TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JARKKO VIRTANEN

SOVELLUSTASON AUTENTIKOIVA AVAINTENVAIHTO WEB OF THINGS -YMPÄRISTÖSSÄ

Diplomityö

Tarkastaja: Prof. Jarmo Harju ja DI Joona Kannisto
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 15.1.2014

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
JARKKO VIRTANEN: Sovellustason autentikoiva avaintenvaihto Web of Things -ympäristössä
Diplomityö, 63 sivua
Lokakuu 2014
Pääaine: Tietoliikenneverkot ja protokollat
Tarkastaja: DI Joona Kannisto, Professori Jarmo Harju
Avainsanat: AKA, Digest, Yhdyskäytävä, HTTP, COAP, JSON, autentikaatio, JWT, JWS, JWE, tietoturva, RESTful, Web of Things

Web of Things -ympäristöjen (WoT) hypertekstimäisen sisällön tuomiseen ympärillemme oleville laitteille kustannustehokkaasti tarvitaan resurssitehokkaita ratkaisuja. Resurssiongelmia helpottamaan on kehitetty resurssirajoitteisille optimoitu tiedonsiirtoprotokolla Constrained Application Protocol (CoAP). CoAP perustuu RESTful-arkkitehtuuriin, joka on kehitetty Hypertext Transfer Protocol (HTTP) -protokollan arkkitehtuurista. Yhteinen arkkitehtuuripohja mahdollistaa edeltävien protokollien välillä tiedonsiirron yhdyskäytävän avulla. HTTP - ja CoAP -protokollien välinen tiedonsiirto on tarpeen WoT-ympäristöjen yleistymisessä, koska suurin osa verkon palvelimien toiminnasta perustuu HTTP-protokollaan. CoAP-asiakkaan tietoturvallisen yhteydenoton HTTP-palvelimelle mahdollistavasta sovellustason yhdyskäytävästä ei kuitenkaan ole toteutusta.

Työssä esitellään edellä mainitut protokollat ja kuvataan niiden keinoja tietoturvasta huolehtimiseen ja soveltuvuuteen toimia yhdyskäytävän läpi. Pyrkimyksenä työssä on toteuttaa turvallinen autentikointi sekä tiedonsiirto asiakkaalta palvelimelle ja takaisin yhdyskäytävän läpi. Reunaehtoina ovat resurssirajoitteisten laitteiden vaatima yksinkertaisuus ja pysyminen kerrosmallien sovellustasolla.

Työssä tutkitaan JavaScript Object Notation (JSON) -notaation ja sen tietoturvaksi kehitetyn JSON Web Token (JWT) -esitysmallin soveltuvuutta edellä selvitettyihin tarpeisiin. Autentikointi toteutetaan 3rd Generation Partnership Project (3GPP):n yleistä autentikointiarkkitehtuuria (GAA) käyttäen, jolloin avaimet saadaan SIM-operaattorilta HTTP-Digest-autentikaatioon perustuvalla AKA-Digest-autentikointitavalla.

Suunnitelmien toimivuuden ja tulevien ongelmien havainnoimiseksi toteutettiin valittuja tekniikoita käyttäen demonstraatio, jonka komponentteja olivat CoAP-asiakas, yhdyskäytävä ja HTTP-palvelin. Toteutukset tehtiin Java-kielellä pyrkien käyttämään valmiita ohjelmistokirjastoja mahdollisuuksien mukaan. Toteutusvaiheessa havaitut ongelmat tulivat Digest-autentikaatio-ohjelmistoista, hajallaan olevasta dokumentaatiosta ja uusien ohjelmistokirjastojen keskeneräisyydestä. Digest-autentikaatiota toteuttavat ohjelmistot toimivat epästandardeilla tavoilla. 3GPP-dokumentit oli kohdennettu organisaatioille ja siten vaikeaselkoisia yksittäiselle lukijalle.

Demonstraatiototeutus havaittiin toimivaksi WoT-ympäristöissä. Toteutuksen ja testauksen aikana syntyi erilaisia ideoita, joiden pohjalta toteutus on jatkokehitettävissä todellisiin ympäristöihin.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Programme in Information Technology
VIRTANEN JARKKO: Authentication and key agreement on the application layer in the Web of Things environment
Master of Science Thesis, 63 pages
October 2014
Major subject: Communication Networks and Protocols
Examiner: M.Sc Joona Kannisto, Prof. Jarmo Harju
Keywords: AKA, Digest, Proxy, HTTP, COAP, JSON, authentication, JWT, JWS, JWE, RESTful, Security, Web of Things

Web of Things treats the Internet of Things objects in a manner compatible with the Web, and brings the content from the Web to these devices and vice versa. Web means a system with interlinked mostly hyper textual content all around. Unfortunately, existing protocols that are used to build the Web are not always suitable for constrained devices. One solution for that is the new Constrained Application Protocol, which is designed for constrained devices.

CoAP is based on RESTful architecture which is designed to contain the most desirable features of HTTP, and that is why mapping between HTTP and CoAP protocols should be possible. Mapping is critical for the WoT, because most of the Web servers support only HTTP protocol. However, CoAP and HTTP features are different, which complicates the translation. For instance, there is no commonly agreed way for a CoAP client to do authentication and secured connection to authenticated HTTP server through a transparent proxy server.

The thesis describes main Web related protocols and explains how those protocols handle authentication and secure connections. Aim is to figure out if it is possible to use some established way to secure connection through CoAP client to HTTP server or if not what kind of way is best solution to do that. The chosen restrictions require paying attention to constrained devices and staying on the network application layer.

The experimental part of the thesis investigates the use of JavaScript Object Notation, and JSON Web Token model to secure traffic. With SIM cards there is 3GPP Generic Authenticate Architecture solution, which offers a lightweight solution for strong authentication. GAA solution uses the SIM operator as a trusted party and the operator gives keys to the client and the server using Generic Bootstrapping Architecture (GBA).

The implementation consisted of a CoAP client, an HTTP server and a proxy between them to test how the proposed solution would work. The plan was to use existing software libraries as much as possible. During the work we faced problems which were caused by new unstable software libraries, scattered documentation and Digest software that did not follow standards. 3GPP documents were clearly made for organizations and did not include instructions how to make solution that use 3GPP architectures.

Furthermore, the conducted study gave many ideas for future development. After testing the coded demonstration application, the approach was seen to work well.

ALKUSANAT

Tämä diplomityö on tehty talven 2013–2014 aikana Tampereen teknillisen yliopiston tietotekniikan laitokselle. Työssä mahdollistettiin resurssirajoitteisten Web of Things -ympäristön laitteiden turvallinen autentikointi ja tiedonsiirto protokollien CoAP ja HTTP välillä.

Työ kasvatti ohjelmointiosaamistani, sekä laajensi ymmärrystäni protokollista, autentikaatio ja salaustavoista. Autentikointitavoista varsinkin SIM-pohjainen AKA oli minulle hyppy tuntemattomaan.

Työ ei olisi syntynyt ilman yliopiston antamaa hyvää koulutusta, ohjausta ja resursseja mm. työpisteen osalta. Toteutuksen tekninen osuuden tekeminen mahdollistui lukuisten avointen sovellusten ansiosta. Työn testauksesta tuskin olisi tullut mitään ilman Ericssonin tarjoamaa BSF-palvelinta ja tukea sen käyttöön.

Tärkeimmät kiitokset menevät ohjaajilleni DI Joona Kannistolle ja professori Jarmo Harjulle. Ohjaajiltani sain idean mielenkiintoiselle työlleni ja he ovat sen jälkeen antaneet hyviä neuvoja ja tukea työn eteenpäin viemiseksi, mitä ilman työ tuskin olisi tässä pisteessä. Lisäksi haluan kiittää ystäviäni ja läheisiäni keskusteluista, tuesta ja ohjeista, jotka auttoivat minua monessa vaiheessa.

Tampereella 17.9.2014.

Jarkko Virtanen

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

2G	Second generation of mobile telecommunications technology.
3G	Third generation of mobile telecommunications technology.
3GPP	The 3rd Generation Partnership Project.
AKA	Authentication and Key Agreement.
AKA-Digest	Hypertext Transfer Protocol (HTTP) Digest Authentication Using Authentication and Key Agreement.
AK	Anonymity key.
AMF	Authentication Management Field.
AUTN	Authentication Token.
DTLS	Datagram Transport Layer Security.
BSF	A Bootstrapping Server Function.
B-TID	A bootstrapping transaction identifier.
CK	A cipher key.
CoAP	Constrained Application Protocol.
FC	Carrier Centre Frequency.
GAA	Generic Authentication Architecture.
GBA	Generic Bootstrapping Architecture.
HSS	Home Subscriber Server.
HTTP	Hypertext Transfer Protocol.
HTTP-Basic	Basic access authentication.
HTTP-Digest	Digest access authentication.
IK	The integrity key.
IoT	The Internet of Things.
IMPI	IM Private Identity.
JSON	JavaScript Object Notation.
JWA	JSON Web Algorithms.
JWE	JSON Web Encryption.
JWS	JSON Web Signature.
JWT	JSON Web Token.
KDF	Key Derivation Function.
Ks	Concatenation of session keys (i.e. Ks = IK||CK).
Ks_NAF	KDF (Ks, key derivation parameters).
M2M	Machine to Machine.
MAC-A	Network Authentication Code.
MAC-S	The message authentication code.
NAF	Network Application Function.
SIM	Subscriber Identity Module.
OP	Operant Variant Algorithm Configuration Field.

PKI	A public key infrastructure.
RES	Response to Challenge.
REST	Representational State Transfer.
SQN	Sequence Number.
TCP	The Transmission Control Protocol.
TLS	Transport Layer Security.
TMPI	Temporary identity.
UDP	User Datagram Protocol.
UE	User Equipment.
UICC	Universal integrated circuit card.
URI	Uniform Resource Identifier.
WoT	The Web of Things.
XML	Extensible Markup Language.

1 JOHDANTO

Tämä työ on syntynyt tarpeesta tehdä demonstraatiototeutus vähätehoisten Web of Things -ympäristöjen CoAP-protokollaa [1] käyttävien laitteiden HTTP-yhteensopivasta [2] yhteydenotosta ja vahvasta autentikoitumisesta. Näihin on olemassa useita ratkaisuja, jotka sijoittuvat eritasoille protokollien kerrosmallissa. Usean protokollatason käytöstä voi seurata ongelmia, joten tavoitteena on pysyminen sovelluskerroksella ja ratkaista siitä syntyvät ongelmat käyttäen sovellustason erilaisia mekanismeja.

Verkkoprotokollia voidaan verrata tiimalasimalliin, missä siirtotasolla ja sovellustasolla on useita protokollia ja siinä välillä vain muutama protokolla. Sovellustason protokollista kuitenkin dominoivaan asemaan on noussut hypermedian mahdollistava HTTP-protokolla [2]. Yleistymisen syinä on ihmisläheinen selkokielinen syntaksi ja yksinkertaisuus. Yksinkertaisuutta tukee mm. liikenteen eheyden valvonnan jättäminen alemmille protokollille. HTTP on monipuolinen protokolla ja sisältää mm. erilaisia tapoja käyttäjän tunnistamiseen. Protokollana se ei ole kuitenkaan ongelmaton esim. resurssirajoitteisille laitteille. Selkokielisyys vie siirtotieltä ylimääräistä kaistaa ja protokollan otsikon optioiden rajaton määrä ja niiden käsittely voi aiheutua ongelmiksi resursseiltaan rajoittuneille laitteille.

Yhtenä lähestymistapana ongelmiin on ollut uuden CoAP (Constrained Application Protocol)-protokollan [1] kehittäminen. CoAP-protokollaa kehitettäessä on alusta alkaen pystytty huomioimaan HTTP:n käytöstä syntyneitä ongelmia resurssirajoitteisille laitteille. Soveltuvuutta resurssirajoitteisille laitteille on saavutettu mm. siirtymällä binäärimuotoiseen otsikkorakenteeseen ja sisäänrakennetulla verkon resurssien selvityksellä. Yhtenä sen suunnitteluominaisuutena on ollut mahdollisuus konvertoida liikennettä HTTP- ja CoAP -protokollien välillä.[1]

Tähän mennessä CoAP-laitteelta yhteydenoton autentikaatiota vaativalle HTTP-palvelimelle mahdollistavasta yhdyskäytävästä ei kuitenkaan ole olemassa toteutusta. Tämä johtuu mm. CoAP-protokollasta puuttuvasta sisäänrakennetusta tuesta sovellustason autentikaatiolle. Autentikoinnin mahdollistavan yhdyskäytävän tarve on kuitenkin suuri johtuen CoAP-protokollan tuoreudesta verrattuna HTTP-protokollaan. Suurin osa verkon palvelimista perustuu HTTP-protokollaan, eikä niihin voida odottaa CoAP-toteutusta.[1][3][4]

Resurssirajoitteisilla laitteilla yhdeksi ongelmaksi muodostuvat laskentaa vaativat operaatiot, kuten autentikaatio- tai salaus -operaatiot. Tapoja toteuttaa edellisiä on symmetristä ja epäsymmetristä, joista varsinkin jälkimmäistä pidetään laskennallisesti vaativampana. Vahvoista autentikointitavoista suuri osa perustuu epäsymmetriseen laskentaan. Autentikointiin ja avaintenvaihtoon 3GPP on kehittänyt symmetrisen Generic Authentication Architecture [5] ratkaisun, joka käyttää 3G-operaattoria luotettuna lähteenä. Ratkaisu hyödyntää HTTP-Digest-autentikaatiota [4] käyttäjän tunnistamiseen. [6][7][8]

Työssä kirjallinen osuus on jaettu tämän johdannon jälkeen kuuteen isompaan kokonaisuuteen. Toisessa Web Of Things -luvussa avataan toimintaympäristöä ja siihen liittyviä asioita RESTful-arkkitehtuurista Internetin mediatyyppien kautta tietoturvaan. Työn protokollatason perustuessa sovellustasoon käsitellään kolmannessa luvussa sovellustason mahdollisuuksista vaikuttaa tietoturvaan, tärkeimpinä autentikointi, salaus ja eheys. Neljännessä luvussa kerrotaan miten 3GPP:n autentikointiratkaisu toimii ja miten se eroaa HTTP:n Digest-autentikaatiosta. Viidennessä luvussa kuvataan toteutus. Kuudennessa luvussa mietitään mahdollisia jatkokehityskohteita työhön. Viimeisessä yhteenveto luvussa käydään läpi työn kulku ja keskeiset tulokset.

2 WEB OF THINGS

Web of Things on Internet of Things -käsitteen pohjalta ideoitu käsite. Internet of Things -käsite on alun perin Kevin Astonin määrittelemä [9], missä jokaiseen objektiin on liitetty maailmanlaajuisesti yksilöivä RFID-tunniste. Käsitteen merkityksestä ei ole kuitenkaan olemassa mitään standardia määritystä. Astonin mukaan ihmiset eivät käytä ideoita tai informaatiota vaan bittejä, jotka ovat jotain konkreettista, eli things (asia esine). Suomennoksena voisi olla asioiden tai esineiden muodostama verkko. Yleisestä ottaen käsitteessä nähdään verkon olevan integroituna kaikkialla ja siten jatkuvasti käytettävissä sekä muokattavissa. Verkon kautta pystytään siten saamaan oikeaa ja jäsenneltyä tietoa tarpeisiimme missä ikinä olemmekaan. [10][9][7]

Web of Things poikkeaa edellisestä käsitteestä ajatuksella, missä verkon hypertekstimäinen sisältö tuodaan ympärillämme oleville laitteille käyttämällä sekä uusia, että olemassa olevia erilaisia tekniikoita ja verkon protokollia. Pyrkimyksenä on saada sulautetut laitteet ja erilaiset älylaitteet kytkettyä verkkoon mahdollisimman pienin muutoksin. Näin laitteet voivat lukea muiden hypertekstimäistä sisältöä ja tarjota muille laitteille omaa sisältöään, kuten reaaliaikaista tietoa omasta toiminnastaan. Verkon laitteiden antaessa selattavan käyttöliittymän, niitä pystytään myös hallinnoimaan nykyistä helpommin. Ratkaisuina Web of Things -käsitteen toteuttamiseen on pidetty SOAP-protokollaa [11] ja HTTP-protokollan pohjalta kehitettyä REST (Representational State Transfer) -arkkitehtuuria [12]. Edeltävistä SOAP on palveluorientoitunut ja REST on resurssiorientoitunut. Käyttäjämäärällisesti suhteutettuna SOAP-protokolla on paljon rajoittuneempi verrattuna REST-arkkitehtuureita käyttävien protokollien käyttäjämääriin. REST:in käyttäjämäärän ja mukautuvuuden avulla erilaisille alustoille, sitä voidaan pitää soveltuvana ja siten käytettäväksi pohjana luotaessa WoT-ratkaisuja.[11][13][10]

Web of Things -käsitteen yhtenä perustana on laitteiden välinen liikenne, joten se vaatii laitteiden keskustelemista toistensa kanssa ihmisten puuttumatta toimintaan. Kyseisen kaltaisen toiminnan kuvaamiseen on kehitetty termi Machine to Machine (M2M) [14]. Termi esiteltiin alun perin mobiiliverkon laitteiden yhteistoiminnan kuvaamiseen. Terminä M2M on kuitenkin yleispätevä kuvaamaan kaikenlaisten laitteiden keskinäistä tiedonsiirtoa ilman ihmisten väliintuloa missään vaiheessa.[14]

2.1 RESTful-arkkitehtuuri

RESTful on alun perin Roy Fieldingin [12] määrittelemä arkkitehtuuri. Arkkitehtuurin nimi tulee sanoista ”Representational State Transfer”. Se kehitettiin rinnakkaisesti RFC-2616 HTTP 1.1-protokollan [2] kanssa. Pohjaksi otettiin HTTP 1.0 -protokolla RFC-1945 [3] ja RFC-2396 Uniform Resource Identifiers (URI) [15]. REST-arkkitehtuurin määrittelyllä pyrittiin varmistamaan HTTP-protokollassa havaittujen hyvien ominaisuuksien säilyminen tulevissa HTTP:n versioissa. Samalla myös mahdollistettiin hyvien ominaisuuksien mukaan tuominen muiden protokollien suunnitteluun. Arkkitehtuurin ominaisuuksiksi tulivat HTTP-protokollasta asiakas-palvelin-malli, tilattomuus, välimuistinkäyttömahdollisuus, kerrostettu palvelinarkkitehtuuri ja optiona toiminnollisuuksien laajentaminen asiakkaan suorittamalla ohjelmistokoodilla. URI-standardista määrittelyyn tuli yhdenmukainen rajapinta. RESTful-arkkitehtuuriin perustuvat protokollista nykyisin mm. HTTP ja CoAP.[12][16]

Asiakas-palvelin-mallissa on erotettu asiakas ja palvelin toisistaan, jolloin niiden toiminta ei ole toisistaan riippuvaista. Riippumattomuudesta saavutetaan erilaisia etuja, joista seuraavaksi muutama. Ensimmäisenä tulee mahdollisuus tehdä molempiin muutoksia vaikuttamatta toisen toimintaan. Toisena poistuu asiakkaalta tai palvelimelta tarve pitää kirjaa toisistaan, jolloin kummankaan lukumäärällä ei ole suoraa vaikutusta toisen toimintaan. Mallissa palvelin yksinkertaisesti vastaa asiakkaalta tulevaan resurssipyyntöön resurssiensa puitteissa.[12]

Tilattomuudessa käytetyn protokollan tasolla ole sisällytettynä mitään tilaa asiakkaan tai palvelimen osalta. Protokollalla lähetetty pyyntö sisältää kaikki tarvittavat tiedot vastauksen tekemiseen. Palvelimen tietojen pysyessä samana asiakkaan toistaessa resurssipyyntönsä tulisi vastauksen olla identtinen edelliseen verrattuna.[12]

Välimuistinkäyttömahdollisuus tarkoittaa sitä, että asiakas voi tallentaa saamansa vastaukset muistiin. Vastauksia ei tarvitse näin hakea jatkuvasti uudestaan ja asiakas voi ryhtyä toimimaan palvelimena, joka välittää resursseja eteenpäin. Palvelimelle välimuistinkäyttömahdollisuus asettaa vaatimuksen merkata tiedon säilymisaika, jotta asiakas pystyy pitämään tiedon validina päivittämällä riittävän usein.[12]

Kerrostetussa palvelinarkkitehtuurissa useat palvelimet voivat jakaa tekemiään tehtäviä. Asiakkaalla ei ole yhteysosoitteen lisäksi tietoa resurssipyynnön vastaanottajasta tai siihen vastaavasta palvelimesta.[12]

Yhdenmukainen rajapinta tarkoittaa RESTful-arkkitehtuurissa palveluiden ajattelemista resursseina, joihin viittaa aina yksikäsitteinen URI. Yhdenmukainen rajapinta pitää sisällään määritteet resurssin identifiointi, resurssin manipulointi URI:in kautta, itsestään kuvaavat viestit ja hypermedian toimiminen ohjelmistotasolla.[12]

Resurssien identifiointi jakaa resurssin ja sen palauttaman sisällön kahdeksi eri asiaksi. Resurssi on asia, mihin otetaan yhteyttä URI:n kautta ja resurssi hallinnoi esim. tietokantaa. Resurssi ei kuitenkaan palauta hallinnoimaansa tietokantaa, vaan tekee siitä esim. XML [17]- tai JSON [18] -esitysmallin, minkä resurssi palauttaa asiakkaalle. Näin resurssi identifioi hallinnoimaansa tietoa.[12]

Resurssien manipulointi tarkoittaa resurssien salliessa ja tukiessa mahdollisuutta muokata resurssin sisältöä. Manipulointi tapahtuu resurssin käyttämien esitysmuotojen kautta määritellyillä metodeilla, joita esim. HTTP:n tapauksessa ovat GET, POST, PUT ja DELETE. Resurssit voivat kuitenkin määritellä metodeista mitä ne toteuttavat.[12]

Itseään kuvaavat viestit tarkoittavat jokaisen viestin sisältävän riittävän paljon tietoja itsestään oikean tulkinnan mahdollistamiseksi. Keinoina oikean tulkinnan saavuttamiseen ovat mm. Internetin mediatyypit [19], eli MIME-tyypit. MIME-tyyppi määritteellä kerrotaan tavat tulkita sisältö. Viestin vastaanottava ohjelma vertaa saamansa viestin tyyppiä tukemiensa MIME-tyyppien listaan ja löytäessään vastaavuuden, suorittaa viestin sisällön vastaavuuden kertomien ohjeiden mukaisesti.[12]

Hypermedian toimiminen ohjelmistotasolla tarkoittaa oletusta, missä saatu hypermediasivu sisältää validit uudet yhteysosoitteet määritellyksi ajaksi. Tällöin asiakkaalla ei ole tarvetta tietää osoitteita lukuun ottamatta yksinkertaisia osoitteita tunnettuihin resursseihin. Asiakas ei voi olettaa aikaisemmin palvelimelta saatujen osoitteiden olevan voimassa myöhemmin määritellyllä hetkellä.[12]

Asiakkaan päässä suoritettavalla ohjelmistokoodilla tarkoitetaan palvelimen mahdollisuutta lähettää asiakkaalle suoritettavaa koodia. Näin voidaan laajentaa asiakkaan mahdollisuuksia käsitellä tietoa. Asiakkaan suorittaessa palvelimelta saatua koodia palvelimen sijaan voidaan täten vähentää palvelimen laskentaa, sekä olla vähemmän riippuvaisia verkon toiminnasta.[12]

2.2 HTTP-protokolla

HTTP (Hypertext Transfer Protocol) [2] on hypertekstin siirtoon suunniteltu sovellustason protokolla. HTTP-protokolla ei ole riippuvainen kuljetuskerroksen protokollista, mutta se kuitenkin olettaa käytettävän luotettavaa protokollaa, jolloin protokolla varmentaa paketin läpimenon. Luotettavan protokollan vaatimuksesta johtuen sitä yleensä käytettään TCP-protokollan päällä. TCP-protokollan avulla voidaan tehdä HTTP-protokollasta puuttuvaa vuonhallintaa ja eheystarkistuksia. HTTP-protokollaa käytetään yleisesti IANA:n [20] sille rekisteröimän portin 80 kautta.[2][20]

Liikenne perustuu asiakaan lähettämään pyyntöön palvelimelle, mihin palvelin vastaa tietojensa pohjalta. HTTP:sta on olemassa 1.0 [3]- ja 1.1 [2] -standardit. Suurimpana muutoksena jälkimmäisessä on mahdollisuus uudelleen käyttää luotuja TCP-yhteyksiä, jonka avulla voidaan vähentää latenssia.[2][3]

HTTP-pyyntö ja vastaus rakentuvat kahdesta osasta, joista ensimmäinen on HTTP-otsikko (header) ja jälkimmäinen hyötykuorma, eli runko (body). Osat ovat tyypiltään tekstikenttiä. Header-kentän muoto riippuu asiakas-palvelin-mallin mukaan, onko lähettäjänä asiakas vai palvelin. Asiakkaan tapauksessa header-kenttä alkaa resurssin manipulointimetodilla, jota seuraa resurssin URI ja käytetty HTTP-standardi. Palvelin vastaa pyyntöön kertomalla käyttämänsä HTTP-standardin ja sen jälkeen vastauksen tilakoodin ja sen tekstimuotoisen kuvauksen. Header-osa sisältää edellisten jälkeen attribuutteina nimi-arvo-pareja, kuten Content-Type ja Content-Length. Content-Type kertoo sisällön MIME-tyypin ja Content-Length sisällön pituuden. Nimi-arvo-parien pohjalta reagoidaan HTTP-viestiin ja tulkitaan varsinainen pyydetty resurssi. Header-kenttään sisällytettyjen pyyntö/vastaus-attribuuttien määrää ei ole rajoitettu. Attribuutteja voidaan luoda mielivaltaisesti, mutta standardissa on määritelty yleisimmät. Header-kentän maksimikokoa ei ole määritetty, kuitenkin yleensä tietoturvasyistä sitä käsittelevät ohjelmat määrittävät kentälle jonkun maksimikoon. Runko-osa on hyötykuormalle ja siinä kuljetetun tiedon muotoa ei ole rajoitettu mitenkään.[2][21]

Resurssien pyyntö- ja manipulointi -metodeita voidaan määritellä vapaasti, mutta HTTP 1.1 -standardiin on määritelty metodit OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE ja CONNECT. OPTIONS-metodilla voidaan tehdä kysely resurssin ominaisuuksista. GET-metodilla pyydetään resurssia palauttamaan tietonsa. HEAD-metodi toimii samoin kun GET, mutta sillä pyydetään vain otsikko-osa. POST-metodilla voidaan täydentää resurssin tiedot, eli esim. resurssi palauttaa tyhjän lomakkeen ja asiakas täydentää siihen tiedot ja palauttaa lomakkeen resurssille POST-metodilla. PUT-metodilla täydennetään olemassa oleva resurssi tai luodaan uusi resurssi annetun URI:n paikalle, jollei sitä vielä ole olemassa. DELETE-metodilla voidaan tuhota olemassa oleva resurssi. TRACE-metodi palauttaa annetun pyynnön takaisin asiakkaalle, jolloin pystytään havainnoimaan paketin pyyntöketju. CONNECT-metodilla asiakas voi pyytää yhdyskäytävää vaihtamaan liikenne dynaamisesti tunneliin, jolloin mahdollistuu TLS-yhteydet. Metodeista GET ja HEAD on määritelty turvallisiksi metodeiksi, eli niitä käyttämällä ei voi vaikuttaa resurssien toimintaan.[2]

Statuskoodeja HTTP-protokollaan on määritelty useita. Koodit ovat kolmenumeroisia, joista ensimmäinen määrittää statuksen tyypin ja jälkimmäiset kaksi tarkentavat tyyppiä. Ykkösellä alkavat ovat informatiivisia, kuten 100 on Continue. Kakkosella alkavat kuvaavat toiminnan hyväksymistä kuten 200 on OK ja 202 on Accept. Kolmosella alkavat merkitsevät uudelleenohjausta kuten 301 on Moved Permanently. Nelosella alkavat merkitsevät virhettä asiakkaanpyynnössä kuten 401 on Unauthorized ja 403 on Forbidden. Numerolla viisi alkavat ilmaisevat palvelimen virhettä kuten 500 on Internal Server Error.[2]

2.3 CoAP-protokolla

CoAP (Constrained Application Protocol) [1] on resurssirajoitteisille, eli pienitehoisille ja vähävirtaisille laitteille kehitetty sovellustason protokolla. Protokollan kehityksen lähtökohdiksi on otettu pyrkimys mahdollisimman tehokkaaseen laitteiden väliseen kommunikointiin paikoissa, joissa ei välttämättä löydy kiinteää sähkö- tai tietoliikenneverkkoa.[1]

CoAP perustuu IETF RESTful-arkkitehtuuriin [12] samoin kuten HTTP-protokolla. Protokollana CoAP on hyvin lähellä HTTP:tä ja siten protokollan vaihto HTTP:hen liikenteen aikana on usein mahdollista pienin muutoksin ja rajoitteineen. Rajoitteet tulevat mm. HTTP:n laajemmasta tuesta erilaisille asioille, kuten autentikoinnille. Tällöin autentikointi parametreja ei voida suoraan siirtää CoAP-protokollan kuljetettavaksi.[1][2]

Kuljetuskerroksen protokollana CoAP:in alla on suunniteltu käytettäväksi UDP-protokollaa. UDP on TCP:tä kevyempi, eikä myöskään vaadi verkkoa kuormittavaa tietoa viestien perillemenosta, mikä esim. antureiden lähettämän liikenteen osalta on harvemmin tarpeen. Oletusporttina CoAP-protokollassa on käytössä 5683.[1][20][22]

Tärkeimpänä kehityskriteerinä CoAP-protokollassa on ollut hitaiden verkkojen siirtokapasiteetin maksimointi hyötykuormalle. Keinoina maksimointiin ovat olleet mm. otsikkotietojen pakkaaminen binääriseen muotoon ja pyrkimys eroon pakettien pirstaloitumisesta. Pakettien pirstaloituessa joudutaan aina lisäämään tietoa, miten ne kasataan uudelleen.

Protokollaan on myös lisätty asioita, joita ei HTTP-protokollasta löydy. Näitä asioita ovat ryhmälähetys, sisäänrakennettu resurssienselvitys ja asynkroninen tiedonsiirto. Ryhmälähetyksessä tieto voidaan kerralla lähettää kaikille verkosta löytyville laitteille. Näin voidaan esim. kerralla pyytää kaikilta verkon laitteilta tilannetieto. Resurssien selvityksellä voidaan taas nopeasti tarkistaa mitä resursseja verkosta löytyy. Resurssien selvitys on tarpeen, koska resursseiltaan rajoittuneet laitteet voivat olla vain hetken aktiivisia tai laitteiden välinen verkko voi kadota aina välillä. Asynkroninen tiedonsiirto on myös verkon resurssien satunnaisesta aktiivisuudesta johtuen tarpeen. Asynkronisuudella pystytään näin jossain määrin hallitsemaan tilanteita, missä ollaan lähettämässä tietoja laitteelle, joka voi hetkellisesti kadota verkosta.[1]

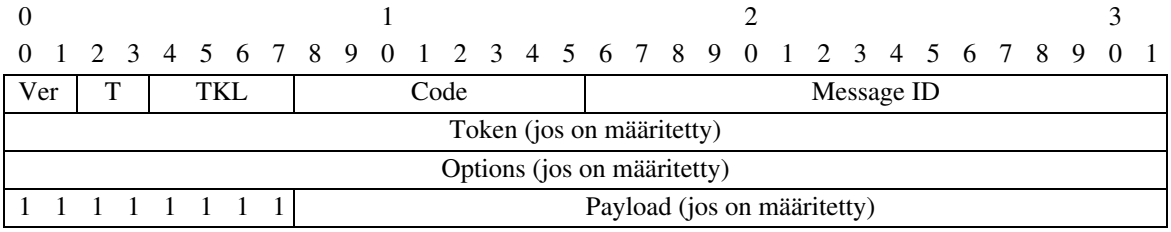
CoAP-resurssit rakentuvat pääresurssin "/.well-known/core" ympärille. Tehtäessä resurssipyyntö kyseiseen resurssiin, se palauttaa kaikki resurssit, jotka ovat liitetty siihen. Näin mahdollistetaan hakutoiminto resurssien kesken. Pääresurssiin voidaan myös liittää muiden palvelimien pääresursseja, jolloin kyselyllä resurssiin palautetaan usean pääresurssin tiedot.[23]

CoAP-protokollan tärkeimmät suunnitellut ominaisuudet ja vaatimukset ovat lueteltu alla:

- Täyttää M2M-tiedonsiirron vaatimukset.
- UDP-protokollan käyttö ja siihen optionalisesti tuki unicast- ja multicast -viesteille.
- Asynkronisesti tapahtuva viestien välitys.
- URI:n ja sisältötyyppien tuki.
- Välityspalvelimien ja välimuistin käyttömahdollisuus.
- Mahdollisuus vaihtaa tilattomasti protokollien HTTP ja CoAP välillä.
- Turvallisuuspuoli voidaan toteuttaa käyttäen IETF RFC-6347 [24] määrittelemää ”Datagram Transport Layer Security” (DTLS) -tekniikkaa hyväksikäyttäen.[1][24]

Valitusta UDP-kuljetusprotokollasta johtuen ongelmiksi CoAP:in käytössä voi syntyä verkon tukkeutuminen, koska ei ole käytössä minkäänlaista vuon hallintaa. Mahdollinen tarve tietää viestin perillemenosta hoidetaan viestityypillä. Protokollaan on määritelty neljä viestityyppiä Confirmable, Non-confirmable, Acknowledgement ja Reset. Viestin sisältäessä Confirmable määritteen, viestiin odotetaan vastausta ja jollei sitä tule tehdään uudelleen lähetys käyttäen exponential backoff-algoritmia. HTTP:n header- ja body -rakenne on korvattu CoAP-protokollassa paketti-rakenteella.[1]

Paketti sisältää binäärimuodossa tunnistustiedot, otsakkeen ja dataosiot alla kuvatun mukaisesti.



Kuva 1 CoAP-paketin rakenne.

Ver: Käytetyn CoAP-standardin versio.

T: Lähetetyn viestin tyyppi, voi olla Confirmable(0), Non-confirmable(1), Acknowledgement(2) ja Reset(3)

TKL: Mahdollisen Token-kentän pituus.

Code: Viestin sisällön tyyppi.

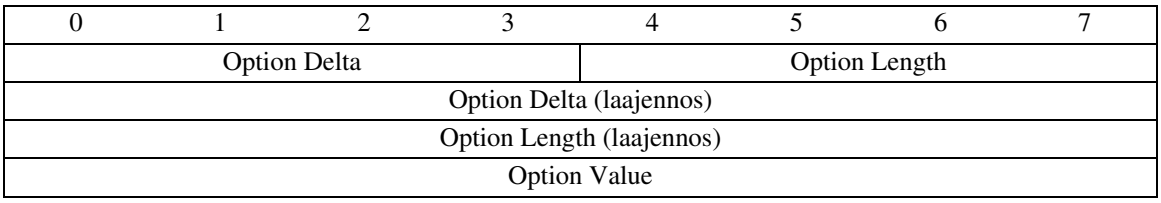
Message ID: Verkon tavujärjestyksessä oleva ID-arvo, jolla tunnistetaan paketin järjestys ja käytetään estämään duplikaatit.

Token: Asiakkaan generoima arvo pyynnön mukaan, minkä palvelin palauttaa vastauksessa identtisenä takaisin. Näin lisätään luottamusta siihen, että asiakas saa lähettämäänsä viestiin vastauksen.

Options: Käytetään laajentamaa viestin kuvausta. Optioita käytettäessä sisällytetään kyseiseen kohtaan alempana kuvattu optio-kenttä.

Payload: Paketin sisältämä tietokuorma.

CoAP tarjoaa useita optioita, joita voidaan liittää viestiin. Optiot toimivat samaan tapaan, kuin HTTP:n otsikon attribuutit. Optioissa määritellään mm. viestin sisältö, eli MIME-tyyppi ”Content-Format” määritteellä, jonka optio-numero on 12. Optioiden numerot on määritelty neljään eri kategoriaan. Ensimmäiset 256 arvoa on jätetty IETF:n päätettäväksi. Arvot 256–2047 on varattu yleisesti käytettyihin julkisiin spesifikaatioihin. Arvot 2048–64999 on varattu yksityiseen tai toimittajakohtaisiin käyttötarkoituksiin. Loput arvot, eli 65000–65535 on varattu testauskäyttöön, eikä niitä saa käyttää julkisesti. Numero ilmoitetaan delta-koodattuna, jolloin option arvoksi muodostuu summa edellisen option delta-arvosta ja nykyisen option etukäteen määritetyn arvon kanssa. Eli jos on olemassa kolme erilaista optiota, joiden arvoiksi on määritelty 3,7 ja 12, tällöin delta-kenttiin sijoitettuna ensimmäisen arvo on 3, seuraavan 10 ja kolmannen 22. Alla optio kentän rakenne:



Kuva 2 CoAP protokollan optio kentän rakenne.

Option Delta: Optio-kentän numero ilmoitettuna delta-koodattuna.
Option Length: Optio kentän pituus tavuina ilmoitettuna.
Option Delta (extended): Alkuperäisen option kaltainen lisäkenttä.
Option Length (extended): Alkuperäisen pituuden kaltainen lisäkenttä.
Option Value: Riippuu, miten optiota käsitellään, voi olla tyhjä, tavujono, numerojono tai merkkijono.[1]

2.4 MIME – Internet media tyyppi

MIME-tyyppi määriteltiin alun perin IETF:n julkaisemassa kolmiosaisessa ”Multipurpose Internet Mail Extension” dokumentissa, joiden RFC:t ovat 2045 [19], 2046 [25] ja 2047 [26]. Kyseisessä määrittelyssä kuvataan kuinka tietomuotoja, jotka eivät ole ASCII-merkkijonoja voidaan sisällyttää sähköpostin osaksi. MIME laajennettiin myöhemmin RFC-6838:ssä [27] media-tyypiksi kattamaan kaikenlaisen verkkoliikenteen tyypittämisen. Näin sovellustason ohjelmat saavat tiedon, miten media tulisi tulkita oikealla tavalla.[19][25][26]

Media-tyypit määritetään tekstimuodossa selkokielisenä ja Id-numerona. Tekstimuoto on esim. text/html, joka kuvaa tiedon olevan tulkittavissa html-muodossa. Saman text/html tyypin id-arvo on numero kolme. Media tyyppien määrää ei ole rajoitettu mitenkään. Uusia tyyppejä voidaan luoda IETF RFC-4288:ssä [28] kuvatun tavan mukaisesti. MIME-tyypeistä johtuen ei tarvitse sitoutua protokollaan suoraan määriteltyihin tyyppeihin kuten tekstiin. Tällöin voidaan käyttää sovellustason mekanismeja, jotka ovat protokollista riippumattomia tekniikoita. Sovellustason mekanismit mahdollistavat näin protokolla riippumattomuuden ja esim. hyötykuorman siirtämisen protokollasta toiseen vaikuttamatta hyötykuormaan. MIME–tyyppi kertoo, minkälaisia mekanismeja tarvitaan tiedon tulkitsemiseen. Tekstimuotoinen tyyppi soveltuu hyvin esim. HTTP-protokollassa käytettäväksi, numero taas soveltuu mm. CoAP-protokollan binääriseen otsikkoon.[19][28]

Työssä ollaan mahdollistamassa Digest-autentikaatioon perustuva kättely CoAP-protokollaa käyttäviltä laitteilta HTTP-protokollaa käyttäville palvelimille. Digest perustuu HTTP-protokollaan, jolloin HTTP:tä käytettäessä ei käytetä ulkopuolista sovellustason mekanismia. CoAP-protokollaan ei ole kuitenkaan määritelty Digest-autentikaatiota. Määrittelemättömyyden seurauksena on loogista siirtää kättelyn vaatimat attribuutit protokollien vaihtovaiheessa jonkun sovellustason mekanismin päälle. Näitä sovellustason mekanismeja ovat mm. XML [17] ja JSON [18]. MIME-tyypillä kerrotaan miten tieto tulee lopulta tulkita. Näin toimien CoAP-protokolla tietää vain kuljettamansa mekanismin tyypin, jonka perusteella autentikaatiota tarvitseva laite osaa tulkita tarvittavaa mekanismia.[1][19][17][18]

XML (Extensible Markup Language) [17] on tiedon merkintäkieli, joka kehitettiin SGML:in (Standard Generalized Markup Language) [29] pohjalta HTML-liikenteen apuna käytettäväksi. XML on rakenteellinen tekstimuotoinen selkokielinen notaatio, missä tiedon merkitys kuvataan itsensä sisään. XML:ää voidaan käyttää tiedon tallentamiseen ja välittämiseen, mihin perustuu muun muassa verkossa käytössä oleva XHTML-esityskieli [30]. XML-media-tyyppi on vaihtoehtoisesti application/xml tai text/xml.[17][31]

Esimerkki XML-dokumentin toteutuksesta alkaa prologilla, joka sisältää käytetyn version ja mahdollisesti muita tietoja, kuten tiedon koodaustavasta.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

Tämän jälkeen tulee käytetty kuvauskieli, minkä pohjalta XML-dokumentin sisäinen rakenne tulkitaan.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Nyt voidaan määritellä elementit, jotka sisältävät dokumentin tiedot.

```
<elementti attribuutti="arvo">
 ...
</elementti>
```

XML:n selväkielisestä syntaksista johtuen se on raskas kevyille laitteille ja siitä on siksi kehitetty erilaisia kevyempiä versioita. Yleisimpiä on W3C (World Wide Web Consortium):n valitsema Binary XML-formaatti EXI (Efficient XML Interchange) [32], missä XML-syntaksista on pyritty tekemään mahdollisimman pieni ja kompakti.[16]

JSON (JavaScript Object Notation) [18] on XML-kielelle kehitetty vaihtoehtoinen tiedon paketointi ja esitysformaatti. Notaationa JSON:ista on pyritty tekemään kevyt ja tekstipohjainen. JSON-objekti voidaan nimensä mukaisesti luoda suoraan JavaScript-objektista. JSON ei ole kuitenkaan millään tavalla kielisidonnainen ja siten riippuvainen JavaScriptin käytöstä. JSON:in avulla voidaan nopeasti paketoida tietoa ja siirtää eteenpäin. JSON-notaation mediatyyppi on application/json. JSON-notaation käyttämät tietotyypit ovat string, numbers, booleans ja null. JSON-notaatiossa tieto esitetään nimi-arvo-attribuutti pareina siten, että arvo voi sisältää uusia pareja. Alla esimerkki JSON-notaatiosta:

```
"www-authenticate" : {
        "type": "digest",
        "challenge": {
            "realm": "jarkko@virtanen.com",
            "qop": "auth, auth-int",
            "nonce": "dcd98b7102dd2f0e8b11d0f600bfb0c093",
            "opaque": "5ccc069c403ebaf9f0171e9517f40e41"
        }
    }
```

[18][28]

Esitysmuotoon on määritelty erilaisia valmiita malleja ja rakenteita esittämään erilaisia asioita, kuten JSON Web Key (JWK) [33], JSON Web Algorithms (JWA) [34], JSON Web Signature (JWS) [35], JSON Web Encryption (JWE) [36] ja JSON Web Token (JWT) [37]. JWK on rakenne, jossa esitetään operaatiossa käytetyt kryptografiset operaatiot ja niiden sisältämät erilaiset avaimet. JWA:han on kuvattuna erilaiset kryptografiset algoritmit, joita voidaan käyttää mm. JWS-, JWK- ja JWE -rakenteissa. JWA määrittää myös sitä käyttävien ohjelmistojen tuen tarpeen JWA:ssa määritellyille algoritmeille. [33][34][35][36][37]

JWT on JSON-notaatioon kehitetty esitysmuoto, joka sisältää dataosioita eri tarkoituksiin. Esitysmuodon tarkoitus on mahdollistaa mm. HTTP-Authorization-kentän header-attribuutin tietojen siirtämisen JSON-objektin sisällä. Näin HTTP-header-attribuutteja voidaan siirtää muiden protokollien välitettäväksi. JWT:n sisältämä tieto voidaan halutessaan allekirjoittaa (JWS) tai kryptata (JWE). JWT-osioihin tieto sisällytetään Base64Url-koodattuina JSON-objekteina. Ensimmäisessä otsikko data-osiossa kerrotaan, mitä kryptografisia operaatioita datalle on tehty. Ensimmäisen dataosion jälkeen tulee otsikon määrittelemät dataosiot [37][35][36]

Käytettäväksi sovellustason esitystavaksi Digest-autentikaation siirtämiseen työhön valittiin JSON-objekti. Suurin syy valintaan on JSON:in yksinkertaisuus ja siten helppo käytettävyys. XML-kielen valitseminen olisi johtanut valintoihin käytettävästä kuvauskielestä, sekä onko siihen saatavilla riittävää laitetukea. HTTP-Digest-autentikaation konvertointi XML-kieleen olisi mahdollisesti ollut myös JSON-objektiin verrattuna haastavampi toteuttaa.

2.5 Tietoturvan huomiointi Web of Things -ympäristössä

Web of Things -ympäristöjen tietoturvaongelmat eivät paljoa eroa muunlaisten verkkojen ongelmista. Tietoturvaongelmia on tahallisia ja tahattomia ja niihin varautumalla saavutetaan verkon pysyminen toiminnassa ja verkkolaitteiden kyky vastata resurssipyyntöihin. Tahallisia ongelmia ovat mm. erilaiset palvelinestohyökkäykset ja pyrkimykset päästä käsiksi verkkoliikenteeseen. Verkon liikenteeseen käsiksi pääseminen voi mahdollistaa liikenteen seurannan ja muokkaamisen. Palvelinestohyökkäykset ovat häiritsemisyrityksiä, joilla pyrkimyksenä on haitata verkon toiminnollisuutta. Keinoina liikenteen häiritsemiseen on mm. ylimääräistä liikenteen luominen tai resurssipyyntöjen tekeminen, jolloin resurssi joutuu tekemään ylimääräistä työtä. Tahattomia ongelmia ovat mm. viat laitteissa, ohjelmistoissa tai yhteyksissä. Ongelman voi aiheuttaa esimerkiksi langattomissa yhteyksissä etäisyyksien kasvaminen liian suureksi, jolloin yhteys katkeaa. Näihin voidaan yrittää varautua erilaisilla kahdennuksilla, palomuurauksella ja protokollan toimintaan liittyvillä ratkaisuilla. [38][39]

Turvallisuutta voidaan lisätä käyttämällä erilaisia sovellustason ja kuljetuskerroksen tietoturvaratkaisuja. Kuljetuskerroksen ratkaisut ovat valitettavasti riippuvaisia kuljetuskerroksen protokollista. Luotetuille kuljetuskerroksen protokollille, kuten TCP [40] on kehitetty päällä käytettäväksi esim. TLS-tunnelointiprotokolla [41]. Epäluotettavien protokollien, kuten UDP [22] päällä käytettäväksi on kehitetty DTLS-tunnelointi [24]. [41][24]

Tietoturvan huomioimiseen kuuluu oleellisesti tietoturvan kolme pääkäsitettä, jotka ovat saatavuus, eheys ja luottamuksellisuus. Edellisten kolmen mahdollistamiseksi tulevat vielä käsitteet autentikointi ja kiistämättömyys. Molempia käsitteitä tarvitaan kolmen pääkäsitteen mahdollistamiseksi. [39][42][10]

Autentikointi on järjestelmän resurssien käytön valvomista ja käyttäjien tunnistamista. Valvonnan ja tunnistamisen jälkeen voidaan sallia vain oikeuden omaavien tahojen resurssien käyttö. Kirjautuneena oloa kutsutaan istunnoksi. Yleensä istunto on voimassa kirjautumisesta istunnon lopettamiseen tai jonkun aikamääreen verran. Resurssia käyttäviä koneita tai henkilöitä kutsutaan identiteeteiksi ja resurssi voi valvoa erilaisilla keinoilla onko identiteeteillä oikeuksia halutun resurssin käyttöön. [39][42]

Identiteetin todentamiseen on useita erilaisia tapoja, jotka yleensä jaetaan kolmeen erilliseen kokonaisuuteen. Ensimmäisenä kokonaisuutena on olemassa oleva tieto, jonka kirjautuva muistaa tai tietää. Tietona on yleensä sovittu salaisuus, joka voi olla esim. salasana tai tunnusluku. Toiseen ryhmään kuuluvat yksilöitävät asiat, joita autentikoituva pitää hallussaan. Yksilöivällä asialla voidaan tämän jälkeen yhdistää autentikoituva tiettyyn identiteettiin. Erilaisia käytössä olevia kohteeseen yksilöiviä asioita ovat mm. fyysinen SIM-siru tai kertakäyttöinen avainlista. Kolmantena ryhmänä on biometrinen tunniste. Biometrinen tunniste on jonkinlainen kirjautujan ominaisuus, joita muiden on kirjautuessa vaikea matkia tai väärentää. Ominaisuuksina ihmisillä voivat olla mm. silmän iiris tai nimen kirjoitustapa. Koneilla olevia ominaisuuksia voivat olla mm. erilaiset laitetunnukset tai tunnettu aika tehdä jonkinlainen laskutoimitus. [39][42]

Kaikilla tavoilla on erilaisia ongelmia. Salaisuus voidaan onnistua selvittämään esim. vakoilemalla tai arvaamalla. Yksilöivän asia, kuten SIM-siru voi hajota tai se voidaan onnistua varastamaan. Yksilöivän asian pitäisi myös aina olla lähettyvillä käytön mahdollistamiseksi. Jollei tunnistautuja ole kiinteästi samalla paikalla, tästä voi tulla ongelma. Biometrisen tunnistuksen ominaisuus pitää pystyä tallentamaan tarkasti ja biometria voi jostain syystä muuttua. [39][42]

Virhemahdollisuuksien takia autentikaatiota kutsutaan vahvaksi, jos tunnistus perustuu kahden yllä olevan tavan samanaikaiseen käyttöön. Yhteen tapaan perustuvaa kutsutaan heikoksi, koska yhden ominaisuuden keksiminen, väärentäminen tai varastaminen katsotaan liian helpoksi.[39][42]

Identiteetin varmentamiseen käytetään monesti ihmisten kohdalla tunnusta ja sille muistettavaa salasanaa. Laitteiden osalta salasanan tilalla käytetään jollain tavalla laitteelle tallennettua tai välitettyä bittijonoa, jolloin puhutaan avaimesta. Työn käsitellessä WoT-ympäristöjä, joissa laitteiden välinen tiedonsiirto on oleellista, käytetään jatkossa selkeyden vuoksi vain termiä avain.

Kiistämättömyys tarkoittaa jollain keinolla tiedon varmentamista alkuperäiseksi, sekä sen todistamista jälkikäteen, jolloin tietoa voidaan pitää autenttisena. Autenttisuus tarkoittaa tietojen pysymistä alkuperäisinä eli esim. kukaan ole pystynyt väärentämään tietoja antamalla omia tietoja alkuperäisinä tai manipuloimaan jotenkin olemassa olevia tietoja.[39][42]

Saatavuus tarkoittaa tiedon olemista saatavilla etukäteen määritellyn vasteajan puitteissa. WoT-ympäristöissä saatavuus on haastava käsite, koska laitteet voivat aktivoitua vain tiettyinä aikoina, eikä ole takeita yhteyden toiminnasta niinä aikoina. Yhtenä ratkaisuna WoT-ympäristöissä ovat välityspalvelimet, jotka pyrkivät varastoimaan tietoja kasvattaen näin saatavuutta. [39][42]

Eheys tarkoittaa tiedon pysymistä identtisenä ajan kulumisesta tai paikanvaihtumisesta huolimatta. Tietoliikenteen osalta viesti pysyy identtisenä koko matkan lähettäjältä lähetettävälle. Tällöin tiedonsiirtoon liittyvä protokolla jollain keinolla estää tai havaitsee tiedon muuttumisen matkalla. Estäminen on havaitsemista vahvempi käsite, mutta monesti riittää havainto tiedon muuttumisesta, jolloin tieto voidaan hylätä tai käyttää tietäen virheen mahdollisuudesta. WoT-ympäristöissä liikenne on monesti antureiden antamia arvoja, jolloin virheellinen arvo voidaan hylätä. Eheyttä tavoiteltaessa erilaisten salausoperaatioiden jälkeen tieto pitää pystyä palauttamaan identtisenä takaisin ymmärrettävään muotoon. Eheyden toteutumista pystytään tarkastelemaan mm. symmetrisen tai epäsymmetrisen avainparin avulla lasketuilla kryptografisilla tiivistesummilla tai digitaalisella allekirjoituksella.[39][43][8][42]

Kryptografisena tiivistesummana käytetään yleensä MAC (Message Authentication Code) -arvoja, josta käytössä tiivistepohjaiseksi kehitetty HMAC (Hash-based Message Authentication Code). Tiivistettä käytettäessä avaimen avulla lasketaan lähettävästä tiedosta kryptografinen tiivistesumma, mikä tämän jälkeen liitetään viestin mukaan. Avaimen haltija voi aina toistaa kryptografisen tiivistesumman laskemisen ja siten identtisellä arvolla havaita tiedon pysyneen samana. Epäsymmetrisessä tapauksessa tiedon lähettäjä voi julkaista julkisen avaimen, minkä pohjalta lukijat voivat tarkastaa tiivisteen vastaavan alkuperäistä. Symmetristä avainta käytettäessä tahot, jotka tietävät avaimen voivat tarkistaa tiedon aitouden. [39][43][8]

Digitaalinen allekirjoitus on hyvin samanlainen kuin tiiviste, mutta siinä teksti salataan yksityisellä avaimella, tätä kutsutaan julkisen avaimen allekirjoitukseksi. Tällöin julkisen avaimen haltijat voivat purkaa viestin ja näin havaita viestin autenttisuuden ja alkuperäisen tekijän. Käytössä olevissa algoritmeissa yleensä viesteistä tehdään kryptografinen tiivistesumma ja vasta tiiviste allekirjoitetaan. Tällainen tapa on viestien allekirjoittamiseen verrattuna nopeampaa ja siten yleensä käytetty.[39][43][8]

Luottamuksellisuudella tarkoitetaan toimijan halua olla riippuvainen toisesta osapuolesta ja kokea olonsa turvalliseksi, vaikka on mahdollisuus negatiivisista seurauksista. Luottamukseen liittyy vahvasti ongelma, millä perusteella luottamus rakentuu. Luotanko vain käyttäjään, joka ilmoittaa omaavansa kyseiset asiat vai käyttäjään, jonka ilmoituksen varmistaa joku kolmas tai jopa useampi taho. Vielä parempi, jos luotan tahoihin, jotka varmistavat käyttäjän ilmoituksen. Keinoja tällaisen luottamuksen rakentamiseen on mm. sertifikaatit tai listat luotetuista tahoista. Sertifikaatit myöntää taho, jonka toiminta on mahdollisesti niin läpinäkyvää, että siihen voidaan luottaa tai se on niin iso, että siihen halutaan luottaa. Luotetun tahon lista voi olla esim. 3G-operaattori, joka pitää kirjaa asiakkaistaan. WoT-ympäristöissä luottamusta tarvitaan mm. asiakkaan kirjautuessa palvelimelle. Asiakkaan tarvitsee luottaa palvelimen olevan väittämänsä palvelin ja palvelimen luottaa asiakkaan olevan kuka sanoo olevansa.[42][39]

Luottamuksen ja kiistämättömyyden saavuttamiseen yksi keino on tiedon salaaminen. Salaus on tietojen muuttamista jollain keinoilla muiden ymmärtämättömään tai havaitsemattomaan muotoon. Erilaisia keinoja tiedon piilottamiseen ovat mm. symmetrisen tai epäsymmetrisen salauksen, lasketun kryptografisen tiivisteen tai steganografian käyttäminen. Edellä olevia tapoja voidaan myös yhdistellä paremman lopputuloksen saavuttamiseksi.[39][8]

Salaustapoja käytettäessä tieto yleensä muutetaan toiseksi käyttäen vaihtoehtoisesti symmetristä tai epäsymmetristä salausalgoritmia ja tietynmittaista salausavainta. Algoritmit perustuvat julkisiin ja salaisiin ratkaisuihin. Monesti julkisia pidetään turvallisimpina, koska silloin kaikilla on mahdollisuudet analysoida ja etsiä virheitä niiden toteutuksista. Symmetrisessä salauksessa samalla avaimella voidaan sekä salata, että purkaa salaus. Epäsymmetrisessä tarvitaan kaksi avainta, joista toisella voidaan salata ja toisella purkaa. Epäsymmetrisen avaimen etuna on mahdollisuus julkistaa avaimen toinen osa, jolloin vaihtoehtoisesti suuri joukko tahoja voi salata ja vain yksi purkaa tai vain yksi salata ja monet purkaa. Symmetrisen avaimen käyttö on laskennallisesti vähäresurssisempi ratkaisu. Käyttötarkoituksesta riippuen symmetristä ja epäsymmetristä avainta käytetään tiedon salaamiseen. Monesti symmetrisiä ja epäsymmetrisiä algoritmeja voidaan yhdistää molempien hyvien puolien saavuttamiseksi. Yhdistäessä itse viesti salataan symmetrisellä avaimella ja sen jälkeen symmetrinen avain salataan julkisella avaimella. Näiden operaatioiden jälkeen epäsymmetrisen avainparin yksityisen avaimen haltija voi avata viestin.[39][8][7][43]

Symmetrisessä salauksessa yleensä käytetään lohkosalausalgoritmeja, jolla pyritään peittämään selko ja kryptotekstin välinen riippuvuus. Lohkosalausalgoritmit käyttävät jollain moodilla salattuja lohkoja, jotka sen jälkeen algoritmista riippuvalla tavalla sekoitetaan keskenään. Lohkon pituus voi olla esim. 64bit tai 128bit. Lohkoon sijoitetaan lohkon pituinen selkoteksti joka salataan moodista riippuvalla tekniikalla lohkoon syötetyllä tietynmittaisella avaimella. Avain voi olla esim. 128 bittiä pitkä ja ulostulona on samanpituinen salateksti. Lohkon ja avaimen pituudella pystytään monesti vaikuttamaan salauksen turvallisuuteen. Lohkot käyttävät monesti toiminnassaan hyväksi alustusvektoria, joka on yleensä satunnainen merkkijono. Alustusvektorilla pyritään satunnaistamaan salausta esim. tekemällä XOR-operaatio selkotekstin kanssa ennen moodiin menemistä. Johtuen lohkojen määritetystä pituudesta ja selkotekstien satunnaisesta pituudesta käytetään erilaisia algoritmeja täydentämään selkoteksti turvallisesti lohkon mittaiseksi. Täydentävän algoritmin toiminta on kriittistä, koska lohkojen täydentäminen jollain tietyllä arvolla voi mahdollistaa selkotekstin hyökkäyksen. Lohkosalauksista käytetyimpiä on avoimeen standardiin perustuva AES (Advanced Encryption Standard) [44], joka syntyi Yhdysvaltojen standardoimisviraston NIST kilpailun pohjalta. AES-salauksessa lohkomoodeina voidaan käyttää mm. CBC ja HMAC ja CMAC, joista kaksi jälkimmäistä tarjoaa myös luottamusta alkuperää kohtaan. [43][8][39]

Epäsymmetrisissä käytetään yleensä modulaarisia laskutoimituksia, jotka perustuvat ajatukseen modulo-operaatioiden käänteisoperaatioiden monikäsitteisyydestä. Tällöin alkuarvoja tuntemattoman on vaikeaa tehdä laskutoimitusta takaisinpäin. Edeltävien pohjalta saadaan luotua algoritmista riippuen eri operaatioilla julkinen ja salainen avain. Yleisimpiä käytössä olevia ovat Diffie-Hellman [45] ja RSA [46]s. RSA perustuu suurien alkulukujen tulon tekijöiden jakamisen vaikeuteen. Diffie-Hellman diskreetin logaritmin laskemisen vaikeuteen. [43][8][39]

Kryptografista tiivistesummaa käytettäessä ei ikinä liikuteta oikeaa tietoa, vaan siitä laskettua tunnistearvoa. Näin voidaan toimia esim. kirjautumistilanteissa, missä palvelimelle ei tarvitse varastoida oikeaa avainta, vain tiiviste riittää. Kryptografinen tiiviste helpottaa tiedon säilyttämistä, koska tiivisteestä on hyvin vaikea päätellä alkuperäistä tietoa.[8][43][39]

2.6 TLS-tunnelointi

TLS (The Transport Layer Security) [41] on verkon kuljetuskerroksen turvallisuuteen kehitetty protokolla. TLS-protokollan suunnittelun pohjana on ollut pyrkimys mahdollistaa asiakas-palvelinpään liikenteen suojaus välimieshyökkäystä, pakettien väärentämistä ja viestivarkauksia vastaan. Protokolla on suunniteltu käytettäväksi luotettavan yhteyden tarjoavien protokollien päällä, kuten TCP. Edeltävästä johtuen TLS-protokollaa käytetään esim. HTTP-protokollan parina turvaamaan sen liikennettä. TLS perustuu X.509-varmenteiden käyttöön ja niiden luotettavuuteen. Palvelimet osoittavat asiakkailleen luotettavuutensa varmenteilla, jonka joku luotettava varmenteen tarjoaja on allekirjoittanut. TLS-protokolla jakaantuu kahteen osaan ”The TLS Record Protocol” ja ”The TLS Handshake Protocol”.[41]

Record Protocol tarjoaa yhteyden turvaamiseen kahta pääominaisuutta. Ensimmäinen ominaisuus on yhteyden salausta symmetrisellä avaimella. Toinen on autenttisuuden selvittäminen sisällöstä lasketun tiiviste-funktion avulla.[41]

Handshake Protocol on suunniteltu ylempien protokollien paketointiin, eli toteuttaa tunnelin asiakkaan ja palvelimen välille. Tunnelin syntymisen jälkeen sovellustason protokollat aloittavat liikenteensä. Kättelyprotokollalla on kolme perusominaisuutta. Ominaisuuksista ensimmäinen on autentikoituvien tahojen autentikointi käyttäen asymmetristä tai julkisen avaimen salausta. Toinen ominaisuus on jaetun avaimen neuvottelu siten, ettei edes yhteyden välissä oleva henkilö saa sitä selville. Kolmas ominaisuus on neuvottelun luotettavuus, jolloin kukaan tunkeutuja ei voi siihen vaikuttaa.[41]

2.7 DTLS-tunnelointi

DTLS (Datagram Transport Layer Security) [24] on kuljetuskerroksen Datagram-protokollien turvallisuuteen kehitetty protokolla. Datagram-protokolla tarkoittaa protokollaa, mikä ei anna takeita toimituksesta, perillemenoajasta tai pakettien perillemenojärjestyksen säilymisestä. Tällainen protokolla on mm. UDP. Edellisistä syistä johtuen Datagram-protokollissa ei voida käyttää aikaisemmassa luvussa kuvattua TLS-protokollaa. Suurimpana syynä TLS:n kyvyttömyys sietää pakettien sekoittumista tai katoamista. Tämän takia TLS-protokollasta kehitettiin DTLS, joka pyrkii olemaan mahdollisimman vastaava protokolla. Protokolla on suunniteltu torjumaan välimieshyökkäyksen, viestien lukemisen ja vääristämisen. Protokollaa käytetään mm. turvaamaan UDP-protokollan päällä toimivan CoAP-protokollan liikennettä.[24]

DTLS-protokollan suurimmat erot TLS:n verrattuna ovat salauksen riippumattomuus muista paketeista ja kättelyn ymmärrys pakettien mahdollisesta katoamisesta matkalla. Näiden toteuttamiseen DTLS:ään lisättiin eksplisiittinen järjestysnumero, jonka perusteella paketit voidaan järjestää ja mahdollisesti pyytää uudestaan. Toinen on DTLS-paketin hajottaminen pieniksi kokonaisuuksiksi, jolloin kokonaisuus mahtuu yhteen Datagram-pakettiin. Näin estetään yhden DTLS-paketin hajoaminen useaan välityspakettiin kuljetuskerroksen toimesta.[24]

DTLS-kättelyyn tehtiin kolme oleellista muutosta TLS-protokollaan verrattuna. Muutoksista ensimmäinen on tilallisen keksin käyttöönotto. Palvelin lähettää keksin asiakkaalle, jonka on vastattava palvelimelle viestillä, joka sisältää keksin. Tällaisella menettelyllä pyritään vaikeuttamaan palvelinestohyökkäyksen tekemistä, koska tällöin asiakaskin joutuu lähettämään paluukuormaa. Toinen on järjestysnumero, joka kasvaa jokaisessa pyynnön vaiheessa, jolloin voidaan havaita pakettien puuttuminen tai väärä järjestys. Kolmantena on uudelleenlähetysajastin, joka luo ajastimen odottamaan seuraavaa viestiä ja jollei sitä kuulu, tehdään uudelleenlähetyspyyntö.[24]

3 SOVELLUSTASON TIETOTURVA

Sovellustaso on mm. OSI [47] -, sekä TCP/IP [48] -mallin ylin kerros. Sovellustason tiedonsiirrosta ovat vastuussa tietoja siirtävät sovellukset, jotka rakentavat oman toimintansa mm. protokollien TCP ja UDP päälle. Sovellustason tietoturvassa ollaan siten kiinnostuneita esim. HTTP- ja CoAP -protokollien kuljettamasta tiedosta, sekä niiden käyttämistä mekanismeista ja esitysmalleista. Esitysmallit ja niihin liittyvät mekanismit toimivat näin vielä hieman sovellustason yläpuolella. HTTP- ja CoAP -protokollien omien mekanismien lisäksi yleisesti tunnettuja päällä käytettäviä esitysmalleja ovat mm. JSON (JavaScript Object Notation) [18] ja XML (Extensible Markup Language) [17].[38][47][17][18][49]

Sovellustason tietoturvaa voidaan hoitaa alempien kerrosten tietoturvaratkaisuilla, kuten tunnelointiprotokollilla, joita ovat esimerkiksi aiemmin mainitut TLS [41] ja DTLS [24]. Alempien kerrosten ratkaisuissa on kaksi isompaa ongelmaa. Ensimmäisenä ne eivät ota kantaa päällä kulkevien protokollien tietoturvaan, eli joku sovellustason ohjelma voi pahantahtoisesti vaikuttaa sovellustason liikenteeseen. Toisena tunnelointiratkaisut aiheuttavat riippuvuuden sovellustasolta alempien tasojen liikenteeseen. Muutos tunnelointitason protokollaan tällöin estää tunneloinnin käyttämisen.[41][24]

Protokollarakenteeseen tehtävät muutokset harvemmin ovat toivottavia. Verkkoliikenteessä syntyy kuitenkin tilanteita, joissa tarvitsee muuttaa sovellustason tai kuljetustason protokollaa. Tällaisia tilanteita syntyy verkon tai ohjelman tukiessa vain tiettyä protokollaa. Edeltävissä tilanteissa on pakko käyttää yhdyskäytävää muuntamaan protokollia toisiksi. Palomuuraukset voivat myös muokata liikennettä pyrkiessään analysoimaan paketteja. Yhdyskäytävät ja palomuuraukset voivat näin rikkoa alemman tason protokolliin tai paketteihin sidotut turvallisuusratkaisut. Turvallisuusratkaisuihin käytetyt tunneloinnit hajoavat mahdollisesti joko kokonaan tai sen verran, että turvaprotokollat havaitsevat sisältönsä muuttuneen ja hylkäävät liikenteen. Esimerkiksi TLS:n [41] tapauksessa yhdyskäytävän mahdollisuudet ovat tunneloida liikenne suoraan läpi tai ottaa liikenne vastaan omalla sertifikaatilla ja toimittaa se sen jälkeen eteenpäin. Rikkoutumisen ollessa todennäköistä tai mahdollista sovellustason tietoturvaratkaisuiden käyttö on ainoa mahdollisuus. Käytettävistä alempien kerrosten ratkaisusta riippuen sovellustasolla ei ole ollenkaan tai vain vähän tietoa siitä turvataanko liikennettä alemmilla tasoilla. Halutessaan varmistua liikenteen turvallisuudesta sovellusten on järkevää pyrkiä turvaamaan liikennettään sovellustason ratkaisuilla.[41][50][51][52]

Sovellustason tietoturvalla pystytään vaikuttamaan tietoturvan kolmeen käsitteeseen, jotka olivat saatavuus, eheys ja luottamuksellisuus. Keinoja vaikuttaa kolmeen tietoturvan käsitteeseen sovellustasolla on protokollien omat suojaustekniikat sekä niiden päällä käytettävien esitysmallien erilaiset ratkaisut.[42][39]

3.1 Autentikointi sovellustasolla

Sovellustason autentikointi on oleellista, koska silloin autentikoidutaan vain kyseiselle sovellukselle. Tästä syystä autentikointiratkaisut ovat sovelluskohtaisia ratkaisuja. Jollei sovellustaso itse tarjoa autentikointitapaa, yleensä käytetään alemman kerroksen tai päällä toimivan mekanismin ratkaisuja. Alempana toimivat TLS [41] ja DTLS [24] mahdollistavat keskustelun sovelluksen kanssa, mutta ne eivät ole millään tavalla puhtaita sovellustason ratkaisuja. Niiden hyvä puoli on koko sovellustason autentikointi. Päälläkäytettävät mekanismit taas autentikoivat päällä olevan liikenteen, mutta jättävät itse sovellustason tunnistamattomaksi. Protokollien omiin ratkaisuihin kuuluu mm. heikon autentikaation HTTP:n Basic- ja Digest -autentikaatiot[4], jolla voidaan vaikuttaa autentikointiin ja siten luottamukseen. Haettaessa vahvaa autentikaatiota yksi hyvin soveltuva ratkaisu on esimerkiksi 3GPP tarjoama GAA (Generic Authentication Architecture)[5]. GAA:n käyttämässä AKA-autentikaatiossa käyttäjä tunnistetaan Digest-autentikaatiosta pohjautuvalla tavalla ja luottamus tulee operaattorilta.[41][24][5][4]

Tunnistustapoja käytettäessä yleensä käyttäjälle näkymättömissä toimii salausta tarjoavia symmetrisiä tai epäsymmetrisiä -protokollia, jotka toimivat käytetyllä protokollatasolla. Näiden avulla pyritään tekemään turvallinen avaintenvaihto, missä kirjautumistiedot pyritään siirtämään turvallisesti verkon yli ohi ulkopuolisten katseilta.[39]

Käyttäjä voi pyrkiä tekemään itsestään myös tunnistettavan verkossa, pyrkimällä merkitsemään lähettämänsä liikenteen jollakin tavalla. Monesti tunnistus hoidetaan käyttämällä yhteyteen liittyviä asioita, kuten portti, IP- tai MAC -osoite. Edelliset tavat ovat kuitenkin alempien protokollatasojen ratkaisuja. Sovellustasolla voidaan käyttää asiakkaalle räätälöityä URI:a tai palvelin voi lähettää asiakkaalle tiedon, kuten keksin, minkä asiakas sisällyttää viesteihinsä. Yksilöivän URI:n tai sisällytetyn tiedon avulla palvelin tunnistaa asiakkaan. Asiakas ja palvelin voivat myös käyttää omaa tai yhdessä sovittua allekirjoitusta. Allekirjoittamalla viestit voidaan pitää luottamusta siihen, että vastapuoli tietää viestin tulevan oikealta taholta.[39]

3.1.1 Basic-autentikointi

Basic-autentikaatio on alkuperäisessä HTTP 1.0-protokollassa RFC-1945 [3] määritelty autentikointitapa. Basic-autentikaatiota kehitettäessä verkon salakuuntelua ei nähty ongelmana, joten luottamuksellisuutta ei huomioitu. Autentikaatiotapana siitä tuli näin hyvin yksinkertainen ja kevyt. Käytössä olevista autentikaatiotavoista Basic on yksi yksinkertaisimmista. Tekniikkana se ei vaadi minkäänlaista tilatietoja käyttäjän ja palvelimen välille. Autentikaatiossa tunnus ja avain lähetetään palvelimelle pelkästään Base64Url-koodattuna, joten tunnus ja avain voidaan selvittää helposti. Avaimen ja tunnuksen selkokielisyydestä johtuen Basic-autentikaatiota ei voi suositella käytettäväksi ainoana ratkaisuna.[4][3]

Esimerkki autentikaatiosta käyttäjän tehdessä tietopyynnön palvelimelle:

```
GET /dir/index.html HTTP/1.0
Host: localhost
```

Palvelimen halutessa Basic-autentikointia se vastaa käyttäjän lähettämään pyyntöön otsakkeella, mihin on sijoitettu autentikoiduttavan palvelimen tieto Realm-arvona:

```
WWW-Authenticate: Basic realm="WallyWorld"
```

Käyttäjä vastaan pyyntöön Base64-koodatulla “käyttäjänimi:avain” tekstikentällä. Käyttäjän tunnus on Alladin ja avaimena on ”open sesame” WallyWorld palvelimelle. Käyttäjä yhdistää tunnuksen ja avaimen ”:” merkillä ja tekee sille Base64-koodauksen, jolloin vastauksesta tulee alla oleva:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Vastauksen vastatessa palvelimella olevaa tunnus-avain-paria yhteys hyväksytään, jolloin palvelin vastaa asiakkaalle viestillä ”200 OK”.[4]

3.1.2 Digest-autentikointi

Digest on IETF RFC-1945:n [3] Basic-autentikaatiosta RFC-2069:ssä [53] jatkokehitetty autentikointitapa. Kehitettäessä Digest-autentikaatiota tietoturvaa on pyritty huomioimaan. Digest perustuu haastevaste-tekniikkaan, jossa avain lähetään MD5-muotoisena kryptografisena tiivisteenä. Tekniikkana Digest ei ole vahva autentikointi, mutta turvallisempi verrattuna Basic-autentikaatioon. Suurimpana ongelmana on, ettei sillä vaikuteta itse siirtokanavaan, jolloin välimies-hyökkäyksen huomioiminen tai estäminen on mahdotonta.[4]

Alkuperäisessä standardissa palvelimen haaste lähetetään alla olevan BNF-syntaksin mukaisesti:

```
WWW-Authenticate        = "WWW-Authenticate" ":" "Digest"
                        digest-challenge

digest-challenge        = 1#( realm | [ domain ] | nonce |[
                        digest-opaque ] |[ stale ] | [ algorithm
                        ] )

realm                   = "realm" "=" realm-value
realm-value             = quoted-string
domain                  = "domain" "=" <"> 1#URI <">
nonce                   = "nonce" "=" nonce-value
nonce-value             = quoted-string
opaque                  = "opaque" "=" quoted-string
stale                   = "stale" "=" ( "true" | "false" )
algorithm               = "algorithm" "=" ( "MD5" | token )
```

Arvoista Realm-attribuutti kertoo asiakkaalle, minkä tunnus-avain-parin palvelin haluaa asiakkaalta. Domain on kenttä, mihin voidaan listata URI-osoitteita, joihin autentikaatio on lähetettävä. Nonce-arvo on palvelimen määrittämä satunnaisarvo, jonka asiakas sisällyttää MD5-funktioon tiivisteen laskemisen aikana lisäämään turvallisuutta. Opaque on arvo, jonka asiakas toimittaa muuttumattomana takaisin palvelimelle. Stale-arvolla palvelin voi pyytää asiakkaalta uuden vasteen. Voidaan käyttää silloin kun asiakkaan käyttämä Nonce-arvo on väärä. Algorithm-arvo kertoo mitä käytettävistä tavoista rakentaa Response-arvo käytetään. Alkuperäiseen on määritelty vain MD5, joten kenttä voidaan jättää tyhjäksi. [53]

Asiakkaan vaste lähetetään haasteeseen alla olevan BNF-syntaksin mukaisesti:

```
Authorization           = "Authorization" ":" "Digest" digest-
response

digest-response         = 1#( username | realm | nonce | digest-
                        uri | response | [ digest ] | [
                        algorithm ] | opaque )

username                = "username" "=" username-value
username-value          = quoted-string
digest-uri              = "uri" "=" digest-uri-value
digest-uri-value        = request-uri
response                = "response" "=" response-digest
digest                  = "digest" "=" entity-digest

response-digest         = <"> *LHEX <">
entity-digest           = <"> *LHEX <">
LHEX                    = "0" | "1" | "2" | "3" | "4" | "5" |
                        "6" | "7" | "8" | "9" | "a" | "b" | "c"
                        | "d" | "e" | "f"
```

Arvoista Username on käyttäjän käyttäjätunnus. Digest-uri on palvelimen osoite, johon otetaan yhteyttä. Response on asiakkaan tunnuksesta ja avaimesta, sekä palvelimen arvoista laskettu vastaus. Digest-kenttä on optionaalinen ja sen avulla voidaan toimittaa sisällölle tehtyjä Digest-attribuutteja. Näin toimien voidaan luoda luottamusta sisällön alkuperää kohtaan. Response-attribuutin vastausarvo lasketaan BNF-syntaksin mukaisesti alla olevalla tavalla:

```
response-digest   =     MD5(HA1 ":" nonce ":" HA2)
HA1               =     MD5(käyttäjänimi ":" realm-arvo ":"
                  avain)
HA2               =     MD5(yhteys-metodi ":" yhteys URI)
```

Edeltävässä syntaksissa ”yhteys-metodi” on asiakkaan käyttämä yhteydenottokeino, eli HTTP:n tapauksessa yleensä GET. Digest-algoritmissa omat ja palvelimelta saadut arvot palautetaan vasteessa takaisin Response-arvon lisäksi. [53]

Myöhemmin RFC-2069 [53] on korvattu RFC-2617:lla [4] ”HTTP Authentication:Basic and Digest Access Authentication”, johon on yhdistetty Basic-autentikaatio ja tehty pieniä muutoksia alkuperäiseen Digest-autentikaatioon. Muutokset koskevat joidenkin attribuuttien lisäyksiä ja taas poistoja haasteeseen ja vasteeseen. Muutoksilla tavoitellaan suurempaa turvallisuutta ja luottamusta autentikointiin. Näillä keinoilla mm. selkotekstihyökkäyksen toteuttaminen on aikaisempaa haasteellisempaa. RFC-2069:n määrittelemää tapaa kuitenkin käytetään, jollei haaste sisällä RFC-2617 määriteltyjä asioita ja käytettävää algoritmia ei ole määritelty tai se on määritelty arvolla ”MD5”.[4]

Haasteeseen lisäyksinä mukaan tuodut attribuutit ovat ”qop” ja ”auth-param”. Jälkimmäinen arvoista on tulevaisuuden laajennuksia varten. Tällöin haaste BNF-syntaksi on seuraavanlainen:

```
challenge          =  "Digest" digest-challenge

digest-challenge   = 1#( realm | [ domain ] | nonce |[ opaque ]
                   |[ stale ] | [ algorithm ] |[ qop-options ] |
                   [auth-param] )

domain             = "domain" "=" <"> URI ( 1*SP URI ) <">
URI                = absoluteURI | abs_path
nonce              = "nonce" "=" nonce-value
nonce-value        = quoted-string
opaque             = "opaque" "=" quoted-string
stale              = "stale" "=" ( "true" | "false" )
algorithm          = "algorithm" "=" ( "MD5" | "MD5-sess" |
                   token)
qop-options        = "qop" "=" <"> 1#qop-value <">
qop-value          = "auth" | "auth-int" | token
```

Vasteeseen tuodut lisäykset attribuutteina ovat käyttäjän luomat ”cnonce”, ja ”nonce-count”. Algoritmi kenttää on lisätty uusi MD5-sess vaihtoehto. Vasteesta on poistettu ”digest”-attribuutti. Alla on vaste muutosten jälkeen BNF-syntaksin avulla esitettynä:

```
credentials        = "Digest" digest-response
digest-response    = 1#( username | realm | nonce | digest-uri |
                   response | [ algorithm ] | [cnonce] | [opaque]
                   | [message-qop] | [nonce-count]  | [auth-
                   param] )

username           = "username" "=" username-value
username-value     = quoted-string
digest-uri         = "uri" "=" digest-uri-value
digest-uri-value   = request-uri   ; As specified by HTTP/1.1
message-qop        = "qop" "=" qop-value
cnonce             = "cnonce" "=" cnonce-value
cnonce-value       = nonce-value
nonce-count        = "nc" "=" nc-value
nc-value           = 8LHEX
response           = "response" "=" request-digest
request-digest     = <"> 32LHEX <">
LHEX               =  "0" | "1" | "2" | "3" | "4" | "5" | "6" |
                   "7" | "8" | "9" | "a" | "b" | "c" | "d" | "e"
                   | "f"
```

Lisäyksistä qop (Quality of Protection) -arvo kuvaa minkälaista laatua autentikoinnille vaaditaan. Arvot voivat olla Auth tai Auth-int. Auth tarkoittaa pelkkää autentikaatiota ja lisäys int tulee Integrity-sanasta, jolloin autentikaatioon tuodaan mukaan eheyttä ja kiistämättömyyttä header- ja body -rakenteiden välille. Cnonce (Client nonce) on asiakkaan luoma satunnaisarvo, jota käytetään tuomaan luottamusta autentikaatioon. Nonce-count, eli Nc on kasvava järjestysluku. Luvulla mahdollistetaan autentikoinnin voimassaolonaikana uudet pyynnöt palvelimelle kasvattamalla vain järjestyslukua. MD5-sess-algoritmin tapauksessa käyttäjänimi, realm-arvo ja avain sisällytetään HA1-funktioon MD5-tiivisteenä. [4]

Qop-arvon ollessa ”auth” tai ”auth-int” vaste-arvo lasketaan alla olevasti:

```
response-digest   =     MD5(HA1 ":" nonce ":" nonceCount ":"
                  clientNonce ":" qop ":" HA2)
```

Jos algoritmiksi on määritelty ”MD5-sess”, silloin tunnukselle ja avaimelle tehdään kahteen kertaan MD5-laskenta. HA1 lasketaan algoritmissa alla olevalla tavalla:

```
HA1   =     MD5(MD5(käyttäjänimi ":" realm-arvo ":" avain) ":"
      nonce ":" cnonce)
```

Qop-arvon ollessa ”auth-int”, HA2 funktio lasketaan alla olevalla tavalla:

```
HA2   =     MD5(Method ":" digest-uri ":" MD5(entityBody))
```

Digest-haaste-vaste viestinnästä perusluonteinen esimerkki, missä asiakas ottaa yhteyden palvelimelle. Esimerkissä asiakkaan käyttäjätunnus on ”testi” ja avain on ”a”. Asiakas aloittaa yhteyden oton tekemällä pyynnön ja kertomalla yhteystavastaan:

```
GET /priv/index.html HTTP/1.1
User-Agent: Jakarta Commons-HttpClient/3.1-rc1
Host: 192.168.0.112
```

Palvelin vastaa haasteella, missä WWW-Authenticate-attribuutin ala-attribuutteina ovat Digestin vaatimat kentät:

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.4.4
Date: Sat, 14 Jun 2014 16:19:48 GMT
Content-Type: text/html
Content-Length: 194
Connection: keep-alive
WWW-Authenticate: Digest algorithm="MD5", qop="auth",
realm="Tunnistus", nonce="5e504eea539c7624"
```

Asiakas vastaa authorization-attribuutilla, jonka ala-attribuutteina ovat palvelimelta saadut tiedot lisättynä asiakkaan tiedoilla:

```
GET /priv/index.html HTTP/1.1
Authorization: Digest username="testi", realm="Tunnistus",
nonce="5e504eea539c7624", uri="/priv/index.html", algorithm=MD5,
response="4017d6064e33b11686666c1d11886d7d", qop=auth,
nc=00000001, cnonce="d3eab7233a2341d8e3cee77005907b91"
User-Agent: Jakarta Commons-HttpClient/3.1-rc1
Host: 192.168.0.112
```

Lopulta tunnuksen ja avaimen vastatessa palvelimen arvoja hyväksyvä vastaus ja autentikaatio on valmis:

```
HTTP/1.1 200 OK
Server: nginx/1.4.4
Date: Sat, 14 Jun 2014 16:19:50 GMT
Content-Type: text/html
Content-Length: 53
Last-Modified: Tue, 04 Feb 2014 18:16:13 GMT
Connection: keep-alive
Authentication-Info: qop="auth",
rspauth="f0f601d594c1f29095f476e206628c48",
cnonce="d3eab7233a2341d8e3cee77005907b91", nc=00000001
ETag: "52f12e6d-35"
Accept-Ranges: bytes
```

3.2 Salaus ja eheys sovellustasolla

Sovellustasolla salausta voidaan toteuttaa protokollien omilla ratkaisuilla ja sovellustason päällä toimivien mekanismien avulla. Käsittelemissäni REST-protokollissa salauksen toteuttamiseen suoraan ei ole olemassa ratkaisuja, jolloin salauksen toteuttaminen jää sovellustason mekanismeille. Eheyden toteuttamiseen HTTP- tai CoAP -protokollissa ei ole kunnollista ratkaisua, koska ne luottavat alempien protokollien käyttämiin ratkaisuihin. Molemmat protokollat kuitenkin sovellustasoa ajatellen vaativat lisättyjä dataobjektitason mekanismeja eheyden valvontaan. Luottamukseen ja eheyteen voidaan vaikuttaa esitysmalleihin toteutetuilla ratkaisuilla. Ratkaisuja ovat JSON-puolella mm. JWT:hen (JSON Web Token) [37] pohjautuvat ratkaisut. XML-puolella vastaavat ratkaisut ovat SAML (Security Assertion Markup Language) [54] tai SOAP-laajennus WS-Security (Web Services Security) [55]. SAML:ia ja WS-Securityä voidaan käyttää toistensa kanssa yhdessä paremman turvan aikaansaamiseen. Saatavuuteen pystytään vaikuttamaan mm, protokollien uudelleenlähetysalgoritmeilla ja erilaisilla välimuistiratkaisuilla.[2][1][37][54][55]

Työhön valitusta JSON-toteutuksesta johtuen salauksen ja eheyden toteutukseen käytetään JWT:hen perustuvia JWS- ja JWE -tekniikoita. Edellisten sarjatietue voidaan tehdä kahdella eri tavalla, joista ensimmäinen on yhteensopiva sarja ja toinen on JSON-sarja. Yhteensopivassa muodossa lopputuloksena on URL-turvallinen merkkijono, jolloin sitä voidaan siirtää URI:n mukana pelkäämättä merkkijononhajoamista. JSON-sarjassa esitysmuotona on JSON-objekti. Objektin hyvänä puolena siihen voidaan sisällyttää useita allekirjoituksia tai salauksia, mutta haittapuolena on, että se ei ole URL-turvallinen ratkaisu. [35][36]

3.2.1 JWS-allekirjoitus

JWS (JSON Web Signature) [35] on JSON-rakenteiden suojaamiseen kehitetty tekniikka ja se on määritelty IETF:n ”JSON WEB Signature” draftissa. Tekniikan avulla JSON-rakenne voidaan digitaalisesti allekirjoittaa tai laskea sille HMAC-arvo. Yleensä JWS-notaatiota käytetään allekirjoittamaan digitaalisesti JSON Web Token -paketti. JSON-rakenteesta allekirjoitettu rakennepaketti koostuu Base64Url-koodatuista otsikosta, hyötykuormasta ja allekirjoitusavaimesta, jotka lopulta tehdään yksi sarjatietue.[35][37]

Otsikko pitää sisällään tiedon käytetyistä kryptografisista algoritmeista, jotka on määritelty JWA-spesifikaatiossa. Tämän pohjalta tiedetään miten hyötykuorma on allekirjoitettu tai minkälainen MAC-operaatio sille on tehty. Otsikko rakentuu kahden osan yhdistelmästä, joista toinen on suojattu ja toinen suojaamaton. Suojattu sisältää osat, jotka ovat eheydellisesti suojattu allekirjoituksella tai MAC-operaatiolla ja suojaamaton sisältää osat, joille ei ole tehty mitään operaatiota.[35][34]

JWS-yhteensopiva merkkijono koostuu kolmesta osasta pisteellä erotettuna alla olevan mukaisesti:

```
Base64Url(UTF8(JWS suojattu otsikko)),
Base64Url(JWS hyötykuorma) ja
Base64Url(JWS allekirjoitus)
```

Otsikko sisältää halutut otsikkotiedot, kattaen JWS-suojatun otsikon, mahdollisen tiedon JSON-sarjasta ja JWS suojaamattoman otsikon. Hyötykuorma on allekirjoitettava JSON-paketti. Allekirjoitus on otsikosta ja hyötykuormasta laskettu allekirjoitus. Kaiken ollessa valmis, paketista tehdään sarjatietue ja se voidaan toimittaa eteenpäin.

Alla on esimerkki JWS rakenteen tekemisestä. Algoritmina on HMAC SHA-256 ja hyötykuormana alla oleva JSON-rakenne:

```
{"iss":"joe",
     "exp":1300819380,
     "http://example.com/is_root":true}
```

Ensimmäisenä tehdään JSON-otsikko, jossa tyypiksi on asetettu JWT ja algoritmiksi JWA pohjalta HS256.

```
{"typ":"JWT",
      "alg":"HS256"}
```

JSON-otsikko ja hyötykuorma muunnetaan UTF-8-muotoiseksi tavutaulukoksi ja sille tehdään Base64Url-koodaus. Edeltävät yhdistetään pisteellä ja tehdään niistä ASCII-tavutaulukko. Nyt yhdistetystä otsikosta ja hyötykuormasta lasketaan tavutaulukomuodossa olevan avaimen avulla HMAC-tiiviste ja tehdään sille Base64Url-koodaus. Edeltävien vaiheiden jälkeen voidaan tehdä lopullinen JWS-yhteensopiva sarjatietue, joka tulee pistämällä kaikki yhteen pisteellä erotettuna:

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJqb2UiLA0KICJ
leHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp
0cnVlfQ.dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

[35][34]

Edellisen purkaminen tapahtuu nyt purkamalla osien Base64Url-koodaus ja tulkitsemalla otsikon ohjeita. [35][34]

3.2.2 JWE-salaus

JSON Web Encryption [36] on JSON-rakenteiden salaamiseen kehitetty tekniikka ja se on määritelty IETF:n ”JSON WEB Encryption” draftissa. JWE:n alla oleva rakenne koostuu viidestä Base64Url-koodatusta osasta, jotka käyttävät hyväkseen JWA-spesifikaatiota [34] määrittämään käytetyt salausalgoritmit: [36][34]

```
Base64Url(UTF8(JWE suojattu otsikko)),
Base64Url(JWE salausavain),
Base64Url(JWE alustusvektori),
Base64Url(JWE salateksi), ja
Base64Url(JWE autentikointimerkkiarvo)
```

Esimerkki, missä selkoteksti "Live long and prosper." salataan käyttäen AES Key Wrap- ja AES_128_CBC_HMAC_SHA_256 -algoritmia. Salausavain salataan AES Key Wrap-algoritmilla, jotta saadaan JWE-salausavain. Selkoteksti salataan käyttäen AES_128_CBC_HMAC_SHA_256-algoritmia. Tällöin otsikko määritellään alla olevasti:

```
{"alg":"A128KW","enc":"A128CBC-HS256"}
```

Edellä olevasta tehdään UTF8-muodossa oleva Base64Url-merkkijono:

```
eyJhbGciOiJBMTI4S1ciLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0
```

Tämän jälkeen luodaan satunnainen 256-bittinen sisällönsalausavain CEK (Content Encryption Key), joka salataan AES Key Wrap -algoritmilla. Näin saadaan symmetrinen JWE-salausavain. Salausavain JWK-muodossa, tuottaa alla olevan JSON-rakenteen:

```
{"kty":"oct",
      "k":"GawgguFyGrWKav7AX4VKUg"
}
```

Edellä olevan tavutaulukkomuodoksi Base64Url-koodauksen jälkeen tulee:

```
6KB707dM9YTIgHtLvtgWQ8mKwboJW3of9locizkDTHzBC2IlrT1oOQ
```

Satunnaisavaimen jälkeen tehdään satunnainen 128 bittiä pitkä JWE-alustusvektori, joka on Base64Url-muodossa:

```
AxY8DCtDaGlsbGljb3RoZQ
```

Kryptataan selkoteksti käyttäen AES_128_CBC_HMAC_SHA_256 algoritmia. Algoritmissa salausavain CEK toimii salausavaimena, JWE-alustusvektorina ja ylimääräisenä datana käytetään JWE-otsikkoa. Salatun tekstin ja siitä lasketun avaimellisen tiivisteen tavujonojen ASCII-muodon Base64Url-koodatut arvot ovat:

```
KDlTtXchhZTGufMYmOYGS4HffxPSUrfmqCHXaI9wOGY
```

ja

```
U0m_YmjN04DJvceFICbCVQ
```

Sarjallistettu JWE-kryptattu lopputulos saadaan lopulta Base64Url-koodattujen arvojen lopputuloksena, jolloin tulee alla olevan näköinen merkkijono:

```
eyJhbGciOiJBMTI4S1ciLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0.6KB707dM9YTI
gHtLvtgWQ8mKwboJW3of9locizkDTHzBC2IlrT1oOQ.AxY8DCtDaGlsbGljb3RoZ
Q.KDlTtXchhZTGufMYmOYGS4HffxPSUrfmqCHXaI9wOGY.U0m_YmjN04DJvceFIC
bCVQ
```

Purkaminen tapahtuu vastakkaisessa järjestyksessä tulkitsemalla otsikon ohjeet, mitä algoritmeja on käytetty.[36][34][33]

4 AKA AUTENTIKOINTI JA AVAINTENVAIHTO

AKA (Authentication and Key Agreement) [56] on 3G-autentikointiin kehitetty ratkaisu. AKA mahdollistaa kirjautuvan vahvan autentikoinnin ja istunnon luottamuksellisen avaintenvaihdon UMTS (Universal Mobile Telecommunications System) palveluita tukevissa verkoissa. Avaintenvaihto perustuu symmetristä kryptografiaa käyttävään haaste-vaste-mekanismiin. Sovellusten käyttöön AKA:sta on kehitetty AKA-Digest-autentikaatio, jonka avulla sovellukset pystyvät tekemään vahvan autentikoitumisen tavoitellulle palvelimelle. [56][5]

4.1 Eroavaisuudet Digest-autentikaatioon

RFC-2617:ssä [4] määritelty Digest-autentikaatio on toiminut pohjana AKA-Digest-spesifikaatiolle. AKA-algoritmin Digest-toteutus on pyritty pitämään identtisenä esikuvansa kanssa. Tehdyillä lisäyksillä on pyritty kasvattamaan virheensietoisuutta, sekä parannettu avaimesta laskettavan tiivisteen turvallisuutta. Lopputuloksena on pienillä muutoksilla toteutettu laajennus alkuperäiseen Digest-autentikaatioon. Alla olevat mallit on määritelty käyttäen BNF-syntaksia.[56]

Isoin muutos on uuden algoritmin määrittely, joka on nimetty AKA-algoritmiksi. Algoritmi annetaan haasteessa alla olevan esimerkin mukaisesti:

```
algorithm               =  "algorithm" EQUAL ( aka-namespace
                                             / algorithm-value )
aka-namespace           =  aka-version "-" algorithm-value
aka-version             =  "AKAv" 1*DIGIT
algorithm-value         =  ( "MD5" / "MD5-sess" / token )
```

Edellä olevan mukaisesti algoritmi on perustapauksena määritelty seuraavasti: algorithm=AKAv1-MD5, jossa ”v1” indikoi versiota. Toisena muutoksena on AUTS (Resynchronisation Token) -parametrin lisääminen vasteeseen. Asiakkaan havaitessa synkronointiongelman, eli lasketut avaimet eivät täsmää, se lisää vastaukseensa AUTS-parametrin. Palvelimen saadessa AUTS-parametrin, se tietää tarpeesta luoda ja toimittaa uusi haaste asiakkaalle.[56]

Haaste sisältää Digest-algoritmiin kuuluvan Nonce-arvon. Perinteisestä satunnaisarvosta poiketen AKA-algoritmissa Nonce on määritelty sisältämään RAND (Random Challenge) ja AUTN (Authentication Token), sekä mahdollisia palvelinkohtaisia arvoja. Edelliset arvot on sisällytetty peräkkäisinä merkkijonoina Base64-koodattuna. RAND- ja AUTN -arvot ovat 128 bittiä pitkiä ja kyseisten arvojen jälkeisiä palvelimesta riippuvaisten arvojen maksimikokoa ei ole määritetty. Arvojen esittämisestä alla esimerkki:

```
nonce                 =  "nonce" EQUAL ( aka-nonce
                            / nonce-value )
aka-nonce             =  LDQUOT aka-nonce-value RDQUOT
aka-nonce-value       =  <Base64 encoding of RAND, AUTN, and
                            server specific data>
```

[56][5]

AUTN-arvo jakaantuu vielä kolmeen eri arvoon. Ensimmäinen 48 bittinen osuus on toteutettu XOR-operaatiolla arvoista SQN (Sequence Number) sekä AK (Anonymity Key). Seuraava 16 bittiä pitkä osuus on AMF (Authentication Management Field) arvo. Viimeinen 64 bittiä pitkä osuus on MAC-A (Network Authentication Code)-arvo.[56][57][58][59]

Nonce-parametri sisältämine arvoineen on AKA-algoritmissa suunniteltu siirrettäväksi SIM-kortille tai sitä emuloivalle ohjelmistolle. Kortilla tai emulointiohjelmistolla sijaitsee operaattorin asettamat 128 bittinen OP (Operant Variant Algorithm Configuration Field) -arvo ja käyttäjän avain. [58][59]

Tietojen siirron jälkeen kaikki tarkistus- ja avain -arvot lasketaan käyttäen AKA-algoritmin määrittämää Milenage-algoritmia. Milenage-algoritmiin on määritelty kahdeksan eri funktiota, joita käyttäen saadaan laskettua autentikointiin liittyvät arvot. Näistä funktioista seitsemän palauttaa konkreettisen autentikointiin liittyvän arvon ja yksi apufunktio laskee arvon, jota tarvitaan muissa funktioissa. Apufunktiolla lasketaan OPc-arvo operaattorin SIM-kortille sisällyttämästä OP-arvosta, sekä asiakkaan salaisesta avaimesta. AKA-algoritmi on suunniteltu turvalliseksi riippumatta OP-arvon julkisuudesta, mutta operaattorit voivat saada lisäturvaa pitäessään sen salassa. OPc-arvolla vaikeutetaan erilaisten salausanalyysien ja väärennösten onnistumista. Apufunktion laskennan jälkeen päästää laskemaan taulukossa yksi esitellyt funktiot f1, f2, f3, f4,f5, f1* ja f5*.[58][59]

Funktio	Parametri 1	Parametri 2	Parametri 3	Parametri 4	Parametri 5	Lopputulos
F1	Salainen avain	RAND	OPc	SQN	AMF	MAC-A
F2	Salainen avain	RAND	OPc			RES
F3	Salainen avain	RAND	OPc			CK
F4	Salainen avain	RAND	OPc			IK, AK
F5	Salainen avain	RAND	OPc			AK
F1*	Salainen avain	RAND	OPc	SQN	AMF	MAC-S
F2*	Salainen avain	RAND	OPc			AK-S
OPc	Salainen avain	OP				OPc

Taulukko 1 Milenage-algoritmin funktiot ja niiden ottamat arvot ja mitä ne palauttavat.

Funktioiden lopullisista arvoista funktion F1 tuloksena tulevaa MAC-A-arvoa voidaan verrata AUTN-arvosta saatuun MAC-A-arvoon ja täten päätellä onko saatu nonce-arvo paikkaansa pitävä. Funktion F2 tuloksena tulevaa RES (Response to Challenge) -arvoa käytetään palvelimelle palautettavana avaintiivisteeseen sijoitettavana käyttäjän avaimena. Arvo sijoitetaan AKA-Digest-algoritmiin binäärisenä. Funktioista F3 ja F4 saatavat CK (Cipher Key) - ja IK (Integrity Key) -arvot yhdistämällä asiakas luo itselleen Ks-avaimen, mitä tarvitaan GAA-toteutuksessa. Funktioista F4 ja F5 saatavaa AK:ta (Anonymity Key) käytetään aikaisemmin määritettyyn XOR-operaatioon, jolla selvitetään SQN-arvo. Funktiosta F1* saatavaa MAC-S (The message authentication code) -arvoa käytetään AUTS-parametrin laskemiseen tekemällä SQN:n ja AK-S:n välillä XOR-operaatio ja yhdistämällä se MAC-S-arvoon. Funktiolla F2* lasketaan AK-S-arvo, jota tarvittiin edellä mainittuun XOR-operaatioon. [58][59][57]

4.2 GAA-arkkitehtuuri autentikoimiseen

GAA (Generic Authentication Architecture) [6] on 3GPP:n (3rd Generation Partnership Project) ratkaisu 3G-yhteyksien avulla toteutettavaan autentikointiin. GAA jakaantuu kahteen osaa jaetun salaisuuden ja sertifikaatteja käyttävään julkisen ja yksityisen avaimen ratkaisuihin. Jaetun salaisuuden ratkaisun periaatteet on kuvattu 3GPP TS33.220 [5] ja sertifikaatteihin perustuvan 3GPP TS33.221 [60] -spesifikaatiossa. [6][61]

Jaetun salaisuuden mahdollistavan autentikaation kehittäminen johtui 3GPP:n tarpeesta luoda vahva käyttäjätunnistus ja avaintenvaihto SIM-laitteille. Jaetun salaisuuden lähtökohtana on GBA (Generic Bootstrapping Architecture) -arkkitehtuurin [5] symmetrinen autentikointi. Siinä asiakas ja kolmannen osapuolen palvelin autentikoituvat operaattorille ja näin luottavat toisiinsa operaattorilta saatavan yhteisen avaimen ansiosta. Luottamus perustuu kaikkien osapuolien luottamukseen operaattoria kohtaan. Asiakas sopii käytettävästä avaimesta käyttäen vaihtoehtoisesti AKA- tai GBA-Digest -arkkitehtuuria [56]. [5][61]

Sertifikaatteihin perustuvan ratkaisun pohjana on SSC (Support for Subscriber Certificates) -määrittely [60]. Ratkaisussa asiakas käyttää operaattoria PKI (Public Key Infrastructure) -portaalina saadakseen tarvittavan sertifikaatin. Sertifikaatti haetaan operaattorilta käyttäen olemassa olevaa avainparia ja jollei niitä ole etukäteen asetettu voidaan niiden hakemiseen käyttää mm. GBA:ta. [5][61]

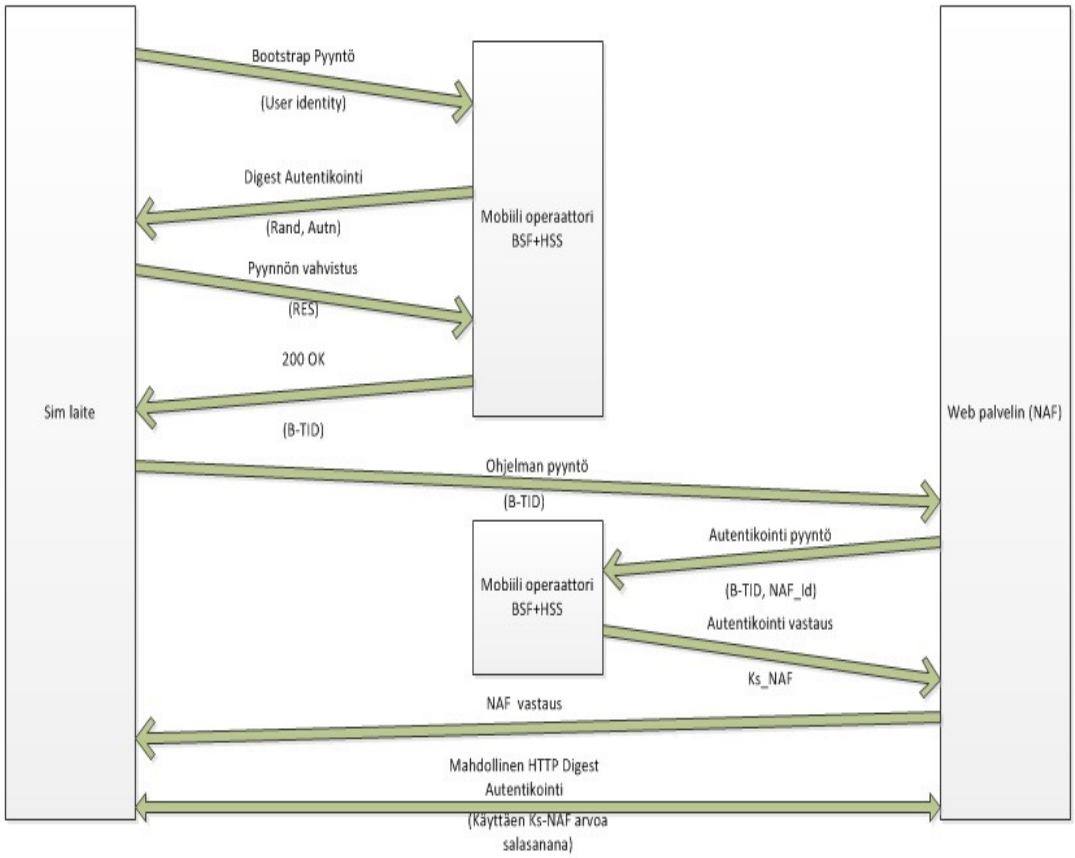
GBA:ta pidetään vahvana autentikoinnintapana, koska siinä on käytössä operaattorilta saadut tunnistearvot. AKA-tapauksessa sirun sisältämä 2G- tai 3G -laite ja sen käyttäjältä vaatima PIN-koodi. GBA-Digest-tapauksessa operaattorin arvot ja TLS-yhteys. Työn perustuessa resurssirajoitteisten laitteiden käyttöön ja teoreettisenkin ylimääräisen laskennan minimoimiseen käsittelen työssä vain GBA:han perustuvaa jaetun salaisuuden ratkaisua. [5][61]

4.3 GBA-arkkitehtuuri luottamuksen rakentamiseen

GBA:sta on olemassa kolme hieman erilaista versiota, joita voidaan kutsua 2G (GBA-ME)-, 3G (GBA-U)- ja GBA-Digest -ratkaisuiksi. Alkuperäinen 2G-versio on toiminut 3G- ja GBA-Digest -arkkitehtuurien pohjana, joten myöhemmät vain muokkaavat alkuperäistä toteutusta. 2G-versio perustuu SIM-kortin päällä toimivaan ohjelmaan, jonne operaattori on tallentanut avaimensa. Uudemmassa 3G-versiossa avaimet on talletettu USIM-ohjelmaan, joka sijaitsee UICC (Universal integrated circuit card) -piirillä SIM-kortissa. Itsenäisestä piiristä johtuen 3G-ratkaisua pidetään tietoturvallisempana. GBA-Digest on ratkaisu tilanteisiin, missä 2G- tai 3G -arkkitehtuureja ei voida käyttää SIM-kortin puuttumisen vuoksi. Tämän käyttäminen kuitenkin olettaa, että asiakkaalla on mahdollisuus käyttää HTTP-protokollaa yhteydenottoon. Asiakas (UE) voi aloittaa GBA-yhteyden NAF:in (Network Application Function) kanssa suoraan sen tukiessa GBA:ta. Muussa tapauksessa yhteydenotto täytyy aloittaa ilman GBA-parametreja. Parametrittomaan pyyntöön vastaukseksi GBA-palvelin vastaa tarvitsevansa GBA-yhteydenoton. GBA-käytön ollessa varmaa UE aloittaa bootstrapping-autentikaationeuvottelun operaattorin BSF (Bootstrapping Server Function) -palvelimen kanssa.[5][61][62]

Bootstrapping-autentikaatio jakautuu kahteen osaan. Ensin asiakas autentikoituu operaattorilleen kuvan 3 mukaan käyttäen jotain kolmesta algoritmista. Asiakas ja operaattori sopivat avaintenvaihdosta, eli kuvassa tehty ylempi yhteydenotto operaattorille. Autentikoinnin onnistuttua UE saa operaattorilta B-TID (Bootstrapping Transaction identifier) -arvon.[5][61][63]

Autentikoinnin jälkeen UE lähettää autentikoiduttavalle palvelimelle (NAF) B-TID-arvon, näkyy kuvassa ensimmäisenä pyyntönä NAF-palvelimelle. B-TID-arvon perusteella NAF voi hakea sitä vastaavan avainmateriaalin BSF:ltä. Edellinen näkyy kuvassa 3 NAF-palvelimen pyyntönä operaattorin palvelimelle. Operaattorin löytäessä oikeat avainmateriaalit, BSF palauttaa ne NAF-palvelimelle. Avainmateriaalin saamisen jälkeen NAF-palvelimella ja UE:lla on yhteinen salaisuus, jonka avulla ne voivat sopia keskenään tietojen vaihdosta. Yhteistä salaisuutta voidaan käyttää mm. kuvan 3 viimeisen yhteydenoton mukaisesti SIM-laitteen ja NAF-palvelimen välillä. Kuvan 3 mukaisessa ratkaisussa toteutetaan Digest-autentikointi yhteisen salaisuuden avulla UE:n ja NAF:in välillä. [5][61][63]



Kuva 3 Yleisen autentikointi arkkitehtuurin mukainen toteutus.[63]

UE ja BSF käyttävät yhteisen salaisen avaimen luomiseen KDF (Key Derivation Function) -funktiota. KDF-funktio jakaantuu useaan eri funktioon riippuen sisään otettavien parametrien määrästä ja minkälainen lopputulos halutaan. Funktion sisällä luodaan yhtenäinen merkkijono annetuista parametreista. Merkkijonossa aina ensin tulee parametri ja parametrin jälkeen tulee kyseisen parametrin pituus ilmoitettuna kahden oktetin pituisena. Funktion ensimmäisenä parametrina on aina spesifikaatiot toisistaan erottava FC (Carrier Centre Frequency) -numeroarvo, jonka jälkeen tulee annetut parametrit. 3GPP 33.220 spesifikaatiossa FC-arvona käytettävä väli on 0x00-0x0F, kuitenkin kaikissa löydetyissä esimerkeissä käytetään arvoa ”0x01”. Saadusta merkkijonosta lopulta lasketaan HMAC-SHA-256-tiiviste paluuarvoksi, minkä jälkeen se Base64Url-koodataan.[5][63][64]

4.3.1 2G-arkkitehtuuri

Autentikaatio tapahtuu kuvan kolme mukaisella tavalla ja siten onnistuneessa autentikaatiossa tehdään ensin kaksi erillistä pyyntö-vastaus-yhteydenottoa UE:n ja BSF:n välillä. Pyyntöjen jälkeen SIM-laite aloittaa yhteydenoton NAF-palvelimelle, joka omien tarkistusten jälkeen ilmoittaa asiakkaalle yhteyden onnistumisesta. Ensimmäisessä pyynnössä operaattorille asiakas ilmoittaa halusta AKA-autentikaatioon. Toisessa pyynnössä asiakas osoittaa operaattorin antamien parametrien ja omien tietojen avulla tekemiensä laskujen avulla olevansa väittämänsä asiakas.[61][5]

Näistä ensimmäisellä yhteydenottokierroksella operaattorille UE aloittaa lähettämällä käyttäjätunnuksensa. Tämän tunnuksen pohjalta BSF hakee käyttäjäpalvelimiltaan HSS (Home Subscriber Server) käyttäjän autentikointiasetukset ja autentikointivektorin. Autentikointivektori koostuu satunnaisesta merkkijonosta (RAND), autentikaatiotokenista (AUTN), oletetusta vastauksesta (XRES), salausavaimesta (CK) ja eheysavaimesta (IK). Edellä mainituista arvoista operaattori lähettää asiakkaalle RAND- ja AUTN -arvot haasteeksi. [61][5][63]

Asiakkaan saatua haasteen, se ajaa AKA-algoritmin käyttäen SIM-korttiaan. AKA-algoritmin avulla varmistetaan BSF:ltä saatu AUTN-arvo ja lasketaan RES-arvo, sekä tila-avaimet CK ja IK. Avaimet CK ja IK yhdistämällä asiakas saa itselleen avaimen (Ks). [61][5][63]

Arvojen laskemisen jälkeen UE aloittaa toisen pyyntö-vastaus-yhteydenoton lähettämällä toisen pyynnön. Pyynnön mukana asiakas toimittaa laskemansa RES-arvon BSF:lle. BSF pystyy nyt varmistamaan UE:n aitouden vertaamalla RES-arvoa autentikointivektorin XRES-arvoon. Arvojen vastatessa toisiaan BSF hyväksyy yhteyden ja luo B-TID-arvon RAND-muuttujasta ja BSF tunnisteesta. B-TID-arvon ja avaimen elinajan BSF lähettää UE:lle hyväksyvän vastaus viestin ”200 OK” mukana. Viestin pohjalta asiakas tietää kaiken olevan kunnossa ja voi näin aloittaa yhteydenoton kolmannen osapuolen NAF-palvelimelle. [61][5][63]

Ks-avaimen pohjalta UE voi nyt laskea Ks_NAF-avaimen käyttäen KDF-funktiota [5]. KDF-funktion parametrit on määritelty muodossa KDF(Ks, ”gba-me”, RAND, IMPI, NAF_ID). Arvoista Ks ja tekstiarvon ”gba-me” lisäksi RAND on satunnaisluku, IMPI (IM Private Identity) on salausavain ja NAF_ID on yhdistelmä FQDN (Fully Qualified Domain Name) - ja Ua security protocol identifier -arvoista. Ua on viisi oktettia pitkä binääri-lukujono. Eri käyttötarkoituksia varten on Ua:n arvoksi määritelty eri jonoja, oletuksena voidaan pitää nollista koostuvaa merkkijonoa. [61][5][63]

UE aloittaa yhteydenoton NAF:ille pyynnöllä, jonka mukana se toimittaa aikaisemmin saamansa B-TID-arvon, kuvassa 3 ”Ohjelman pyyntö”. Tapaa toimittaa B-TID-arvo ei ole määritelty, joten se on NAF:in ja UE:n välinen asia. [61][5][63]

Saatuaan B-TID-arvon UE:lta, NAF lähettää autentikointi pyynnön BSF:lle, kuvassa 3 ”Autentikointi pyyntö”. Pyynnössä NAF lähettää tietonsa ja UE:lta saamansa B-TID-arvon. Operaattori vastaa viestiin B-TID-arvoa vastaavilla avainarvoilla. NAF-palvelimen pyynnössä BSF:lle mukana menee NAFid (NAF Identifier), joka sisältää yhteysosoitteen ja Ua:n salausprotokollatunnisteen. NAFid:n pohjalta operaattori pystyy tunnistamaan NAF-palvelimen ja varmistamaan sen aitouden. B-TID-arvon pohjalta BSF tunnistaa UE:n. Operaattorin havaitessa kaiken olevan kunnossa se lähettää Ks_NAF-avaimen ja bootstrapping ajan sekä avaimen voimassa oloajan NAF:ille. Jollei jotain tietoja löydy tai ne eivät ole oikeat BSF lähettää hylkäysviestin NAF:ille, joka tämän jälkeen lähettää hylkäysviestin UE:lle. [61][5][63]

Onnistuneen autentikoinnin jälkeen asiakkaalla ja NAF-palvelimella on identtiset Ks_NAF-avaimet, eli yhteinen salaisuus. Salaisuuden avulla asiakas ja NAF voivat nyt salata käyttämänsä liikenteen tai toteuttaa esim. kuvan 3 viimeisen vaiheen, eli Digest autentikaation toistensa välillä käyttäen Ks-NAF-arvoa avaimena. [61][5][63]

4.3.2 3G-arkkitehtuuri

3G eli GBA-U-arkkitehtuurin suurin muutos 2G-arkkitehtuurin on laskutoimitusten toteuttaminen SIM-laitteen UICC-piirillä. Tällöin piiri antaa eteenpäin vain oleelliset tiedot, jolloin esim. avaimet CK ja IK jäävät pysyvästi UICC-piirille. Näin estetään tilanne, missä joku ulkopuolinen taho pystyisi jotenkin hyödyntämään laskettuja arvoja.

Autentikaatio menee 2G-arkkitehtuurin mukaisesti, mutta nyt BSF:n lähettämän haasteen RAND- ja AUTN -arvot SIM-laite ohjaa suoraan SIM-kortin UICC-piirille. UICC laskee RAND- ja AUTN -arvoista CK-, IK- ja RES -arvot. RES-arvon UICC antaa SIM-laitteelle lähetettäväksi BSF:lle. Nyt sekä BSF ja UICC ovat kykeneviä laskemaan Ks_NAF-avaimen, kuten GBA-ME tapauksessa.

GBA-U:n toinen ero 2G-arkkitehtuuriin tulee avaimien luomisessa, missä yhden Ks_NAF-avaimen sijaan lasketaan avainten luontifunktiossa kaksi erillistä avainta. Avaimet ovat Ks_ext_NAF ja Ks_int_NAF. Avaimista ensimmäinen vastaa 2G-toteutuksen Ks_NAF-avainta. Käyttöproseduuri 3G-arkkitehtuurissa menee UE ja NAF:in välillä samoin, kuten 2G-ratkaisussa. Kahden erillisen avaimen laskeminen kuitenkin mahdollistaa NAF-palvelimen tukiessa jommankumman tai molempien käyttämistä. Kahden avaimen ratkaisulla voidaan lisätä joissain määrin tietoturvaa, koska asiakas ja palvelin voivat mm. salata ja allekirjoittaa liikenteen eri avaimilla. Toisena keinona on yhdistää avaimet jollain tavalla, jolloin saavutetaan pidempi ja siten tietoturvallisempi avain. [61][5]

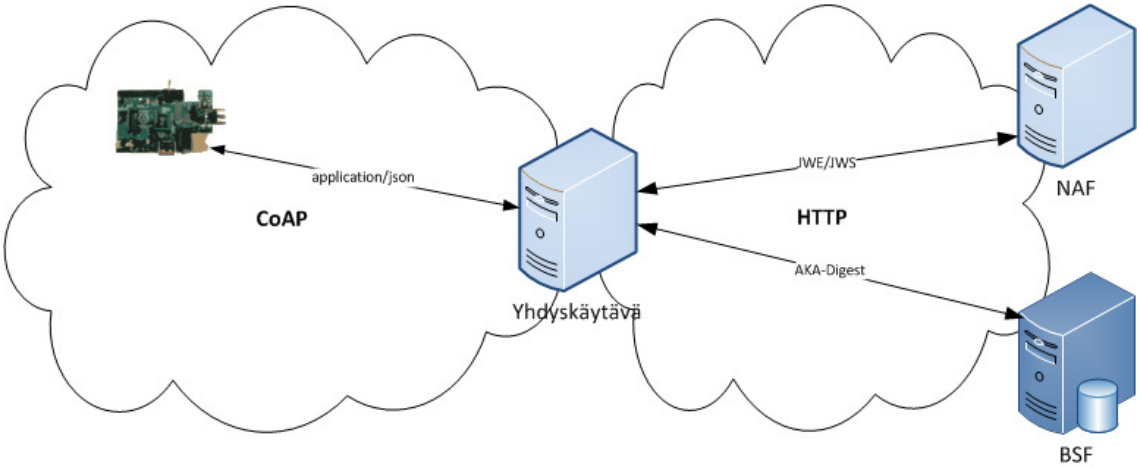
4.3.3 GBA-Digest-arkkitehtuuri

GBA-Digest-arkkitehtuuri on kehitetty tilanteisiin, missä ei ole mahdollista käyttää 2G- tai 3G -autentikointia ja avaintenvaihtoa. Arkkitehtuurissa oletus on, ettei ole olemassa SIM-laitetta tai siihen verrannollista keinoa säilöä operaattorikohtaisia arvoja. Operaattorikohtaisten arvojen puuttumisen takia Milenage-algoritmia ei voida käyttää ja siten AKA-algoritmin käyttämisestä tulee mahdotonta. Vaihtoehdoiksi jää käyttää HTTP-Digest-algoritmia autentikoitumiseen. Tällöin tietoturvan taso laskee, joten spesifikaatioon on tuotu vaatimus TLS-yhteyden luomisesta. TLS-yhteys tarvitaan eheyttä ja kiistämättömyyttä varten, joten TLS:n tarjoaman salauksen käyttö ei ole pakollista.[5]

Haaste-vaste-viestintä kulkee nyt Digest-spesifikaation mukaisesti käyttäjältä BSF:lle. Yhteydenotossa käyttäjä laittaa Authorization-pyyntöönsä käyttäjätunnisteen Username-kenttään ja sisällyttää tokenin kertomaan GBA-Digest-yhteydestä. Käyttäjätunniste on TMPI (Temporary identity), jos TMPI on liitetty IMPI-arvoon, muuten käytetään suoraan IMPI-arvoa. Digest-vasteen response-kenttään UE generoi avaimen KDF-funktiolla. Funktion sisään menevät: A1-arvo, teksti ”GBA_Digest_RESP” ja TLS master -avain. A1-arvo on muodostettu käyttäjänimestä, avaimesta ja realm-arvosta, joista lasketaan HMAC-arvo. UE:n vasteen ollessa kunnossa BSF vastaa ”200 OK” viestillä ja toimittaa mukana B-TID-arvon. Nyt UE ja BSF luovat Ks-arvon samalla KDF-funktiolla lisäämällä funktion loppuun Response-arvon. Saatuaan vastauksen BSF:ltä, UE voi aloittaa yhteydenoton NAF-palvelimelle. [5]

5 TOTEUTUS

Toteutuksen tavoitteena on mahdollistaa kuvan 4 mukaisen asiakas-anturin autentikointi, avaintenvaihto ja tietojensiirto kuvan NAF-palvelimelle. NAF ja operaattorin kirjautumispalvelin BSF toimivat HTTP-protokollalla ja kuvan anturi käyttää CoAP-protokollaa. Kuvan 4 BSF-palvelin käyttää autentikointiin AKA-Digest-autentikointia ja anturi, sekä NAF suojaavat liikenteensä BSF:ltä saadulla avaimella.



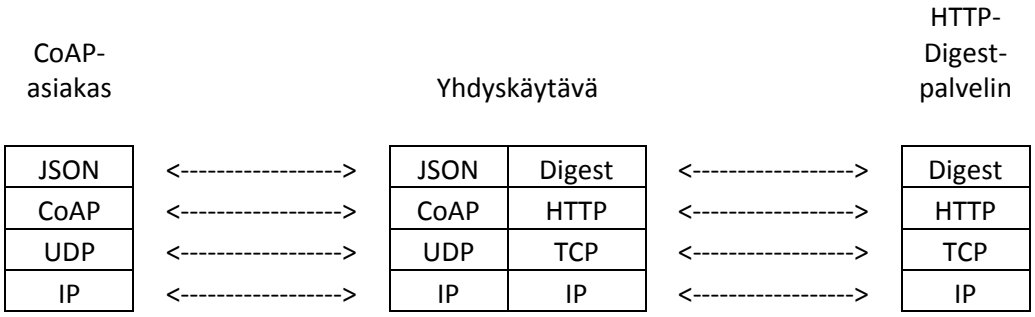
Kuva 4 Yleiskuva toteutuksen mukaisesta CoAP- ja HTTP -protokollien verkkoyhteydestä.

Työssä pääasiana on luoda kokeiluluontoinen kuvan 4 mukainen yhdyskäytävä, sekä sen testaamiseen tarvittavat anturi ja NAF-palvelin. Yhdyskäytävän tarkoitus on mahdollistaa anturin tiedonsiirto ja autentikointi BSF- ja NAF -palvelimille. Kuvan 4 protokollista CoAP käyttää kuljetuskerroksella UDP-protokollaa ja HTTP käyttää TCP-protokollaa tiedon siirtämiseen. Tällöin yhdyskäytävän läpi kulkevan liikenteen protokollapino on kuvan 5 mukainen.

CoAP-asiakas		Yhdyskäytävä			HTTP-palvelin
CoAP	<---------->	CoAP	HTTP	<---------->	HTTP
UDP	<---------->	UDP	TCP	<---------->	TCP
IP	<---------->	IP	IP	<---------->	IP

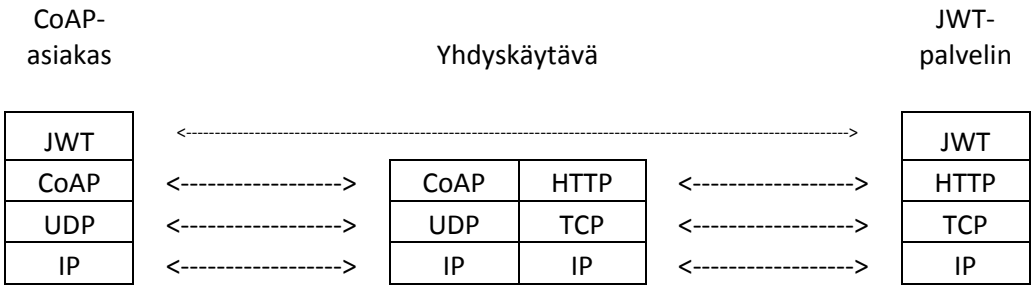
Kuva 5 Asiakkaan ja HTTP-palvelimen käyttämä protokollapino normaalissa yhteydenotossa yhdyskäytävän läpi.

Tiedon välittäminen vaatii luotettavaa autentikaatiota ja avaintenvaihtoa. Näiden mahdollistamiseen resurssirajoitteisilla laitteilla on suositeltavaa käyttää symmetristä avainta mm. laskennan minimoimiseksi, jolloin HTTP- ja AKA -Digest-autentikointitavat sopivat käytettäväksi. Protokollista CoAP ei suoraan tue minkäänlaista Digest-autentikointia, joten sen käyttämistä varten tarvitaan päällä käytettävää esitysmallia. Työssä käytettäväksi esitysmalliksi aikaisemmin valittiin JSON, jolloin yhdyskäytävä muuntaa Digest-attribuutit havaitessaan JSON-notaatioksi. Asiakkaalta JSON-muodossa tulevat attribuutit yhdyskäytävä siirtää Digest-vasteeksi HTTP-pyynnön otsakkeeseen. Näin toimien yhdyskäytävän läpi kulkevan liikenteen protokollapino on alla olevan kuvan 6 mukainen:



Kuva 6 Asiakkaan ja HTTP-palvelimen käyttämä protokollapino Digest-autentikaatiossa yhdyskäytävän läpi.

Autentikoinnin tapahtumisen jälkeen tavoite on kuljettaa liikenne CoAP-asiakkaan ja HTTP-palvelimen välillä. Pakettien allekirjoittaminen ja salaamisen toteuttamiseen valittiin JWT-paketointi ja sen mahdollistamat JWS, sekä JWE. JWT-paketteja käytettäessä liikenne kulkee kuvan 5 mukaisesti, mutta nyt päälle kulkee JWT-esitysmallikerros. Päällä käytettävän esitysmallin tapauksessa yhdyskäytävän läpi kulkeva liikenne menee alla kuvatun kuvan 7 mukaisesti:



Kuva 7 Asiakkaan ja JWT-suojausta käyttävän palvelimen protokollapino normaalissa yhteydenotossa yhdyskäytävän läpi.

Yhdyskäytävän toiminnan osoittamiseen ja testaamiseen tarvitaan asiakasimplementaatiota, eli kuvan 4 anturia osoittamaan valitun tavan mukaisen yhdyskäytävän toiminta. Asiakkaan tehtäviksi tulee tehdä CoAP-pyynnöt kuvien 5, 6 ja 7 mukaisesti ja käsittelemään pyyntöihin tulevat vastaukset. Vastausten mahdollisesti sisältäessä JSON-notaatioon sijoitettuja HTTP- tai AKA -Digest haasteita on asiakkaan rakennettava niihin vastaukset ja lähetettävä takaisin palvelimelle. Näiden jälkeen asiakkaan on kyettävä siirtämään liikenne JWT-käyttäen.

Työn toteutukset päätettiin tehdä Java-kielellä. Java valikoitui valmiiden kirjastojen helpon saatavuuden vuoksi, jolloin se soveltuu hyvin demonstraatiotarkoituksiin.

5.1 HTTP-CoAP-yhdyskäytävän toteutus

Yhdyskäytävän toteutus jakautui aluksi kahteen itsenäiseen kokonaisuuteen, joiden yhdistämisellä yhdyskäytävä toteutettiin. Kokonaisuuksista ensimmäinen oli CoAP-liikenteen vastaanotto ja lähettäminen takaisin CoAP-asiakkaalle. Toisena oli HTTP-yhteyden luominen ja palvelimelta saatavien autentikointitietojen vastaanottaminen ja niihin vastaaminen.

5.1.1 CoAP-protokollan vastaanotto ja lähetys

CoAP-protokollan toteuttamiseen Java-kielessä löytyy useita erilaisia kirjastoja. Kirjastoja ovat mm. NCoAP [65], JCoAP [66] ja Californium [67]. Kokeiluissa Californium osoittautui toteutuksellisesti laajimmaksi verrattuna muihin kirjastoihin ja samalla helppokäyttöiseksi, joten se tuli valittua pohjaksi toteutukselle.[65][66][67]

Californium tarjoaa luokat asiakkaan ja palvelimen toteutuksille ja mahdollisti kaikki tarvitut toiminnot. Vaikkei työssä olekaan tarvetta DTLS-tuelle, senkin tuki löytyy Californiumista. Californiumin tarjoaa valmiita esimerkkejä, joiden avulla pystyi saman tien keskittymään palvelimen resurssien luomiseen, sekä testaamaan niihin asiakkaan yhteydenottoja.[67]

Yhdyskäytävän toteutus lähti malliresurssista, jossa luodaan nimen mukaan tietoja sisältäviä resursseja rekursiivisesti resurssien sisään. Esimerkiksi URI-osoitteessa aina ”/” merkillä kerrotaan alkavan uuden aliresurssin. Näin yhdyskäytävä voidaan toteuttaa rakentamalla peräkkäisiä aliresursseja. Ennestään tuntemattoman osoitteen jälkeen luodaan sen niminen resurssi ja jos tulee ”/” merkki, aloitetaan aina uuden aliresurssin luominen. Työssä ”Yhteys” on osoite resurssiin, mikä kutsuttuna luo sen perään annetun määrän aliresursseja. Kutsut resurssiin tapahtuvat käyttäen CoAP-protokollan resurssimetodeita GET, PUT, POST ja DELETE. Alla esimerkkipyyntö:

```
”GET coap://localhost:5683/Yhteys/www.tut.fi/pop”
```

Ensimmäisenä URI:ssa tulee käytetyn protokollan määrite, joka on ”coap”, määritettä seuraa palvelimen sijainti. Yhdyskäytävä sijaitsee tässä tilanteessa samalla koneella portissa 5683. Portin jälkeen tulee toteutettu ”Yhteys” resurssi, minkä perään sijoitetaan tarvittu määrä aliresursseja, joista ensimmäinen on ”www.tut.fi” ja sen aliresurssina on ”pop”. Pyynnöllä haetaan rekursiivisesti viimeisen resurssin tiedot ja jollei aliresursseja ole luotu, ne luodaan tavoitellessa viimeistä aliresurssia.

Resurssipuun rakentumisen jälkeen resurssi selvittää resurssin koko nimen mukaan lukien edeltävät resurssit. Resurssin koko nimen selviämisen jälkeen tehdään yhteyspyyntö siihen osoitteeseen HTTP-luokalle. Pyynnössä resurssin nimi on tuleva yhteysosoite. Toteutuksessa asiakkaan on suunniteltu aloittavan yhteydenoton GET-pyynnöllä, jolloin sille luodaan resurssi. Resurssi tekee HTTP-luokalle pyynnön ja odottaa siltä vastauksen, vastauksen jälkeen yhdyskäytävä ohjaa vastauksen asiakkaalle. Vastauksen sisältäessä Digest-haasteen asiakas luo sille Digest-vasteen ja päivittää luomaansa resurssiin tietoja PUT-komennolla. Yhdyskäytävän toteutuksessa PUT-komennon odotetaan sisällyttävän Digest-header-attribuutteja pyyntöön. PUT-komennolla yhdyskäytävän resurssi kutsuu samoin HTTP-asiakasta, kuten GET-pyyntö, mutta välittää MAP-rakenteessa sille Digest-attribuutit. Resurssi odottaa HTTP-asiakkaalta vastauksen ja välittää vastauksen CoAP-asiakkaalle. Vastauksiin yhdyskäytävä pyrkii sijoittamaan oikean MIME-tyypin, jolloin esim. autentikaation tapauksessa viestiin sisällytetään JSON:in MIME-tyyppi. Asiakas pystyy nyt MIME:n perusteella lukemaan viestin JSON-sisällön.

Viimeisen kuvan 7 tapauksessa tarvitaan POST-metodia, joka ottaa vastaan nimi-arvopareja sisältävän JSON-merkkijonon. Tämä nimi-arvopareja sisältävä JSON-merkkijono muutetaan MAP-rakenteeksi ja välitetään pyyntönä HTTP-luokan POST-metodille. Pyyntöön saamansa vastauksen POST-metodi lähettää merkkijonona takaisin asiakkaalle. Työn edetessä edellinen osoittautui ongelmalliseksi, koska ollaan välittämässä vain yhtä merkkijonoa. Tämän takia mahdollistettiin POST-rakenteen toteutus siten, että se ottaa vastaan merkkijonon ja välittää sen HTTP-luokalle.

Toteutusvaiheen testauksessa Californium-kirjaston versio 13 kieltäytyi luomasta uusia resursseja, mutta onneksi ongelma korjaantui uudemman version 18 myötä. Version myötä kirjastoa oli muutettu reilumman puoleisesti, joten se pakotti kirjoittamaan yhdyskäytävän toteutusta uudelleen. Näiden ongelmien jälkeen toteutukseen vaaditut asiat ovat toimineet hyvin.

5.1.2 HTTP-protokollan lähetys ja vastaanotto

HTTP-asiakasluokan vaatimuksiksi asetettiin alussa kolme asiaa. Ensimmäinen oli HTTP-yhteyden luominen ja saadun vastauksen palauttaminen takaisin. Seuraavana asiana oli Digest-autentikaation hallitseminen ja sen header-kentän attribuuttien talteenotto, sekä muokkaaminen. Kolmantena oli POST-datan lähettämien ja siihen vastauksen saaminen, millä mahdollistetaan autentikoinnin jälkeinen liikenne. Edeltävillä saavutetaan kuvien 5, 6 ja 7 toteutusvaatimukset HTTP-lähetyksen osalta.

HTTP-yhteyden luomiseen löytyy erilaisia kirjastoja Javalle, kuten Java net [68], Apachen HTTP Client [69] ja Apachen HTTP Components [70]. Esimerkkien pohjalta havaittiin Apachen HTTP Client -kirjasto riittävän yksinkertaiseksi, mutta kuitenkin tarpeeksi laajaksi työhön.[70][69][68]

HTTP-luokan toteutuksen suunnitteluvaiheessa päädyttiin kolmen metodin ratkaisuun, missä ensimmäinen luo GET-pyynnön sille annettuun osoitteeseen ja sen jälkeen palauttaa saadun vastauksen MAP-tietorakenteeseen laitettuna. Tietorakenne sisältää mahdollisesti palvelimelta saatavan Digest-parametrin ja muun sisällön. Toinen metodi tehtiin Digest-authorization-attribuuttien välittämiseen. Metodi ottaa vastaan osoitteen ja MAP-rakenteen ja lopulta palauttaa HTTP-palvelimen vastauksen merkkijonona. Tämä metodi odottaa edellisestä poiketen nyt MAP-rakenteeseen Digest-attribuutteja, jotka voidaan toimittaa HTTP-pyynnössä eteenpäin. Vastauksena HTTP-palvelimelta metodi odottaa saavansa HTTP-sisältöä, minkä se lopulta palauttaa pyytäjälle. Kolmas POST-metodi ottaa vastaan MAP-rakenteen, joka sisältää nimi-arvo-pareja ja siirtää ne POST-pyyntöön ja lähettää saamaansa osoitteeseen. Välittämisen jälkeen parametri odottaa saavansa HTTP-sisältöä, jonka se lopulta välittää eteenpäin.

Työn kuluessa lähdettiin korvaamaan POST-metodia rakenteella, joka ottaa vastan lähetettävän merkkijonon ja MAP-rakenteen mahdollisia HTTP-header-atribuutteja varten. Edeltävä rakenne mahdollistaa keksin sisältämän POST-pyynnön tekemisen BSF:lle ja toimii yksinkertaisemmin UE:n ja NAF:in välillä, kun lähetetään JSON-merkkijonoa.

5.2 JWT-tietoturvan toteutus

JWT-tietoturvassa työn osalta oleellista oli viestien allekirjoitus ja salaus. Allekirjoitus pystyttiin toteuttamaan JWS-notaatiolla ja salaus JWE-notaatiolla. Javaan on erilaisia kirjastoja toteuttaa molempia, kuten Jsoncrypto [71] ja Nimbus JOSE+JWT [72]. Edeltävistä valittiin työhön jälkimmäinen. Nimbus mahdollistaa laajan skaalan erilaisia JWT-operaatioita, joten sen avulla työtä pystyy tarvittaessa jatkokehittämään.

B-TID-arvon sisällyttämiseen JWE- tai JWS -notaatioissa ei ole mitään suoraa tapaa. Yksi selkeä tapa olisi kehittää uusi algoritmi JWA-draftin mukaisesti [34], mihin kuuluisi B-TID-arvo. Algoritmi sisältäisi B-TID-arvon, mahdollisia kriittisyysparametreja ja ohjeet kuinka vastapää tulkitsee sisällön tai käyttää BSF:ltä tulevaa avainta. Kuitenkin työssä algoritmin kehittäminen menisi työn aiheen ulkopuolelle, joten päädyimme sijoittamaan B-TID-arvo otsakkeeseen omaan B-TID-kenttään. Tämä oli helppo toteuttaa, koska otsakkeeseen saa sijoittaa rajattomasti vapaavalintaisia kenttiä, joilla voi tarkentaa sisältöä. Yhtenä arvona voisi sisällyttää ”crit” attribuuttiparin, jolla voitaisiin kertoa tarvittava GBA-autentikointi tyyppi. Kentän avulla vastaanottaja tietäisi esim. sisällön olevan GBA-riippuvainen, kuitenkin työn osalta asia oli selvä, joten sitä ei sisällytetty. Muutoksen jälkeen otsikko näyttää allekirjoituksen tapauksessa alla olevalta:

```
{
      ”typ”:”JWT”,
      ”alg”:”HS256”,
      ”btid”:”VUQTvtcXP8ftP6xYpp7PYg==@p133.piuha.net”
}
```

Työhön ongelmia toi Nimbus-kirjaston keskeneräisyys. Kirjastoon tuli kiitettävällä vauhdilla päivityksiä, joten kaikki tarvittavat toimi lopulta ongelmitta, mutta alkuun mikään ei tuntunut toimivan. Kirjaston yksi iso ongelma on sen käyttämien kirjastojen määrä, joita ei ole sisällytetty itse kirjastoon. Työhön on edellisen takia jouduttu sisällyttämään ylimääräisiä kirjastoja, kuten Javan mail-kirjasto, jotta Nimbus tekee haluttuja asioita.

5.3 CoAP-asiakkaan toteutus

CoAP-asiakasta toteutettiin samanaikaisesti yhdyskäytävän kanssa testaamisen mahdollistamiseksi. Asiakkaan toteutuksen pyrkimyksenä oli yksinkertainen toteutus, jolla pystytään havainnollistamaan yhdyskäytävän toiminta. Näitä varten asiakasluokan on pystyttävä tekemään vaaditut CoAP-yhteydenotot, sekä HTTP- ja AKA -Digest vaatimusten mukaisen vasteen luonti. Myöhemmässä vaiheessa tehtiin toteutus, jolla mahdollistettiin tiedon lähettäminen käyttäen JWT-notaatiota.

Asiakasluokka käyttää hyväkseen JSON-operaatioihin liittyvissä asioissa Json-luokkaa ja CoAP-protokollaan liittyvissä asioissa Coaptoteutusta. Coaptoteutus on CoAP-asiakasluokka, joka toteuttaa tehdyn yhteyspyynnön. Ohjelman toiminnot jakautuvat näin pääluokkaan StartCoap, missä määritetään pyyntötapa, käytetty osoite ja mahdollinen sisältö pyyntöön. Sisältöä tai saatua vastausta voidaan halutessaan käsitellä Json-luokan avulla. Coaptoteutus lopulta tekee yhteyspyynnön ja palauttaa saamansa vastauksen pääluokalle.

Pääluokka vastaa Digest-operaatioihin liittyvistä asioista. Näitä ovat tyhjän Digest-vastepyynnön luominen, vasteen tekeminen saatuun haasteeseen ja AKA-autentikaation tapauksessa SIM-kortissa tehtävän laskennan emuloiminen. Emulointi käsittää kaksi asiaa Milenage-funktioiden laskemisen ja niistä avaimen muodostamisen AKA-autentikaation ensimmäiseen vaiheeseen ja KS-NAF-arvon laskemisen.

Milenage-funktioiden toteutukseen löytyi valmis kirjasto ”ICT ADAMANTIUM” projektista, mikä pyrkii rakentamaan rajapintoja erilaisten multimediatoteutusten välille. Kirjaston avulla pystytään laskemaan Milenage-funktioiden arvot.[73][74]

KS-NAF-arvon laskenta toteutettiin työhön. KS-NAF-arvo rakentuu merkkijonoista ja niiden pituudesta, sekä kaiken aloittavasta ensimmäisestä arvosta, joka on ”01”. Ensimmäisen arvon jälkeen tulee haluttu merkkijono ja sen jälkeen merkkijonon pituus kahdella oktetilla ilmoitettuna. Työn tapauksessa ensimmäinen oktetti on aina nolla, koska pituus ei tule kasvamaan yhtä oktettia pidemmäksi. Ensimmäistä arvoa seuraa GBA-tyyppi, mikä työssä on GBA-ME, eli 2G-AKA-autentikaatio. Tyypin jälkeen tulee BSF:n lähettämä RAND-merkkijono. RAND-arvon jälkeen tulee IMPI ja viimeisenä NAFID. Syntyneelle merkkijonolle lasketaan HMAC256-arvo käyttäen Milenagesta saatujen CK- ja IK -avaimien yhdistelmää.

Pääluokan Main-metodin tehtäviksi jää luoda pyyntö ja tulostaa saatu vastaus. Vastaus sisältäessä Digest-haasteen Main-metodi ohjaa sen metodille, joka tekee vasteen, jonka jälkeen Main lähettää vasteen sisältämän pyynnön palvelimelle.

5.4 NAF-palvelimen toteutus

HTTP-protokollaa käyttävä NAF-palvelin tehtiin toteutuksen viimeisenä osana. Palvelimen avulla pystyttiin osoittamaan AKA-digest-autentikaatiossa saatavien avainarvojen käytettävyys CoAP-asiakkaan ja HTTP-palvelimen välillä. Käytettävyys osoitettiin tekemällä HTTP-protokollaa käyttävät JWE- ja JWS -palvelimet, joiden autentikointitapa on AKA-Digest. Palvelimet ottavat vastaan käyttämänsä esitysmallin, lukevat esitysmallista B-TID-arvon, pyytävät sitä vastaavan KS-NAF-arvon operaattorilta ja tulkitsevat sen avulla esitysmallin sisällön. Sisältönä palvelimet odottavat saavansa tekstiä, minkä ne kääntävät ympäri ja palauttavat takaisin asiakkaalle KS-NAF-arvolla allekirjoitettuna tai salattuna. Asiakas pystyy tämän jälkeen havainnoimaan kaiken onnistuneen käyttämällään avaimella.

Palvelin käyttää HTTP-pyyntöjen vastaanottamiseen Oraclen HttpServer-kirjastoa [75]. POST-pyynnön saamisen jälkeen palvelin avaa sen sisältönä olevan JWT-paketin header-osan attribuuttina olevan B-TID-arvon työssä toteutetulla JSON-luokalla. Saatuaan B-TID-arvon palvelin tekee alla näkyvän Curl-ohjelmalla toteutetun mukaisen POST-pyynnön Ericssonin BSF-testipalvelimelle:

```
curl -i -X POST -H "X-EricssonLabs-APIKEY: snakeoilapikey" -H
"Content-Type: text/xml" -d "<?xml version=\"1.0\"
encoding=\"UTF-8\"?><requestBootstrappingInfoRequest
xmlns=\"http://www.3gpp.org/GBASErvice\"><B-
TID>VUQTvtcXP8ftP6xYpp7PYg==@p133.piuha.net</B-TID>
<nafid>cDEzMy5waXVoYS5uZXQAAAAAAA==</nafid><gsid>0</gsid><gbaUAw
are>True</gbaUAware>
</requestBootstrappingInfoRequest>"
http://p133.piuha.net:8080/bsf/requestBootstrappingInfo
```

Pyyntöön sijoitetaan asiakkaan lähettämä B-TID-arvo ja NAF palvelin NAFID. Palvelin saa alla näkyvän vastauksen lähettämäänsä pyyntöön:

```
<?xml version="1.0" encoding="UTF-8"?>
<requestBootstrappingInfoResponse
xmlns="http://www.3gpp.org/GBASErvice"><impi>tut.test1@p133.piuh
a.net</impi><meKeyMaterial>6Dd5MShgV63ZasFGqg62KsqsCCfzPmjqmkr4+
VXNlPQ=</meKeyMaterial><uiccKeyMaterial>rgoHMpx6SXhsBejqTg0HW28c
/ETcJSmnRIqUqLH8+2g=</uiccKeyMaterial><keyExpiryTime>2014-04-
29T09:27:34.440Z</keyExpiryTime><bootstrappingInfoCreationTime>2
014-04-
29T08:27:34.440Z</bootstrappingInfoCreationTime><ussList><![CDAT
A[<?xml version="1.0" encoding="UTF-8"?>
<ussList />
]]></ussList></requestBootstrappingInfoResponse>
```

Viestin mukana tulee XML-muodossa AKA-Digest-autentikaation 3G-tavan molemmat avaimet ja avainten voimassaoloaika. Avaimista meKeyMaterial vastaa NAF-palvelimen asiakkaan lähettämää avainta. Avaimen saatuaan NAF purkaa JSON-luokan avulla saamansa viestin ja luo uuden JWE- tai JWS -viestin käyttämällä samaa avainta.

Tehty NAF on demonstraatio todellisesta NAF-palvelimesta, koska NAF-palvelimen ja BSF:n välillä tulisi olla suojattu yhteys, kuten TLS [41]. TLS-yhteyden avulla operaattori pystyy siirtämään avainmateriaalin turvallisesti palvelimelle. Tällä hetkellä tieto liikkuu salaamattomana ymmärrettävässä muodossa, minkä seurauksena väärennetty NAF-palvelin voi pystyä esiintymään oikeana palvelimena. Tämä tosin syntyy tilanteessa, missä asiakas ei varmenna palvelinta.

5.5 HTTP-autentikaation välittäminen

HTTP-autentikaation välittäminen CoAP-protokollassa vaatii CoAP-protokollan puuttuvasta tuesta johtuen ylimäärästä tietoa mitä ollaan välittämässä ja miten se tulkitaan. Selkeänä ratkaisuna on käyttää sovellustason mekanismeja. Toteutuksessa päädyttiin ratkaisuun, jossa autentikaatio siirretään osana hyötykuormaa. Toisenlaisena vaihtoehtona olisi käyttää CoAP-protokollan otsikon optio-kenttää. Sovellustason esitysmuodoksi aikaisemmin valittiin JSON, joten pyrkimyksenä oli löytää siihen sopivat kirjastot, tietotyypit ja tavat tehdä sopiva konversio header-kentästä JSON-notaatioon.

Konversion toteuttaminen ja samalla testaaminen jakaantui kolmeen kokonaisuuteen. Kokonaisuuksista ensimmäinen oli RFC-2617 [4] mukaisen Digest-autentikoinnin siirtäminen JSON-rakenteen päälle. Ensimmäisen vaiheen jälkeen oli toteutuksen testaaminen HTTP-Digest palvelimeen ja mahdolliset muutokset. Testipalvelimen ohjelmistoksi valikoitui NGINX-palvelinohjelmistoon [76] Digest toteutuksen mahdollistavan HttpAuthDigestModule [77] käyttäminen. NGINX-ohjelmiston valintaan johti Apachen vastaava toimimattomuus halutunlaisesti. Kolmantena vaiheena oli AKA-Digest toteutuksen testaus, sekä mahdolliset muutokset yhdyskäytävään. Kolmannen testausosion apuna oli Ericssonin AKA-Digest-palvelin.[77][4]

Aliluvuissa esitetyt esimerkit osoittavat saadut Digest-haasteet ja tehdyt vasteet niihin ja mahdollisesti syntyneet ongelmat. Edeltävistä syistä johtuen esimerkit eivät siten ole samoista haaste-vaste-yhteydenotoista.

5.5.1 Autentikaation siirto sovellustasolle

Autentikaation siirto yksinkertaistettuna jakaantuu kahteen osaan. Ensimmäisenä tulee sopivan JSON-kirjaston löytäminen Javaan, millä sopivan JSON-rakenteen luominen ja lukeminen onnistuvat helposti. Toisena on autentikaatiolle sopivan JSON-rakenteen suunnittelu. Rakenteeseen on onnistuttava sijoittamaan ja taas lukemaan helposti saatavat Digest-attribuutit.

Javaan on olemassa useita erillisiä kirjastoja JSON:in käyttämiseen. Oletuksena kirjastot tulkitsevat JSON-muodon Java-kieleen alla olevan taulukon 2 mukaisesti:

String	Java.lang.String
Number	Java.lang.Number
True|False	Java.lang.Boolean
Null	Null
Array	Java.util.List
Object	Java.util.Map

Taulukko 2: JSON-tietotyypit ja niitä vastaavat Java-kielen tietotyypit.[78]

Erilaisten JSON-kirjastojen suurimmat toteutukselliset erot tulivat kokeilujen perusteella taulukkojen, kuten Arrayn ja Objectin rakentamisesta. Erot näkyvät moniulotteisuudessa ja mitä tiedostotyyppejä hyväksytään edeltävien sisään tai sallivatko kirjastot Object-tietotyypin sisään esim. String- ja Number -tietotyyppejä. JSON-tietotyyppejä voi kirjaston asennuksen jälkeen käyttää myös suoraan Javassa, jolloin ei olisi tarvetta käyttää Javan sisäänrakennettuja tyyppejä.

Tietotyyppien muuntamiseen käyttöön valittiin Jackson JSON-kirjasto [79]. Valintaan johti Jacksonin tuki useampiulotteisille MAP-tietorakenteille, eli moniulotteisesta MAP-rakenteesta tulee moniulotteinen JSON-rakenne ja toisinpäin. Useampiulotteisten MAP-rakenteiden tuki oli olennaista, koska toteutuksen sisäiseen käyttöön oli valittu MAP-tietorakenne. Yhtenä vaihtoehtona olisi ollut käyttää sisäisessä toteutuksessa suoraan JSON-Objectia. JSON-Objectin sijaan koettiin kuitenkin tietojen välittämiseen luokkien välillä helpoimmaksi ratkaisuksi yhden MAP-rakenteen toteuttamiseen. Näin toimien MAP voi sisältää useita erilaisia rakenteita, joista yksi voi esim. olla Digest, mikä taas sisältää sen attribuutit omissa rakenteissaan. [79]

Useampiulotteisten tietorakenteiden avulla pystyttiin JSON-rakenteessa mahdollistamaan vaihtoehtoisesti esim. Basic- tai Digest -algoritmien käyttö. Ylimmällä tasolla voidaan esim. määritellä käytössä oleva algoritmi, jolloin algoritmista riippuvan toteutukseen siirtymiseen ei tarvitse lukea koko tietorakennetta.

HTTP-header kentän authenticate- ja authorization -attribuutit päädyttiin sijoittamaan alla olevien mallien mukaisiin tietorakenteisiin. Rakenteessa ylimpänä kerrotaan onko malli www-authenticate vai authorization. Seuraavalla tasolla määritetään autentikaation tyyppi, eli tässä tapauksessa Digest. Lopulta alimmalla tasolla annetaan tyypin attribuutit. Tällöin www-authenticate haasteen attribuutit näyttävät JSON-rakenteessa alla olevalta:

```
{"www-authenticate" : {
        "type": "digest",
        "challenge": {
            "realm": "di@tyo.com",
            "qop": "auth, auth-int",
            "nonce": "dcd98b7102dd2f0e8b11d0f600bfb0c093",
            "opaque": "5ccc069c403ebaf9f0171e9517f40e41"
        }
    }
}
```

Autentikaatiota toiseen suuntaan muutettaessa yhdyskäytävä tulkitsee alla olevan authorization mallin Digest attribuuteiksi CoAP-protokollan yhteydenotossa:

```
{"authorization": {
        "type": "digest",
        "digest-response" :
        {
            "username": "jarkkovirtanen",
            "realm" : " di@tyo.com ",
            "nonce": "dcd98b7102dd2f0e8b11d0f600bfb0c093",
            "uri": "/dir/index.html",
            "qop": "auth",
            "nc:00000001,
            "cnonce":"0a4f113b",
            "response":"6629fae49393a05397450978507c4ef1",
            "opaque":"5ccc069c403ebaf9f0171e9517f40e41"
        }
    }
}
```

Siirrettäessä JSON-rakenteesta Digest-attribuuteiksi pitää yhdyskäytävässä huomioida attribuuttien järjestys. Järjestyksen pitäisi mennä RFC-2617:ssä [4] määritellyn järjestyksen mukaisesti, jonka jälkeen siirto tapahtuu onnistuneesti.

5.5.2 Autentikoituminen Digest-palvelimelle.

Autentikoinnin toteuttaminen lähti liikkeelle Digest-palvelimen pystyttämisestä testausta varten, jotta myöhemmin AKA-Digestin testaus olisi helpompaa. NGINX-palvelimelle tehtiin testisivu, jonka tunnuksena oli ”testi” ja avaimena ”a”. Suojattu testisivu sisälsi ”Secret” tekstin, minkä sai näkyviin autentikoinnin onnistuessa. Tämän jälkeen lähdettiin testaamaan tehdyllä asiakkaalla yhdyskäytävän läpi yhteydenottoa palvelimelle.

Palvelin palauttaa Digest-haasteessa seuraavat parametrit Wireshark-kaappauksen perusteella:

```
WWW-Authenticate: Digest algorithm="MD5", qop="auth",
realm="Tunnistus", nonce="16f67f0252fde619"
```

Haaste eroaa RFC-2617:ssä [4] määritetystä attribuuttien järjestyksestä, jonka pitäisi olla realm, nonce, algorithm ja qop. Järjestyksellä ei kuitenkaan ole asiakkaan kohdalla merkitystä, koska se pyrkii vain poimimaan merkkijonosta oikean attribuutin ja sen arvon. RFC-2617:n mukaisen vastauksen palvelin kuitenkin hylkää. Hylkäyksen seurauksena yhteydenottoa palvelimelle lähdettiin testaamaan Firefox-selaimen avulla. Firefox palauttaa Wireshark-kaappausten perusteella vasteen alla olevassa järjestyksessä, minkä palvelin hyväksyy:

```
Authorization: Digest username="testi", realm="Tunnistus",
nonce="39d2afe052fdd97d", uri="/priv/index.html", algorithm=MD5,
response="860bf16b79b6ac817b82fbb93fc42f62", qop=auth,
nc=00000001, cnonce="1150561b6d9e8864"
```

Edeltävä eroaa RFC-2617:n [4] mukaisesta, minkä pitäisi olla username, realm, nonce, uri, response, algorithm, cnonce, qop ja nc. Firefoxin mukaisen attribuuttijärjestyksen toteutuksen jälkeen autentikaatio meni kuitenkin onnistuneesti läpi.[4]

5.5.3 Autentikoituminen AKA-Digest-palvelimelle.

AKA-autentikoinnin ensimmäinen vaihe menee RFC-3310:n mukaan identtisesti HTTP-Digest-autentikaation verrattuna lukuunottamatta attribuuttieroja ja tapaa laskea avain. Edellisen perusteella lähdettiin tekemään samanlaista yhteydenottoa Ericssonin BSF-palvelimeen ”http://p133.piuha.net:8080/bsf/bootstrap”. Vastaukseksi ensimmäiseen yhteydenotto pyyntöön saatiin kuitenkin:

```
”HTTP Status 400 - Malformed Authorization header:
Authentication header is missing”
```

Ongelmaa selvitellessä löytyi Miikka Poikselän ja Georg Mayerin kirjasta ”The Ims IP Multimedia Concepts And Services” [80] kuvaus SIP-AKA-autentikaatiosta. Kirjan mukaan yhteydenotto pitää alkaa tyhjällä Authorization-pyynnöllä:

```
Authorization: Digest username="tobias private@home1.fr",
realm="home1.fr",
nonce="",
uri="sip:home1.fr",
response=""
```

Muutoksen toteuttaminen vaati yhdyskäytävään uuden luokan toteuttamista. Uusi luokka lähtee tekemään yhteydenottoa Digest-Authorization-kentällä ja palauttaa saadun www-authenticate-Digest-pyynnön. Muokkauksen jälkeen BSF alkoi vastata alla olevan header-kentän mukaisella haasteella:

```
HTTP/1.1 401 Unauthorized
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=6628CF479E39962D566C848C4D1300C9;
Path=/bsf/; HttpOnly
WWW-Authenticate: Digest
realm="ims.ericsson.com",nonce="GESAkdFAM3ZAciqHPldYighweiRPQAAA
LGKx+mL50V8=",algorithm="AKAv1-MD5",qop="auth-int"
Content-Type: text/html;charset=utf-8
Content-Length: 954
Date: Fri, 11 Apr 2014 09:41:01 GMT
```

Haasteessa ihmetystä aiheutti ensi näkemältä auth-int vaatimus, mikä osoittautui vaatimukseksi sisällyttää md5Hex-arvo tyhjästä merkkijonosta Response-arvon laskentaan. Vastaukseksi vasteeseen tuli:

```
HTTP Status 500 - Could not find session data
```

Vastaus tulkittiin palvelimen haluksi saada jonkinlaisen keksin takaisin. Haaste sisälsi kyllä keksin mutta sitä ei Digest-haasteessa pitäisi tarvita. Vaatimus todennäköisesti ei johdu itse BSF:n toteutuksesta, vaan kirjastosta, jonka päälle BSF on toteutettu. Keksin käytön vaatimuksesta seurasi kuitenkin ongelma keksin siirtämisestä CoAP-liikenteeseen, koska CoAP ei suoraan tue keksejä, kuten ei autentikaatiotakaan. Ongelma ratkaistiin luomalla uusi JSON attribuutti ”Cookie”, joka sisällytettiin Authorization JSON-rakenteen rinnalle:

```
Cookie={JSESSIONID=6628CF479E39962D566C848C4D1300C9}
```

Tällöin kokonaisuudessaan JSON paketti näyttää seuraavalta:

```
{"www-authenticate":
      {"challenge":
            {"qop":"auth-int",
            "nonce":"GZaArG4jqT3m8gZi259cUL7v1E++HQAAv5cVhISjAIQ=
            ",
            "realm":"ims.ericsson.com",
            "algorithm":"AKAv1-MD5"
            },
      "type":"digest"
      },
"Set-Cookie":
{"JSESSIONID":"6628CF479E39962D566C848C4D1300C9"}
}
```

Muutosten jälkeen keksi pystyttiin kierrättämään asiakkaankautta takaisin BSF-palvelimelle. Kuitenkin vieläkin tuli alla oleva virheilmoitus, joten jostain syystä vaste ei vieläkään riittänyt:

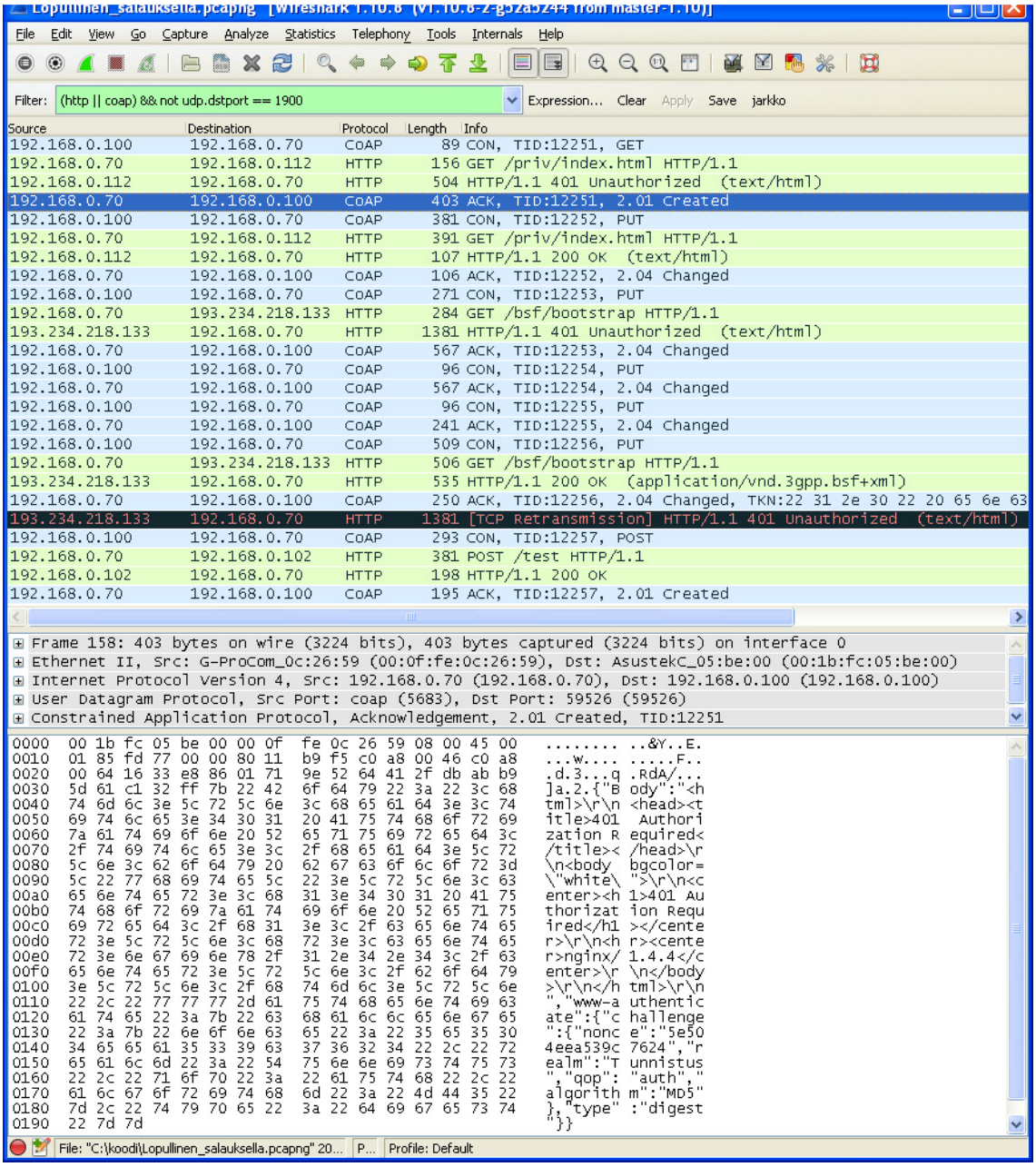
```
HTTP/1.1 403 Forbidden
```

Ongelma vaikutti lähes mahdottomalta, kunnes löytyi NVS(NGM Virtual Stickies) [13] projekti. Projektissa toteutettiin RESTful-arkkitehtuuria hyväksikäyttäen HTTP:sta yhdyskäytävä SIP-protokollaa käyttävälle Android-alustalle. Projektin toteutuksesta löytyi valmiit luokat SIP-AKA-Digest-autentikaation toteuttamiseen. Projektin perusteella vasteeseen tehty response-arvo pitää sijoittaa vasteen sisään muista merkkijonomuodossa olevista arvoista poiketen tavutaulukkomuodossa. Edeltävä oli kyllä dokumenteissa mainittu, mutta muista poikkeavana asiana jäänyt huomioimatta. Toteutetun muutoksen jäljiltä autentikaatio meni onnistuneesti läpi ja BSF palauttaa ”200 OK” viestin, minkä header-osassa on autentikaatiotiedot ja vastauksen rungossa B-TID-arvo sijoitettuna XML-rakenteeseen. XML-rakenne ei työn tavoitteiden kannalta ollut toivottavaa, koska siitä seuraa resurssirajoitteisille laitteille vaatimus parsia sekä JSON-, että XML -rakenteita. Yhtenä vaihtoehtona olisi ollut siirtää B-TID-arvo JSON-rakenteeseen yhdyskäytävässä, jolloin asiakkaalle kaikki autentikointidata olisi näkynyt loogisesti JSON-rakenteessa. Edeltävän toteutus olisi kuitenkin tarkoittanut yhdyskäytävän osalta HTTP-body-osioiden lukemista ja tätä ei nähty kannattavana. Ongelman yhdyskäytävän osalta olisi myös aiheuttanut auth-int-arvon varmentaminen. Demoympäristössä B-TID-arvon parsiminen asiakkaan osalta oli kuitenkin yksinkertaista XML-parsereilla, joista työssä käytettiin Sun Microsystemsin Xerces-projektin DOMParseria [81]. Parsinnan jälkeen AKA-autentikaation ensimmäinen vaihe, eli autentikointi BSF:lle on tehty.[13]

Toisessa vaiheessa tehtäväksi tuli rakentaa KS-NAF-arvo. KS-NAF-arvolla vaihtoehtoisesti allekirjoitetaan tai salataan JWT-paketti, jonka otsikko sisältää saadun BTID-arvon ja dataosiot asiakkaan sisällyttämää sisältöä. Edeltävän paketin asiakas lähettää NAF-palvelimelle, jonka odotetaan palauttavan sisältö käännettynä takaisin. Johtuen NAF-palvelimen käyttämästä samasta JSON-luokasta, jota asiakaskin käytti, sisältö tuli onnistuneesti takaisin käännettynä.

5.6 Sovelluksen tietoliikenteen tarkastelu

Liikenne kulkee kuvan 8 esimerkkikaappauksen mukaisella tavalla yhdyskäytävän läpi. Kuvassa 8 CoAP-asiakas toimii IP-osoitteessa 192.168.0.100, yhdyskäytävän IP-osoite on 192.168.0.70, Digest-palvelimen IP on 192.168.0.112, BSF vastaa osoitteesta 193.234.218.133 ja NAF osoitteesta 192.168.0.102.



Kuva 8 Yhdyskäytävän kuljettama liikenne, sisältää HTTP-Digest, AKA-Digest ja JWE-tiedonsiirrot.

Ensimmäisenä asiakas ottaa yhteyden Digest-palvelimeen. Palvelimelta asiakas saa haasteen autentikoitua. Haasteen yhdyskäytävältä asiakkaalle lähtevä liikenne on avattu kuvan alimmassa laatikossa. Asiakkaan vasteen jälkeen palvelin hyväksyy yhteydenoton ”200 OK”-viestillä. Tämän jälkeen asiakas aloittaa AKA-neuvottelun BSF-palvelimen kanssa lähettämällä sille tyhjän vasteen. Vasteeseen BSF lähettää haasteen ja asiakas tekee AKA-Digest laskennan ja toimittaa vastauksen BSF-palvelimelle. BSF vastaa samoin kuten Digest-palvelin ”200 OK” viestillä sillä poikkeuksella, että nyt rungon mukana tulee XML-muodossa BTID-arvo. Nyt asiakas pystyy tekemään POST-pyynnön NAF-palvelimelle. Palvelin vastaa onnistumisella ”200 OK” viestillä ja yhdyskäytävä ohjaa vastauksen takaisin asiakkaalle.

6 JATKOKEHITYS JA POHDINTA

Työ jakaantui viiteen kokeelliseen osakokonaisuuteen, CoAP-asiakkaaseen, yhdyskäytävään, NAF-palvelimeen, autentikointiin ja autentikoinnin hyödyntämiseen. Kokeellisuudesta johtuen työ toteutettiin Javalla ja kehitys tapahtui ketteriä ohjelmistokehitysmalleja mukaillen samalla kun työ pyrittiin pitämään diplomityön rajoissa. Edellisistä syistä johtuen kaikkiin osiin jäi asioita, joita ei toteutettu tai ne olisi työn myöhemmissä vaiheissa analysoituna ollut järkevämpi toteuttaa toisella tavalla. Näistä asioista kerrotaan seuraavissa aliluvuissa.

Valituksi ohjelmointikieleksi tuli Java, koska sille on olemassa paljon valmiita toteutuksia, joita voidaan käyttää työn apuna. Java-ohjelmat pyörivät kuitenkin aina virtuaalikoneen päällä, mikä aiheuttaa raskautta. Haettaessa optimaalisia ratkaisuja resurssirajoitteisille laitteille Java ei ole paras vaihtoehto, joten ainakin asiakas pitäisi kirjoittaa uudelleen tehokkaammalla ohjelmointikielellä.

Asiakkaan toteutuksessa lähtökohtana oli aiheuttaa tietynlaista liikennettä ja havaita saapuuko vastaus odotetusti. Kehitystapa johti kasvavaan listaan asioita, joissa lähetetään pyyntö ja odotetaan sille vastaus. Vastauksen tultua edelliseen pyyntöön asiakas lähettää uudenlaisen pyynnön. Listaperustaisesta kehityksestä seurasi kaksi isompaa ongelmaa, joita olisi hyvä pyrkiä kehittämään jatkossa. Ensimmäisenä on käyttöliittymä, joka on tekstipohjainen ja suunniteltu listan vaiheiden ajamiseen. Asiakkaan käyttäminen muuhun kuin testaustarkoitukseen vaatisi käyttöliittymän uudelleen suunnittelua. Toinen asia on tilattomuus, asiakkaalle työssä riitti muistaa edellinen vaihe. Säännöllisessä käytössä mm. Digest-autentikaatioon perustuvat asiat vaatisivat kuitenkin autentikointi-parametrien muistamista ja siten tilan ylläpitämistä. Muutosten jälkeen asiakasta voisi käyttää moneen muuhunkin käyttötarkoitukseen.

Yhdyskäytävä on testauksissa hallinnut laaja-alaisesti tiedon välittämistä. Seuraava kehitysaskel valitussa toteutustavassa olisi pyrkiä rakentamaan MIME-tyypeille ja HTTP-paluukoodeille yleisluonteinen tapa välittää ne yhdyskäytävän läpi. Tämän hetkiset tavat eivät kovin hyvin sovellu yleiseen liikenteeseen. Työn lähtökohtana oli HTTP-protokollan CoAP-protokollalle soveltumattomien osien siirtäminen JSON-rakenteen päälle rungossa välitettäväksi. Jatkokehityksen osalta voisi miettiä onko tällainen tapa oikea vai pitäisikö miettiä enemmän CoAP-header-rakenteen käyttöä. Header-osiota pystytään tehokkaammin käyttämään mm. Pass Through -ratkaisulla. Pass Through -ratkaisussa määritellään CoAP-optio nimeltä ”HTTP-header” ja sen jälkeen kaikki HTTP-header-sisältö sijoitetaan siihen. Tapana tämä on yksinkertainen ja selkeä siten, ettei header-sisältö päädy runkoon, kuitenkin samalla HTTP:n selkokielisen syntaksin sijoittaminen CoAP-protokollaan hävittää CoAP-protokollan hyötyjä suhteessa HTTP-protokollaan. Yksi jatkoselvityksen aihe on valita paras tapa muun muassa keksien ja autentikaation välittämiseen.

JSON-rakenne on tällä hetkellä syvä, mikä ihmissilmälle on selkeä, mutta sen käsittely on matalaa raskaampi. Seuraavaksi rakennetta voisi lähteä muuttamaan litteämmäksi, jolloin HTTP:n header-attribuuteista tehtäisiin esim. aina yksi JSON-lohko alla olevan esimerkin mukaisesti:

```
{"header":
    {
        "inout":"www-authenticate",
        "qop":"auth-int",
        "nonce":"GZaArG4jqT3m8gZi259cUL7v1E++HQAAv5cVhISjAIQ=",
        "realm":"ims.ericsson.com",
        "algorithm":"AKAv1-MD5",
        "type":"digest"
    },
    {
        "JSESSIONID":"6628CF479E39962D566C848C4D1300C9"
    }
}
```

NAF-palvelin vaikutti pitkään asialta, mitä ei tulla toteuttamaan johtuen TLS-vaatimuksesta, jonka toteutus oli rajattu työn ulkopuolelle. Kuitenkin, kun havaittiin, ettei BSF-testipalvelin vaadi TLS-yhteyttä, NAF-palvelimen toteuttamisesta tuli yksinkertaista. Jatkokehityksenä yhteyden suojaus NAF- ja BSF -palvelimien välillä tarvitaan toteutettuna esim. TLS-yhteyden tai WS-Securityn avulla. Palvelin ei tällä hetkellä myöskään pidä tilaa siihen yhteyden ottaneista asiakkaista tai BSF-palvelimelle tehdyistä B-TID-pyynnöistä. Tilan toteuttamisella mahdollistettaisiin BSF:n kuormittamisen vähentäminen ja riippuvuuden poistaminen siitä esim. yhteyskatkosten aikana.

Autentikointi toteutettiin asiakkaan ja yhdyskäytävän läpi käyttäen Digest- ja AKAv1 -autentikointiratkaisuja. Seuraava looginen eteneminen olisi autentikaatiotapojen laajentaminen esim. AKAv2-toteutukselle. AKA-Digest-toteutuksen osalta BSF palauttaa B-TID-arvon XML-dokumenttina rungossa, mikä JSON-toteutuksen osalta ei ollut toivottavaa. Nyt UE joutuu tekemään sekä JSON- että XML -tulkkausta. Asiaan työn osalta ei voi vaikuttaa, mutta optimaalisena ratkaisuna B-TID-arvo tulisi ”200 OK” viestin mukana header-kentässä. Yksi jatkokehityksessä tutkittava tapa olisi tulkata XML-rakenne jo yhdyskäytävässä JSON-rakenteeksi, jolloin UE:n ei tarvitse tulkata XML-rakennetta. Tulkkaus jo yhdyskäytävässä vaatisi header-rakenteen avaamisen lisäksi body-osion lukemista. Muutos vain header-rakenteesta kiinnostuneesta yhdyskäytävästä kaiken lukevaan voisi kuitenkin johtaa moneen uuteen ongelmaan. B-TID-arvon luku yhdyskäytävässä estäisi auth-int-arvon mahdollistaman viestin varmentamisen asiakkaan osalta.

Autentikoinnin hyödyntämistä työssä käytettiin UE:n ja NAF-palvelimen keskinäisen liikenteen suojaamiseen. Työssä KS-NAF-arvoa käytetään suoraan avaimena JWT-paketeissa. Ratkaisuun päädyttiin palvelimen tilattomuudesta johtuen, kuitenkin turvallisempi ratkaisu olisi KS-NAF-avaimen avulla luoda asiakkaalle ja palvelimelle yhteinen avain. Näin operaattori ei pääsisi avaamaan kuljetettavia viestejä, sekä voitaisiin yhden autentikoinnin jälkeen poistaa riippuvuus operaattorista. Yhteisen avaimen luomisesta tulee ongelma JWT-rakenteen suhteen, koska tällä hetkellä JWT:ssä ei ole tukea symmetriseen avaintenvaihtoon. Symmetrinen avaintenvaihto vaatisi myös tilan ylläpitämistä molemmilta osapuolilta. Asymmetrisenä avaintenvaihto voisi toimia tilattomasti toisen osapuolen toimesta. Toisena jatkokehityksen aiheena on B-TID-arvon välittäminen. Tällä hetkellä se kulkee JWT-otsikossa, mutta se voitaisiin määrittää JWA-algoritmissa esitettäväksi tietyllä tavalla, jolloin siitä tulisi yleiskäyttöisempi.

7 YHTEENVETO

Työ syntyi tarpeesta mahdollistaa Web of Things -käsitteen kuvaamassa ympäristössä toimivan laitteen autentikointi ja avaintenvaihto, sekä sen jälkeinen tiedonsiirto turvallisesti sovellustasolla. WoT-käsitteellä tarkoitetaan ympäristöä, jossa laitteet ovat ympärillämme ja välittävät hypertekstimäistä sisältöä keskenään. Ympärillämme olevat laitteet harvemmin ovat resurssitehokkaita tai nopean tiedonsiirtoyhteyden päässä, jolloin tarvitaan resurssioptimoituja ratkaisuja. WoT-ympäristöjen yleistymisen edellytyksenä on riittävä yhteensopivuus hypertekstin siirtämiseen tarkoitetulle HTTP-protokollalle, jolloin on saatavilla riittävästi valmista sisältöä.

Selkokielisen HTTP-protokollan kehityksen yhteydessä protokollan hyvien ominaisuuksien säilyttämiseksi kehitettiin RESTful-arkkitehtuuri. Arkkitehtuuria voidaan käyttää pohjana uusille protokollille haettaessa HTTP:ta optimaalisempaa ratkaisua johonkin tarpeeseen. CoAP-protokolla on RESTful-arkkitehtuurista kehitetty protokolla resurssirajoitteisiin ympäristöihin, jolloin se soveltuu hyvin WoT-ympäristöihin. CoAP-protokollan kehittämisessä yhtenä pyrkimyksenä on ollut maksimoida hyötykuorman osuus siirrosta, johon on päästy käyttämällä binääristä otsikkorakennetta.

Yhteinen arkkitehtuuri HTTP:n ja CoAP:in välillä antaa mahdollisuuden jonkintasoiseen tiedonsiirtoon yhdyskäytävän avulla. Tarkastelun kohteena olevat protokollat tukevat MIME-tyypin käyttöä kuvaamaan liikuttamaansa sisältöä, jolloin MIME:llä mahdollistetaan protokollien ulkopuolisten mekanismien käyttö. Ulkopuolisilla mekanismeilla pystytään siten kompensoimaan protokollien eroja siirtämällä tietoja mekanismien kuljetettavaksi ja kertomalla MIME:llä miten mekanismeja tulkitaan.

Tietoturvasta puhuttaessa huomioidaan käsitteet saatavuus, eheys ja luottamuksellisuus. Edellisten mahdollistamiseksi tulevat vielä käsitteet autentikointi ja kiistämättömyys. Tietoturvan käsitteiden saavuttamiseen HTTP-protokollassa käytetään monesti TLS-tunnelointia ja CoAP-protokollassa TLS:stä datagram-protokollille kehitettyä DTLS-tunnelointia. Edellä mainitut tunnelointiratkaisut toimivat kuitenkin sovellustason alla, joten ne eivät soveltuneet työssä käytettäväksi.

Sovellustasolla tietoturvaan liittyvien käsitteiden huomioimiseen on erilaisia tapoja, joista osa perustuu protokollien omiin toteutuksiin ja osa päällä toimiviin mekanismeihin. Autentikoinnin osalta toimivia ovat HTTP:n heikon autentikoinnin Basic- ja Digest -tekniikat, joille ei kuitenkaan ole tukea CoAP-protokollassa. CoAP-protokollassa autentikointi attribuutit täytyy siirtää yhdyskäytävän avulla päällä käytettäviin mekanismeihin, kuten JSON-rakenteeseen. Rakenteessa tämän jälkeen kerrotaan sisällön sisältävän autentikoinnin mahdollistavat attribuutit. Salauksen ja eheyden toteutukseen ja valvontaan sovellustasolla tarvitaan päällä toimivia mekanismeja, kuten JWT:tä, joka perustuu JSON-notaatioon. JWT-rakenteen JWE-määritys vastaa salauksesta ja JWS-määritys allekirjoituksesta.

3GPP on kehittänyt vahvan autentikoinnin GAA-arkkitehtuurin, jossa autentikoidutaan operaattorin tietojen ja PIN-koodin avulla. Ratkaisussa perustuu GBA-arkkitehtuurin symmetriseen avaintenvaihtoon. Asiakas autentikoituu ensin operaattorille käyttäen HTTP-Digest-autentikaatiosta kehitettyä AKA-Digest-autentikaatiota. Operaattorilta asiakas saa B-TID-tunnisteen, jonka asiakas välittää haluamalleen palvelimelle. Palvelin saa tunnistetta vastaan operaattorilta saman avaimen, jonka asiakas on tietojensa pohjalta laskenut. Näin molemmilla osapuolilla on yhteinen avain käytettävissään.

Teorian jälkeen työssä toteutettiin Java-kielellä SIM-korttia emuloiva CoAP-asiakas, NAF-palvelin, sekä yhdyskäytävä CoAP- ja HTTP -protokollien välille. Yhdyskäytävä mahdollistaa tiedonsiirron protokollien välillä ja tekee siirron Digest-attribuuttien ja JSON-notaation välillä. Asiakas ja NAF käyttävät kirjautumiseen AKA-Digest-autentikaatiota ja välittävät liikenteensä toistensa välillä JWT-esitysmallia käyttäen vaihtoehtoisesti salaten tai allekirjoittaen. Toteutuksen aikana havainnoitiin eteen tulevia ongelmia ja selviteltiin toimivia ratkaisuja ongelmien korjaamiseksi.

Toimivan demonstraatiototeutuksen pohjalta jatkoa ajatellen ongelmat ovat CoAP- ja HTTP -protokollien tilakoodien yhdistämisessä ja erilaisten HTTP-header rakenteiden välittämisessä. Valittu JSON-rakenne mahdollistaa header-attribuuttien siirtämisen yksinkertaisesti eteenpäin, vaikkei se toteutuksena välttämättä ollut paras mahdollinen. Käytetyt tekniikat ovat vielä nuoria, mistä seurasi ohjelmistokirjastojen keskeneräisyyttä, kuitenkin nopean kehityksen ansiosta ongelmat häviävät nopeasti. AKA-autentikaatio havaittiin toimivaksi autentikointiratkaisuksi. Toteutusaikainen ongelma oli 3GPP-spesifikaatiot, jotka oli selvästi suunniteltu suurille organisaatioille, eikä pienille projekteille. Dokumenteissa oli jatkuvasti viittauksia toisiin dokumentteihin, mikä teki lukemisesta ja ongelmien selvittämisestä hankalaa.

Havaittujen ongelmien ja kehitysehdotusten jälkeen voi todeta työn täyttäneen tavoitteet ja vaatimukset. WoT-ympäristöjen autentikointi ja avaintenvaihto onnistuu resurssirajoitteisilta laitteilta hyvin sovellustason puitteissa.

LÄHTEET

[1] Z. Shelby, K. Hartke, and C. Bormann, “Constrained Application Protocol (CoAP),” *IETF Internet-draft*. [Online]. Saatavilla: http://tools.ietf.org/html/draft-ietf-core-coap-03. [Luettu: 07-Sep-2013].

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” *IETF RFC-2616*. Saatavilla: http://tools.ietf.org/html/rfc2616.

[3] T. Berners-Lee, R. Fielding, H. Frystyk, “Hypertext Transfer Protocol -- HTTP/1.0,” *IETF RFC-1945*. Saatavilla: http://www.isi.edu/in-notes/rfc1945.txt.

[4] P. J. Leach, J. Franks, A. Luotonen, P. M. Hallam-Baker, S. D. Lawrence, J. L. Hostetler, and L. C. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” *IETF RFC-2617*. Saatavilla: http://tools.ietf.org/html/rfc2617.

[5] 3GPPP, “TS 33.220; Generic Authentication Architecture (GAA); Generic bootstrap-ping architecture(GBA),” 2013. [Online]. Saatavilla: http://www.3gpp.org/ftp/Specs/html-info/33220.htm.

[6] 3GPP, “TR 33.919 V11.0.0 (2012-09) 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Generic Authentication Architecture (GAA); System description (Release 11).” [Online]. Saatavilla: http://www.qtc.jp/3GPP/Specs/33919-b00.pdf.

[7] M. Sethi, J. Arkko, A. Keränen, “End-To-End Security for Sleepy Smart Object Networks.” [Online]. Saatavilla: http://www.scribd.com/doc/171229489/End-To-End-Security-for-Sleepy-Smart-Object-Networks. [Luettu: 29-Apr-2014].

[8] B. Schneier, *Applied cryptography*, 2th ed. Wiley, 1997, p. 784.

[9] K. Ashton, “That ‘Internet of Things’ Thing - RFID Journal.”[Online]. Saatavilla: http://www.rfidjournal.com/articles/view?4986. [Luettu: 08-May-2014].

[10] D. Guinard, V. Trifa, “Towards the Web of Things: Web Mashups for Embedded Devices.” [Online]. Saatavilla: http://www.vs.inf.ethz.ch/publ/papers/dguinard_09_WOTMashups.pdf. [Luettu: 28-Apr-2014].

[11] W3C, “SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).” . Saatavilla: http://www.w3.org/TR/soap12-part1/.

[12] R. T. Fielding and R. N. Taylor, “Principled design of the modern Web architecture,” *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002.

[13] H. Al-Shammari, P. Crane, J. Munster, W. Wong, *NVS NGN Virtual Stickies*. Information Science Department University of Otago called ODMIE, 2009, pp. 1–30.

[14] Margaret Rouse, “machine-to-machine (M2M).” [Online]. Saatavilla: http://whatis.techtarget.com/definition/machine-to-machine-M2M.

[15] T. Berners-Lee, R. Fielding, “Uniform Resource Identifiers (URI): Generic Syntax,” *IETF RFC-2396*. Saatavilla: http://www.ietf.org/rfc/rfc2396.txt.

[16] M. Laine, “RESTful Web Services for the Internet of Things.” [Online]. Saatavilla: http://media.tkk.fi/webservices/personnel/markku_laine/restful_web_services_for_the_internet_of_things.pdf.

[17] W3C, “Extensible Markup Language (XML) 1.0 (Fifth Edition).” Saatavilla: http://www.w3.org/TR/REC-xml/.

[18] D. Crockford, “The application/json Media Type for JavaScript Object Notation (JSON),” *IETF RFC-4627*. Saatavilla: http://www.ietf.org/rfc/rfc4627.txt.

[19] N. Borenstein and N. Freed, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,” *IETF RFC-2045*. Saatavilla: http://tools.ietf.org/html/rfc2045.

[20] IANA, “Service Name and Transport Protocol Port Number Registry.” [Online]. Saatavilla: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. [Luettu: 08-May-2014].

[21] IANA, “Message Headers.” [Online]. Saatavilla: http://www.iana.org/assignments/message-headers/message-headers.xhtml. [Luettu: 09-May-2014].

[22] J. Postel, “User Datagram Protocol,” *IETF RFC-768*, 1980. Saatavilla: http://tools.ietf.org/html/rfc768.

[23] Z. Shelby, “Constrained RESTful Environments (CoRE) Link Format,” *IETF RFC-6690*. Saatavilla: http://tools.ietf.org/html/rfc6690.

[24] E. Rescorla and N. Modadugu, “Datagram Transport Layer Security Version 1.2,” *IETF RFC-6347*. Saatavilla: http://tools.ietf.org/html/rfc6347.

[25] N. Borenstein and N. Freed, “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types,” *IETF RFC-2046*. Saatavilla: http://tools.ietf.org/html/rfc2046.

[26] K. Moore, “MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text,” *IETF RFC-2047*. Saatavilla: http://tools.ietf.org/html/rfc2047.

[27] J. Klensin, T. Hansen, and N. Freed, “Media Type Specifications and Registration Procedures,” *IETF RFC-6838*. Saatavilla: http://tools.ietf.org/html/rfc6838.

[28] N. Freed and J. C. Klensin, “Media Type Specifications and Registration Procedures,” *IETF RFC-4288*. Saatavilla: http://tools.ietf.org/html/rfc4288.

[29] W3C, “Overview of SGML Resources.” [Online]. Saatavilla: http://www.w3.org/MarkUp/SGML/Overview.html. [Luettu: 19-Jul-2014].

[30] W3C, “XHTML 1.0: The Extensible HyperText Markup Language (Second Edition).” Saatavilla: http://www.w3.org/TR/xhtml1/.

[31] M. Murata, Dan Kohn, “XML Media Types,” *IETF RFC-3023*. Saatavilla: http://tools.ietf.org/html/rfc3023.

[32] W3C, “Efficient XML Interchange (EXI) Format 1.0 (Second Edition).” Saatavilla: http://www.w3.org/TR/exi/.

[33] M. Michael Jones, “JSON Web Key (JWK),” *IETF Internet-draft*, 2014. [Online]. Saatavilla: http://tools.ietf.org/html/draft-ietf-jose-json-web-key-25. [Luettu: 28-Apr-2014].

[34] M. Michael Jones, “JSON Web Algorithms (JWA),” *IETF Internet-draft*. [Online]. Saatavilla: https://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms-16. [Luettu: 28-Apr-2014].

[35] J. Bradley, N. Sakimura, and M. Jones, “JSON Web Signature (JWS),” *IETF Internet-draft*. [Online]. Saatavilla: https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-06. [Luettu: 28-Apr-2014].

[36] J. Hildebrand and M. Jones, “JSON Web Encryption (JWE),” *IETF Internet-draft*. [Online]. Saatavilla: https://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-00. [Luettu: 28-Apr-2014].

[37] M. Jones, P. Tarjan, Y. Goland, N. Sakimura, J. Bradley, J. Panzer, and D. Balfanz, “JSON Web Token (JWT),” *IETF Internet-draft*. [Online]. Saatavilla: https://tools.ietf.org/html/draft-jones-json-web-token-10. [Luettu: 28-Apr-2014].

[38] Valtiovarainministeriö, “Valtion tietohallinnon internet-tietoturvallisuusohje 1/2003,” 2003. Saatavilla: http://www.vm.fi/vm/fi/04_julkaisut_ja_asiakirjat/01_julkaisut/05_valtionhallinnon_tietoturvallisuus/39680/39681_fi.pdf.

[39] Bruce Schneier, *Secrets & Lies*. Wiley, 2000, p. 432.

[40] Information Sciences Institute University of Southern California, “Transmission Control Protocol Darpa Internet Program Protocol Specification,” *IETF RFC-793*. Saatavilla: http://www.ietf.org/rfc/rfc793.txt.

[41] T. Dierks, E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” *IETF RFC-5246*. Saatavilla: http://tools.ietf.org/html/rfc5246.

[42] M. Linden, “Identiteetin- ja pääsynhallinta,” *Tampereen teknillinen yliopisto luentomoniste*, 2012. Saatavilla: http://www.cs.tut.fi/~linden/iam-pruju.pdf.

[43] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design Principles and Practical Applications*. John Wiley & Sons, 2011, p. 384.

[44] AES, “Standards.” [Online]. Saatavilla: http://www.aes.org/standards/. [Luettu: 19-Jul-2014].

[45] E. Rescorla, “Diffie-Hellman Key Agreement Method,” *IETF RFC-2631*. Saatavilla: http://www.ietf.org/rfc/rfc2631.txt.

[46] RSA Laboratories, “PKCS #1 v2.2: RSA Cryptography Standard,” 2012. [Online]. Saatavilla: http://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf.

[47] I. Network Sorcery, “Protocols.” [Online]. Saatavilla: http://www.networksorcery.com/enp/Protocol.htm. [Luettu: 29-Apr-2014].

[48] R. Braden, “Requirements for Internet Hosts -- Application and Support,” *IETF RFC-1123*. Saatavilla: https://www.ietf.org/rfc/rfc1123.txt.

[49] R. Braden, “Requirements for Internet Hosts -- Communication Layers,” *IETF RFC-1122*. Saatavilla: http://asg.web.cmu.edu/rfc/rfc1122.html.

[50] Sean Wilkins, “OSI and TCP/IP Model Layers | OSI and TCP/IP Model Layers | Pearson IT Certification.” [Online]. Saatavilla: http://www.pearsonitcertification.com/articles/article.aspx?p=1804869. [Luettu: 29-Apr-2014].

[51] J. Jarmoc, “SSL / TLS Interception Proxies and Transitive Trust,” 2012. [Online]. Saatavilla: http://www.secureworks.com/cyber-threat-intelligence/threats/transitive-trust/.

[52] R. Oppliger, *SSL and TLS: Theory and Practice*. Artech House, 2009, p. 257.

[53] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, L. Stewart, “An Extension to HTTP : Digest Access Authentication,” *IETF RFC-2069*. Saatavilla: http://www.ietf.org/rfc/rfc2069.txt.

[54] S. Cantor, J. Kemp, R. Philpott, J. Hughes, H. Lockhart, M. Beach, R. Metz, R. Randall, T. Wisniewski, P. Austel, R. L. B. Morgan, P. C. Davis, J. Hodges, F. Hirsch, P. Madsen, and A. Anderson, “Assertions and Protocols for the OASIS Security Assertion Markup Language,” 2005. Saatavilla: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[55] K. Lawrence, C. Kaler, A. Nadalin, R. Monzillo, and P. Hallam-baker, “Web Services Security : SOAP Message Security 1 . 1,” 2006. Saatavilla: http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[56] A. Niemi, J. Arkko, V. Torvinen, “Hypertext Transfer Protocol (HTTP) Digest Authentication Using Authentication and Key Agreement (AKA),” *IETF RFC-3310*. Saatavilla: http://www.ietf.org/rfc/rfc3310.txt.

[57] ETSI, “TS 133 102 - V7.1.0 - Universal Mobile Telecommunications System (UMTS); 3G security; Security architecture (3GPP TS 33.102 version 7.1.0 Release 7),” 2006. Saatavilla: http://www.etsi.org/deliver/etsi_ts/133100_133199/133102/07.01.00_60/ts_133102v070100p.pdf.

[58] 3GPP, “TS 35.205 V11.0.0 (2012-09), 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 1: General,” 2012. [Online]. Saatavilla: http://www.qtc.jp/3GPP/Specs/35205-b00.pdf.

[59] 3GPP, “TS 35.206 V11.0.0 (2012-09), 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification.” [Online]. Saatavilla: http://www.3gpp.org/DynaReport/35206.htm.

[60] ETSI, “TS 133 221, Generic Authentication Architecture (GAA) – Support for subscriber certificates,” 2014. [Online]. Saatavilla: http://www.etsi.org/deliver/etsi_ts/133200_133299/133221/06.03.00_60/ts_133221v060300p.pdf.

[61] T. Olkkonen, “Generic Bootstrapping Architecture.” [Online]. Saatavilla: http://www.tml.tkk.fi/Publications/C/22/papers/Olkkonen_final.pdf.

[62] Q. Zhang, *Pre-Shared Key Transport Layer Security Protocol with Generic Bootstrapping Architecture Support*. Helsinki University of Technology, Department of Electronics, communication and Automation, 2008, p. 60.

[63] Ericsson, “Ericsson Labs | Documentation.” [Online]. Saatavilla: http://labs.ericsson.com/apis/mobile-web-security-bootstrap.

[64] 3GPP, “TS 36.104 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception (Release 8),” 2007. Saatavilla: http://www.qtc.jp/3GPP/Specs/36104-800.pdf.

[65] Ncoap-core, “ncoap-core.” [Online]. Saatavilla: http://media.itm.uni-luebeck.de/people/kleine/maven/ncoap/1.8.0-SNAPSHOT/project-info.html. [Luettu: 05-May-2014].

[66] The University of Rostock, “jCoAP is a Java Library implementing the Constrained Application Protocol (CoAP) - Google Project Hosting.” [Online]. Saatavilla: https://code.google.com/p/jcoap/. [Luettu: 05-May-2014].

[67] M. Kovatsch, M. Lanter, Z. Shelby, “Californium (Cf) CoAP framework - Java CoAP Implementation.” [Online]. Saatavilla: http://people.inf.ethz.ch/mkovatsc/californium.php. [Luettu: 05-May-2014].

[68] Oracle, “java.net (Java Platform SE 7).” [Online]. Saatavilla: http://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html. [Luettu: 05-May-2014].

[69] The Apache Software Foundation, “HttpClient - HttpClient Home.” [Online]. Saatavilla: http://hc.apache.org/httpclient-3.x/. [Luettu: 05-May-2014].

[70] The Apache Software Foundation, “Apache HttpComponents - Apache HttpComponents.” [Online]. Saatavilla: http://hc.apache.org/index.html. [Luettu: 05-May-2014].

[71] Google, “jsoncrypto - Implement the JSON WebToken Encryption and Signing Standards - Google Project Hosting.” [Online]. Saatavilla: https://code.google.com/p/jsoncrypto/. [Luettu: 24-Jun-2014].

[72] Connect2id, “Nimbus JOSE + JWT.” [Online]. Saatavilla: http://connect2id.com/products/nimbus-jose-jwt. [Luettu: 24-Jun-2014].

[73] Ict-adamantium, “FP7 ICT Project ADAMANTIUM.” [Online]. Saatavilla: http://www.ict-adamantium.eu/. [Luettu: 26-Jun-2014].

[74] Ict-adamantium, “Milenage.” [Online]. Saatavilla: http://www.ict-adamantium.eu/SIP-Agent/javadoc/net/java/sip/communicator/sip/security/Milenage.html. [Luettu: 26-Jun-2014].

[75] Oracle, “HttpServer (Java HTTP Server).” [Online]. Saatavilla: http://docs.oracle.com/javase/7/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/HttpServer.html. [Luettu: 24-Jun-2014].

[76] Nginx, “nginx news.” [Online]. Saatavilla: http://nginx.org/. [Luettu: 03-Aug-2014].

[77] Nginx, “HttpAuthDigestModule - Nginx Community.” [Online]. Saatavilla: http://wiki.nginx.org/HttpAuthDigestModule. [Luettu: 05-May-2014].

[78] Tutorialspoint, “JSON with Java.” [Online]. Saatavilla: http://www.tutorialspoint.com/json/json_java_example.htm. [Luettu: 08-May-2014].

[79] FasterXML, “JacksonHome - FasterXML Wiki.” [Online]. Saatavilla: http://wiki.fasterxml.com/JacksonHome. [Luettu: 05-May-2014].

[80] M. Poikselkä and G. Mayer, *The IMS: IP Multimedia Concepts and Services*. John Wiley & Sons, 2009, p. 560.

[81] The Apache Software Foundation, “com.sun.org.apache.xerces.internal.parsers: public class: DOMParser.” [Online]. Saatavilla: http://www.docjar.com/docs/api/com/sun/org/apache/xerces/internal/parsers/DOMParser.html. [Luettu: 12-Jul-2014].