ pahimman tapauksen kokonaiskommunikointikompleksisuuden sietäen enintään $N/2$ hyökkääjää [4]. Yksittäisen käyttäjän kommunikointikompleksisuus on $O(N)$.

Vaikka kommunikointikompleksisuutta onkin saatu madallettua, niin $O(N)$ kompleksisuus ei ole vielä riittävän pieni, että PVSS-protokollaa voitaisiin käyttää lohkoketjusovelluksessa, jossa satunnaisuuden tuottamiseen osallistuu tuhansia osallistujia, sillä yksittäisen osallistujan transaktiokulut kasvavat liian suuriksi. Salaisien osien lisäksi yksittäisen käyttäjän pitää julkaista lohkoketjuun osien todisteita, jotta julkinen varmentaminen on mahdollista. Tämä lisää kommunikaatiokompleksisuuteen vakion, jolla on vaikutusta käytännön toteutuksen kannalta. Osien jakaminen vaatii tallennustilaa lohkoketjusta, mikä on erityisen kallista käyttäjälle. Jos PVSS-protokollana toteutettuun lohkoketjusovellukseen osallistuminen on ilmaista, niin protokolla on altis Sybil-hyökkäykselle. Yksi hyökkääjä voi tällöin esiintyä helposti yli t eri käyttäjänä.

3.5 Merlin-ketju

Merlin-ketju (Merlin chain) on sekvenssi $V_1, V_2..., V_n$, missä arvo V_x on arvon V_{x+1} tiiviste, jota voidaan käyttää lohkon tiivisteen kanssa vaatimukset 1-4 täyttävän protokollan rakentamiseen [20]. Protokolla estää kaivajien sekä osallistujien yritykset vaikuttaa tuotettavaan satunnaislukuun ja se pyrkii toimimaan jatkuvana satunnaislukujen lähteenä, majakkana. Viidessä seuraavassa kappaleessa kuvataan tiivistetysti protokollan toiminta.

Ennen osallistumista jokainen osallistuja tuottaa yksityisen satunnaisen arvon V_n, tallentaa tämän ja generoi arvosta Merlin-ketjun. n valitaan niin, että Merlin-ketjussa on niin monta arvoa kuin osallistuja ikinä tarvitsee protokollan suorittamiseen tulevaisuudessa. Osallistuja paljastaa ketjun arvoja alkaen arvosta V_1. Paljastettuaan arvon V_x osallistujan on pakko paljastaa seuraavaksi arvo V_{x+1} tai löytää törmäys tiivisteelle V_x, minkä oletetaan olevan laskennallisesti mahdotonta. Merlin-ketju pakottaa osallistujan käyttämään ketjun ennalta määräämiä, mutta muille osallistujille satunnaisia, arvoja. Täten osallistuja ei voi reagoida muiden osallistujien arvoihin vaihtamalla omaa arvoaan.

Olkoon seuraavana tuotettava satunnainen arvo R_x. Edellinen arvo on R_{x-1} ja edellisen arvon lohko $B(R_{x-1})$. Yksinkertaisessa mallissa on yksi osallistuja, tuottaja, joka syöttää

älysopimukselle Merlin-ketjun arvoja yhdessä aikaleiman kanssa. Arvo V_x hyväksytään sen täyttäessä tietyt ehdot. Arvon on oltava validi Merlin ketjun arvo, eli $H(V_x) = V_{x-1}$. Nykyisen lohkon lohkonumeron on oltava niin paljon suurempi kuin lohkon, jossa edellinen satunnaisluku tuotettiin, että voidaan olla varmoja edellisen lohkon muuttumattomuudesta. Sopimus tuottaa satunnaisen arvon laskemalla tiivisteen osallistujan arvosta, sekä edellisen satunnaisluvun lohkon seuraajan tiivisteestä, $R_x = H(\, V_x || H(B(R_{x-1}) + 1)\,)$. Satunnaisen arvon yhteydessä sopimus tuottaa myös aikaleiman, joka on aika, jolloin tuottaja pystyi ensimmäisen kerran simuloimaan tuotettavan arvon.

Protokolla estää kaivajia manipuloimasta satunnaista arvoa, sillä kaivajilla ei ole tiedossa arvo V_x lohkon kaivamisen hetkellä, eikä arvoa siten voi simuloida. Protokolla estää tuottajan tekemät manipuloimisyritykset käyttämällä Merlin-ketjua ja täten pakottamalla tuottajan syöttämään ennaltamäärättyjä arvoja. Vaatimukset 1-4 täyttyvät, kunhan tuottaja ei paljasta Merlin-ketjua etukäteen kaivajille [20].

Yhdistämällä monen tuottajan arvot ehkäistään tuottajien ja kaivajien yhdessä tapahtuvan manipuloinnin uhka [20]. Tällöin R_x koostuu N tuottajan satunnaisesta arvosta. Tuottajat suorittavat yhden tuottajan protokollan kerran ja jäävät odottamaan kunnes kaikki tuottajat ovat tuottaneet satunnaisen arvon. Tuottajien satunnaisista arvoista otetaan XOR, mikä muodostaa protokollan lopullisen satunnaisen arvon.

Protokolla täyttää vaatimukset, kunhan vähintään yksi tuottaja toimii rehellisesti ja kaikki tuottajat paljastavat arvonsa. Ongelmana on se mitä tehdään tuottajille, jotka eivät paljasta Merlin-ketjun arvojaan. Paljastamatta jättäminen lukitsee koko protokollan suorituksen ja ongelma on ratkaistava, jotta protokolla täyttää vaatimuksen 1. Protokollan kehittäjät ehdottavat, että arvonsa salaavat tuottajat voi poistaa tuottajien joukosta ja tuottajan Merlin-ketjun voisi palauttaa ratkaisemalla vähintään tietyn ajan vaativan ongelman (Timelock-puzzle) [20].

Jos tuottaja vain poistetaan ja sivuutetaan, ilman tämän ketjun palauttamista, kohtaa protokolla ongelman, sillä lohkoketjuympäristössä hyökkääjä voi luoda uusia tilejä uusilla Merlin-ketjuilla, joilla osallistua tuottajina. Hyökkääjä lopettaa arvojen julkaisemisen aina kohdatessaan epäsuotuisen arvon ja pystyy täten vääristämään lopullisen satunnaisen arvon jakaumaa.

Toisin sanoen, osallistumisen ollessa ilmaista protokolla on altis Sybil-hyökkäykselle, jossa hyökkääjä voi manipuloida satunnaista arvoa, mutta ei kuitenkaan päättää arvoa kun protokollaa suorittaa vähintään yksi rehellinen osallistuja.

Sybil resistenssiä voisi parantaa vaatimalla osallistujilta vakuus. Jos tuottaja ei paljasta arvoaan tietyssä ajassa, niin hän menettää vakuutensa. Uusien tuottajien luominen olisi tällöin taloudellisesti kallista. Tuottajien historiaa voidaan myös seurata ja luoda avoin tilastointi tuottajan maineelle. Maineikkaita tuottajia palkittaisiin protokollan mukaisesta toiminnasta. Mitä muuttumattomampi ja maineikkaampi tuottajien joukko on, sitä turvallisemmin protokolla toimii. Tuottajien muuttumaton joukko ei kuitenkaan sovellu lohkoketjusovelluksiin, joissa käyttäjien joukko muuttuu jatkuvasti.

Merlin-protokollan yksittäisen osallistujan kommunikointikustannus on vakio. Käyttäjän pitää lähettää yhden satunnaisluvun tuottamiseksi vain yksi transaktio, jossa tämä paljastaa Merlin-ketjun arvonsa.

Caucus on toinen Merlin-ketjuja käyttävä protokolla [2]. Merlin-ketjun sijasta kirjoittajat käyttävät nimeä *tiivisteketju*. Caucus yrittää ratkaista konsensusmekanismin johtajan valinnan ongelman tuottamalla satunnaisuutta, jonka mukaan johtaja valitaan. Satunnaisuuden tuottaminen tapahtuu kierroksissa. Valittu johtaja vastaa seuraavan satunnaisluvun tuottamisesta paljastamalla järjestysluvultaan kierrosnumeroa vastaavan tiivisteketjunsa arvon. Protokolla vaatii satunnaisen arvon konfigurointia varten ja kirjoittajat esittävät, että tähän voidaan käyttää esimerkiksi PVSS-protokollaa. Caucus on sovellettavissa PoS-lohkoketjun konsensusmekanismiin, mutta sitä voi käyttää myös lohkoketjusovelluksessa. Kirjoittajat esittävät älysopimustoteutuksen, joka kykeni kirjoitushetkellä 2018 tuottamaan yhden satunnaisluvun 0.1 dollarin kustannuksella. Kommunikointikompleksisuus konfiguroinnin jälkeen on Merlin-ketju-protokollan tapaan O(1), mutta Caucus protokollan arvot ovat manipuloitavissa. Valittu johtaja voi vääristää satunnaisuuden jakaumaa olemalla julkaisematta epäsuotuista tiivisteketjun arvoaan, jolloin protokollan aikakatkaisu siirtyy seuraavaan kierrokseen ja uuden johtajan valintaan. Caucus jättää tarkoituksella kyseisen ongelman ratkaisun tulevalle tutkimukselle, mutta kirjoittavat esittävät, että manipuloinnin estämiseen voitaisiin käyttää protokollaan osallistumiseen vaadittavaa vakuustalletusta [2].

Caucus-protokollan manipulointia voisi ehkäistä käyttämällä VDF:ää. Johtaja julkaisisi tiivisteketjun arvonsa lohkoketjuun ja lopullinen satunnainen arvo saataisiin VDF:stä. Varsinaisen tuloksen voisi julkaista kuka tahansa. Tällöin valittu johtaja ei voisi simuloida satunnaista arvoa, eikä täten tehdä päätöstä julkaisemisesta. VDF:n viiveen myötä kahden satunnaisluvun tuottamisen välinen aika kuitenkin kasvaisi.

Caucus-protokollassa on pienempi riski protokollan pysähtymiselle verrattuna Merlin-protokollaan siksi, että Merlin-protokollan tapauksessa satunnaisluvun tuottamiseksi pitää odottaa

kaikkia osallistujia, kun taas Caucus-protokollassa tuotettavana oleva satunnaisuus riippuu yhdestä osallistujasta, valitusta johtajasta. Merlin-protokolla on toteutettu älysopimuksena [20], kuten myös Caucus protokolla [2], mutta kirjoitushetkellä onnistuin löytämään vain Caucus-protokollan toteutuksen. Saatavilla oleva valmis toteutus helpottaa merkittävästi protokollan käyttöönottoa lohkoketjusovelluksessa.

3.6 Homomorfinen salaus

Homomorfinen Salaus (Homomorphic Encryption, HE) mahdollistaa tiettyjen laskutoimitusten suorittamisen salatulla datalla ilman, että salausta puretaan [34]. Laskutoimitusten tulos vastaa samaa kuin, että laskutoimitukset olisi tehty salaamattomalla datalla. Täysin Homomorfisessa Salauksessa (Fully Homomorphic Encryption, FHE) salatulla datalle voidaan suorittaa mitä tahansa laskentaa. Osittain Homomorfinen Salaus (Partially Homomorphic Encryption, PHE) mahdollistaa vain tiettyjen laskutoimitusten suorittamisen salatulle datalle.

HERB protokolla käyttää satunnaisuuden tuottamiseen Osittain Homomorfista Salausta ja se tuottaa oletuksiensa rajoissa vaatimukset 1-4 täyttäviä satunnaislukuja [10]. HERB olettaa, että osallistujia on $N = 3t + 1$, joista hyökkääjiä on enintään t. HERB konfiguroidaan hajautetulla avainten generoinnilla (Distributed Key Generation, DKG [24]), jossa osallistujat generoivat asymmetrisen salauksen avainparin, jonka salainen avain on jaettu N osallistujalle ja salaisen avaimen käyttämiseen salauksen purkamiseksi vaaditaan $t + 1$ avaimen osaa. Osallistujat kommitoivat satunnaisuutta salaten sen käyttäen jaettua salaista avainta vastaavaa yhteistä julkista avainta. Vähintään $t + 1$ salattua kommitointia lasketaan yhteen, minkä jälkeen $t + 1$ osallistujaa purkaa yhteenlaskettujen kommitointien salauksen. Purettu yhteenlaskettu kommitointi toimii satunnaisena arvona.

DKG protokollasta on olemassaoleva älysopimustoteutus, jossa yksittäisen käyttäjän kommunikaatiokompleksisuus on $O(n)$ [26]. Konfiguroinnin jälkeen yksittäisen HERB osallistujan tarvitsee älysopimustoteutuksessa lähettää satunnaisluvun tuottamiseksi kaksi transaktiota [10]. Itse toteutus ei ole kuitenkaan triviaali sillä se vaatii kryptografista On- ja Off-chain laskentaa. HERB on myös altis Sybil-hyökkäyksille, sillä jos protokollaan rekisteröinti on lähes kulutonta, voi hyökkääjä osallistua protokollaan monella tilillä ja täten saada helposti yli t osaa yksityisestä avaimesta. Protokollaan ei voi myöskään liittyä, eikä poistua, vapaasti kesken satunnaislukujen tuottamisen, vaan uusien osallistujien pitää suorittaa DKG konfigurointi uudestaan.

HERB protokollasta nostetaan esiin toisista protokollista poikkeava ominaisuus. Salaisen avaimen osien ryhmä voi olla erillinen satunnaisuutta kommitoivien ryhmästä [10]. Tämä on mahdollista, sillä satunnaisuus salataan käyttäen julkista avainta. Jos kuitenkin käytännön toteutuksessa kommitointi on ilmaista, kaikille avointa ja kommitoidut salaisuudet puretaan kun niitä on saapunut esimerkiksi vähintään $t+1$, niin yksittäinen hyökkääjä voi pahimmassa tapauksessa vastata kaikista kommitoinneista ja täten päättää satunnaisen arvon. Tämä onnistuu niin, että hyökkääjä lähettää heti kommitointi-ikkunan avauduttua eri tileillä $t+1$ kommitointia, niin suurilla transaktiokuluilla, että vain hyökkääjän kommitoinnit menevät läpi. Hyökkäys olisi taloudellisesti kallis, mutta se voisi olla kannattava palkinnon ollessa riittävän suuri.

4 Analyysi

Luvussa vertaillaan tutkielmassa esiteltyjä menetelmiä ottaen huomioon niiden soveltuvuus erilaisiin lohkoketjusovelluksiin. Satunnaisuuden tuottamiseksi lohkoketjusovellukseen on monta keinoa, kuten myös lohkoketjusovelluksilla on monia eriäviä vaatimuksia. Lohkoketjusovellusten erilaiset vaatimukset tekevätkin satunnaisuutta tuottavien menetelmien vertailusta haastavaa eikä käsitellyistä menetelmistä löydy kaikkiin tilanteisiin sopivaa tapaa.

Taulukkoon 4.1 on koottu käsiteltyjä menetelmiä, joilla lohkoketjusovellus saa hyödynnettäväkseen satunnaisuutta. Taulukossa on menetelmän nimi, tekstimuotoisia huomioita, arvio menetelmän konfiguroinnin ja yhden satunnaisluvun tuottamisen kommunikaatiokompleksisuudesta, määrä joka käyttäjistä voi olla hyökkääjiä niin että menetelmä täyttää vaatimukset 1-4, sekä tieto siitä onko menetelmä lupavapaassa lohkoketjuympäristössä Sybil resistentti. Ei Sybil resistentit menetelmät vaativat käytännön toteutuksessaan mekanismin Sybil-hyökkäyksen torjumiseen, jotta toteutus täyttää vaatimukset 1-4.

Toteutuksen kannalta yksinkertaisin menetelmä on käyttää pelkkää lohkon tiivistettä. EVM-ympäristössä sen käyttö tapahtuu yhdellä Solidity-kielen funkiokutsulla [29]. Vaikka toteutus ei ole turvallinen, niin monelle lohkoketjusovellukselle se voi olla riittävän turvallinen. Menetelmä voi sopia esimerkiksi pienille sovelluksille, jotka eivät käsittele taloudellista arvoa, sekä sovelluksille joissa käyttäjä joutuu suorittamaan jonkinlaisen identifioinnin tai käyttäjien joukko on suljettu ja ennalta tiedossa.

Chainlink VRF, Homomorfinen salaus, Caucus ja Merlin-ketju ovat menetelmiä, joista on älysopimustoteutus, mutta Merlin-ketjun älysopimustoteutusta ei löytynyt kirjoitushetkellä. Vaikka nämä menetelmät ovatkin selvästi tiivisteen käyttämistä monimutkaisempia, niin ne ovat turvallisempia ja valmiit toteutukset helpottavat niiden käyttöönottoa. Chainlink VRF tarjoaa valmiit älysopimusrajapinnat [9], mutta sovelluksen pitää kehittää malli palvelun rahoittamiseksi jotta se voi tilata satunnaislukuja. VDF:n käytännön toteutuksessa suurin haaste on itse viivefunktion toteuttamisessa niin, että laitteistoerojen merkitys saadaan mahdollisimman pieneksi. PVSS menetelmän satunnaisluvun tuottamisesta ei löytynyt valmista lohkoketjusovellukseen sovellettua toteutusta eikä toteutus ole triviaali.

Menetelmästä riippumatta käyttäjän pitää lähettää vähintään yksi transaktio osallistuak-

seen lohkoketjusovellukseen. Chainlink VRF, lohkon tiiviste VDF:llä sekä Caucus ovat menetelmiä, joissa yhden satunnaisluvun tuottamiseksi vaaditaan liittymisen lisäksi vain yksi transaktio keneltä tahansa osallistujalta. Merlin-ketjua hyödyntävä menetelmä vaatii jokaiselta käyttäjältä kommitointitransaktion liittymisen lisäksi, kuten myös kommitointi VDF:n kanssa. Homomorfinen salaus vaatii $t + 1$ käyttäjältä yhden transaktion salauksen purkamiseksi.

Homomorfisessa salauksessa, sekä Caucuksessa konfiguroinnin kompleksisuus on O(N). Vaikka konfiguroinnin kustannus on suuri varsinkin isoihin sovelluksiin, niin menetelmiä voidaan käyttää myös suuremman käyttäjämäärän sovelluksissa sillä konfigurointi tapahtuu vain kerran. PVSS ei skaalaudu suuren käyttäjämäärän sovelluksiin sillä satunnaisluvun tuottamisen kompleksisuus on O(N). Muiden menetelmien konfigurointikustannus on vakio. Pienestä konfigurointikustannuksesta on hyötyä esimerkiksi jos sovellus tarvitsee olemassaolonsa aikana vain yhden satunnaisluvun.

Chainlink VRF ja lohkon tiiviste VDF:n kanssa eivät riipu sovelluksen käyttäjistä, joten ne kestävät N haitallista sovelluksen käyttäjää. Kommitointi VDF:n kanssa, Merlin-ketju ja Caucus tuottavat satunnaisen luvun kunhan vähintään yksi käyttäjä on rehellinen. Caucus tosin vaatii esimerkiksi PVSS konfiguroinnin, mikä kestää parhaillaan $N/2$ hyökkääjää. Homomorfinen salaus joutuu myös tekemään oletuksen maksimissaan $N/3$ hyökkääjästä. Käyttäjistä riippumattomissa menetelmissä sovelluksen käyttäjien joukko voi muuttua tiheästi, kun taas esimerkiksi Merlin-ketjua hyödyntävä protokolla on sitä turvallisempi mitä vähemmän käyttäjien joukko muuttuu. Staattisempaa käyttäjäjoukkoa suosivat protokollat soveltuvat satunnaisuuden majakan toteuttavaan sovellukseen. Majakan tapauksessa ongelmana on kuitenkin se, kuinka käyttäjiä kannustetaan osallistumaan majakan toimintaan, joka aiheuttaa käyttäjälle transaktiokuluja.

Homomorfista salausta ja PVSS:ää käyttävät menetelmät ovat erityisen alttiita Sybil-hyökkäyksille jos sovelluksen käyttäjäksi liittyminen on halpaa suhteessa hyökkäyksellä saavutettaviin voittoihin. Molemmissa hyökkääjä voi saada helposti oletusta suuremman osan jaetuista salaisuuksista, minkä seurauksena hyökkääjä voi ennustaa satunnaisen arvon. Menetelmien Sybil resistenssiä pitää parantaa jos niitä halutaan käyttää lohkoketjusovelluksessa, mikä lisää käytännön toteutuksen monimutkaisuutta. Merlin-protokolla on altis lievemmälle Sybil-hyökkäykselle, jolla voi manipuloida satunnaisuutta. Chainlink VRF:n ja VDF:n eri käyttötapojen tapauksessa riskiä Sybil-hyökkäykselle ei ole, mutta Chainlink VRF:n tapauksessa sovellus luottaa Chainlink verkkoon.

Caucus protokollan käyttämä vakuustalletus on yksi tapa parantaa Sybil resistenssiä. Va-

kuustalletuksen käyttäminen ei kuitenkaan tee protokollasta täysin Sybil resistenttiä ja talletuksen koon määrittäminen on vaikeaa. Mitä suurempi talletus on, sitä korkeamman taloudellisen riskin hyökkäyksiä sovellus kestää, mutta korkea vakuus rajaa käyttäjien joukkoa pienemmäksi, sillä kaikilla käyttäjillä ei ole varaa suureen vakuuteen. Tehokas tapa satunnaisuutta tuottavien protokollien ja niitä käyttävien lohkoketjusovellusten Sybil resistenssin lisäämiseen on käyttää rekisteriä, joka liittää osoitteeseen luonnollisen henkilön. Rekisterin rakentaminen ei ole kuitenkaan triviaalia lohkoketjussa. Ethereum ketjussa esimerkkinä toimii hajautettu Proof of Humanity* rekisteri. Lohkoketjusovelluksen Sybil resistenssin parantaminen on aihe, joka vaatii lisää tutkimusta.

Chainlink VRF ja lohkon tiivisteen käyttäminen ilman VDF:ää eivät vaadi Off-chain laskentaa, minkä myötä käytännön toteutus on yksinkertaisempi verraten muihin menetelmiin. Tiivisteketjua hyödyntävissä protokollissa pitää päättää kuinka käyttäjän tiivisteketju varmuuskopioidaan. Tämä vaatii Off-chain toteutuksen jos varmuuskopiointia ei jätetä täysin käyttäjän vastuulle. VDF:n, PVSS:n ja homomorfisen salauksen tapauksessa toteutettavana on kryptografista Off-chain laskentaa. Toteutukset vaativat muun muassa salaisten osien hallinnan ja validointien toteuttamisen käyttäjän laitteella. Käyttäjän on pystyttävä validoimaan protokollan suoritus laitteellaan.

Pelkän lohkon tiivisteen ja Chainlink VRF:n käyttäminen soveltuvat parhaiten reaaliaikaisiin sovelluksiin, joissa satunnaisluku pitää saada nopeasti. Käyttäjien syötteeseen nojaavien protokollien on määriteltävä jokin viive kauan käyttäjien on mahdollista syöttää satunnaisuuttaan. Tämän viiveen määrittely ei ole kuitenkaan helppoa, sillä liian lyhyt viive saattaa rankaista rehellisiä käyttäjiä, joilla on esimerkiksi hidas yhteys. Liian pitkä viive satunnaisuuden tuottamisessa taas ei välttämättä sovellu reaaliaikaiseen sovellukseen. VDF nojaa viiveeseen turvallisen satunnaisuuden takaamiseksi, eikä satunnaislukua saada yhtä nopeasti kuin esimerkiksi lohkon tiivisteen tapauksessa.

4.1 Muu tutkimus ja puutteet

Hajautetun satunnaisuuden tuottamiseen tarkoitettuja protokollia vertaillaan viimeaikaisissa tutkimuksissa [5, 4, 27]. Tutkielmassa ei käyty läpi mainituissa tutkimuksissa esiintyviä menetelmiä, jotka käyttävät BLS [8] allekirjoituksia. Tutkielmassa ei esitelty myöskään peliteoreettisia menetelmiä satunnaisuuden tuottamiseen.

*https://www.proofofhumanity.id/

Taulukko 4.1: Menetelmiä satunnaisuuden saamiseen lohkoketjusovellukseen

Menetelmä	**Huomioitavaa**	**Kustannus**	**Hyökkääjiä**	**Sybil resistentti**
Chainlink VRF	Maksaa Chainlink virtuaalivaluuttaa. Valmis älysopimus rajapinta.	$O(1)$ Gen.	N	Kyllä
Lohkon tiiviste + VDF	VDF turvallisen viiveen määrittely. Kaksi transaktiota, lohkonumeron päättäminen ja VDF:n tuloste. Yksinkertaisin On-chain toteutus.	$O(1)$ Konf. $O(1)$ Gen.	N	Kyllä
Kommitointi + VDF	VDF turvallisen viiveen määrittely. Vaatii kaikkien kommitoinnin.	$O(1)$ Konf. $O(1)$ Gen.	$N-1$	Kyllä
PVSS	Korkea kommunikointikustannus. Monimutkainen toteutus.	$O(N)$	$N/2 - N/3$	Ei
Merlin-ketju	Kuinka toimia tuottajien kanssa, jotka eivät paljasta Merlin-ketjun arvoaan? Merlin-ketjun varmuuskopiointi.	$O(1)$ Konf. $O(1)$ Gen.	$N-1$	Ei, vain manipuloitavissa
Caucus	Samat kuin Merlin-ketjun kanssa. Olemassaoleva älysopimustoteutus. Pienempi pysähtymisriski verrattuna Merlin-ketjuun.	$O(N)$ Konf. $O(1)$ Gen.	$N/2 - N/3$ Konf. $N-1$ Jälkeen	Vakuustalletus
Homomorfinen salaus	Vaativa off-chain laskentaan nojaava toteutus. Osittain valmis toteutus saatavilla.	$O(N)$ Konf. $O(1)$ Gen.	$N/3$	Ei

5 Yhteenveto

Tutkielmassa esiteltiin ja vertailtiin menetelmiä, joilla lohkoketjusovellus voi saada käyttöönsä satunnaisuutta. Satunnaisuutta tuottavan menetelmän satunnaisuuden pitää olla julkisesti todennettavaa, manipuloimatonta ja ennustamatonta. Satunnaisuuden hyödyntäminen ei ole triviaalia julkisessa, deterministisessä ja lupavapaassa lohkoketjuympäristössä. Menetelmien monimutkaisuus vaihtelee aina yksinkertaisista tiivistefunktioita käyttävistä, monimutkaisempia hajautettuja kryptografisia metodeja käyttäviin menetelmiin. Metodien käytöllä pyritään pakottamaan käyttäjät toimimaan sovitulla tavalla, tai vähentämään hyökkääjien mahdollisuuksia. Sovellus voi saada satunnaisuuden ulkopuolelta tai sovelluksen käyttäjät voivat tuottaa itse satunnaisuutta. Tutkielman pääpaino oli sovelluksen käyttäjien itse tuottamassa satunnaisuudessa. Tutkielma toimii myös yleiskatsauksena hajautetun satunnaisuuden tuottamisen ongelmaan.

Eri menetelmien tekemät oletukset, ominaisuudet ja heikkoudet vaihtelevat, minkä myötä menetelmien soveltuvuus eri sovelluksiin vaihtelee. Vaihtelevuuden myötä tutkielmassa ei pyritty löytämään yhtä ongelmaa ratkaisevaa menetelmää vaan keskityttiin vertailemaan menetelmien ominaisuuksia ja heikkouksia niin, että vertailusta olisi apua uutta satunnaisuutta vaativaa lohkoketjusovellusta kehittäessä. Menetelmien analysoinnissa käytettiin mittareina yhdelle käyttäjälle koituvia kommunikointikustannuksia sekä kuinka monta epärehellistä käyttäjää menetelmä kestää. Analysoinnissa nostettiin myös huomioita menetelmien käytännön toteutuksiin liittyen.

Tutkielmassa käytiin läpi hyökkäyksiä, joita menetelmiä kohtaan voidaan kohdistaa. Osasta satunnaisuutta tuottavista menetelmistä huomattiin, että ne tarvitsevat tavan menetelmän Sybil resistenssin parantamiseksi jotta protokollan voi toteuttaa turvallisesti. Sybil resistenssin parantaminen lohkoketjusovelluksessa nostettiin yhdeksi jatkotutkimuksen aiheeksi.

Lähteet

[1] *Algorand (ALGO) Blockchain Explorer*. URL: https://algoexplorer.io/ (viitattu 26. 03. 2022).

[2] S. Azouvi, P. McCorry ja S. Meiklejohn. "Winning the caucus race: Continuous leader election via public randomness". *CoRR* abs/1801.07965 (2018). URL: http://arxiv.org/abs/1801.07965.

[3] M. Belotti, N. Božić, G. Pujolle ja S. Secci. "A vademecum on blockchain technologies: When, which, and how". *IEEE Communications Surveys Tutorials* 21.4 (2019), s. 3796–3838. DOI: 10.1109/COMST.2019.2928178.

[4] A. Bhat, A. Kate, K. Nayak ja N. Shrestha. "OptRand: Optimistically responsive distributed random beacons". *Cryptology ePrint Archive* (2022).

[5] A. Bhat, N. Shrestha, Z. Luo, A. Kate ja K. Nayak. "Randpiper–reconfiguration-friendly random beacons with quadratic communication". Teoksessa: *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*. 2021, s. 3502–3524.

[6] M. Blum. "Coin flipping by telephone a protocol for solving impossible problems". *SIGACT News* 15.1 (tammikuu 1983), s. 23–27. ISSN: 0163-5700. DOI: 10.1145/1008908.1008911. URL: https://doi.org/10.1145/1008908.1008911.

[7] D. Boneh, J. Bonneau, B. Bünz ja B. Fisch. "Verifiable delay functions". English (US). Teoksessa: *Advances in Cryptology – CRYPTO 2018 - 38th Annual International Cryptology Conference, 2018, Proceedings*. Toim. A. Boldyreva ja H. Shacham. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag, 2018, s. 757–788. ISBN: 978-3-319-96883-4. DOI: 10.1007/978-3-319-96884-1_25.

[8] D. Boneh, B. Lynn ja H. Shacham. "Short signatures from the Weil pairing". Teoksessa: *International conference on the theory and application of cryptology and information security*. 2001, s. 514–532.

[9] *Chainlink Documentation*. URL: https://docs.chain.link/ (viitattu 29. 04. 2022).

[10] A. Cherniaeva, I. Shirobokov ja O. Shlomovits. "Homomorphic encryption random beacon". *Cryptology ePrint Archive* (2019).

[11] J. R. Douceur. ”The sybil attack”. Teoksessa: *International workshop on peer-to-peer systems*. Springer. 2002, s. 251–260.

[12] *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. 2014. URL: https://ethereum.org/en/whitepaper/ (viitattu 26. 03. 2022).

[13] etherscan.io. *Ethereum Gas Tracker | Etherscan*. en. URL: http://etherscan.io/gastracker (viitattu 22. 03. 2022).

[14] M. J. Fischer, N. A. Lynch ja M. S. Paterson. ”Impossibility of distributed consensus with one faulty process”. en. *Journal of the ACM* 32.2 (huhtikuu 1985), s. 374–382. ISSN: 0004-5411, 1557-735X. DOI: 10.1145/3149.214121. URL: https://dl.acm.org/doi/10.1145/3149.214121 (viitattu 17. 03. 2022).

[15] Y. Gilad, R. Hemo, S. Micali, G. Vlachos ja N. Zeldovich. ”Algorand: Scaling Byzantine Agreements for Cryptocurrencies”. Teoksessa: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP ’17. New York, NY, USA: Association for Computing Machinery, 2017, s. 51–68. ISBN: 978-1-4503-5085-3. DOI: 10.1145/3132747.3132757. URL: https://doi.org/10.1145/3132747.3132757.

[16] O. Goldreich. ”Computational complexity : a conceptual perspective”. Teoksessa: *Computational complexity : a conceptual perspective*. Cambridge ; Cambridge University Press, 2008, s. 283–348. ISBN: 0-511-40077-2.

[17] T. Hanke, M. Movahedi ja D. Williams. ”DFINITY Technology Overview Series, Consensus System” (toukokuu 2018). URL: http://arxiv.org/abs/1805.04548 (viitattu 18. 03. 2022).

[18] S. Al-Kuwari, J. H. Davenport ja R. J. Bradford. *Cryptographic hash functions: Recent design trends and security notions*. Cryptology ePrint Archive, Report 2011/565. 2011.

[19] C. Lesaege, W. George ja F. Ast. ”Kleros: Research Challenges in Decentralized Justice” (maaliskuu 2020).

[20] P. Mell, J. Kelsey ja J. Shook. ”Cryptocurrency smart contracts for distributed consensus of public randomness”. Teoksessa: Proceedings of the 19th International Symposium on Stabilization, Safety, ja Security of Distributed Systems, Boston, MA, 2017. DOI: https://doi.org/10.1007/978-3-319-69084-1_31.

[21] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. URL: http://bitcoin.org/bitcoin.pdf.

[22] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen ja E. Dutkiewicz. "Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities". *IEEE access : practical innovations, open solutions* 7 (2019), s. 85727–85745. ISSN: 2169-3536.

[23] D. Papadopoulos, D. Wessels, S. Huque, M. Naor, J. Včelák, L. Reyzin ja S. Goldberg. "Making NSEC5 practical for DNSSEC". *Cryptology ePrint Archive* (2017).

[24] T. P. Pedersen. "A threshold cryptosystem without a trusted party". Teoksessa: *Workshop on the theory and application of of cryptographic techniques*. Springer. 1991, s. 522–526.

[25] C. Pierrot ja B. Wesolowski. "Malleability of the blockchain's entropy". en. *Cryptography and Communications* 10.1 (tammikuu 2018), s. 211–233. ISSN: 1936-2455. DOI: 10.1007/s12095-017-0264-3. URL: https://doi.org/10.1007/s12095-017-0264-3 (viitattu 24. 03. 2022).

[26] P. Schindler, A. Judmayer, N. Stifter ja E. Weippl. "Ethdkg: Distributed key generation with ethereum smart contracts". *Cryptology ePrint Archive* (2019).

[27] P. Schindler, A. Judmayer, N. Stifter ja E. Weippl. "HydRand: Efficient Continuous Distributed Randomness". Teoksessa: *2020 IEEE Symposium on Security and Privacy (SP)*. ISSN: 2375-1207. Toukokuu 2020, s. 73–89. DOI: 10.1109/SP40000.2020.00003.

[28] A. Shamir. "How to Share a Secret". *Commun. ACM* 22.11 (marraskuu 1979). Place: New York, NY, USA Publisher: Association for Computing Machinery, s. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[29] *Solidity documentation*. URL: https://docs.soliditylang.org (viitattu 26. 03. 2022).

[30] M. Stadler. "Publicly verifiable secret sharing". Teoksessa: *Advances in cryptology — EUROCRYPT '96*. Lecture notes in computer science. ISSN: 0302-9743. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, s. 190–199. ISBN: 3-540-61186-X.

[31] *State of the DApps — DApp Statistics*. en-US. URL: https://www.stateofthedapps.com/stats (viitattu 03. 05. 2022).

[32] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer ja B. Ford. "Scalable Bias-Resistant Distributed Randomness". Teoksessa: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, s. 444–460. DOI: 10.1109/SP.2017.45.

[33] G. Wood. ”Ethereum: A secure decentralised generalised transaction ledger” (). URL: https://ethereum.github.io/yellowpaper/paper.pdf (viitattu 08. 05. 2022).

[34] X. Yi. *Homomorphic encryption and applications*. SpringerBriefs in computer science. Cham: Springer, 2014. ISBN: 3-319-12229-0.