Teemu Pöytäniemi

Suurten tiedostojen siirto ja talletus eri pilvialustoilla

Diplomityö

Tieto- ja säjkötekniikan tiedekunta

Mika Välimäki

TIIVISTELMÄ

Teemu Pöytäniemi: Suurten tiedostojen siirto ja talletus eri pilvialustoilla

Diplomityö

Tampereen yliopisto

Tietotekniikan tutkinto-ohjelma

2021

Tiivistelmä on suppea, 1 sivun mittainen itsenäinen esitys työstä: mikä oli ongelma, mitä tehtiin ja mitä saatiin tulokseksi. Kuvia, kaavioita ja taulukoita ei käytetä tiivistelmässä.

Laita työn pääkielellä kirjoitettu tiivistelmä ensin ja käännös sen jälkeen. Suomenkieliselle kandidaatintyölle pitää olla myös englanninkielinen nimi arkistointia varten.

Tässä pohjassa tiivistelmää varten 2 omaa tekstityyppiä: tunnistetiedoille tyyli CoverBodyText2 ja tiivistelmätekstille Abstract, jossa riviväli on 1.0. Otsikkotyyppi on Heading (no number), joka tekee automaattisesti sivunvaihdon (Page break before). Samaa otsikkotyyppiä käytetään mm. sisällysluettelossa. Lähdeluettelossa on identtinen tyyppi hieman eri nimellä, jolloin se voidaan poimia sisällysluetteloon. Sivunumeroja varten etusivun lopussa pitää olla Section Break ja tiivistelmän yläotsakkeen (header) asetus Link to Previous pois päältä, ja lisäksi sivunumeron muotoilusta Start at i (eikä Continue).

Avainsanat: Tiivistelmä-tekstin jälkeen.

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

abstract

:

Tampere University

The abstract is a concise 1-page description of the work: what was the problem, what was done, and what are the results. Do not include charts or tables in the abstract.

Put the abstract in the primary language of your thesis first and then the translation (when that is needed).

This document template has two text styles for abstract. BibInfo is for bibliographical information above whereas the rest uses the style Abstract, which has line spacing of 1.0. The style Heading (no number) is used in the frontmatter before actual text and it makes the necessary preceding page break. Similar style is used in the bibliography with slightly different name in order to include it in the table of contents. The title page must end with Section Break to get pages numbered correctly. Moreover, the header on this page turns off the setting Link to Previous and formats the page numbers to Start at 1 (instead of Continue).

Keywords: After Abstract-text

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

ALKUSANAT

Tämä dokumenttipohja on laadittu Tampereen yliopiston tekniikan alan opinnäytetöitä varten. Mallipohja perustuu aikaisemmalla Tampereen teknillisen yliopiston pohjalle, mutta se on päivitetty vuonna 2019 toimintansa aloittavaa Tampereen yliopistoa varten.

Alkusanoissa esitetään opinnäytetyön tekemiseen liittyvät yleiset tiedot. Tapana on myös esittää kiitokset työn tekemiseen vaikuttaneille henkilöille ja yhteisöille. Alkusanat eivät kuulu arvioinnin piriin, mutta niissä ei silti ole sopivaa moittia tai kritisoida ketään. Alkusanojen pituus on enintään 1 sivu. Alkusanojen lopussa on päivämäärä, jonka jälkeen työhön ei ole enää tehty korjauksia.

Tampereella, 15.2.2019

Päivittäjä

Sisällysluettelo

[1. Johdanto 1](#_Toc73893651)

[2. Teoria 3](#_Toc73893652)

[2.1 Tiedostojen siirto 3](#_Toc73893653)

[2.1.1 HTTP ja WebSocket 3](#_Toc73893654)

[2.1.2 FTP ja FTPS 11](#_Toc73893655)

[2.1.3 SFTP 13](#_Toc73893656)

[2.2 Tiedostojen talletus 15](#_Toc73893657)

[2.2.1 SQL tietokanat 15](#_Toc73893658)

[2.2.2 Object Storage 16](#_Toc73893659)

[2.2.3 Block storage 19](#_Toc73893660)

[2.2.4 File storage 20](#_Toc73893661)

[3. esimerkki projektit **Error! Bookmark not defined.**](#_Toc73893662)

[3.1 Tiedostojen siirto 31](#_Toc73893663)

[3.2 Tiedostojen talletus 33](#_Toc73893664)

[4. Projektien tulokset 34](#_Toc73893665)

[4.1 kustannusvertailu 34](#_Toc73893666)

[4.2 tehokkuusvertailu 42](#_Toc73893667)

[5. yhteenveto 51](#_Toc73893668)

[Lähteet 53](#_Toc73893669)

KUVALUETTELO

[**Kuva 1.** Kuvaaja on hyvä muokata julkaisukelpoiseksi. Vasemmalla on esitetty muokkaamaton kuvaaja ja oikealla muokattu. **Error! Bookmark not defined.**](#_Toc532040187)

[**Kuva 2.** Tyylit kirjoitusohjeen versioissa 11.3 ja 11.5 55](#_Toc532040188)

Tämä luettelo on vapaaehtoinen. Kuvaluettelo lisätään *References > insert Table of Figures* ja sieltä *Options… > Build table of figures based on > Style:Figure Caption*. Myös taulukkoluettelon saa samasta kohdasta, kun valitsee viimeisestä kohdasta tyylin *Table Caption*.

LYHENTEET JA MERKINNÄT

CC-lisenssi Creative Commons -lisenssi

LaTeX ladontajärjestelmä tieteelliseen kirjoittamiseen

SI-järjestelmä ransk. Système international d’unités, kansainvälinen mittayksikköjärjestelmä

TTY Tampereen teknillinen yliopisto

URL engl. Uniform Resource Locator, verkkosivun osoite

*a* kiihtyvyys

**F**voima

*m* massa

.

# Johdanto

Moderni pilvilaskentainfraktuuri sai alkunsa 1990-luvulla, kun ensimmäiset yritykset alkoivat tarjota virtuaalikoneita ja virtuuliverkkoja SaaS(software-as-a-service) palveluna internetissä. Vuoteen 2006 mennessä luotiin Amazon Web Services (AWS) ja julkaistiin Elastic Compute Cloud (EC2) -palvelu. Palvelun avulla asiakkaat voivat vuokrata virtuaalikoneita heidän sovellusten infrastruktuuriksi. Samana vuonna Google julkaisi Google Docsin, jota käytetään asiakirjojen luomiseen, muokkaamiseen ja jakamiseen pilvessä.[ https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0178-3]

Vaikka pilvipalveluiden kasvu alussa on ollut hidasta, viimeisten 10 vuoden aikana pilvipalvelut ovat laajentuneet merkittävästi. Vuoteen 2010 mennessä Amazon, Google, Microsoft ja OpenStack olivat kaikki käynnistäneet pilviosastot. Tämä auttoi saamaan pilvipalvelut yleisesti saataville. Sen jälkeen pilvipalvelut ovat vallanneet suuren osan teknologiateollisuudesta ja pilvisiirrot tai -migraatiot ovat yleistyneet. [https://www.dataversity.net/how-the-cloud-has-evolved-over-the-past-10-years/]

Pilvilaskenta-alustat voidaan jakaa kolmeen ryhmään: julkiseen, yksityiseen ja hybridiin. Julkiset alustat ovat yleisin pilvipalveluiden alusta. Pilviresurssit (kuten palvelimet ja tallennustila) ovat kolmannen osapuolen pilvipalveluntarjoajan omistamia ja ylläpitämiä, ja ne toimitetaan Internetin kautta. Julkisessa pilvessä kaikki laitteistot, ohjelmistot ja muut tukevat infrastruktuurit ovat pilvipalveluntarjoajan omistuksessa ja hallinnassa. Microsoft Azure ja AWS ovat esimerkkejä julkisesta pilvestä. Julkisessa pilvessä jaat samat laitteistot, tallennustilat ja verkkolaitteet muiden organisaatioiden tai käyttäjien kanssa ja käytät palveluita ja hallitset tiliäsi verkkoselaimella. Yksityinen pilvi koostuu pilvipalveluista, joita käyttää yksinomaan yksi yritys tai organisaatio. Yksityinen pilvi voi sijaita fyysisesti omassa palvelinkeskuksessa tai sitä voi isännöidä kolmannen osapuolen palveluntarjoaja. Yksityisessä pilvessä palvelut ja infrastruktuuri ylläpidetään aina yksityisessä verkossa ja laitteistot ja ohjelmistot on omistettu vain yhdelle organisaatiolle. Yksityinen pilvi voi helpottaa organisaation mukauttaa resurssejaan vastaamaan tiettyjä IT-vaatimuksia. Hybridipilvi on alusta, joka yhdistää paikallisen infrastruktuurin tai yksityisen pilvialustan julkiseen pilvialustaan. Hybridialustat mahdollistavat tietojen ja sovellusten liikkumisen kahden ympäristön välillä. Monet organisaatiot valitsevat hybridipilvilähestymistavan liiketoiminnan tarpeiden vuoksi, kuten säännösten ja tietojen riippumattomuusvaatimusten täyttämisen, paikan päällä tapahtuvan teknologiainvestoinnin täyden hyödyn hyödyntämisen tai alhaisen viiveen ongelmien ratkaisemisen vuoksi.

# Teoria

## Tiedostojen siirto

Tässä luvussa käsitellään protokollia, joiden avulla tiedostoja voidaan siirtää pilvialustoille. Ensimmäisessä aliluvussa käsitellään selainten mahdollistamia lataustapoja ja sen jälkeen siirrytään protokolliin, joita erilliset asiakasohjelmat tarvitsevat.

### HTTP

HTTP (Hypertext Transfer Protocol) on internetin taustalla oleva protokolla. Tim Berners-Leen ja hänen tiiminsä vuosina 1989–1991 kehittämä HTTP on käynyt läpi monia muutoksia, jotka ovat auttaneet säilyttämään sen yksinkertaisuuden ja muokkaamaan sen joustavuutta. HTTP kehittyi protokollasta, joka oli suunniteltu vaihtamaan tiedostoja puoliluotettavassa laboratorioympäristössä, nykyaikaiseksi Internet-labyrintiksi, joka kuljettaa kuvia ja videoita korkearesoluutioisina ja 3D:nä.

HTTP on lyhenne sanoista Hypertext Transfer Protocol, joka on sovellusprotokolla, jota on käytetty viestintään World Wide Webissä vuodesta 1989 lähtien. HTTP on menetelmä, jota tietokoneet ja palvelimet käyttävät Internetissä tietojen pyytämiseen ja lähettämiseen. Pohjimmiltaan HTTP on se, mikä mahdollistaa palvelinten ja tietokoneiden kommunikoinnin keskenään.

Tim Berners-Leen kehittämä HTTP pian Internetin käyttöönoton jälkeen kehitettiin alun perin hyvin yksinkertaisessa ympäristössä. Ensimmäinen dokumentoitu HTTP-versio koostui vain yhdestä koodirivistä, joka sisälsi GET-menetelmän ja pyydetyn asiakirjan polun. Vastaus tuotti yhden hypertekstiasiakirjan pyydettyyn polkuun

HTTP on lyhenne sanoista Hypertext Transfer Protocol, joka on protokolla tiedonsiirtoon World Wide Webissä. Se on tietoliikenteen perusta World Wide Webissä, ja sitä käytetään verkkosivujen, kuvien, videoiden ja muiden resurssien siirtämiseen verkkopalvelimien ja verkkoasiakkaiden välillä.

HTTP on asiakas-palvelin-protokolla, mikä tarkoittaa, että se sisältää yhteyden asiakkaan (kuten verkkoselain tai mobiilisovellus) ja palvelimen välillä. Asiakas lähettääHTTP-pyynnön palvelimelle, joka sitten lähettää HTTP-vastauksen takaisin asiakkaalle. Vastaus sisältää pyydetyt tiedot tai resurssit, kuten verkkosivun, kuvan tai videon.

HTTP on tilaton protokolla, mikä tarkoittaa, että jokainen pyyntö ja vastaus ovat riippumattomia eivätkä ole riippuvaisia ​​aiemmista viesteistä. Tämä tekee protokollasta yksinkertaisemman ja skaalautuvamman, mutta se voi myös johtaa suorituskykyongelmiin.

HTTP-protokollasta on julkaistu useita versioita: HTTP/0.9, HTTP/1.0, HTTP/1.1, HTTP/2 ja HTTP/3. HTTP -protokolla sai alkunsa 1980-luvun lopulla ja ensimmäinen internet palvelin kehitettiin CERNissä 1990. Vuonna 1991 ensimmäinen virallinen dokumentoitu HTTP-versio julkaistiin. Versio sai nimen HTTP/0.9 ja se tukee vain GET-pyyntöä, jonka avulla asiakkaat voivat noutaa HTML-asiakirjoja palvelimelta, mutta ei tue muita tiedostomuotoja. HTTP 1.0 julkaistiin vuonna 1996 ja se käyttää yksinkertaista pyyntö-vastausmallia, jossa asiakas lähettää pyynnön palvelimelle ja palvelin lähettää vastauksen takaisin asiakkaalle. Protokolla tukee GET, POST ja HEAD -pyyntöjä ja protokollaan lisättiin otsikot sekä paluu koodit. Otsikoiden avulla protokollalla pystyy siirtämään muita tiedostotyyppejä kuin HTML -dokumentteja. Pyyntö- ja vastausviestit lähetetään verkon yli pelkkänä tekstinä, jossa otsikko ja teksti erotetaan tyhjällä rivillä. Jokaiselle pyynnölle luodaan oma TCP-yhteys ja vastauksen saapuessa TCP yhteys katkaistaan. HTTP/1.1 julkaistiin vuonna 1997 ja siitä julkaistiin päivitetty versio vuonna 1999. HTTP 1.1 tukee pysyviä yhteyksiä, jotka mahdollistavat useiden pyyntöjen ja vastausten lähettämisen peräkkäin saman TCP-yhteyden kautta. Tämä vähentää uusien yhteyksien luomista jokaista pyyntöä varten ja parantaa protokollan suorituskykyä. HTTP/1.1 yksi uusi ominaisuus on HTTP-putki, jonka avulla voidaan lähettää useita HTTP-pyyntöjä yhden TCP-yhteyden kautta odottamatta pyyntöjen vastauksia. Pyyntöjen ketjuttaminen johtaa huomattavaan parannukseen HTML-sivujen latausajoissa, erityisesti korkean viiveen yhteyksissä. Palvelimen on kuitenkin lähetettävä vastauksensa samassa järjestyksessä kuin pyynnöt vastaanotettiin. HTTP/1.1 suurin suorituskykyongelma on HOL- esto (head-of-line). Se syntyy kun samaan kohteeseen on menossa useampi pyyntö, mutta jonon kärjessä oleva pyyntö, joka ei voi noutaa vaadittua resurssiaan, estää kaikki sen takana olevat pyynnöt. HOL- esto tapahtuu myös HTTP-putkessa, koska putkessa lähetettyihin pyyntöihin pitää vastata samassa järjestyksessä. Erillisten rinnakkaisten TCP-yhteyksien lisääminen voi helpottaa ongelmaa, mutta asiakkaan ja palvelimen välisten samanaikaisten TCP-yhteyksien määrä on rajoitettu, ja jokainen uusi yhteys vaatii huomattavia resursseja.

HTTP/2 on päivitetty versio HTTP/1.1-protokollasta, joka otettiin käyttöön vuonna 2015. Se on suunniteltu korjaamaan joitakin HTTP/1.1:n rajoituksia ja suorituskykyongelmia sekä parantamaan verkkoviestinnän nopeutta ja tehokkuutta. Yksi merkittävimmistä HTTP/1.1:n ja HTTP/2:n erottavista ominaisuuksista on binäärikehystyskerros. HTTP/1.1 pitää kaikki pyynnöt ja vastaukset pelkästään tekstimuodossa. HTTP/2 käyttää binaarikehystyskerrosta kaikkien viestien kapseloimiseen binäärimuotoon säilyttäen silti HTTP-semantiikan. Viestien muuntaminen binäärimuotoon sallii HTTP/2:n käyttää uusia siirtotapoja tiedon toimittamiseen, joita ei ole saatavilla HTTP/1.1:ssä. HTTP/2:ssa binäärinen kehystyskerros koodaa pyynnöt ja vastaukset ja leikkaa ne pienemmiksi tietopaketeiksi, mikä lisää huomattavasti tiedonsiirron nopeutta. Toisin kuin HTTP/1.1, jonka on käytettävä useita TCP-yhteyksiä HOL-eston vaikutuksen vähentämiseksi, HTTP/2 muodostaa yhden yhteyden kahden palvelimen välille. Tässä yhteydessä on useita tietovirtoja. Jokainen tietovirta koostuu useista viesteistä pyyntö/vastaus-muodossa. jokainen näistä viesteistä jakaantuu vielä pienempiin yksiköihin, joita kutsutaan kehyksiksi. Yhteys koostuu joukosta binäärikoodattuja kehyksiä, joista jokainen on merkitty tiettyyn tietovirtaan. Tietovirran tunnisteet sallivat yhteyden lomittaa kehykset siirron aikana ja koota ne uudelleen toisessa päässä. Lomitetut pyynnöt ja vastaukset voivat toimia rinnakkain estämättä niiden takana olevia viestejä. Tätä prosessia kutsutaan multipleksaukseksi ja se ratkaisee HTTP/1.1:n HOL- eston. Yksi mahdollinen ongelma on, että useat tietovirrat ovat riippuvaisia tietyn tietovirran valmistumisesta ja joutuvat odottamaan sen valmistumista. Ratkaisuna ongelmaan on tietovirtojen priorisointi, jonka avulla tietovirralle voidaan määrittää prioriteetti ja riippuvuussuhde muihin tietovirtoihin. Palvelin käyttää näitä tietoja riippuvuuspuun luomiseen, jonka avulla palvelin voi määrittää järjestyksen, jossa tietovirtojen kehykset lähetetään.

<https://www.javacodegeeks.com/2023/03/http-1-1-vs-http-2-vs-http-3-key-differences.html> <https://ably.com/topic/http-2-vs-http-3>

<https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference>

<https://www.wallarm.com/what/what-is-http-2-and-how-is-it-different-from-http-1>

HTTP:n uusin versio HTTP/3 on Googlen kehittämä ja se julkaistiin vuonna 2015. Suurin muutos versioon HTTP/2 nähden on taustalla olevan siirtokerroksen verkkoprotokollan TCP:n vaihtaminen QUIC:ksi (Quick UDP Internet Connections). QUIC on Googlen vuonna 2012 julkaisema protokolla, joka käyttää siirtokerroksessa UDP:ta TCP:n sijaan. QUIC on multipleksoitu viestintäalgoritmi, joka kehitettiin paremmaksi algoritmiksi kuin HTTP/2:n käyttämä multipleksi. Vaikka HTTP/2 poistaa HOL- eston HTTP-protokollasta, niin TCP-protokollalla on oma HOL- esto. Jos TCP hukkaa paketin yhdessä HTTP/2 luomassa tietovirrassa, kaikkien muiden tietovirtojen kehysten siirtäminen samassa TCP-yhteydessä estyy kunnes hukatut paketit ovat saatu uudelleen lähetettyä. QUIC-tietovirran data jaetaan pienempiin osiin, joita kutsutaan kehyksiksi. Kehykset lähetetään UDP-paketteina ja ne sisältävät tietovirran tunnuksen ja järjestysnumeron. Vastaanottaja voi järjestää ne uudelleen, jos ne vastaanotetaan eri järjestyksessä mitä lähetettiin. Jos paketti katoaa, QUIC lähettää sen uudelleen aikakatkaisun jälkeen. Logiikka on sama mitä TCP tekee HTTP/2, mutta se ei vaikuta yhteyden muihin tietovirtoihin. <https://www.rfc-editor.org/rfc/rfc9114.txt> https://http.dev/3

### WebDav

WebDAV tulee sanoista Web Distributed Authoring and Versioning, joka on HTTP:n laajennus. WebDAV-protokolla esiteltiin alun perin vuonna 1996 ja se standardointiin ensimmäistä kertaa vuonna 1999. Protokollan nykyinen versio on vuodelta 2007. WEBDAv mahdollistaa HTTP- palvelimen toimimisen tiedostopalvelimena. WebDAV laajentaa HTTP-otsikoiden ja -menetelmien standardijoukkoa, jonka avulla voit luoda, siirtää ja muokata tiedostoja sekä poistaa tai kopioida tiedostoja ja kansioita. WebDAV- palvelimet mahdollistavat tiedostojen versioiden seurannan ja palvelimet on jaettu kahteen luokkaan versioiden seurannan perusteella: luokka 1 ja luokka 2. Luokan 1 WebDAV-palvelimet tarjoavat perushallintaominaisuuksia, kuten mahdollisuuden luoda, kopioida, siirtää tai poistaa tiedostoja ja kansioita. Monet asiakkaat pitävät luokan 1 WebDAV-palvelimia vain luku -muotoisina, koska ne eivät voi suojata tiedostoja samanaikaisilta muutoksilta. Luokan 2 WebDAV-palvelimet voivat lukita tiedostoja ja sallivat tiedostojen samanaikaisen muokkauksen.

WebDav lisää seuraavat HTTP menetelmät HTTP 1.1 standardiin: PROPFIND, PROPPATCH, COPY, MOVE, MKCOL, LOCK ja UNLOCK. WebDav käyttää PUT ja

DELETE menetelmiä, jotka löytyvät jo valmiiksi HTTP-standardista. Protokolla käyttää tiedostoista ja kansioista resurssi nimitystä. PROPFIND menetelmällä pystyy hakemaan tietoja resursseista ja PROPPATCH menetelmä mahdollistaa niiden muokkaamisen. LOCK menetelmä lukitsee halutun resurssin, sillain että toiset käyttäjät eivät pysty tekemään muutoksia siihen. UNLOCK menetelmä purkaa lukituksen. COPY menetelmä kopio resurssin palvelimella ja MOVE menetelmä siirtää sen. resurssin pystyy lataamaan palvilimelle PUT menetelmällä ja poistamaan DELETE menetelmällä ja lataamaan GET menetelmällä.

Kuvassa X on esitetty esimerkkitapaus asiakkaan ja palvelimen välisestä liikenteessä, jossa asiakas lukitsee ensin tiedoston ja muokkaa sitä ja sen jälkeen ja purkaa lukituksen lopuksi.

<http://www.webdav.org/specs/rfc3744.html> <https://www.cloudwards.net/what-is-webdav/> https://easy-software.com/en/glossary/webdav-protocol/

A screenshot of a computer

Description automatically generated

### WebSocket

WebSocket luo kaksisuuntaisen yhteyden asiakkaan ja palvelimen välillen, ja mahdollistaa sen, että palvelin lähettää dataa asiakkaalle ilman erillistä asiakkaan pyyntöä.

Kuvassa 1 on esitelty WebSocketin arkkitehtuuri korkealla tasolla. Aluksi kättelyvaiheeseen käytetään HTTP – protokollaa jonka jälkeen siirrytään WebSocket -protokollaan. Websocket-protokolla muistuttaa TCP-protokollaa, mutta protokollat eroavat kuitenkin toisistaan ja WebSocketilla ei voi muodostaa yhteyttä TCP-serveriin.



Kuva 1 webSocket-protokolla

Asiakas aloittaa kättelyprotokollan lähettämällä HTTP GET pyyynnön palvelimelle. Pyynnössä pitää olla mukana tietyt HTTP-otsikot, jotta palvelin hyväksyy pyynnön. Upgrade-otsikko pitää olla ”websocket” ja Connection-otsikko pitää olla ”Upgrade”. Sec-websocet-version pitää olla 13, joka on nykyinen versio WebSocket standardista. Web-Socket-Protocol otsikko voidaan lisätä ja se kertoo mitä protokollia asiakas tukee. Origin-otsikko lisätään tietoturvan takia ja sen avulla palvelin voi estää luvattomat cross-origin pyynnöt ja se voi hylätä kättely-yritykset listattuihin domaineihin. Pyynnössä on myös mukana Sec-WebSocket-Key otsikko, jonka arvo on satunnainen 16-tavuinen merkkijono, joka on base-64 koodattu. Tämän merkkijonon generointi pitäisi olla kryptografisesti satunnainen ja sen arvoja ei pitäisi pystyä arvaamaan. Jos palvelin hyväksyy kättelyn, niin pyynnön HTTP-vastauskoodi on 101 ja vastauksessa on seuraavat otsikot: Upgrade: ”websocket” ja Connection: ”Upgrade”. Sec-WebSocket-protocol-otsikon arvo on palvelimen valitsema protokolla. Sec-WebSocket-Accept otsikon arvo riippuu pyynnön Sec-WebSocket-Key otsikosta. Palvelin lisää Sec-WebSocket-Key otsikkoon joka kerta vakio GUIDIn 258EAFA5-E914-47DA-95CA-C5AB0DC85B11 ja laskee niiden summasta SHA1-tiivisteen. Tämän tiivisteen palvelin palauttaa Sec-WebSocket-Accept otsikossa. Jos Sec-WebSocket-Accept otsikossa palautuu väärä arvo tai HTTP-pyynnön paluukoodi ei ole 101, asiakas ei avaa yhteyttä.

Kun kättely on onnistuneesti tehty, siirrytään WebSocket-prokollaan. Protokollassa data liikkuu kehyksissä, joiden mallikuva on kuvassa 2. Kehyksessa olevilla numeroilla on seuraavat merkitykset alkaen ensimmäisestä bitistä:

* FIN, 1 bitti: merkkaa onko kehys lähetettävän datan viimeien vai ei. 1 jos on, 0 jos ei ole
* RSV1, RSV2, RSV3 1 bitti kukin. Kaikki ovat nollia jos ei ole lisäosaa kommunikoinnin alussa sovittu.
* Opicode, 4 bittiä. Koodi joka merkkaa kehyksen tyypin
* MASK, 1 bitti. Merkkaa onko siirrettävä data naamioitu vai ei. Jos asiakas lähettää kehyksen tämä bitti täytyy olla 1
* Siirrettävän datan pituus, 7 bittiä. Jos tämä luku on 0-125 se merkkaa siirrettävän datan pituuden. Jos se on puolestaan 126 se tarkoittaa että seuraavat 16 bittiä muostaa luvun, joka on siirrettävän datan pituus. Jos luku on 127 seuraavat 64 bittiä muodostaa luvun, joka on siirrettävän datan pituus. Tämän luvun merkitsevin bitti täytyy olla kuitenkin nolla.
* naamiointiavain, 32 bittiä jos Mask bitti on 1. Tällä avaimella asiakkaan on naamioitava data
* Siirrettävä data, jonka pituus on ilmoitettu siirrettävän datan pituus-arvolla. Se pitää sisällään lisäosa-datan sekä sovellus-datan. Lisäosa-datan pituus on 0 jos lisäosaa ei ole käytössä, muuten lisäosa määrittää tämän datakentän pituuden. Sovellus-datan pituus on siirrettävän datan pituus miinus lisäosa-datan pituus.



Kuva 2: WebSocket kehys

Kun asiakas lähettää kehyksen palvelimelle, MASK bitillä täytyy olla arvo 1, ja 32-bittinen naamiointiavain määriteltynä. Avaimen pitää olla jokaiselle kehykselle eri ja sen generoivan algoritmin kryptografisesti vahva, jotta tulevia arvoa ei voida arvata vanhojen arvojen pohjalta. Datan naamioiminen tapahtuu seuraavalla tavalla 8 bitin lohkoissa: Merkataan lohkon indexiä :llä ja avaimen indeksiä :llä. Indeksit kähtevät nollasta liikkeelle seuraavalla notaatioilla tarkoitetaan 8 bitin lohkoa, joka alkaa indeksin j osoittamasta paikasta. Indeksi kasvaa aina yhdellä siirryttäessä seuraavaan lohkoon ja lohkon aloitusbitti saadaan kertomalla inedeksi kahdeksalla. Avaimen -ideksi saadaan seuraavalla kaavalla

Modulo joudutaan ottamaan kun avaimen pituus on aina 32 bittiä ja datan pituus voi olla pidempi. Siirrettävä naamioitu data saadaan ottamalla XOR operaatio avaimesta ja naamioimattomasta datasta

Datan naamioimisen tarkoitus on tehdä WebSocket-liikenteestä HTTP-liikenteestä poikkeavaa ja arvaamatonta. Muuten verkkoinfrastruktuurilaitteet, joita ei ole päivitetty ymmärtämään WebSocket-liikennettä voivat luulla sitä virheellisesti normaaliksi HTTP-liikenteeksi ja avata tietoturva-aukkoja. Naamioimisella pystytään suojaamaan verkkoinfrastruktuurilaitteet, ja vaikka hyökkääjä pystyisi hallitsemaan asiakaspuolen sekä palvelimen koodia, se ei pysty hallitsemaan liikutettavan kehyksen arvoja, koska ne on naamioimisen ansiosta satunnaisia.

4-bittisellä Opcode-koodilla pystytään merkkaamaan kehyksen tyyppi. Kehys voi olla datan siirtoon tarkoitettu kehys tai hallintakehys. Jos kehys on hallintakehys, sillä on sovittu hallintakoodi Opicode lohkossa ja mahdollisesti siirrettävää dataa. Mahdollisia komentoja on ”close”, ”ping” ja ”pong”. Close-komento aloittaa yhteyden sulkemisprosessin ja se voi sisältää datakentässä viestin, miksi yhteys halutaan sulkea. Kun Close-komento on vastaanotettu, siihen vastataan lähettämällä kehys, jonka komento on myös Close. Close-vastausta voi viivästyttää ja jatkaa datan lähettämistä, mutta silloin ei ole taetta että Close komennon lähettänyt osapuoli pitää yhteyden auki. Ping-komennon tarkoitus on tarkistaa että yhteyden toinen pää ottaa vielä vastaan liikennettä. Kun Ping-komento on vastaanotettu, siihen vastataan lähettämällä Pong-komento. Jos kehys on datan siirtokehys Opcode-koodilla hallitaan siirrettävän datan formaattia. Mahdollisia vaihtoehtoja ovat binääri ja teksti. Tekstin käyttämä merkistö on UTF-8. Opcoden avulla voidaan pilkkoa lähetettävä data useampaan kehykseen, jolloin kaikkien keysten datan formaatti pitää olla sama kuin ensimmäinen kehyksen. Ensimmäinen kehys määrittää Opcodella siirrettävän datan tyypin ja seuraavien kehysten Opcode määrittää että se on jatkoa edelliselle kehykselle. Datan viimeisen kehyksen FIN-bitin arvo on 1, joka kertoo että kyseessä on datan viimeinen kehys. Opicode-koodissa on vapaita komentoja lisäosia varten, joita ei ole määritelty WebSocket-protokollaan. Asiakas voi pyytää lisäosan käyttöä kättelyvaiheessa lisäämällä Sec-WebSocket-Extensions otsikon pyyntöön. Jos serveri hyväksyy lisäosia käyttöön, se lisää saman Sec-WebSocket-Extensions otsikon vastaukseen, johon se listaa hyväksytyt lisäosat.

### FTP ja FTPS

FTP eli File Transfer Protocol on tiedostojen siirtoon käytetty protokolla. Protokolla toimii TCP/IP-yhteyden päällä ja se koostuu asiakkaasta ja palvelimesta. Kuvassa 2 on esitetty FTP protokollan arkkitehtuuri korkealla tasolla. Sekä asiakas että palvelin tarvitsevat ohjelmat, jotka toteuttavat protokollan. Asiakkaan ohjelma voi olla graafinen käyttöliittymä tai se voi toimia komentoriviltä. [2]



Kuva 3 FTP-protokolla

Protokolla käyttää kahta rinnakkaista TCP yhteyttä: hallintayhteyttä ja datayhteyttä. Hallintayhteydellä lähetetään kaikki komennot liittyen käyttäjien hallintaan ja tiedonsiirtoon, ja se on totetutettu Telnet-protokollalla. Telnet on vanha sanomamuotoinen protokolla, joka toimii TCP-prokollan päällä. Telnetillä pystyy muodostamaan yhteyden etätietokoneelle ja sitä on käytetty aikoinaan mm. sähköpostien lähetykseen. Telnet ei salaa sanomiaan, minkä vuoksi protokolla on tietoturvaltaan heikko. Sen käytöstä on luovuttu moderneissa järjestelmissä ja se on korvattu turvallisimmilla protokollilla, kuten SSH. Hallintayhteydellä määritellään datayhteyden ominaisuudet ja sillä avataan haluttu datayhteys. [2]

Datayhteydellä siirretään tiedostot. Yhdellä aktiivisella yhteydellä voidaan siirtää vain yhtä tiedostoa samanaikaisesti. Datayhteys siirtää datan aina 8-bittisinä tavuina joita kutsutaan siirtotavun kooksi. FTP tukee muutamaa eri tiedonsiirtotyyppiä, joina se pystyy esittämään siirrettevän datan. Oletustyyppi on ASCII, jota kaikkien FTP-toteusten pitää tukea. ASCII on tekstin merkistö, jossa jokainen merkki esitetään 7 bittisinä numerona. Tiedosto lähetetään 8-bittisenä NVT ASCII:na jossa eniten merkitsevä bitti on nolla ja sen perässä 7-bittinen ASCII koodi. Vaihtoehtoinen tyyppi lähettää tekstitiedostoja on EBCDIC. Se on 8-bittinen merkistö, joka on pääasiassa käytössä IBM:n suurtietokoneissa. ASCII ja EBCDIC ottavat myös toisen vapaaehtoisen parametrin joka määrittää tekstin pystysuuntaista asettelua ja sitä käytetään tekstin tulostuksen muotoiluun. Oletusarvo Non-print ei ota kantaa muotoiluun. Telnet format control parametriksi annettuna olettaa että tekstistä löytyy vertikaalisia tekstin hallinta komentoja kuten LF ja NL. Carriage control olettaa että tekstistä löytyy ASA (FORTRAN) vertikaali hallinta merkkejä ja hallitsee niiden avulla tekstin muotoilua. Image-tyyppi on binääritiedostojen lähettämiseen tarkoitettu tyyppi. Tiedosto luetaan biteiksi ja bitit lätetään 8-bittisinä tavuina. Lähetyksen loppuun saatetaan tarvita täytetttä jos tiedoston bitit eivät ole kahdeksalla jaollisia. Täyte on pelkästään nollia, ja asiakkaan ja palvelimen täytyy sopia kuinka täyte tunnistetaan oikeasta datasta. Local-tyyppi on image-tyypin laajennos ja sen avulla voidaan muuttaa lähetettävän datan loogisen tavun kokoa annettavalla parametrillä. [2]

FTP tukee kolmea eri tiedostonvälitysrakennetta. Tiedostorakenne, tallennerakenne ja sivurakenne: Tiedostorakenteessa tiedostolla ei oleteta olevan mitään sisäistä rakennetta, vaan tiedosto lähetetään binäärimuodossa. Tallennerakenteessa tiedoston oletetaan koostuvan toistuvista pienemmistä osista. Sivurakenteessa tiedoston oletetaan koostuvan sivuista, jotka ovat indeksoitu. Tätä rakennetta käytetään epäjatkuvien tiedostotyyppien kanssa jotka voivat sisältää useampia sisäisiä tiedostoja jotka tunnistetaan tiedostossa olevalla otsikkodatalla. [2]

Datan siirtämisen aloittaminen edellyttää datayhteyden muodostamista sekä siirtoparametrien sopimisesta. Aluksi asiakkaan pitää avata datayhteyden portti kuuntelemista varten ja lähettää pyyntösanoma hallintayhteydellä. Tämä sanoma pitää sisällään tiedon kumpaan suuntaa dataa ollaan siirtämässä. Palvelin vastaa pyyntöön ottamalla yhteyden asiakkaan datayhteyden porttiin. Seuraavaksi tiedoston siirtotapa pitää sopia. FTP tukee kolmea tapaa: stream, block ja Compressed. Stream lähettää tiedostot jatkuvana virtana. Jos tiedoston tiedostonvälitysrakenne on tiedostorakenne, niin tiedoston päättyminen merkataan sulkemalla datayhteys. Tallennerakenteessa käytetään kaksitavuista hallintakoodia merkkaamaan osien päättymistä tai tiedoston päättymistä. Block-siirrossa data siirretään lohkoina. Kullakin lohkolla on 3-tavuinen otsikko, jossa on tieto lohkon pituudesta sekä lohkon tyypistä. Lohkon tyypillä pystytään merkkaamaan lohkoon tieto siitä, että onko lohkotiedoston viimeinen, tallenteen viimeinen vai uudelleenkäynnistyksen merkki. Compressed-siirto käyttää yksinkertaista pakkausalgoritmia nimeltään run-length encoding joka lähettää datan lohkoissa kuten block-siirrossa. Pakkaamalla data siirrossa saadaan pienennettyä siirrettävän datan määrää, mutta yleensä pakkaus tehdään tehokkaimmilla algoritmeilla jo tiedostoa luodessa tai verkkotasolla modeemeiden toimesta. FTP-protokolla ei sisällä tapaa tarkistaa siirrettävän datan eheyttä, vaan se luottaa TCP-protokollaan. FTP sisältää kuitenkin protokollan tiedoston latauksen jatkamiseen, jos lataus on keskeytynyt. Latauksen jatkamista voi käyttää vain jos siirtotapa on block tai compressed. Latauksen jatkaminen merkataan lohkon otsikkoon sille varatulla koodilla paikassa, josta tiedoston lataus jatkuu. [2]

FTP-protokolla ei salaa kumpaakaan käyttämäänsä yhteyttä joten sitä pidetään nykyään tietoturvattomana. FTPS (FTP Secure) on FTP-protokollan tietoturvallisempi lisäosa, joka käyttää SSL- tai TLS-suojausprotokollaa. Suojauksen päälle kytkeminen voidaan toteuttaa kahdella eri tavalla: implisittisesti ja eksplisiittisesti. Implisiittinen tapa vaatii suojausta jo heti yhteyden luomisvaiheessa. Tämä rikkoo pelkästään FTP-protokollalla toimivien asiakasohjelmien ja palvelimien toimivuuden implisiittisesti suojausta vaativan vastinparinsa kanssa. Eksplitisiittisessä suojauksessa asiakkaan täytyy erikseen pyytää suojattua yhteyttä. Näin palvelin mahdollistaa kommunikoinnin sekä FTP- että FTPS-asiakkaiden kanssa. Eksplisitiittisesti määriteltäessä voidaan myös säätää mitkä yhteydet suojataan, kun implisiittisesti määriteltynä molemmat kanavat ovat suojattuna. Esimerkiksi jos siirrettävä data on jo valmiiksi salattu, pelkästään hallintayhteys voidaan salata. [2]

### SFTP

SFTP-protokolla on tiedostonsiirto- ja tiedostonhallintaprokolla, joka koostuu palvelimesta ja asiakasohjelmasta joka ottaa yhteyden palvelimeen. SFTP-prokolla toimii suojatun turvallisen kanavan avulla ja se suunniteltiin SSH 2-protokollan laajennokseksi. SFTP prokolla ei itsessään suojaa liikennettä, vaan se luottaa alla olevan kanavan suojaukseen. SFTP-yhteyden avulla voidaan siirtää tiedostoja ja tehdä tiedostonhallintaa. Vaikka SFTP-protokolla suunniteltiin SSH 2-prokollan laajennukseksi, sitä voidaan käyttää myös muiden protokollien avulla esimerkiksi SSH-1 protokollan avulla. SSH 1-prokollan käyttö aiheuttaa rajoitteita asiakasohjelmalle, koska sen pitää tietää SFTP-palvelimen binääritiedostojen sijainti palvelimella. SFTP-protokollasta on julkaistu useita versioita, joista käytetyin versio nykyään on versio 3 joka julkastiin vuonna 2001. Vuoden 2001 jälkeen protokollasta on julkaistu versiot 4, 5 ja 6, mutta ne eivät ole laajalti tuettuja ja useat SFTP-palvelimien kehittäjät ovat ilmoittaneet, että heillä ei ole aikomustakaan tukea uusimpia versioita tulevaisuudessa.

SFPT-protokolla lähettää paketit verkon yli muodossa, jossa on alussa tavun kokoinen tila komennolle ja sen jälkeen tulee siirrettävä data. Kuvassa 4 on esitetty SFTP-protokollan arkkitehtuuri, ja siinä näkyy pakettien komentojen nimet. Asikas lähettää aluksi SSH\_FXP\_INIT-komennon jonka datana on haluttu versio protokollasta. Palvelin vastaa SSH\_FXP\_VERSION-komennolla, joka pitää sisällään palvelimen tukemat versiot. Palvelimen ja asiakasohjelman pitää tukea samaa versiota protokollasta, jotta yhteys pystytään muodostamaa. Tiedostojen käsitteleminen tapahtuu niille tarkoitetuilla komennoilla. Esimerkiksi SSH\_FXP\_OPEN komento avaa tiedoston tai luo uuden tyhjän tiedoston. Palvelin palauttaa SSH\_FXP\_OPEN kyselyyn SSH\_FXP\_HANDLE komennon jos SSH\_FXP\_OPEN kysely oli onnistunut. SSH\_FXP\_HANDLE palauttaa datana ID:n jonka avulla tiedosto voidaan siirtää asiakasohjelmalle SSH\_FXF\_READ lukukomennon avulla. Esimerkkejä muista komennoista : SSH\_FXF\_WRITE mahdollistaa tiedostojen kirjoittamisen palvelimelle, SSH\_FXP\_REMOVE mahdollistaa tiedostojen poistamisen palvelimelta. Diagram

Description automatically generated

Kuva 4 SFTP-protokolla

## Tiedostojen talletus

Tässä luvussa käsitellään eri tallennusmuotoja, joita pilvialustat yleisesti tarjoavat.

### SQL-tietokannat

SQL-tietokannat ovat relaatiotietokantoja, joissa tieto on tallennettuna tauluina, jotka koostuvat riveistä ja sarakkeista. Taulut perustuvat relattiomalliin, jonka määrittää predikaattilogikka. Taulu esittää yhden asian tiedot, sarakkeet kuvaavat kokonaisuuden arvoja ja rivit asian ilmentymiä. Tauluissa jokaisella rivillä on uniikki ID, joka pystyy muodostumaan joko yhdestä tai useammasta ssarakkeen arvosta. Taulujen välille pystytään luomaan yhteyksia toisiinsa taulujen avainten avulla.

Tietokannan käytön mahdollistavat tietokannan ohjausjärjestelmät. Ne mahdllistavtavat käyttäjän pääsyn tietokantaan sekä sen päivittämisen ja muokkauksen. Tietokannan ohjausjärjestelmiä on maksullia ja avoimen lähdekoodin järjestelmiä. Suosituimmilla avoimen lähdekoodin ohjausjärjestilmillä on hyvin samllainen pääarkkitehtuuri, mutta toteutuksissa on pieniä eroavaisuuksia. Kuva 5 esittää Postgres tietokannan arkkitehtuuria. Postgres on yksi suosituimmista ilmaisen lähdekoodin tietokannosta ja se on julkaistu vuonna 1995. Postgres toimii asikas-palvelin -mallilla, jossa asikasohjelma ottaa yhteyden ohjausjärjestelmään ja se luo jokaisella asikasyhteydelle oman palvelinprosessin.Kaikilla palveluprosesseilla on pääsy jaettuun muistiin joka on RAM -muistissa. Tietokantakyselyt tehhdään jaettuun muistiin, joka on kopio levyllä olevasta tiedosta. Jaettumuisti on huomattavasti nopeampaa kuin levyltä luku. Lukuoperaatiot kohdisttuvat pelkästään jaettuun muistiin, mutta kirjoitus operaatiot tehdään WAL(Write ahead log) logiin. Ensiksi muutokset tehdään jaettuun muistiin, jonka jälkeen WAL-kirjoittaja – prosessi kirjoittaa muutokset levylle säännöllisenä taustaprosessina. Checkpointer ja writer prosessit päivittävät muutoksetjaetusta muistista levylle sen jälkeen kun muutokset ovat tehty WAL- tiedostoon. Levylle kirjoitus tehdään taustaprosessina pienissä paloissa koska I/O -operaatio on raskas. Jaetusta muistissa pidetään tallessa poistettuja rivejä sekä muuttuneiden rivien vanhoja versioita, koska toiset palveluprosessit saattavat tarvita vanhoja arvoja. Autovacuum – prosessi poistaa vanhentuneet rivit jaetusta muistista. [https://www.postgresql.org/docs/current/routine-vacuuming.html, https://en.wikibooks.org/wiki/PostgreSQL/Architecture] Diagram

Description automatically generated

Kuva 5 Postgresin arkkitehtuuri

### NOSQL

NoSQL tarjoaa mahdollisuuden tallentaa datan tietokantaa muuna rakenteena kuin relaatiomallin tauluina. Ensimmäiset NoSQl tietokannat kehitettiin jo 1960 -luvulla, mutta nimitys NoSQL syntyi 2000 -luvun alussa. NoSQL kannoille ei ole yhtä määritelmää vaan ne jakautuvat useampaan kategoriaan. Yleisinpiin kategorioihin kuuluu mm. erillaiset avain-kuorma- tietokannat, jotka käyttävät hakemisto-tietotyyppiä tiedon tallettamiseen. Data talletetaan tietokantaan uniikin avaimen avulla, johon on linkitetty sitä vastaava kuorma. Toinen yleinen NoSQL tietokanta on dokumentti – tietokanta, joka tallettaa tiedon jossakin standardoitusa tiedostoformaatissa esimerkiksi JSON- tai XML – formaatissa. Dokumentit ovat talletettuna tietokantaan uniikin ID avulla joka tekee dokumentti – tietokannoista avain-kuorma -tietokannan osajoukon.

### Object Storage

Object Storage tarjoaa tallennusmahdollisuuden rakenteettomalle datalle. Tiedostojen tyypillä ja koolla ei ole mitään merkitystä vaan kaikki tiedostot tallennetaan yhteen ämpäriin. Ämpärissä tiedostoilla ei ole minkään sortin kansiorakennetta, vaan jokaisella tiedostolla on ämpärissä uniikki id. Ämpäriin pääsee käsiksi HTTP REST-rajapinnan kautta, joka mahdollistaa tiedostojen hallinnan. Jokainen Object Storagen toteuttava taho voi luoda oman REST-rajapinnan, mutta käytännössä kaikki pilvialusta toteuttavat Amazon AWS S3 ämpärin rajapinnan. S3 oli ensimmäinen Object Storage pilvialustoilla ja sen rajapinta oli hyvin dokumentoitu, jonka seurauksena siitä tuli erittäin suosittu ja muutkin pilvialustat alkoivat tarjoamaan samaa rajapintaa. Object Storage tarjoaa erittäin hyvän skaalautuvuuden, koska ämpärin kokoa voidaan kasvattaa erittäin suureksi, jopa exatavujen suuruiseksi. Object Storage on myös erittäin kustannustehokas ratkaisu, kun vain käytetystä talletustilasta laskutetaan eikä yhtään ylimääräisestä varalla olevasta talletustilasta kuten normaaleissa levytalletusratkaisuissa yleensä joutuu maksamaan.

Nykyään jokaisella suurella pilvialustalla on oma versio Object Storagesta ja niiden sisäiset toteutukset voivat vaihdella. Seuraavaksi on esitelty IBM:n toteutus asialle ja Objectect Storagen arkkitehtuuri on esitetty kuvassa 5. Manager tarjoaa toiminnot admin-käyttäjille ja sen avulla voidaan valvoa ja muokata järjestelmää. Accesser tarjoaa REST-rajapinnan käyttäjille, sekä hoitaa datan salauksen kirjoittaessa ja salauksen purun lukuvaiheessa. Slicestor on vastuussa tiedostosiivujen tallentamisesta, jotka se saa Accesserilta. Slicestoreiden ryhmää kutsutaan Device Setiksi ja Device Settien ryhmää puolestaan Storage pooliksi. Storage poolit voivat koostua Device Seteista, jotka sijaitsevat eri palvelinkeskuksissa. Ämpärit kuvataan sisäisessä arkkitehtuurissa Voulteina ja ne sijaitsevat Storage poolissa. Yksi Storage pool voi pitää sisällään useita Voulteja ja jokainen Voult pitää sisällään Slicestoreita jokaisesta Device Setistä

. 

Kuva 6 IBM Object Storage

Datan tallentamiseen käytetään Information Dispersal Algoritmia (IDA). Algoritmin perusidea on seuraava: Tallennettava datasta luodaan N kappaletta paloja. Tätä lukua kutsutaan algoritmin leveydeksi. Kuitenkaan kaikkia näitä paloja ei tarvita, jotta tiedosto pystytään lukemaan tai kirjoittamaan kokonaisuudessan, vaan pienempi määrä paloja riittää. Read Treshold kertoo määrän kuinka monta palasta tarvitaan alkuperäisen datan lukemiseksi ja Write Threshold kuinka monta palasta tarvitsee olla kirjoitettuna, jotta kirjoitusoperaatio merkataan suoritetuksi. Write Threshold on aina suurempi mitä Read Treshold.

Kuvassa 6 on esitetty tiedoston kirjoitusprosessi. Prosessi lähtee likkeelle Accesser Nodeista, jotka pilkkovat yli 4 MiB kokoiset tiedostot neljäksi erilliseksi siivuksi. Siivut viedään yksitellen loppuprosessin läpi tai pieni tiedosto viedään kerralla kokonaisena. Seuraavaksi dataan lisätään eheyden tarkistusta varten tarkistusarvo tiivistefunktiolla ja data voidaan salata lohkosaluksen avulla. Data viipaloidaan ensiksi Read Treshold arvon mukaisesti viipaleista ja näistä viipaleista luodaan algoritmin leveyden verran paloja joista tarvitsee kirjoittaa algoritmin Write Threshold- määrän mukaisesti paloja, jotta kirjoitusoperaatio merkataan suoritetuksi. Nämä palat lähetetään Slicestor Nodeille tallennettavaksi. Kun kirjoitusoperaatio on merkattu onnistuneesti suoritetuksi, puuttuvat palaset kirjoitetaan asynkroonisesti taustaprosessina. Lukuprosessi on päinvastainen suhteessa kirjoitusprosessiin. Slicestor Nodeilta luetaan Read Treshold- määrän mukainen määrä siivuja ja näistä siivuista palautetaan alkuperäinen tiedosto.



Kuva 7 Tiedoston kirjoitusprosessi

### Block storage

Block storage tallentaa datan vakiokokoisina paloina, jotka se tallentaa yksilöllisen tunnisteen avulla. Tavallinen tietokoneen kiintolevy toimii käyttöjärjestelmässä Block storagen kanssa identtisesti tiedostoja tallentaessa ja tiedostot ovat pilkottu palasiksi kiintolevyllä. Block storage toteutetaan pilvialustoissa tallennusalueverkkojen (SAN) avulla, jotka ovat yleisimpiä varastointiverkkoarkkitehtuureja. SAN yhdistää useita tallennuslaitteita yhdeksi verkoksi, johon tallennustilaa käyttävät laitteet myös liittyvät.

SAN mahdollistaa verkon luomisen muutamalla eri protokollalla, mutta kaikki prokollat käyttävät SCSI-protokollaa. SCSI on vanha, vuonna 1986 luotu protokolla, joka määrittää tiedonsiirtoväylän sekä mitenkä siihen kytketyt laitteet kommunikoivat keskenään. Useat tietokonearkkitehtuurit sisältävät erillisen väylän I/O operaatioille, joiden avulla CPU juttelee I/O laitteiden kanssa. Tämä väylä toteuttaa SCSI-protokollan. SCSI-väylällä voidaan yhdistää useita tallennuslaitteita ja palvelimia, mutta sen skaalautuvuus on erittäin heikko. Luotuun verkkoon voidaan lisätä vain rajallinen määrä laitteita, sekä kaapelin pituudella on maksimi pituus. Versiosta riippuen verkkoon voidaan lisätä 8-16 laitetta sekä laitteet voivat olla toisistaan vain noin 20 metrin päässä toisistaan. Koska SCSI-väylät eivät ole järkeviä tallennusalueverkkojen luomista varten ne on korvattu järkevimmillä väylillä.

Valokuituverkot käyttävät valokuituja tiedonsiirtoon ja Fibre Channel Protocol (FCP) on verkon protokolla. Valokuituverkot ottavat vastaan SCSI-komentoja ja muuttavat ne FCP-protokollan komennoiksi valokuiduissa tapahtuvan siirron ajaksi ja muuttavat ne takasin SCSI-komennoiksi kun siirto on tehty. Tätä varten tarvitsee asentaa oma laiteajurinsa käyttöjärjestelmään. Tallennustilaa käyttävät laitteet ja tallennuslaitteet kytketään samaan valokuituverkkoon, jolloin tiedostonsiirrot eivät rasita normaalia intenetyhteyttä. Tämä prokolla on eniten käytetty ja nopein, koska valokuituverkko on erittäin nopeaa tiedostojen siirtoon. Haittapuolena on verkon erittäin korkea hinta ja verkon ylläpitämiseen tarvittava erikoisosaaminen. Toisista yleisin protokolla on iSCSI, joka käyttää internetyhteyttä valokuituverkon sijasta, ja on huomattavasti halvempi kuin valokuituverkot, mutta hitaampi. iSCSI siirtää SCSI-komennnot TCP-protokollan avulla verkossa. iSCSI on myös tehottomampi tapa siirtää dataa, koska TCP-pyynnöissä on paljon enemmän turhaa dataa mukana suhteuttuna SCSI-komentoihin verrattuna FCP-protokollaan. Muita TCP:tä käyttäviä SAN-protokollia ovat: iFCP ja FICP, mutta ne ovat huomattavasti vähemmin käytetty mitä iSCSI.

Pilvipalveluiden tarjoamat Block storaget ovat lähes poikkeuksessa toteutettu iSCSI protokollan avulla ja ne voi sitoa virtuaalikoneisiin tavallisena kansioina. Block storageihin tallennettu data ei ole sidottu virtuaalikoneen elinkaareen ja data säilyy vaikka virtuaalikoneen poistaisi mihin Block storage oli sidottu.

### File storage

File storage tallentaa datan hierarkisessa rakenteessa käyttäen tiedostoja ja kansioita. File storage mahdollistaa levyn käytön usealle yhtäaikaiselle käyttäjälle ja sen toteutus perustuu samoihin teknologioihin kuin tavallisen verkkolevyn. Yleisin teknologia on NAS, joka on esitetty kuvassa 7. NAS käyttää sisäisesti samoja teknologioita kuin SAN, mutta se lisää oman protokollakerroksen niiden päälle hallitsemaan tiedostoja. NAS-palvelin avaa IP-osoitteen, mihin asiakkaan voivat ottaa yhteyden LAN-verkosta. Tällöin tiedostot liikkuvat samassa verkossa, missä muukin internetliikenne tapahtuu ja tiedostojen siirto voi rasittaa muuta internetliikennettä. NAS edellyttää asiakkailta erillistä asikasohjelmaa, joka on käyttöjärjestelmästä riippuvainen. Yleisin asikasjärjestelmän protokolla on NFS.



Kuva 8 NAS-arkkitehtuuri

## Pilvialustat

Tässä kappaleessa käsitellään projektissä läytettävät julkisen pilven palvelut.

### Microsoft Azure

Microsoft Azure on Microsoftin ylläpitämä pilvipalvelu, joka julkistettiin Microsoftin Professional Developers Conferencessa (PDC) lokakuussa 2008, ja se julkaistiin virallisesti helmikuussa 2010 nimellä Windows Azure vaihtoehdoksi silloin jo tunnetuille Amazon ja Google pilvialustoille. Azuren ensimmäinen versio tarjosi hyvin rajoitetun joukon palveluita, pelkästään pilvipalvelun ASP.NET-verkkosovellusten kehittämiseen ja käyttämiseen, Azure Blob -tietovaraston, Azure SQL -pilvitietokannan ja Azure Service Busin. Vuonna 2014 sen nimi muutettiin nykyiseksi Microsoft Azureksi. Alustan taustatarina alkoi vuonna 2005, kun Microsoft osti Groove Networksin ja Microsoftin kehittäjät alkoivat kehittämään pilvikäyttöjärjestelmää. Azurea koodattiin aluski koodinimella Red Dog ja se julkistettiin vuonna 2008. Se oli tuolloin Windows NT:n laajennus, joka oli suunniteltu toimimaan pilvessä. Se tarjoaa ohjelmistoa palveluna (SaaS), alustan palveluna (PaaS) ja infrastruktuurin palveluna (IaaS) ja tukee monia erilaisia ​​ohjelmointikieliä, työkaluja ja kehyksiä, mukaan lukien sekä Microsoft-kohtaiset että kolmannen osapuolen ohjelmistot ja järjestelmät.[ https://apix-drive.com/en/blog/reviews/microsoft-azure-review]

Azure, kuten muut pilvialustat perustuu virtualisointiteknologiaan. Tietojenkäsittelyssä virtualisointi tarkoittaa virtuaalisen eikä todellisen version luomista jostakin tietokoneen osasta, mukaan lukien virtuaaliset tietokonelaitteistot, tallennuslaitteet ja tietokoneverkkoresurssit.. Suurin osa tietokonelaitteistoista voidaan voidaan virtualisoida ohjelmistossa. Tietokonelaitteisto on yksinkertaisesti joukko ohjeita, jotka on pysyvästi tai puolipysyvästi koodattu piihin. virtualisointikerroksia käytetään yhdistämään ohjelmistoohjeet laitteiston ohjeisiin. virtualisointikerrokset mahdollistavat virtualisoidun laitteiston suorittamisen ohjelmistossa, kuten itse laitteisto. [https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/get-started/what-is-azure]

Pohjimmiltaan pilvialustat ovat joukko fyysisiä palvelimia yhdessä tai useammassa datakeskuksessa. Palvelinkeskukset toteuttavat virtualisoituja laitteistoja asiakkaille. Jokaisen palvelinkeskuksen sisällä on kokoelma palvelimia palvelintelineissä. Jokainen palvelinteline sisältää useita palvelinkortteja ja verkkokytkimen. Nämä tarjoavat verkkoyhteyden ja virranjakeluyksikön (PDU), joka tuottaa virtaa. Telineet on joskus ryhmitelty suuremmiksi yksiköiksi, jotka tunnetaan klustereina. Palvelimen telineet tai klusterit suorittaa virtualisoituja laitteisto-ilmentymiä käyttäjille. Jotkut palvelimet käyttävät kuitenkin pilvihallintaohjelmistoa, joka tunnetaan nimellä kuituohjain. kuituohjain on hajautettu sovellus, jolla on monia vastuita. Se allokoi palveluita, tarkkailee palvelimen ja siinä käynnissä olevien palveluiden kuntoa ja korjaa palvelimia, jos ne kaatuvat. Kukin kuituohjain on yhdistetty pilviorkesteriohjelmistoon jossa pyörii servereitä, Rajapintoja sekä sisäisiä tietokantoja joita Azure vaatii toimiakseen. pilviorkesteriohjelmistossa pyörii muunmuissa palveluita jotka vastaavat käyttäjien tekemiin pyyntöihin hallintakäyttöliittymän kautta. Käyttäjän pyynnöt allokoivat Azure-resursseja ja -palveluita. Ensin käyttöliittymä varmistaa, onko käyttäjällä lupa varata pyydetyt resurssit. Jos näin on, käyttöliittymä tarkistaa tietokannan löytääkseen riittävän kapasiteetin omaavan palvelintelineen, joka kehottaa kuituohjainta varaamaan resurssin [https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/get-started/what-is-azure]

## Tietoturva

Tässä kappaleessa käsitellään tietoturvaan liittyviä protokollia

### RSA

RSA (Rivest–Shamir–Adleman) on julkisen avaimen salausjärjestelmä, joka on yksi vanhimmista yleisesti käytetyistä salausjärjestelmistä. RSA salaa tai allekirjoittaa tiedot kahdella eri avaimella. Yksi avaimista on julkinen avain, joka on kaikkien käytettävissä ja toinen on salainen yksityinen avain. Julkisella avaimella salattujen tietojen salaus voidaan purkaa vain yksityisellä avaimella. Allekirjoittamisessa yksityistä avainta käytetään allekirjoituksen luomiseen ja kuka tahansa voi julkisella avaimella todentamaan että allekirjoitus on tehty yksityisellä avaimella. Kahden avaimen käytön vuoksi julkisen avaimen salaus tunnetaan myös epäsymmetrisenä salauksena, koska salausta ei pureta samalla avaimella millä se luotiin. Symmetrisessä salauksessa samaa avainta käytetään salauksen luomiseen ja purkamiseen.

Nimi RSA tulee Ron Rivestin, Adi Shamirin ja Leonard Adlemanin sukunimistä, jotka kuvailivat algoritmin julkisesti vuonna 1977. RSA:n turvallisuus perustuu käytännön vaikeuteen jakaa kahden suuren alkuluvun tulo tekijöihin. RSA-salauksen rikkominen tunnetaan RSA-ongelmana. Ongelmaa ei ole onnistuttu ratkaisemaan jos käytetään riittävän suurta avainta. RSA on suhteellisen hidas algoritmi. Tästä johtuen sitä ei yleensä käytetä suoraan tietojen salaamiseen, vaan sillä salataan jaettuja avaimia symmetristä salausta varten.

RSA:n perusperiaate on, että on helppoa löytää kolme erittäin suurta positiivista kokonaislukua , ja , siten että kaikille kokonaisluvuille pätee että jakojäännös jaettuna on . Matemaattisesti ilmaistuna kaavalla.

Kun annetaan vain ja , on erittäin vaikea löytää . Kaavassa Kokonaisluvut ja muodostavat julkisen avaimen, on yksityinen avain ja on viesti. Koska ja d voidaan vaihtaa kaavassa päikseen, mahdollistaa se viestien allekirjoittamisen ja allekirjoituksen tarkistamisen samaa algoritmia käyttäen. julkinen avain muodostetaan kahden satunnaisen suuren alkuluvun tulosta ja sen pituus on julkisen avaimen pituus. on positiivinen satunnainen kokonaisluku, jonka arvon valintaväli määritetään Carmichaelin funktion avulla. Yksityinen avain saadaan myös laskettua Carmichael funktion ja julkisen avaimen avulla. Viesti saadaan salattua laskemalla seuraava salakirjoitus käyttäen julkista avainta ja kaavalla

### TLS

TLS on Julkisen avaimen salausmenetelmä, joka on luotu vanhentuneen SSL-protokollan päivitetyksi versioksi. SSL:n ensimmäistä versiota ei koskaan julkaistu tietoturvaongelmien takia, mutta ensimmäinen julkaistu versio SSL-protokollasta oli SSL 2.0, joka julkaistiin vuonna 1995. SSL 2 löytyi heti tietoturvaongelmia ja ne ongelmat korjattiin sueraavana vuonna julkaistussa SSL 3.0 versiossa. TLS 1.0 julkaistiin vuonna 1999 ja siinä ei ole merkittäviä eroja SSL 3.0 versioon. Vuonna 2006 julkaistu TLS 1.1 lisäsi tietoturvaa. TLS 1.2 julkaistiin vuonna 2008 ja nykyinen uusin versio 1.3 julkaistiin vuonna 2018. Uddet versiot ovat parantaneet tietoturvaa poistamalla protokollasta murtuneita salausalgoritmeja ja pakottaneet käyttämään salausalgoritmeissa vahvempia avaimia.

TLS käyttää X.509 -sertifikaattistandardin varmenteita viestintäkumppanin todentamiseen. Kun asikas ottaa yhteyden alvelimeen, palvelin lähettää varmenteen asiakkaalle. Varmenteessa olevat tärkeät tiedot ovat verkkotunnuksen nimi jolle varmenne on myönnetty, Varmenneviranomainen joka on myöntänyt varmenteen ja sen luoma allekirjoitus, voimassaoloaika sekä palvelimen oma julkinen avain. Varmenneviranomainen (CA) on toimija, joka tallentaa, allekirjoittaa ja myöntää digitaalisia varmenteita. Se on kolmannen osapuolen palvelu, johon molemmat osapuolet luottavat. Asiakkaalla on lista Varmenneviranomaisista joihin se luottaa ja niiden julkisista avaimista. Luotetun julkisen avaimen avulla se pystyy varmentamaan palvelimen varmenteen varmenneviranomaisen allekirjoituksen joka on tehty varmenneviranomaisen yksityisellä avaimella. Varmenneviranomaisen tehtävä on varmistaa ennen palvelimen varmenteen allekirjoittamista että palvelin omistaa kysyisen verkkotunnuksen, jolle se on pyytämässä allekirjoitusta. Verkkotunnuksen varmentaminen onnistuu esimerkiksi DNS-tietueiden avulla. Varmenneviranomaisen tekemän allekirjoituksen avulla asiakas pystyy luottamaan palvelimen identiteettiin. Varmenne voidaan myös allekirjoittaa omalla yksityisellä avaimella, jolloin puhutaan itseallekirjoitetuista varmenteista. Silloin asiakas ei voi varmentaa palvelinta varmenneviranomaisten avulla vann luottamus pitää todentaa toisella tapaa. <https://aws.amazon.com/what-is/ssl-certificate/> https://www.ssl.com/article/ssl-tls-self-signed-certificates/

TLS-kättely on prosessi, joka käynnistää TLS:ää käyttävän viestintäistunnon. TLS-kättelyn aikana kaksi kommunikoivaa osapuolta vaihtavat viestejä vahvistaakseen toisensa, sopiakseen käytettävät salausalgoritmit ja sopiakseen istuntoavaimista. TLS versioissa ennen 1.3 versiota ensimmäisenä asiakas lähettää viestin palvelimelle, jossa se ilmoittaa tukemansa TLS versiot ja salaukset. Lisäksi asiakas lähettää satunnais generoidun numeron. Palvelin vastaa tähän viestiin korkeimman TLS version mitä molemmat osapuolet tukevat sekä molempien tukeman salauksen. Palvelin lähettää oman satunnais generoidun numeron sekä SSL-sertifikaatin asiakkaalle, jonka avulla asiakas varmistaa sertifikaatin aitouden. Kun aitous on varmistettu, asiakkaalla on useampi mahdollinen tapa jakaa symmetriseen salaukseen käytettävä avain palvelimen kanssa. Yksinkertaisin on RSA- kättely, jossa asiakas lähettää palvelimelle avaimen salaamalla sen palvelimen julkisella avaimella. Näin palvelin pelkästään pystyy purkamaan salauksen sen omalla yksityisellä avaimellaan. Monimutkaisempi tapa avaimen vaihtoon on käyttää Diffie–Hellman-avaintenvaihtoprotokollaa (DH). DH-protokollassa avainta ei lähetetä salattuna toiselle osapuolelle, vaan molemmat osapuolet laskevat salaisuuden julkisia arvoja käyttäen. DH- protokollassa valitaan alkuluku ja sen generaattori . Molemmat luvut jaetaan julkisesti osapuolten välillä. Molemmat osapuolet valitsevat itselleen salaisen luvun ja laskevat siitä julkisen luvun . Osapuolet jakavat julkiset luvut toisilleen ja laskevat siitä yhteisen salaisuuden kaavlla , jossa on toisen osapuolen julkaisema julkinen luku ja on oma salainen luku. Salaisuuden jakamiseen voidaan myös käyttää DH-protokollasta johdettuja monimutkaisempia algoritmeja, joissa käytetään elliptisiä käyriä ja niiden krptografisia ominaisuuksia. Kun salaisuus on jaettu, asiakas lähettää salatun viestin palvelimelle ja palvelin vastaa siihen salatun viestin. Tämän jälkeen kättelyosuus on valmis ja yhteys jatkuu symmetrisellä salauksella salattuna, jossa salaukseen käytetään palvelimen ja asiakkaan lähettämiä satunnaisia lukuja ja yhteistä salaisuutta.

TLS 1.3 yksinkertaistaa kättelyosuutta merkittäväst sekä parantaa tetoturvaa poistamalla epäturvallisia algoritmeja protokollasta. Yhteisen salaisuuden jakamiseen 1.3 versio edellyttää eteenpäin turvallisuutta. Tämä tarkoittaa jos palvelimen yksityinen salaisuus paljastuu tulevaisuudessa, niin sillä ei pysty purkamaan sillä tehtyjä vanhoja salauksia. Tämän takia edellisessä kappaleessa esitetty RSA- kättely ei ole mahdollista ja DH-protokollassa edellytetään jokaiselle yhteydelle käytettäväksi aina uusia julksia parametrejä. Asiakas lähettää ensimmäisessä viestissä satunnaisgeneroidun numeron, sen tukemat symmetrisen saluksen algoritmit sekä valitsee menetelmän yhteisen salaisuuden jakamiseen ja laskee siihen tarvittavat julkiset luvut. Asiakas ei voi tietää tässä vaiheessa että palvelin tukee valittua menetelmää yhteisen salaisuuden jakamiseen, mutta vaihtoehtoja on 1.3 versiossa vähemmän joten palvelin tukee asiakkaan valitsema mentelmä erittäin todennäisesti. Palvelin vastaa tähän viestiin valitulla symmetrisellä salauksella, omalla satunnaisluvulla sekä valitulla menetelmällä salaisuuden vaihtoon johon se on laskenut omat julkiset arvonsa. Palvelin pystyy tässä vaiheessa laskemaan yhteisen salaisuuden ja on valmis symmetriseen salaukseen. Palvelimen viesti toimii samalla kuittauksena aloittaa salattu yhteys. Asiakas laskee jaetun salaisuuden palvelin vastauksen parusteella ja on valmis symmetriseen salaukseen. Asiakas varmistaa palvelimen varmenteen aittouden ja lähettää kuittausviestin palvelimelle. Kättely on suoritettu ja yhteys jatketaan symmetrisellä salauksella. TLS 1.3 tukee myös TLS-kättelyn entistä nopeampaa versiota, joka ei vaadi ollenkaan edestakaisia viestejä. Jos asiakas ja palvelin ovat muodostaneet yhteyden toisiinsa aiemmin, ne voivat luoda toisen jaetun salaisuuden ensimmäisessä yhteydessä yhteyden jatkamista varten. Palvelin lähettää asiakkaalle myös istuntolipun yhteyden aikana. Seuraavan yhteyden alkaessa asiakas voi käyttää istuntolippua ja yhteyden jatkamiseen luotua salaisuutta ja TLS-salaus pystytään luomaan ilman kättelyvaihetta.

<https://www.comparitech.com/blog/information-security/diffie-hellman-key-exchange/> <https://www.upguard.com/blog/diffie-hellman>

https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/

Symmetriseen salaukseen käyetetään joko lohko- tai virtasalausalgoritmia. Lohkosalausalgoritmi ottaa vakiokokoisen lohkon kokoisen määrän dataa ja salaa sen salatekstiksi. Virtasalauksessa jokainen salattavan tekstin bitti salataan yksi kerrallaan salausnumerovirran vastaavalla bitillä, jotta saadaan salatekstivirran bitti. Lohkosalusta voidaan käyttää virtasalauksen tapaan jos käytetään alustusvektoreita jotka salataan lohkosalauksella ja salattu alustusvektori yhdistetään salattavaan tekstiin.TLS:sän symmetrisessä salauksessa käytetään lohkosalausta virtausalgoritmin tapaan. Salauksen lisäksi algoritmin pitää pystyä todentamaan salatun datan aitous, eli että kukaan ei ole muokannut salattua dataa siirron aikana. Tälläisiä salausjärjestelmiä, jotka pystyvät salaaman datan ja todentamaan salatun data aitouden kutsutaan todennetuksi salaukseksi (AE). Yleisin AE- algoritmi on GCM (Galois/Counter Mode), jota TLS- protokolla käyttää. TLS 1.3 mahdollistaa myös toisen EA- algoritmin käyttämisen CCM (Counter with CBC-MAC), jota aikaisemmat versiot eivät tue.

TLS 1.2 ja aikaisemmat versiot käyttävät useita eri lohko- ja virtasalausalgoritmeja, mutta 1.3 versio käyttää vain yhtä lohkosalausta (AES) ja yhtä virtasalausta (ChaCha20). AES, joka tunnetaan myös alkuperäisellä nimellä Rijndael on Yhdysvaltain kansallisen standardisointi- ja teknologiainstituutin (NIST) vuonna 2008 vahvistama spesifikaatio lohkosalaukseen. AES on nykyään yksi käytetyimmistä lohkosalausalgoritmeista ja siitä on olemassa variaatiot 128, 192  ja 256 bittisille avaimille. Kuvassa 9 on esitetty lohkokaavio AES-GCM todennetun salauksen järjestelmästä, jota TLS 1.3 ja TLS 1.2 käyttää.

A diagram of a computer program

Description automatically generated

Kuva 9 AES-GCM

Lohkokaavion yläosassa lasketaan salattu data AES- algoritmilla ja alaosassa lasketaan varmenne salatusta datasta, jonka avulla pytsytään todentamaan että salattua dataa ei ole muokattu siirron aikana. Salauksessa käytetään alustusvektoria (IV) ja siihen lisätään lohkonumero ja tämä summa salataan AES:in avulla. Varmenne lasketaan Galois'n kuntien avulla ja siihen saadaan luotua avain AES-salaukseen käytetyn avaimen avulla. Varmenteen laskemiseen käytetään salattua dataa, alustusvektoria sekä mahdollista ylimääräistä dataa. AES-GCM mahdollistaa myös ylimääräisen salaamattoman datan lähettämisen salatun datan kanssa ja pystyy myös varmentamaan tämän aitouden ottamalla sen mukaan vermenteen laskemiseen. Tämä on merkattu kaavioon AAD. <https://datatracker.ietf.org/doc/html/rfc8446> <https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf> <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.nrf52832.ps.v1.1%2Fccm.html>

Kuva 10 esittää F5 Labsin tekemän The 2021 TLS Telemetry Report- tutkimuksen tuloksia TLS:n käytetyimmistä salausalgoritmeista. Mukana tutkimuksessa on miljoona käytetyintä nettisivustoa. Salausalgoritmit ovat iloitettu kuvassa muodossa yhteisen salaisuuden jakaminen-symmetrinensalaus-MAC. Tutkimuksen mukaan neljä yleisintä salausta käyttävät kaikki symmetriseen salaukseen AES-GCM- algoritmia, joka esiteltiin edellisessä kappaleessa. Kun lasketaan neljän suosituimman algoritmin käyttöprosentit yhteen saadaan tulokseksi että AES-GCM suojaa 93.2 % kaikista TLS- yhteyksistä. Lisäksi kuvasta huomataan, että yhteisen salaisuuden jakamiseen käytetään DH-protokollan monimutkaisempaa elliptisiin käyriin perustuvaa versiota, joka on merkattu kuvaan lyhenteellä ECDHE. https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report

A screenshot of a computer

Description automatically generated

Kuva 10 Suosituimmat TLS- salausalgoritmit vuonna 2021

# Projektin esittely

Tässä kappaleessa esitellään anonyymisti projekti, johon diplomityön käytännön osuus perustuu. Projekti on anomisoitu asiakkaan toiveen mukaan, joten sitä ei tulla mainitsemaan nimellä. Projektin päämääränä on toteuttaa pilvipalvelu, joka mahdollistaa suurien tiedostojen lataamisen ja niiden automaattisen käsittelemisen ja tulosten siirtämisen eri tietovarastoihin. Projektiin valittiin Oracle Cloud -pilvipalvelu asiakkaan toiveesta. projekti on toteutettu mikropalvelu -arkkitehtuurina käyttäen Docker kontteja, joita ajetaan tuotannossa Oracle Cloudin ylläpitämässä Kubernetes -palvelussa .

Tiedostojen lataamiseen valittiin ratkaisuksi selain pohjainen ratkaisu, mutta myös muita vaihtoehtoja tutkittiin. Ohjelman pitää pystyä lataamaan tiedostoja sekä kansioita ja sen pitää pystyä validoimaan, että tiedostot eivät ole korruptoituneet siirron aikana. Latauksen suuruus on luokkaa 1 – 10 Gb, ja lataus pitää pystyä tekemään normaalitehoisella loppukäyttäjän tietokoneella.

Tiedostot ladataan jaettavalle levylle, joka voidaan jakaa useampien konttien kesken. Latauksen jälkeen toinen kontti käsittelee ladattavat tiedostot ja siirtää tulokset eri tietovarastoihin. Käytettäviä tietovastoja on SQL -tietokannat, jaettavat verkkolevyt sekä Object Storage.

Valitut teknologiat valittiin arvioimalla niiden suorituskykyä, hintaa sekä tietoturvallisuutta. Tässä diplomityössä käytännönosuutena on toteutettu pieniä testipenkkejä, jotka mittaavat vaadittuja omanaisuuksia, sekä on vertailtu eri pilvialustojen hintoja.

Seuraavassa luvussa 4 Tulokset käytetään seuraavia ominaisuuksia vertaillessa lataus- ja talletustapoja. Tietoturva, nopeus, käytetty verkkoyhteys latauksen yhteydessä sekä kustannukset.

Tietoturva käsitellään kootusti yhtenä kappaleena jossa vertaillaan käytettyjen protokollien tietoturvallisuutta ja mahdollisia haavoittuvuuksia käytetyissä teknologioissa.

# TULOKSET

Työssä esitellään useampi pienempi testausprojekti, jotka voidaan jakaa karkeasti kahteen osioon: tiedostojen siirtoon sekä tiedostojen talletukseen. Kokonaisjärjestelmän arkkitehtuuri on esitetty kuvassa 8, joka kattaa sekä tiedostojen siirron että talletuksen. Järjestelmän luomiseen on käytetty Azure-pilvipalvelua ja testejä ajetaan Azuren ulkopuolella paikallisella tietokoneella. Azuressa on käytössä virtuaalikone, jossa pyörii .NET:n avulla toteuttu palvelin, Nginx palvelin, joka pyörittää käyttöliittymää, jota käytetään HTTP-latausten testaamiseen. Käyttöliittymä on toteutettu Angular ohjelmointikehyksen avulla ja se on toteutettu Single Page Application (SPA) arkkitehtuurilla. Nginx-palvelinta käytetään SPA-käyttöliittymän jakamiseen internettiin. Virtuaalikoneella on myös asennettu vsftp-palvelin, joka on FTP palvelin, joka mahdollistaa myös FTPS-yhteyden. OpenSSH-palvelinta käytetään SFTP-yhteyden luomiseen. Virtuaalikoneeseen on liitetty Bock ja File storaget ja ne on sidottu osaksi virtuaalikoneen tiedostojärjestelmää ja niihin pääse käsiksi samaan tapaan kuin virtuaalikoneen omaan kovalevyyn. Blob storageen ja Dynamo-tietokantaan pääsee käsiksi julkisen HTTP rajapinnan kautta. Dynamo-tietokanta on Azuren toteuttama NoSQL-tietokanta, jossa data on tallennettuna json-dokumentteina. Virtuaalikoneeen kanssa samassa virtuaalisessa sisäverkossa on myös Postgres SQL-palvelin. Virtaalikone juttelee SQL palvelimen kanssa C# ohjelmoiintikielellä toteutetun asiakasohjelman avulla.

Diagram

Description automatically generated

Kuva 11 Testiohjelman kokonaisarkkitehtuuri

## Tiedostojen siirto

Tiedostojen siirtoon liittyvät projektit ovat jaoteltu useampaan pienempään projektiin eri tiedostonsiirtotapojen perusteella. FTP-, FTPS- ja SFTP-siirrot muodostavat yhden projektin ja selaimella toimivat protokollat muodostavat oman projektin.

FTP, FTPS ja SFTP siirrot ovat toteutettu kahdella eri ohjelmointikielen kirjastolla. Käytetyt ohjelmointikielet ovat Python sekä C#. Python-ohjelmointikielelllä on FTP ja FTPS kirjasto on Pythonin sisäinen kirjasto siihen ei tarvita kolmannen osapuolen kirjastoja. SFTP-yhteys on luotu Paramiko-kirjastolla, joka on kolmannen osapuolen ilmaisen lähdekoodin kirjasto. C#-koodissa on läytetty kolmannen osapuolen Nuget-paketteja yhteyksien luomiseen. FluentFTP-kirjastoa on käytetty FTP- ja FTPS- yhteyksien luomiseen ja SSH.NET-kirjastoa SFTP-yhteyteen. Projektissa on ladattu erikokoisia tiedostoja käyttäen kumpaakin ohjelmointikieltä. Tiedostojen koot ovat olleet kilotavun ja 10 megatavun välissä. Saman kokoinen tiedosto on ladattu viiteen kertaan ja jokaisen latauksen latausaika on kirjattu muistiin. Näistä latausajoista on laskettu keskiarvo kullekin tiedostokoolle ja keskiarvoja on käytetty tulosten käsittelemiseen

Selaimella toimivia protokollia eli HTTP- ja WebSocket-prokollia on testattu selainpohjaisen käyttöliittymän avulla. Kuvassa 9 on näyttökuva käyttöliittymän HTTP-latauksesta

Graphical user interface, text, application

Description automatically generated

Kuva 12 HTTP-latausnäkymä

Käyttöliittymällä pystyy valitsemaan eri asetuksia käytettäviin protokolliin. HTTP-latauksessa tiedostoa ei lueta kokonaan selaimeen muistiin, vaan se luetaan palasina. Jos tiedostoa ei palotelttaisi, koko tiedosto pitäisi lukea selaimen käyttämään RAM-muistiin ja se rajoittaisi merkittävästi lähetettävän tiedoston maksimikokoa. Tiedoston paloittelemisen avulla lähetettävällä tiedostolla ei ole maksimikokoa rajoittavaa tekijää. Käyttöliittymässä on valittavana 5kB-, 50kB- ja 500kB-kokoiset palaset. Kukin palanen läheteään omana POST pyyntönä palvelimellle ja viimeisen palasen jälkeen palvelin kokoaa palaset yhteen ja muodostaa kokonaisen tiedoston palvelimella.Toinen valittava ominaisuus on lähetettävän palasen koodaus. Kaikissa koodauksissa tiedoston palanen luetaan selaimen File API:n avulla ArrayBuffer-tyyppisenä selaimen muistiin. Arraybuffer on datan esitys kahdeksan bittisenä numeroina taulukkotietorakenteessa. Data voidaan siirtää käyttäen eri koodauksia siirrossa. Base64-koodauksessa binäärinen data muunnetaan merkkijonoksi. Base64 koodisto käyttää 64-merkkistä aakkostoa johon binääridata muunnetaan. Number array-valinta lähettää binäärisen datan merkkijonona ilman koodausta. Tämä vaihtoehto kasvattaa huomattavasti siirrettävän tiedoston kokoa, koska jokaisen tavun jokainen numero esitetään UTF-8 merkkinä siirron aikana. Base64 ja Number array siirtävät datan json-formaattina, jonka merkistö on UTF-8. Form-data formaatti siirtää datan binäärimuodossa ja dataa ei tarvitse lähettää merkkijonona.

WebSocket-protokollalla on mahdollista valita siirrettävän datan formaatti. Valittavana on merkkijono sekä binääri. Merkkijono käyttää UTF-merkistöä siirrettävän datan koodauksena ja binääri käyttää binääriformaattia tiedonsiirtoon.

Käyttöliittymää ohjataan testeissä Selenium-kirjastolla, jolla pystyy automatisoimaan selaimen käyttöä. Seleniumin avulla selainta ohjataan toistamaan aina samat valinnat. Kun tiedosto on ladattu onnistuneesti selaimeen tulee latausaika näkyviin, jonka Selenium poimii talteen.

## Tiedostojen talletus

Tiedostojen talletukseen on toteutettu .NET-palvelin joka suorittaa talletusoperaatiot sekä Python ohjelma joka tekee pyynnön .NET-palvelimelle ja muodostaa tuloksista csv-tiedoston sekä kuvaajan.

# Projektien tulokset

Projektien tulokset ovat jaettu suorituskyvyn sekä kustannusten mukaan. Kustannusvertailussa otetaan kantaa talletusratkaisujen kustannuksiin Azure-pilvipalvelussa ja vertaillaan eri talletusratkaisujen kustannuksia. Tehokkuusvertailussa vertaillaan talletus- sekä lataustapoja ajallisessa näkökulmassa sekä lataustapojen käyttämää verkon määrää.

## Kustannusvertailu

Kustannusvertailu on jaettu kahteen osaan: Pilvialustojen talletusratkaisuihin ja tiedostojen siirtoon. Tiedostojen siirron kustannuksia mitataan epäsuorasti kuinka paljon siirrettävän tiedosotn koko kasvaa siirron aikana eri protokollilla. Siirrettävän datan määrä voi vaikuttaa internetyhteyden hintaan sekä pilvipalvelut laskuttavat virtuaalikoneille tulevasta ja lähtevästä datasta.

Tutkimus tehtiin käyttämällä WireShark-ohjelmistoa, joka avoimen lähdekoodin pakettianalysaattori. Wireshark on monialustainen, ja se käyttää pcap:ia pakettien sieppaamiseen.

### Pilvialustojen talletusratkaisujen kustannukset

Tässä osiossa vertailtaan Azuren tarjoamien talletusratkaisujen hintoja.

Azure Cosmos on NOSQL tietokanta sekä vektoritietokanta, joka tallentaa dokumentteja JSON – muodossa. Dokumentit indeksoidaan ja niitä on mahdollista hakea ja muokata kyselyillä. Azure Cosmos laskuttaa kolmesta eri käyttötyypistä: laskenta, tallennustila ja kaistanleveys. Azure tarjoaa ilmaiset 25 GB talletustilaa jokaiselle azuren tilille. Laskennan laskutus tapahtuu pyyntöyksikköjen avulla (RU). RU on tietokannan operaatioihin vaadittavan laskennan, muistin ja IO:n abstrakti mitta. eri tietokantaoperaatiot kuluttavat pyyntöyksikköjä tietyn välin sisällä. Tietyn operaation vaadittavaan pyyntöyksikköjen määrään vaikuttaa useampi tekijä esimerkiksi kirjoitettavan/luettavan asian koko sekä indeksien käyttäminen kasvattaa vaadittavien Pyyntöyksikköjen määrää. Vaadittavien pyyntöyksikköjen määrä on deterministinen, eli sama operaatio, samalla tietokannalla vaatii aina saman verran Pyyntöyksikköjä. Azure Cosmos DB laskuttaa pyyntöyksiköillä (RU) sekunnissa mitattuna (RU/s) tai pelkästään käytetyistä pyyntöyksiköistä. Laskutus riippuu tietokannan valitusta skaalautuvuusmallista. palveliton malli laskuttaa pelkästään käytetyistä pyyntöyksiköistä 0.235 euroa miljoonasta pyyntöyksiköstä. Kiinteäsuorituskyky-mallissa tietokantaan kiinnitetään kiinteä RU/s arvo, ja siitä veloitetaan 0.0075 euroa tunnilta sataa RU/s yksikköä kohden. Automaattisessa skaalaus- mallissa tietokannan RU/s arvo skaalautuu automaattisesti kuorman mukana 10% -100% välillä annetusta maksimi RU/s arvosta. Tunnin maksimi RU/s arvosta veloitetaan 0.008 euroa tunnilta sataa RU/s yksikköä kohden. Talletustila maksaa 0.235 euroa kuukaudessa giga tavulta. Kaistanleveydestä joutuu maksamaan pelkästään tietokannasta ulos siirtyvästä datasta. Ensimmäiset 5 GB on ilmaista, jonka jälkeen hinta on 0.05 euroa giga tavulta. https://azure.microsoft.com/en-us/pricing/details/cosmos-db/autoscale-provisioned/

<https://learn.microsoft.com/en-us/azure/cosmos-db/request-units>

Azure tarjoaa täysin ylläpidetyn SQL tietokannan ja varmistaa sen saatavuuden ja suorituskyvyn. Tietokannoissa maksaa laskenta ja talletustila. laskentaan on tarjolla kaksi eri mallia: virtuaaliytimeen (vCore) ja Tietokantatapahtumayksikköön (DTU) perustuva ostomalli. Virtuaaliytimeen (vCore) perustuva malli tarjoaa valinnan varattujen tai palvelimettomien laskentamallien välillä. Valmistetun laskentamallin avulla voi valita tarkan määrän virtuaaliytimiä, jotka on aina varattu laskentaa varten. Palvelimettomassa laskentamallissa virtuaaliytimiä voidaan automaattisesti skaalata konfiguroitavalla laskenta-alueella. Palvelimeton laskentamalli sulkee automaattisesti tietokannan epäaktiivisina aikoina, jolloin tietokantaan ei ole yhtään istuntoa avattuna. Suljetusta ajasta laskennasta ei laskuteta olleenkaan, ja tietokanta käynnistyy udelleen kun uusi istunto luodaan. Virtuaaliytimiä on tarjolla eri arkkitehtuurilla ja niillä on hieman eri hinnat. Virtuaaliytimen määrä on valittavissa 2 ja 128 väliltä ja yhden virtuaaliytimen hinta pienenee hieman kun valitaan suurempi määrä ytimiä. Yhden ytimen hinta kuukaudessa on 100-120 euroa. Palvelimeton malli laskuttaa ytimestä sekuntihintaa ja varattu malli puolestaan minuuttihintaa. Tietokantatapahtumayksikkö (DTU) edustaa suorittimen, muistin, lukujen ja kirjoitusten abstraktia mittaa. DTU-pohjaisessa ostomallissa on mahdollista valita laskentatehon määrä joka ilmoitetaan käytettävissä olevilla tietokantatapahtumayksiköillä. Valittuun laskentatehoon on kiinnitetty kiinteä määrä mukana olevaa tallennustilaa, kiinteä säilytysaika varmuuskopioille ja kiinteä hinta. Lisämuistia on mahdollista ostaa hintaan kuuluvan muistin lisäksi. Laskentatehoa voidaan lisätä tietokannan luomisen jälkeen lisäämällä tietokantatapahtumayksikköjä ja se aiheuttaa vain lyhyen katkoksen tietokannan käyttämiseen. DTU-pohjaisen laskentatehon voi muuttaa virtuaaliytimen tehoon suhteella 100 DTU-yksikköä vastaa yhtä virtuaaliydintä.

Laskentateho on valittavissa 5-400 DTU-yksikön välistä ja käytettävissä oleva talletustila vaihtelee 2 GB - 4 TB välissä. Hinta taas vaihtelee 4.5 ja 14515 euron välillä. Virtuaaliytimeen perustuva malli laskuttaa talletustilasta 0.232 euroa kuukaudessa giga tavulta. Tietokantatapahtumayksikköön perustuvassa mallissa hintaan sisältyy tietty määrä levytilaa ja lisälevytila maksaa 0.16 euroa kuukaudessa giga tavulta https://learn.microsoft.com/en-us/azure/azure-sql/database/dtu-benchmark?view=azuresql

https://learn.microsoft.com/en-us/azure/azure-sql/database/resource-limits-dtu-single-databases?view=azuresql

https://azure.microsoft.com/en-us/pricing/details/azure-sql-database/single/

Azure Blob Storage on Microsoftin Object Storage ja se on suunniteltu rakenteettoman datan tallentamiseen. Azure Blob Storage tarjoaa HTTP-rajapinnan, jonka avulla tiedostoja pystytään muokkaamaan. Useille ohjelmistokielille on tehty valmiit asiakaskirjastot sekä CLI työkaluja. Blob Storage laskuttaa talletustilasta sekä operaatioista. Talletustilan ja operaatioiden hinta määräytyy valitun palvelumallin mukaan. valittavia malleja on viisi: korkealuokkainen, kuuma, lämmin, kylmä ja arkisto. Tallennuksen gigatavujen hinnat nousevat arkistosta kohti korkealuokkaista jokaisella askeleella. Puolestaan operaatiot halpenevat arkistosta kohti korkealuokkaista jokaisella askeleella. Korkealuokkaisella mallilla on pienin latenssi ja se kasvaa kohti arkistoa. korkealuokkaisenmallin talletus tila maksaa 0.13869 euroa giga tavulta ja on melkein kymmenen kertaa kalliimpi kuin seuraava kuumamalli. Hinnat halpenevat edelleen kohti arkistoa mentäessä ja arkisto maksaa 0.00092 euroa giga tavulta. korkealuokkainen malli tarjoa halvimmat luku- ja kirjoitusoperaatiot. korkealuokkaisen kirjoitusoperaatioiden hinta on 0.0211 € kymmentä tuhatta operaatiota kohden ja on noin kuusi kertaa halvempi kuin arkistomallin hinta. Lukuoperaatioiden hinta puolestaan muuttuu huomattavasti enemmän mallien välillä. korkealuokkaisen hinta on 0.0017 € kymmentä tuhatta operaatiota kohden. Kylmän mallin hinta on noin 100 kertainen korkealuokkaiseen nähden ja arkiston noin 4000 kertaa kalliimpi.

<https://azure.microsoft.com/en-us/products/storage/blobs/> <https://azure.microsoft.com/en-us/pricing/details/storage/blobs/>

Azure Files on Microsoftin File Storage palvelu. Palvelu mahdollistaa tiedostojen yhtäaikaisen käyttämisen eri käyttöjärjestelmien välillä. Valittavia palvelumalleja on neljä: korkealuokkaien, Tapahtuma optimoitu, kuuma, viileä. Korkealuokkaien tarjoaa pienimmän latenssin ja se kasvaa kohti kylmää mallia. Korkealuokkaisella talletustila maksaa eniten ja se halpenee mentäessä kohti kylmää. Korkealuokkaisen hinta on 0.1480 € varatulta giga tavulta ja kylmän mallin hinta on noin 10 kertaa halvempi. Luku- ja kirjoitusoperaatiot ovat korkealuokkaisessa ilmaisia, mutta muissa malleissa maksullisia. Tapahtuma optimoidussa on halvimmat hinnat ja hinnat kasvavat kun siirrytään kylmää mallia kohti. Kylmän mallin hinnat ovat keskimäärin noin kymmenen kertaa kalliimmat kuin Tapahtuma optimoidun. <https://azure.microsoft.com/en-us/pricing/details/storage/files/>

Azure Managed Disks on Microsoftin Block storage, jota voidaan käyttää virtuaalikoneiden kanssa. Levyvaihtoehtoja on neljä: Ultra Disk, korkealuokkaien SSD, SSD ja HDD. Ultra Diskin latenssi on kaikista pienin ja se on uuden generation SSD- levy. Korkealuokkaisen SSD:n ja SSD:n välillä on pieni suorituskyky- ja hintaero. Levyille on ilmoitettu kuinka monta kirjoitus/luku- operaatiota voidaan suorittaa (IOPS) sekunnissa sekä nopeus, jolla data voidaan siirtää ulos tai sisään, ilmoitettuna MB sekunnissa (MB/s). Azure tarjoaa mahdollisuuden parantaa levyn IOPS- ja MB/s-suorituskykyä hetkellisesti. Tätä kutsutaan purskeeksi ja maksimi yhtäkestoinen aika on sille 30 minuuttia. Purskeen käyttö kuluttaa krediittejä ja kun kaikki krediitit on käytetty purske loppuu. Kun levyä käytetään alle sen IOPS- ja MB/s-suorituskyvyn krediittejä kerääntyy uutta pursketta varten. Pienillä alle 128 GB levyillä korkealuokkisen SSD:n suorituskyky on huonompi kuin tavallisen SSD:n, mutta purske on paljon tehokkaampi. Mitä suurempi levyn koko sitä tehokkaampi korkealuokkainen SSD on verrattuna vastaavan kokoiseen tavalliseen SSD levyyn. Korkealuokkaisen SSD levyn hinta on noin 2 kertaa suurempi kuin vastaavan kokoisen SSD-levyn. Korkealuokkainen SSD levy laskuttaa vielä erikseen IO operaatioista 0.001850 € jokaista 10,000 operaatiota kohden tunnissa. IO-operaatioilla on maksimi hinta kuukaudessa ja se on noin levyn talletuskapasiteetin hinta. HDD-levyn hinta on noin puolet vastaavan kokoiseen SSD-levyyn nähden. HDD-levyn IOPS-suorituskyky on melkein sama kuin vastaavan kokoisen HDD-levyn, mutta siirrettävän datan määrä on huomattavasti pienempi. Ultra diskin laskutus toimii pikkaisen erillä lailla, siinä käyttäjä voi valita vapaasti halutun IOPS- ja MB/s-suorituskyvyn ja levyn talletuskoon. Jokaisesta valitusta komponentista laskutetaan erikseen ja levyn kustannukseksi tulee näiden summa. <https://azure.microsoft.com/en-us/pricing/details/managed-disks/> <https://learn.microsoft.com/en-us/azure/virtual-machines/disk-bursting>

Eri talletusratkaisujen hintoja vertaillaan kolmella eri talletuskoolla: 16 GB, 256 GB ja 1 TB. Talletusratkaisujen pelkästään talletustilasta laskuttaama hinta kuukaudessa on esitetty taulukossa 1.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 16 GB | 256 GB | 1 TB |
| Managed Disks/ korkealuokkainen SSD | 2.22€ | 31.96€ | 113.61€ |
| Managed Disks/ SSD | 1.11€ | 17.76€ | 71.01€ |
| Managed Disks/ HDD | 0.70€ | 10.48€ | 37.87€ |
| Files/ korkealuokkainen | 2.37€ | 37.89€ | 148€ |
| Files/ tapahtuma optimoitu | 0.88€ | 14.2€ | 55.5€ |
| Files/ viileä | 0.22€ | 3.55€ | 13.9€ |
| Blob Storage/ korkealuokkainen | 2.22€ | 35.50€ | 138.69€ |
| Blob Storage/ kuuma | 0.27€ | 4.28€ | 16.7€ |
| Blob Storage/ arkisto | 0.01€ | 0.24€ | 0.92€ |
| Cosmos | 3.71€ | 59.40€ | 232€ |
| SQL | 3.71€ | 59.40€ | 232€ |

Taulukosta 1 näkee että tietokantojen (Cosmos ja SQL) hinnat ovat samat ja selvästi muita ratkaisuja kalliimmat. Kaikkien muiden talleyusratkaisuiden kalliimpien ratkaisuiden hinnat ovat keskenään lähellä toisiaan. Halvempien ratkaisuiden hinnat eroavat toisistaan enemmän. Kaikista halvin ratkaisu on Blob Storagen arkisto. Sen hinta verrattuna halvimpiin Managed Disksiin tai Files palveluihin on useita kymmeniä kertoja halvempi. Blob Storagen arkiston IO- operaatiot puolestaan ovat kymmeniä kertoja kalliimmat kuin minkään muun talletusratkaisun, eli se sopii parhaiten tiedostoille, joita ei tarvitse juuri olleenkaan muokata tai lukea.

Talletusratkaisuiden IO-kuormasta syntyviä kustannuksia, on paljon vaikeampi vertailla keskenään, koska talletusratkaisuilla on käytössä eri mittoja operaatioiden vaadittavan suorituskyvyn mittaamiseen. Taulukossa 2 on esitetty eri talletusratkaisuiden hinta miljoonalle lukuoperaatiolle ja miljoonalle kirjoitusoperaatiolle. Joillekkin talleyusratkaisulle on jouduttu arvioimaan hinta, koska talletusratkaisu ei suoraan ilmoita hintaa yksittäisille operaatiolle. Arvioidun hinnan edessä on käytetty tilde-merkkiä(~).

|  |  |  |
| --- | --- | --- |
|  | Miljoona lukuoperaatiota | Miljoona kirjoitusoperaatiota |
| Managed Disks/ korkealuokkainen SSD | 0€ | 0€ |
| Managed Disks/ SSD | 0.19€ | 0.19€ |
| Managed Disks/ HDD | 0€ | 0€ |
| Files/ korkealuokkainen | 0€ | 0€ |
| Files/ tapahtuma optimoitu | 0.14€ | 1.39€ |
| Files/ viileä | 1.21€ | 12.02€ |
| Blob Storage/ korkealuokkainen | 0.17€ | 2.11€ |
| Blob Storage/ Kuuma | 0.47€ | 6.01€ |
| Blob Storage/ arkisto | 600.0€ | 12.02€ |
| Cosmos | 0.23€ | ~0.50€ |
| SQL | ~0.34€ | ~0.70€ |

Cosmos tietokannan lukuoperaatioiden hinnan voi laskea suoraan, koska jokainen lukuoperaatio käyttää aina vain yhden RU-resurssin. Kirjoitus- operaatio voi käyttää eri määrän RU-resursseja riippuen tietokannan koosta ja kirjoitettavan tiedoston koosta. Taulukossa on arvioita että kirjoitus-operaatio käyttäisi keskimäärin 2 RU-resurssia. SQL- tietokannan IO-operaatoiden hinta on vieläkin vaikemapi arvioida, koska vastaava laskennan abstraktimitta DTU-yksikkö laskutetaan eri tavalla. DTU- yksiköistä laskutetaan käytettävistä yksiköistä kuukaudessa. Taulukun hinta on arvioitu 5 DTU-yksikön tietokannalla arvioiden että yksi lukuoperaatio vie yhden yksikön ja operaatio käyttää yksikköä sekunnin ajan. Näin viiden yksikön tietokannalla pystyttäisiin suorittamaan noin 13 miljoonaa lukuoperaatiota kuukaudessa. Kuukauden hinta on jaettu 13, josta on saatu miljoonan lukuoperaation hinta. Kirjoitus operaatioiden on arvioitu käyttävän resursseja tuplasti ja näin ollaan päädytty taulukon hintaan. Nämä hinnat edustavat minimihintaa, ja edellyttyvät että tietokannasta käytettäisiin 100% teholla. Todelliset hinnat tulisivat näin ollen olemaan kalliimpia, riippuen tietokannan käyttöasteesta.

Kun vertailee taulukkoa 1 ja taulukkoa 2 niin huomaa selkeästi logiikan kylmien ja kuumien talletusratkaisuiden välillä. Kuumat ratkaisut ovat IO-suorituskyvyltään tehokkaita ja talletustila kustantaa enemmän, mutta IO-operaatiot ovat halpoja. Kylmät ratkaisut ovat päin vastoin erittäin halpoja talletustilaltaan, mutta IO-operaatiot ovat erittäin kalliita ja latenssi erittäin suuri. Kun halutaan optimoida talletusratkaisun hintaa, täytyy miettiä useampia asioita, jotka vaikuttaa hintaa. Kuinka paljon IO-operaatioita suoritetaan ja onko pieni latenssi ja hyvä suorituskyky kriittinen ominaisuus. Jos IO-operaatioita on paljon, mutta nopeus ei ole kriittinen tekiä, optimaalisin hintaratkaisu löytyy lämpimän päädyn ratkaisuista. Jos IO-operaatioita ei ole paljon ja nopeus ei ole kriittinen tekijä, kannattaa valita kylmämalli. Taulukoista erottuu selvimmin Blob Storagen arkisto- malli. Sen talletustilakustannukset ovat selvästi muita ratkaisuja halvemmat, mutta lukuoperaatiot ovat puolestaan luokkaa tuhat kertaa kalliimmat kuin minkään muun ratkaisun. Tietokantojen(Cosmos ja SQL) kustannukset ovat kalliimmat kuumilla ratkaisuilla, mutta ne mahdollistavat monimutkaisten kyselyjen tekemisen ja rinnakkaisen käytön turvallisesti.

### Tiedostojen siirron kustannukset

Tiedostojen siirron kustannuksia vertaillaan epäsuorasti: kuinka paljon eri siirtoprotokollat kasvattavat tiedoston kokoa siirrettäessä. Siirrettävän datan kasvu aiheuttaa lisää kustannuksia verkkoliikenteen kasvaessa. Pilvialustat velottavat virtuaalikoneen tulevasta ja lähtevästä datasta.

Tutkimuksessa käytettiin WireShark nimistä ohjelmistoa. Se on avoimen lähdekoodin pakettianalysaattori, jonka avulla voidaan tutkia verkkoliikennettä. Wireshark on monialustainen ohjelmisto ja se käyttää PCAP:ia (Packet Capture) pakettien sieppaamiseen. PCAP on verkkokäytäntö, joka kopioi verkossa kulkevia datapaketteja niiden liikkuessa verkon poikki ja tallentaa syntyneet kopiot analysointia varten. Wireshark tarjoaa käyttöliittymän kaapattujen pakettien analysointiin, jonka avulla projektin tietoliikennettä on tutkittu.

Projektissa tutkittiin tietoliikennettä tiedostoa siirtäessä molempiin suuntiin: palvelimelle ja palvelimelta kulkevaa liikennettä. Molempiin suuntiin kulkenut data laskettiin yhteen ja vertailu tehtiin tällä summatulla arvolla. Vertailussa on mukana kaikki projektissa käytetyt protokollat ja niitä on käytetty C# -kirjastoilla. Vertailu tehtiin kahden erikokoisella tiedostolla. Tiedostojen koot olivat 1M tavua sekä 200M tavua. Kuvassa X on esitetty vertailu 1 M tavun tiedostolla sekä kuvassa Y on esitetty vertaiulu 200M tiedostolla. Tiedoston kasvu on kuvissa ilmoitettu prosenttiaalisena kasvuna verrattuna alkuperäiseen tiedostoon.

A graph of different colored bars

Description automatically generated

A graph of different colored bars

Description automatically generated

1 M tiedosto kasvaa hieman enemmän siirtäessä kuin suurempi 200 M tiedosto kaikilla muilla protokollilla, paitsi FTP:llä jolla se pysyy melkein samana. 1M tiedoston kasvuprosentti on 3.7 - 6.7 % välillä ja 200 M tiedoston 4.0 - 4.7 % välillä. SFTP kasvaa eniten kummallakin tiedostokoolla. FTP kasvaa vähiten pienemmällä tiedostolla ja WebSocket puolestaan isommalla tiedostolla. Salatut protokollat kasvavat kaikista eniten isommolla tiedostolla ja pienemmällä tiedostolla SFTP ja HTTPS kasvavat eniten, mutta FTPS kasvaa vähemmän kuin salaamattomat HTTP tai WebSocket. Pienellä tiedostolla kasvuun vaikuttaa enemmän esimerkiksi protokollan yhteydenavaus ja käyttäjänhallinan prosessit. Esimerkiksi HTTP protokollaa kasvattaa HTTP -pyynnön vastaus. Pienemmällä tiedostokoolla tämä vastaus on noin 1% tiedoston koosta.

[https://www.forbes.com/advisor/business/software/what-is-pcap/]

## Tehokkuusvertailu

Kuva 12 esittää FTP-, FTPS- sekä SFTP-latausten latausaikojen tulokset Python- ja C#-kirjastoilla toteutettuna. Kuvasta nähdään että SFTP on selvästi FTPtä ja FTPStä hitaampi molemmilla ohjelmointikielillä suurimmilla tiedostoilla. Pienellä tiedostokoolla SFTP ei ole hitaampi kuin FTP tai FTPS, mutta ero kasvaa voimakkaasti kun tiedoston koko kasvaa. Python koodissa SFTP:n suorituskyky on selvästi huonompi, kuin mitä C#-koodilla on, mutta FTP- ja FTPS-latausten kohdalla ohjelmointiielellä ei ole merkitystä. FTP ja FTPS suorituskyvyt eroavat toisistaan vain vähän. Pienillä tiedostoilla FTP on hieman nopeampi koska aloitusprosessi on lyhyempi, koska yhteyttä ei salata.

Chart, line chart

Description automatically generated

Kuva 13 FTP, FTPS ja SFTP latausten tulokset

Kuva 13 esittää HTTP latauksien latausajat. Latausten pyynnön koko on esitetty x-akselilla ja latausaika on esitetty y-akselilla. Pylväiden väri puolestaan merkitsee eri pyynnön koodausta. Kuvaajasta näkee selvästi että pyynnön koon suurentuessa latausaika pienenee erittäin voimakkaasti kaikilla koodauksilla. Pienempi pyynnön koko aiheittaa useampia pyyntöjä palvelimelle, joka hidastaa latausta merkittävästi. 5 kB:n kokosilla pyynnöillä hitain koodaus on base64, vaikka teoriassa array on kaikista hitain koska se kasvattaa eniten siirrettävän datan kokoa.

Chart, bar chart

Description automatically generated

Kuva 14 HTTP-latausten latausajat

Kuva 14 esittä HTTP -latausten lähetettävän datan määrää suhteessa tiedoston kokoon. Lähetettävä data on mitattu Wireshark -ohjelmalla jota käytetään verkkojen analyysiin. Siirrettävässä datassa on mukana kaikki TCP- tasolla siirtnyt data, joten se katttaa siirretävän tiedosotn sekä HTTP- pyyntöön tarvittavan datan. Kun käytetään pientä 5kb kokoista pyyntöä lähetettävän datan määrä kasvaa reilusti kaikilla koodauksilla. Teoriassa form-data on kaikista tehokkain tapa lähettää dataa ja data siirtyy binäärimuodossa. Form-data kasvattaa lähetettävän datan määrää vain muutamalla prosentilla kun lähetettävän pyynnön koko on suuri. Base64 koodaus kasvattaa lähetettävän tiedoston kokoa noin 33% kun käytetään suurta pyyntöä. Array koodaus kasvattaa siirrettävän datan 100% suuremmaksi mitä alkuperäinen tiedosto, jonka takia se ehdottomasti huonoin tapa siirtää tiedostoja HTTP protokollan avulla. Ainoastaan 500 kB on nähtävissä selkeasti teoreettinen tulos että form-data on nopein jonka jälkeen base64 koodaus on noin 33% hitaampi sekä array useampi kertoja hitaampi.

Chart, bar chart

Description automatically generated

Kuva 15 HTTP -latausten tiedoston kasvu lähettäessä

Kuva 15 esittää Webocket-latauksen tulokset. Websocket-yhteys luotiin .NET:n SignalR-kirjaston avulla. SignalR-kirjasto on osa .NET joten sitä ei tarvitse erikseen asentaa palvelimelle, mutta selaimeen se tarvitsi asentaa npm-pakettina. Signal käyttää konepellin alla selaimen Websocket API:a, mutta se luo korkeamman tason rajapinnan, mahdollistaen palvelinpään funktioiden kutsumisen käyttöliittymästä websocket-yhteyden avulla. Binääriviestintää varten SignalR kirjastoon joutuu asentamaan lisäosan Nuget paketin nimeltään MessagePack sekä käyttöliittymään saman nimisen npm-paketin. Lataus suoritettiin sekä merkkijono viestinä että binäärinä ja testissä käytettiin kolmea erikokoista tiedostoa. Latausten ajat olivat hyvin samanlaiset riippumatta käytettiinkö merkkijono- vai binääriformaattia. Pienimmillä tiedostoilla merkkijono oli hitusen nopeampi, mutta suuremmalla tiedostolla binääri oli hieman nopeampi. Teoriassa binääriformaatti pitäisi olla nopeampi, koska siirrettävä data on pienempi.

Chart

Description automatically generated

Kuva 16 Websocket-lataus

**Error! Reference source not found.** esittää siirrettävän datan suuruutta verrattuna alkuperäiseen tiedostoon. Siirrettävä data on mitattu TCP -tasolla, samaan tapaan kuin HTTP -pyyntöjen siirrettävän datan suuruutta vertailtaessa. String -formaatilla lähetettäessä siirrettävä data on kaksinkertainen alkuperäiseen tiedsotoon nähden ja binary -formattissa siirrettävä data 11% suurempi mitä alkuperäinen tiedosto.

Chart

Description automatically generated

Kuva 17 Websocket -latauksen siirrettävän datan kasvu

Kuva 17 esittää tiedostojen talletusnopeuksia eri talletusratkaisuilla. Kuvan y-akseli on logaritminen ja siitä erottuu selvästi etttä Disk ja File Storage ovat huomattavasti nopeimpia talletusratkaisuja. Tähän vaikuttavia tekijöitä on sisäverkon verkkoliikenne ja ne eivät vaadi mitään kirjautumismekanismia. SQL-talletus on seuraavaksi nopein. Siinä data siirretään TCP yhteydellä sisäverkossa, mutta SQL-kyselyt tarvitsevat kirjautumis- ja datan eheysprosessit jotka hidastavat tiedostojen tallettamista. Blob Storage sekä Cosmos käyttävät julkisen verkon HTTP API:a tiedostojen tallentamiseen ja ovat siksi ehdottomasti hitaimpia talletuspaikkoja. Blob Storage mahdollistaa binääritiedostojen tallentamisen binäärimuodossa, mutta Cosmos käyttää pelkästään json-dokumenttejä ja tiedostojen koko kasvaa merkittävästi kun on tallennettuna tekstinä. Cosmoksen json-dokumentin maksimikoko on 2MB joten Cosmoksen suorituskykyä ei voitu testata 10 MB tiedostoilla.

Chart, line chart

Description automatically generated

Kuva 18 Tiedostojen talletusnopeuksia

Yksi tutkimuksista on vertailla tiedostojen tallentamista SQL-kantaan binäärisenä blob tyyppina. Toinen mahdollinen vaihtoehto on tallentaa SQL-tietokantaan tiedoston tiedostopolku ja tallentaa tiedosto kovalevylle tietokannan sijaan. Kuva 18 esittää tutkimuksen tulokset. Tutkimuksessa vertailtiin tiedostojen lukua, kirjoitusta, poistoa sekä tiedosto infon lukua tietokannasta. Tutkimuksessa käytettiin 2 tietokanta taulua jotka ovat muuten identtiset, mutta toisessa tiedosto on tallennettuna tietokantaan ja toisessa linkin avulla tiedostojärjestelmään. Tiedostoinfo on tallennettuna samalla rivillä tiedoston tai sen linkin kanssa ja tyypiltään se on varchar. Tuloksista nähdään selvästi että tiedoston luku selvästi nopempi koska erillistä levylukuoperaatiota ei tarvitse tehdä tietokanta kyselyn jälkeen. Infon luku on hieman nopeampi taulussa jossa on linkki, johtuen että rivien indeksointi on hieman nopeampaa RAM muistissa koska rivien koko on pienempi. Kirjoitus- ja poistonopeudet ovat lähes samat riippumatta tallennustavasta. Toinen huomioita asia tiedostojen tallentamisessa levyjärjestelmään on tietokannan eheys. Kun tiedostot tallennetaan tietokantaan niin tietokanta pysyy eheänä tiedostojen suhteen. Levylle tallennettaessa eheys saattaa rikkoontua koska tietokantaoperaatio ja levyoperaatio eivät ole saman tietokantatransaktion sisällä

Chart, bar chart

Description automatically generated

Kuva 19 Tiedostojen tallentaminen SQL-tietokantaan ja kovalevylle

## Tietoturvavertailu

Tietoturvavertailussa vertaillaan eri talletustapojen sekä tiedonsiirtoprotokollien tietoturvallisuutta. Käsitellyistä tiedonsiirtotavoista suojaamattomia protokollia ovat HTTP, FTP, WebSocket ja WebDav. Niiden käyttöä ei suositella nykyään olleenkaan, vaan niiden tilalla pitäisi käyttää suojattuja versioita HTTPS, FTPS WebsocketS ja WebDavS. Kaikki suojaamattomat protokollat eivät salaa tiedonsiirtoa palvelimen ja asiakkaan välillä tämän takia kuka tahansa joka pääsee verkkoliikenteeseen käsiksi voi siepata ja lukea sen. Lisäksi suojaamattomat protokollat eivät tarjoa mitään todennusta, mikä tekee niistä haavoittuvia välimieshyökkäyksille (MITM). MITM -hyökkäys on yleinen termi, jossa tekijä päsee kuuntelemaan ja muokkaamaan kahden käyttäjän välistä keskustelua. Tarkoituksena on salakuunnella keskustelua tai esiintyä toisena osapuolena muokkaamalla keskustelua. jolloin se vaikuttaa tavalliselta tiedonvaihdolta hyökkäyksen kohteeksi joutuneille ja ne uskovat kommunikoivan suoraan keskenään, vaikka hyökkääjä on asettunut kahden käyttäjän väliin.

HTTP- protokollassa on lisäksi haavoittuvuuksia, joita ei pystytä torjumaan TLS- salauksella. BREACH haavoittuvuus johtuu HTTPP- protokollan pakkausalgoritmista. HTTP-pakkaus suoritetaan tavallisesti deflate-algoritmin avulla. Kun dataa pakataan tällä algoritmilla, kaikki toistuvat tavusekvenssit havaitaan, eikä niitä toisteta pakatussa datassa. Sen sijaan toistuva tavusekvenssi tallennetaan vain kerran, samoin kuin osoittimet, jotka osoittavat mistä sama sekvenssi löytyy uudelleen. Tämä vähentää lähetettävien tavujen määrää. Pakatun datan pituus on kuitenkin näkyvissä myös salattuna, ja tämä on yksi peruselementeistä, joka tekee BREACH-hyökkäyksen mahdolliseksi. Jotta sovellus olisi haavoittuvainen BREACH-hyökkäykselle, palvelimen HTTP-pakkaus täytyy olla kytkettynä päälle. hyökkääjällä tulee olla keino tarkastella uhrin liikennettä ja lisäksi hyökkääjällä on oltava mahdollisuus lähettää HTTP-pyyntöjä haavoittuvaan palvelimeen. Hyökkäys toimii lähettämällä pyyntöjä palvelimelle saadakseen tietoja pakatusta ja salatusta vastauksesta. Vastauksen otsikoita ei pakata, joten tällä hyökkäyksellä niitä ei pystytä kaappaamaan. Esimerkiksi evästeet siirretään otsikoissa, mutta CSRF-avaimet ovat saatavilla tällä hyökkäyksellä. Esimerkiksi CSRF-avaimen varastamiseksi hyökkääjän tulee tietää missä muodossa CSRF-avain lähetään HTTP- pyynnöissä ja vastauksissa. esimerkiksi voidaan käyttää CSRF avaimelle muotoa token=123456789. Hyökkääjä aloittaa ensimmäisen numeron arvaamisen ja lähettää palvelimelle pyyntöjä joissa on pelkästään mukana arvattavan CSRF- avaimen ensimmäinen numero. Hyökkääjä lähettää kaikki mahdolliset numerot palvelimelle. Palvelimen vastauksista lyhin on se missä numero on arvattu oikein. Tämä prosessi toistetaan jokaiselle numerolle erikseen kunnes koko CSRF- avain on saatu muodostettua. Hyökkäykseltä pystyy suojautumaan laittamalla palvelimen HTTP-pakkauksen pois päältä. https://www.acunetix.com/blog/articles/breach-attack/

Kaikkien protollien suojaamiseen käytetään samaa TLS-salausprotokollaa, joten ne ovat kaikki haavoittuvia TLS-protokollan tietoturvaongelmille. Vanhoissa SSL/TLS versioissa on useita tiedossa olevia haavoittuvuuksia alle on esitelty muutama yleisesti tiedossa oleva hyökkäys protokollia vastaan.

RC4 hyökkäys symmetristä salausta vastaan. RC4 on symmetrinen virtasalausalgoritmi, joka julkistettiin vuonna 1987. RC4 on ollut mahdollinen symmetrinen salaus jo ensimmäisessä SSL 2.0 versiossa ja se poistettiin TLS 1.3 versiosta. Toisin kuin nykyaikaisessa virtasalauksessa, RC4 ei ota erillistä satunnaista alustusvektoria avaimen rinnalla. Tämä tarkoittaa, että jos yhtä pitkäaikaista avainta käytetään useiden virtojen turvalliseen salaamiseen, alustusvektorin ja pitkän aikavälin avain on yhdistetään RC4:n salausavaimen luomiseksi. Ensimmäiset haavoittuvuudet RC4 algoritmista löydettiin vuonna 2001 ja 2005. Haavoittuvuudet mahdollistivat löytämään käytetyn salausavaimen suuresta määrästä samalla avaimella salattuja viestejä, jos avain ja alustusvektori oli vain liitetty toisiinsa salausavaimen luomisessa. Huolimatta RC4:n tietoturvaa rikkovista hyökkäyksistä, RC4:ään perustuvia SSL- ja TLS-salausohjelmistoja pidettiin edelleen turvallisina ennen vuotta 2013 sen perusteella, miten niitä käytettiin SSL:ssä ja TLS:ssä. Vuonna 2011 RC4 suositeltiin BEAST-hyökkäyksen kiertäjäksi. Vuonna 2013 julkistetut uudet hyökkäysmuodot osoittivat lopullisesti TLS:n RC4:n rikkomisen toteutettavuuden. Hyökkäys vaatii 2^34 salattua viestiä RC4:n murtamiseen ​​TLS:ssä ja SSL:ssä. Hyökkäyksestä julkaistiin vuonna 2015 uusi versio, joka vaatii enää 2^26 viestiä salauksen murtamiseen. <https://beaglesecurity.com/blog/vulnerability/the-rc4-algorithm-in-transport-layer-security-and-secure-sockets-layer.html>

BEAST on selainhaavoittuvuus, joka julkaistiin vuonna 2011 ja se koskee SSL 3.0:aa ja TLS 1.0:aa. Hyökkääjä voi purkaa kahden osapuolen välillä vaihdetun tiedon salauksen hyödyntämällä haavoittuvuutta lohkosalauksen ketjuttamisen CBC (Cipher Block Chaining) toteutuksessa. Tämä on asiakaspuolen hyökkäys, joka on välimieshyökkäys. Hyökkääjä lisää paketteja TLS-virtaan. Tämän avulla he voivat arvata muokatun viestin kanssa käytetyn alustusvektorin ja siten purkaa salauksen. Jotta BEAST-hyökkäys onnistuisi, hyökkääjällä on oltava jonkinlainen hallinta uhrin selaimessa. Selainten modernit versiot estävät BEAST- hyökkäyksen. RC4 virtasalaus on immuuni BEAST-hyökkäykselle. Siksi RC4:ää käytettiin laajalti keinona lieventää BEAST-hyökkäystä palvelinpuolella ennen kuin se todettiin murretuksi vuonna 2013. <https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/>

POODLE on SSL 3.0 haavoittuvuus, joka johtuu BEAST:in tavoin lohkosalauksen ketjuttamisen CBC (Cipher Block Chaining) toteutuksesta. POODLE on välimieshyökkäys ja edellyttää, että hyökkääjän JavaScript -koodi ajetaan uhrin selaimessa. POODLE haavoittuvuus johtuu lohkosalausalgoritmien täytetarpeesta. Salattavan datan pitää olla lohkosalausalgoritmeissa lohkokoon monikerta, muuten viimeinen lohko pitää täyttää täytteellä loppuun. Jos data on täsmälleen lohkokoon monikerta, silloin viimeinen lähetettävä lohko on pelkästään täytettä. Palvelin hylkää pyynnön jos täyte on viallinen pyynnössä. POODLE hyökkäys perustuu siihen että hyökkääjä pääsee muokkaamaan lähetettävän pyynnön kokoa ja tietää siten pyynnön täytteen. Täytettävä sisältävä lohko on viimeisenä ja hyökkääjä vaihtaa viimeisen lohkon ja murrettavan lohkon paikkaa ja lähettää pyynnön useaan kertaan palvelimelle muuttaen pyynnön kokoa jokaisella pyynnöllä. Jos palvelin hyväksyy pyynnön, niin murrettavan lohkon bitit vastaavat täytelohkon täytettä vastaavia bittejä. Hyödyntämällä tätä haavoittuvuutta hyökkääjä saa murrettua SSL:n salauksesta lohkoja. Käyttäjän evästeiden varastaminen on yksi esimerkki onnistuneesta POODLE- hyökkäyksestä. Esimerkiksi 16 bittisen evästeen varastamiseen tarvitaan maksimissaan 4096 pyyntöä, jos hyökkääjä tietää tarkalleen missä kohtaa pyyntöä evästeet lähetetään. <https://www.acunetix.com/blog/web-security-zone/what-is-poodle-attack/> <https://www.acunetix.com/blog/articles/poodle-gives-final-bite-puts-sslv3-rest/>

DROWN on SSL 2.0 version haavoittuvuus, joka johtuu protokollan liian heikosta symmetrisestä salauksesta. Hyökkääjän on ensiksi kuunneltava useita satoja SSL-yhteyksiä uhrin ja palvelimen välillä. Yhteyksien tarvitsee jakaa jaettusalaus RSA-kättelyllä. Hyökkääjä lähettää palvelimelle muokattuja versioita uhrin salatuista RSA-kättelyistä. Tapa jolla palvelin vastaa kuhunkin näistä muokatuista kättelyistä riippuu siitä muodostuuko salatusta viestistä oikean muotoinen viesti salauksen purkamisen jälkeen. Salaus käyttää vain 40 bittistä avainta RSA-salauksessa joten hyökkääjä pystyy vertaamaan palvelimen vastausta kaikkiin mahdollisiin vastauksiin joita on 2^40 kappaletta. Tällä tavalla hyökkääjä pystyy päätteleen oliko muokattu versio salatusta kättelystä oikein muodostettu vai ei. <https://drownattack.com/>

Tiedossa olevat TLS-haavoittuvuudet koskevat pääasiassa vanhentuneita TLS-versioita, joiden käyttämistä ei enää suositella. RC4- salauksen haavoittuvuus koskee myös turvallista TLS 1.2 versiota. Kuva 20 esittää Acunetix Web Application Vulnerability Report- tutkimuksen tuloksia vuodelta 2020. Tutkimuksen on tehnyt Invicti niminen tietoturvayhtiö ja siinä on testattu 5000 palvelimen tietoturvaa. Tutkimukseksen perusteella palvelimet sallivat edelleen TLS-yhteyden muodostamisen protokollan vanhoilla versioilla. POODLE- hyökkäys edellytttää SSL 3.0 versiota ja DROWN puolestaan vanhempaa SSL 2.0 versiota. Silti palvelimista 3.9% oli haavoittuvia POODLE- hyökkäykselle ja 0.7% oli haavoittuvia DROWN- hyökkäykselle. BREACH- hyökkäys johtuu HTTP- protokollan pakkausalgoritmista ja sille oli haavoittuvia 3.9% palvelimista. 7.7% palvelimista salli symmetriseksi salaukseksi käyttää murrettua RC4- algoritmia. RC4- algoritmia on mahdollista käyttää kaikissa muissa versioissa paitsi TLS 1.3 versiossa. Sen käyttö pitäisi erikseen kieltää TLS- palvelimen asetustiedostossa, jotta haavoittuvuudelta vältyttäisiin. Lisäksi 30.7 % palvelimista salli TLS 1.0 version käyttämisen joka on haavoittuvainen muun muassa BEAST- hyökkäykselle.

https://www.acunetix.com/white-papers/acunetix-web-application-vulnerability-report-2020/

A pie chart with different colored sections

Description automatically generated

Kuva 20 Vuoden 2020 AWAVR- tutkimuksessa löydetyt TLS/SSL- haavoittuvuudet

Kuva 21 esittää F5 Labsin tekemän The 2021 TLS Telemetry Report- tutkimuksen tuloksia HTTPS- palvelimien tukemista SSL/TLS- versioista. Tutkimuksessa on käytetty miljoona suosittua HTTPS-sivustoa. Tutkimuksesta huomataan että SSL 3.0- version käyttämisen mahdollistaa noin muutama prosentti tutkituista palvelimista. Nämä palvelimet ovat todennäköisesti haavoittuvia POODLE- hyökkäykselle. TLS 1.0 tuki on noin 50% palvelimista, mikä on huomattavasti suurempi luku mitä Acunetix Web Application Vulnerability Report- tutkimukseksessa saatu 30.9% on. TLS 1.1 on muutaman prosentin tuetumpi versio mitä TLS 1.0- versio. Ehdottomasti tuetuin versio on TLS 1.2, joka saavuttaa melkein 100% tuen. Uusin versio 1.3 saavutti noin 60% tuen tutkimuksessa. Vaikka F5 Labsin tekemä tutkimus ei suoraan paljasta tietoturvahaavoittuvuuksia TLS- prokollassa, niin se antaa hyvän kuvan kuinka laajasti protokollan vanhentuneet tietoturvaongelmia sisältäviä versioita käytetään jotka mahdollistavat TLS-protokollaan kohdistettuja hyökkäyksiä. https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report

A graph of different colored lines

Description automatically generated

Kuva 21 HTTPS- palvelimien tukemat TLS- versiot vuonna 2021

Kun vertaillaan FTPS- ja SFTP-protokollia tietoturvan näkökulmasta, oleellisen asia on käytettävien porttien määrä. SFTP käyttää vain yhtä porttia tiedonsiirtoon, mutta FTPS tarvitsee kaksi. Lisäksi jokainen rinnakkainen yhteys tarvitsee oman porttinsa. Tämän takia FTPS -palvelimelle määritellään usein sallittujen porttien joukko palomuuriin. Palomuurit voidaan katsoa putkena, jotka määrittelevät sallitun ja kielletyn verkkotoiminnan yksityisessä verkossa. Palomuurit luovat tarkastuspisteitä verkkoliikenteen ohjaamiseksi, jolloin ne tarkistavat ohjelmoitujen parametrien perusteella verkkoliikennettä, ja toimivat niiden mukaisesti. Mitä vähemmän portteja on avattu, sitä vähemmän järjestelmässä on haavoittuvia kohtia. Jos käytössä on palomuuri, niin se pitää konffiguroida hyväksymään kaikki FTPS-palvelimen käyttämät portit, mikä lisää uhkia. Lisäksi salaus kiinnitetään FTPS -protokollaan TLS- tai SSL-tekniikoilla FTP -protokollan päälle. Tämän ulkoisen salauksen seurauksena palomuuriratkaisut eivät pysty havaitsemaan, mitä porttia käytetään ja miksi. Yksiporttiset SFTP-asetukset ovat ihanteellisia käytettäväksi palomuurin rinnalla. Se muodostaa yhden yhdistetyn yhteyden asiakkaan ja palvelimen välille. Palomuuri voi tarkkailla tämän yhteyden poikkeavuuksia, epäilyttäviä merkkejä ja muita uhkamerkkejä. [https://www.spiceworks.com/tech/networking/articles/sftp-vs-ftps/]

# yhteenveto

# Lähteet

1. <https://developer.mozilla.org/en-US/docs/Web/API/File>, viitattu 13.1.2021
2. <https://tools.ietf.org/html/rfc959>, viitattu 28.3.2021
3. <https://ieeexplore-ieee-org.libproxy.tuni.fi/stamp/stamp.jsp?tp=&arnumber=8068716>, viitattu 28.3.2021

Liite A: MS Wordin tekstityylien käyttö

Tyylien määrä pitää pyrkiä minimoimaan, jotta niitä on helppo käyttää ja muokata. Valitettavan monet kirjoittajat käyttävät Wordin tekstityylejä epäjohdonmukaisesti. Yleisesti ottaen tekstiä ei kuulu asemoida lisäämällä rivin- tai sivunvaihtoja, vaan nämä asiat tulee säätää tyyliasetuksista. Samoin kuvien, taulukoiden tai viitteiden numeroiminen käsin johtaa ongelmiin. Yleinen ongelmien lähde on tekstin kopioiminen muista dokumenteista. Usein teksti kannattaa tuoda ilman muotoiluja *Paste > Paste special > Unformatted text*, sillä muuten tyylien määrä räjähtää. Tekstin ulkoasun muokkaaminen näennäisen kätevästi Wordin yläreunan nappuloilla johtaa usein samankaltaisiin ongelmiin. Dokumentin sisällä tekstiä kopioidessa tai siirtäessä muotoilun voi toki säilyttää.

Avaamalla *Apply Styles*, näet käytetyt ja mahdolliset tekstityylit, missä niitä on käytetty ja voit muokata niitä tai luoda uusia. Valitsemalla vielä alareunasta *Options…* voit valita näytetäänkö todella käytetyt vai tässä dokumentissa mahdolliset. Lisäksi voi koiruuttaan valita kaikki Wordin tarjoamat, mutta tiettävästi kukaan ei ole vielä tarvinnut esimerkiksi 9. tason otsikoita. Lisäksi voit valita näytetäänkö pelkät kappaleasetukset (esim. *Normal*, *BibItem*…), fonttimuotoilut (*Normal + Italic*) ja listamuotoilut.

Toisinaan tekstin sekaan jää roikkumaan esimerkiksi lihavoituja kuvia, kursivoituja 10 pisteen rivinvaihtoja tai peräti punainen rivinvaihto jne. Jokaisesta listatusta tyylistä voi onneksi valita *Select all X instances*, jolloin näkee missä sitä on käytetty. Näin on helppo päästä eroon kummallisuuksista.. Esimerkiksi TTY:n opinnäyteohjeessa oli versiossa 11.3 (Kuva 1a) yli 130 tyyliä, mm. liki identtiset *Body text*, *Body text+Not italic, Normal*, *Plain text*, *After:6pt*, *After:6pt,Line spacing:Single*, *Before:1pt*, *Before:1pt,After:0pt*, *Before:3pt* ja niin edelleen. Tyylien määrä puolittui, kun nuo korvattiin yhdellä tyylillä *Normal*  ja muitakin turhia poistettiin (Kuva 1b).

Lisäksi voit valita muotoilumerkit näkyviin (engl. *Show paragraph marks and other hidden formatting symbols)* painamalla täytetyn P-kirjaimen peilikuvan näköistä ikonia yläpalkissa. Pedantit ihmiset voivat sitten poistaa tuplavälilyönnit, ylimääräiset rivinvaihdot, rivinvaihtoa edeltävät välilyönnit, ylimääräiset tabulaattorit ja muut roskat. Tarkista myös tekstin kopioinnista huolimatta kaikki lainausmerkit ovat samanlaiset, mielellään “…” eikä "…".

|  |
| --- |
|  |
| *a) Versio 11.3. Kiinnitä erityishuomio lukuisiin Arial-pohjaisiin tyyleihin, numeroituihin ja muihin listoihin sekä vasemmalle tasattuihin teksteihin.* |
|  |
| *b) Versio 11.5. Edelleen otsikkotasot 4-9 ovat tarpeettomia, mutta valitettavan vaikeita poistaa.*   1. Tyylit kirjoitusohjeen versioissa 11.3 ja 11.5 |

Jos dokumentin tulostaminen pdf-tiedostoon heikentää kuvien laatua, valitse asetus *High Quality Printing* eikä S*tandard*, tai käytä *File* > *Save as Adobe PDF*.

Muotoiluasetukset ovat muuttuneet Wordin versiossa 2013, mikä saattaa aiheuttaa ongelmia otsikoita ympäröivän tyhjän tilan kanssa ellei käytössä ole *Compatibility mode*, katso esimerkiksi https://answers.microsoft.com/en-us/office/forum/office\_2013\_release-word/where-can-i-find-suppress-extra-line-spacing-at/70bf7ca3-a884-40c4-ab59-34d2a04a1a8f