OULUN AMMATTIKORKEAKOULU

Saku Rautio

TIETOLIIKENTEEN SALAUSMENETELMÄT SEKÄ SULAUTETTUJEN OHJELMISTOJEN KEHITYSTYÖKALUJEN EVALUOINTI

TIETOLIIKENTEEN SALAUSMENETELMÄT SEKÄ SULAUTETTUJEN OHJELMISTOJEN KEHITYSTYÖKALUJEN EVALUOINTI

Saku Rautio
Opinnäytetyö
Kevät 2018
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä: Saku Rautio
Opinnäytetyön nimi: Tietoliikenteen salausmenetelmät sekä sulautettujen ohjelmistojen kehitystyökalujen evaluointi
Työn ohjaaja: Veijo Väisänen, Petri Soininen
Työn valmistumislukukausi ja -vuosi: Kevät 2018
Sivumäärä: 84

Tämä opinnäytetyö on suoritettu kahdessa osassa. Ensimmäinen osuus on kymmenen opintopisteen laajuinen kokonaisuus. Toinen osuus vastaa viiden opintopisteen laajuista kokonaisuutta ja se toteutettiin yrityksen toimeksiantona.

Opinnäytetyön ensimmäinen osuus käsittelee tietoliikenteessä käytettäviä salausmenetelmiä. Ensimmäisessä osassa tutkitaan käytettyjä nykyaikaisia salausmenetelmiä ja lopuksi implementoidaan yksi niistä Arduino-alustalle.

Toinen osa tehtiin toimeksiantona RD Velho Oy:lle. Yhtiö on suunnittelutoimisto, joka tarjoaa tuotekehitys- ja suunnittelupalveluita teknologiateollisuuden alalla toimiville yrityksille. Tavoitteena oli evaluoida kolmea työkalua sulautettujen ohjelmistojen ohjelmistokehitystä varten. Lopputuloksena oli odottaa yhden uuden työkalun esilletuloa ja odottaa käsiteltävien työkalujen kypsymistä.

Asiasanat: salausmenetelmät, sulautettu ohjelmisto, työkalujen evaluointi, koosteopinnäytetyö

SISÄLLYS

1 JOHDANTO

Oulun ammattikorkeakoulussa opinnäytetyö on mahdollista toteuttaa joko yhtenä laajempana kokonaisuutena tai viiden opintopisteen kokonaisuuksissa. Jälkimmäinen opinnäytetyö on niin sanottu koosteopintotyö. Tämän koosteopinnäytetyön ensimmäinen osa suoritettiin vuoden 2016 keväällä. Toinen osa valmistui vuoden 2018 keväällä. Ensimmäinen osa on tutkielma nykyajan tietoliikenteen salausmenetelmistä. Toinen osa on kolmen eri työkalun evaluointi sulautettujen ohjelmistojen ohjelmistokehitystä varten ja se suoritettiin yrityksen toimeksiantona.

Koosteopinnäytetyön ensimmäisessä osassa käsitellään kolmea salausmenetelmää, joista yksi implementoidaan Arduino-alustalle. Tämä ensimmäinen osa on kymmenen opintopisteen suuruinen.

Koosteopinnäytetyön toinen osa oli RD Velho Oy -yrityksen toimeksianto. Siinä tutkitaan kolmea eri työkalupakettia ja evaluoidaan, mikä niistä olisi paras RD Velhon käyttöön. Työkalupaketit esitellään aluksi ja sitten niistä tehdään vertailu, josta vedetään johtopäätökset jatkoa varten. Tämä toinen osa on viiden opintopisteen suuruinen.

2 OPINNÄYTETYÖN ENSIMMÄISEN OSAN ESITTELY

Ensimmäisessä osassa käsitellään pääasiallisesti kolmea eri tietoliikenteen salausmenetelmää: DESiä, AESia ja RSAta. Vaikka DES onkin jo vanhentunutta tekniikkaa, on se hyvä olla mukana antamassa pohjan AESille. Lopussa vielä tehdään implementaatio Arduino-alustalle AESista, joka salaa annetun merkkijonon ja purkaa salauksen antaen annetun merkkijonon takaisin. Näin todistetaan implementoidun salauksen toimivuus.

Työssä perehdytään enemmän salauksien matemaattisiin periaatteisiin ja toimintatapoihin eikä niinkään käyttötarkoituksiin tai niiden jatkokehitykseen.

3 OPINNÄYTETYÖN TOISEN OSAN ESITTELY

Toisen osan aiheena on tehdä evaluointi RD Velholle sulautettujen ohjelmistojen kehitystyökaluista kahdesta kaupallisesta työkalupaketista ja tarkastella vapaasti levitettävistä ohjelmistoista vaihtoehdot kaupallisille työkalupaketeille. Tarkasteltavista työkaluista etsitään tarvittavat ja halutut toiminnallisuudet, jotka tarkastellaan ja esitellään. Lopussa tehdään työkalujen välillä vertailu parhaimmasta vaihtoehdosta RD Velholle.

Työssä käsiteltiin arkaluonteisia tietoa, jota ei haluttu julkaistavaksi. Nämä arkaluonteiset tiedot poistettiin julkaistavasta versiosta.

4 YHTEENVETO

Koosteopinnäytetyö koostui kahdesta suorituksesta, jotka yhdistettynä vastaavat 15 opintopisteen suoritusta. Osaopinnäytetöiden aiheet eivät liittyneet toisiinsa, sillä ensimmäisen osan aihe tuli omasta mielenkiinnosta ja halusta tehdä juuri kyseisestä aiheesta opinnäytetyö. Ensimmäisen opinnäytetyön aihe oli itseäni opettava ja hyödyllinen tulevaisuutta varten. Toisen opinnäytetyön aihe vaihtui kahdesti ennen päätymistä RD Velhon toimeksiantoon. Toimeksi annettu aihe ei sinällään ollut mielenkiintoinen, mutta se opetti tekemään evaluointeja ja toimimaan yrityksessä enemmän itsenäisesti.

LIITTEET

Liite 1 Tietoliikenteen salausmenetelmät

Liite 2 Sulautettujen ohjelmistojen kehitystyökalujen evaluointi

LIITE 1

Saku Rautio

TIETOLIIKENTEEN SALAUSMENETELMÄT

TIETOLIIKENTEEN SALAUSMENETELMÄT

Saku Rautio
Opinnäytetyö, osa 1
Kevät 2016
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

SISÄLLYS

SANASTO

Bitti	Tulee englanninkielen sanasta bit (binary digit). Luku, joka on tietotekniikassa informaation määre ja jonka kantaluku on 2.
Bittisiirto	Biteistä koostuvan jonon arvojen siirto järjestyksessä uudelle paikalle.
Brute force	Salauksenpurkumenetelmä, jossa avainta ei yritetä löytää vaan kokeillaan purkaa salaus isolla määrällä avaimia
For-loop	Tapahtumaketju, jossa annetaan paremetreina muuttuja, sen ehtolauseke, joka kertoo kuinka monta kertaa tapahtumaketjua tehdään, sekä komento, mitä jokaisella kierroksella tehdään. Esimerkiksi $\mathrm{for(int\,i = 0;\ i < 5;\ i++)}$ -silmukassa alustetaan kokonaislukumuuttuja *i* ja sen arvoksi asetetaan 0, silmukkaa tehdään niin kauan kuin *i* on vähemmän kuin 5 ja jokaisella kierroksella *i*:n arvoa kasvatetaan yhdellä.
Globaalimuuttuja	Funktioiden ja metodien ulkopuolella määritelty muuttuja, jota voidaan käsitellä kaikissa ohjelman sisäisissä funktioissa.
Heksadesimaali	Tulee englanninkielen sanasta *hexadecimal*. Luku, joka on myös informaation määre, mutta sen kantaluku on 16.
IBM	International Business Machines, yhdysvaltalainen tietotekniikan yritys.

Modulo	Jakojäännöksen määrittäminen tavalla $a + b \equiv c\ (mod\ n)$, jossa *a* ja *b* ovat vapaavalintaisia elementtejä ja *c* on jakojäännös, kun jaetaan elementillä *n*. Elementtiä *n* kutsutaan nimellä *modulus*. Esimerkkinä *a* on yksi, *b* on kaksi ja *n* on kaksi: $1 + 2 \equiv c\ (mod\ 2)$, saadaan *c*:n arvoksi luku 1, sillä $1 + 2 = 3$ ja $3 \div 2 = 1\ jää\ 1$.
NIST	National Institute of Standards and Technology. Yhdysvaltain kauppaministeriön (United States Department of Commerce) alainen virasto, joka kehittää standardeja ja tekniikoita.
NSA	National Security Agency, Yhdysvaltain tiedustelulaitos.
Tavu	Kahdeksan bitin kokoinen jono perättäisiä, yksittäisiä, bittiä.
While-loop	Tapahtumaketju, jota tehdään niin kauan, kunnes annettu ehto on epätosi.
XOR	Eksklusiivinen OR-operaatio, ns. *joko-tai*.

1 JOHDANTO

Tämän päivän tietoliikenne on vilkasta ja monipuolista. Yhä enemmän ja enemmän ihmiset käyttävät internettiä ja tietokoneita niin koti- kuin työelämässäkin. Internet of Things, eli asioiden internet, tuo lisää laitteita ja lisää tietoliikennettä turvattavaksi, samalla tuoden lisää laskentatehoa maailmaan. Alati kasvavan tietoliikenteen määrä tuo ongelmia valtion hallinnoille ja yrityksille, kuten myös kuluttajille. Kuinka olla varma oman tietoliikenteen turvallisuudesta uhraamatta sen eheyttä?

Jo muinaisaikaan on omaa viestintää haluttu salata toiselta osapuolelta, kuten vihollisarmeijalta sodassa. Näin tehtiin käyttämällä tulta ja väri aineita tai tulen savun sekvensoimisella. Roomalaiset käyttivät Caesarin salausta piillottaakseen omat suunnitelmansa, viikingit samantapaista aakkosten korvaamista avaimen kirjainten perusteella, ranskalaiset 1800-luvulla Claude Chappen keksimää optista lennätintä, saksalaiset Enigma-salauslaitetta ja nyt tietoliikennettä salataan useilla menetelmillä.

Opinnäytetyön tavoitteena on kartoittaa yleisimpien tietoliikenteen salausmenetelmien toimintaa ja niiden tehokkuuksia. Samalla käydään läpi niiden historiaa ja katsellaan tulevaisuuden näkökulmasta, ovatko käytettävät salausmenetelmät käyttökelpoisia. Idea opinnäytetyöhön tuli omasta kiinnostuksesta aiheeseen ja halusin oppia tarkemmin tietoliikenteen salaamisesta, mitä se pitää sisällään ja miten se tehdään. Myöhemmin lisättynä tavoitteena on rakentaa toimiva toteutus jostakin työssä käsitellyistä menetelmistä Arduino Uno -alustalle.

2 DATA ENCRYPTION STANDARD -SALAUS

Data Encryption Standard:in (DES) juuret kantavat 1960-luvun loppupuolelle, jolloin IBM:n tutkija nimeltä Horst Feistel kehitti omaa salainta, Feistelin salainta, joka perustuu toistuviin kierroksiin, f-funktioon sekä kaksipuoliseen XOR-vaihtoon. Vähän sen jälkeen, kun Feistel esitteli tuotostaan, alkoi IBM kehittämään salausohjelmaa pankeille. Feistelin esityksestä muokattiin yhdelle piirille toimiva 64 bitin avaimen sisältävä versio 128 bitin alkuperäisversiosta. (1, s. 87.)

IBM pani ns. kaiken likoon salaimeensa ja päätti ottaa yhteyden NSA:han eli yhdysvaltain tiedustelulaitokseen nimeltä National Security Agency. IBM halusi tietotaitoa ja kehitysapua salaimeensa. NSA vaati vastineeksi valtuudet tehdä muutoksia salaukseen ja IBM:n oli pidettävä muutoksista tiedot pois julkisuudesta. Avainpituus muutettiin 56 bittiin luultavasti siksi, että NSA:kin saisi salauksen purettua. (1, s. 88.)

Salaus valmistui vuonna 1975 ja siitä tuli myös Yhdysvaltojen virallinen salaustandardi vuonna 1977 (1, s. 88).

Salausta on käytetty sen julkaisuaikanaan pankkien tietojärjestelmissä, johon se oli suunnattu, USA:n valtion sensitiivisen tiedon salaukseen. Salausta on käytetty myös myöhemmin sirukorteissa, SIM-korteissa ja verkkolaitteissa, jotka vaativat tietoliikenteen salaamista, kuten modeemit, reitittimet ja verkkoon yhdistyvät digiboxit. (2.)

2.1 DES:n salaus

Salattava selväteksti paloitetaan 64 bitin lohkoihin, jolle tehdään permutaatio (Initial permutation(IP)) eli lohkon bitit järjestetään eri tavalla, hieman kuten Caesarin salauksessa. Permutointi tapahtuu ottamalla lohkon bitti ja siirtämällä se IP-taulukon mukaisesti uuteen osoitteeseen. (Kuva 1.)

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

KUVA 1. Initial permutation -taulukko (3, s. 10)

Permutoinnin jälkeen lohko jaetaan kahtia 32 bitin kokoisiin lohkoihin nimeltä L0 ja R0 eli Left-0 ja Right-0. Näitä lohkoja käydään salauksessa 16 kertaa permutoimalla ja käyttämällä f-funktiota vuorottaisiin lohkoihin permutoidulla avaimella. (3, s. 9).

Sama operaatio tehdään 56-bittiselle avainlohkolle. Se alussa jaetaan kahteen 28-bittiseen lohkoon, L ja R, joille jokaisella kierroksella tehdään bittisiirto vasemmalle yhdellä tai kahdella bitillä kierroksen numeron mukaan ja ylivuotavat bitit liitetään takaisin perään. Yhden bitin verran siirretään vain kierroksilla 1, 2, 9 ja 16, muulloin siirretään kahdella. Näin saadaan molemmat lohkot kiertämään kerran ympäri. (1, s. 90.)

Permutointi määräytyy selvätekstin permutoinnin tapaisella taulukolla. Ensimmäinen permutointi tapahtuu *permuted choice 1* -taulukolla (kuvat 2 ja 3), jossa jaetut lohkot permutoidaan erikseen ja seuraavat permutoinnit tapahtuvat *permuted choice 2* -taulukolla (kuva 4), jossa siirretyt avaimet yhdistetään ja 48 bittiä permutoidaan taulukon mukaisesti. Permutoitua lohkoa merkitään K_n , jossa *n* on kierroksen numero. (3, s. 11.)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36

KUVA 2. Permuted choice 1 left -taulukko (3, s. 19)

63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

KUVA 3. Permuted choice 1 right -taulukko (3, s. 19)

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

KUVA 4. Permuted choice 2 -taulukko (3, s. 21)

Jokaisella selvätekstin salauskierroksella käydään läpi f-funktio, jonka parametreina ovat kierroksen R_n ja K_n . Alussa laajennetaan R_n-lohko 48-bittiseksi laajennuspermutaatiolla kuvan 5 taulukon mukaisesti, jossa puolet biteistä kopioidaan uusiin osoitteisiin. (1, s. 90.)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

KUVA 5. Laajennuspermutaatiotaulukko (3, s. 13)

Laajennuksen jälkeen uusi R_n viedään XOR-operaation läpi K_n –lohkon kanssa. Lohko jaetaan 6 bitin palasiin, jolloin saadaan 8 bittiyhdistelmää 48 bitin lohkosta. Jokainen 6 bitin palanen viedään substitution-laatikon läpi, joita on kahdeksan, jokaiselle kombinaatiolle omanlaisensa. Laatikossa on bittikombinaatioarvoja 16 bittiin asti, koska laatikosta ulostuleva bittikombinaatio on 4-bittinen. (1, s. 90.)

NSA oli juurikin näihin laatikoihin puuttunut, kun se avusti IBM:ää salauksen teossa. NSA valitsi tarkoin substitution-laatikoiden sisällöt, jotta salaus olisi heille helpompi purkaa. Substitution-laatikoiden sisältö on siis tarkasti valittu, mutta niiden sisäinen järjestys on erilainen laatikoiden välillä. Substitution-laatikot haittaavat selvästi differentiaalisen kryptoanalyysin käyttämistä salauksen purkamiseen. (1, s. 87.)

Laatikkoon tulevan bittikombinaation ensimmäinen ja viimeinen bitti asettavat käytettävän rivin numeron, neljä keskimmäistä kertovat sarakkeen numeron. Jokaisella laatikolla on neljä riviä, jotka vastaavat 2^2-kombinaatioita, ja 16 saraketta, jotka vastaavat 2^{14}-kombinaatiota. (1, s. 90, 91.)

Esimerkkinä seuraava tilanne: bittiyhdistelmä 100011 ja laatikko S3. (Kuva 6.)

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

KUVA 6. S3-laatikko (3, s. 17)

Omasta bittikombinaatiosta saadaan riviksi 11_{bin} eli viimeinen ja sarakkeeksi 0001_{bin} eli toinen. Näin saadaan ulostulevaksi bittikombinaatioksi 10_{dec} eli 1010_{bin}.

Kokonaisuudessaan ulos tulee siis 32 bittiä, kun käytössä on kahdeksan laatikkoa ja neljä ulostulevaa bittikombinaatiota. Bitit yhdistetään ja ne vielä permutoidaan valitulla taulukolla (kuva 7).

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

KUVA 7. Permutointitaulukko S-laatikosta tulleelle lohkolle (3, s. 15)

F-funktiosta saatu arvo lopuksi viedään XOR-operaation läpi kierroksen senhetkisen L_n -arvon kanssa, josta tulee seuraavan kierroksen R_n eli R_{n+1}. Viimeisen kierroksen jälkeen arvot R_{16} ja L_{16} yhdistetään ja viedään vielä viimeisen permutaation läpi, joka on alkupermutaation käänteispermutaatio, käyttämällä omaa taulukkoa (kuva 8.) Näin saadaan yksi selvätekstilohko salattua. (3, s. 9-11.)

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

KUVA 8. Käänteispermutaatiotaulukko (3, s. 10)

2.2 DES:n purku

DES:n purku tapahtuu pääosin samalla tavalla kuin selvätekstin salaaminen. Salatun lohkon vastaanottaja tekee prosessin alusta loppuun käänteisillä avainlohkoilla, eli $K_1 \rightarrow K_{16}$, $K_2 \rightarrow K_{15}$ jne. DES:n symmetrisyyden ansiosta vastaanottaja purkaa salauksen helposti, pelkästään avainlohkojen järjestystä muuttamalla. (3, s. 12.)

2.3 DES:n eri muodot

DESiä voidaan ketjuttaa ja yhdistellä monella tapaa. Seuraavat ovat yleisimmät tavat, joita käytetään salauksen parantamiseksi. Suurin syy käyttää näitä muotoja on salauksen heikkous nykypäivänä. Jo vuonna 1998 todettiin, että DESiä ei voida käyttää nykyisessä muodossaan, kun Electronic Frontier Foundation rakensi tietokoneen, joka purki salauksen 56 tunnissa. (4, kappale 8.)

DESillä on myös neljä huonoa avainta, sillä ne antavat ulostulona selvätekstiä. Nämä avaimet antavat kierrosavaimiksi pelkkiä ykkösiä tai nollia. Nämä avaimet koostuvat heksadesimaaleina luvuista, jotka toistuvat kahdeksan kertaa peräkkäin avainkombinaatiossa: 01 01 01 01 01 01 01 01. Muut heksadesimaaliluvut ovat FE, 1F ja E0. (1, s. 92.)

3DES on DESin muoto, jossa DESiä ketjutetaan kolme kertaa peräkkäin käyttäen kolmea tai kahta eri avainta. Näin salaus on nyt kolme kertaa varmempi, sillä avainpituus kasvaa 168 bittiin, mutta kääntöpuolena on sen hitaus, joka on täten myös kolme kertaa hitaampi. Yksi tapa käyttää tätä menetelmää on tehdä salausta kolme kertaa peräkkäin jakamalla avain kolmeen osaan, joilla jokainen salauskerta salataan yhdellä avaimella kolmesta. Toinen tapa käyttää 3DES:iä on ajaa se ensin avaimella 1 salaussuuntaan, avaimella 2 purkusuuntaan ja lopuksi avaimella 3 salaussuuntaan. (1, s. 95.)

DES-X on toinen, mutta vähemmän raskas moodi. Siinä otetaan mukaan kaksi uutta 64 bitin avainta. Ensimmäiselle avaimelle tehdään XOR-operaatio selvätekstilohkon kanssa ennen salausta ja toinen avain ulostulevan salatun lohkon kanssa. Näin avainmahdollisuudet kasvavat ennestään uusien avainten ansiosta ja DESin sisäiseen toimintaan ei tarvitse puuttua. (1, s. 95.)

.

3 ADVANCED ENCRYPTION STANDARD -SALAUS

Vuonna 1997 NIST ilmoitti, että se halusi DESin tilalle uuden ja vahvemman salauksen, ja järjesti samanlaisen kilpailutuksen kuin DESin kehitysvaiheessa. Neljä vuotta myöhemmin se valitsi Rijndael-nimisen salauksen, jonka olivat kehittäneet Joan Daemen ja Vincent Rijmen Belgiasta. Uusi salaustandardi sai nimekseen Advanced Encryption Standard, lyhyesti AES. (1, s. 95, 96.)

AES-menetelmää käytetään nykyään salaamaan USA:n salaista tietoa SECRET-tasolle. TOP SECRET -taso vaatii joko 192- tai 256-bittisen avaimen. (5, s. 2.) Tietoliikenteessä sitä käytetään FTPS-, HTTPS-, SFTP-, AS2-, WebDAVS- ja OFTP-protokollissa (6). Muita käyttökohteita ovat esimerkiksi palomuurit, reitittimet ja tietoliikenneprotokollat kuten, SSL tai TLS (7).

3.1 AES:n rakenne

AES:n sisääntulevan selvätekstilohkon koko on 128 bittiä, joka on myös sama koko ulostulolle. Avaimen koko voi olla 128, 192 tai 256 bittiä. Eri avainko'oille on omat merkinnät: AES-128, AES-192 tai AES-256, jossa luku kertoo avaimen koon. Avaimen bitit indeksoidaan välillä *0* ... *n-1*, jossa *n* on avaimen koko. AES:n sisällä avain jaetaan numeroituihin tavuihin siten, että ensimmäiset kahdeksan bittiä muodostavat tavun numero 1, seuraavat kahdeksan bittiä tavun numero 2 ja niin edespäin. AES:n kierrosten määrän määrää avaimen koko. 128-bittisellä se on 10, 192-bittisellä se on 12 ja 256-bittisellä se on 14. (8, s. 5, 14.)

AES:n toimintaan kuuluu sen avaimen laajentaminen pienempiin osiin, jotta niitä voidaan käyttää salattavan lohkon sekoitukseen. Tämä voidaan tehdä taustalla tai rinnalla samaan aikaan, kun itse salausta ajetaan (1, s. 96). Koko salauksen toiminta perustuu myös äärelliseen kuntaan eli Galois'n kuntaan GF(p^m), jossa $p^m = 2^8$. Tälle kunnalle käytetään sen kertolaskussa moduluksena jaotonta polynomia (ts. alkulukupolynomi) $x^8 + x^4 + x^3 + x + 1$, joka vastaa

binääriyhdistelmää 100011011_{bin} ja heksadesimaalia $11B_{hex}$, jotta ulostuloarvo mahtuisi kyseiseen kuntaan. (8, s. 10-12.)

3.2 Avaimen prosessi

Alussa tehdään tavutaulukko nimeltä *palaset,* jonka koko, olkoon se *palakoko*, saadaan jakamalla avaimen koko 32:lla. Näin kooksi saadaan joko 4, 6 tai 8. *Palaset*-taulukon palasten arvot saadaan siten, että avainlohkon ensimmäiset neljä tavua muodostavat yhden palasen, seuraavat neljä toisen ja niin edelleen, kunnes koko avainlohko on saatu pilkottua *palaset*-taulukkoon. (8, s. 9, 10, 14, 19.)

Tämän jälkeen voidaan käyttää prosessin helpottamiseksi *kierros*-muuttujaa, jonka arvo on alussa *palakoko,* ja *palaset*-taulukkoon lisätään uusia arvoja, jotka saadaan seuraavanlaisesti rekursiivisesti käyttäen *pala-apu*-muuttujaa ja kierrosmuuttujaa *kierros*. Seuraavaa prosessia käydään läpi niin kauan, kunnes *kierros* on yhtä suuri kuin $4 * (kierrosten\ määrä + 1)$. (8, s. 19, 20.)

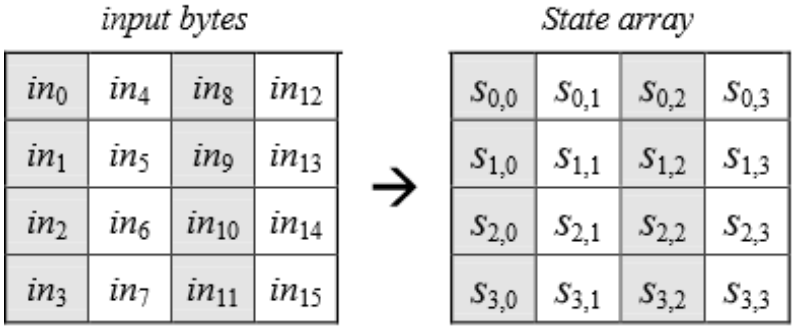
Otetaan *palaset*-taulukosta palanen *pala-apuun*, jonka indeksi on kierros − 1. Jos *kierros* ja sen jakojäännös *palakoosta* on nolla, kierroksen *pala-apu*-muuttujan tavut siirretään vasemmalle siten, että ylivuotava tavu siirtyy taakse. Saatu palanen viedään kuvan 10 *Substitution*-taulukon läpi ja siitä saadulla arvolla tehdään XOR-operaatio *vakiotaulukon kanssa,* joka sisältää neljä tavua myös. Sen sisältämä arvo haetaan indeksillä: $indeksi = kierros/palakoko$. *Vakiotaulukon* ensimmäinen arvo lasketaan seuraavasti: $SA^{(indeksi-1)}$, loput arvot ovat nollia (00_{hex}). SA on sekoitusmatriisin ensimmäinen arvo eli 02_{hex}. Jos kuitenkin *palakoko* on suurempi kuin kuusi (eli käytetään 256 bittistä salausavainta) ja $kierros\ mod\ palakoko = 4$, niin *pala-apu* lähetetään S-laatikon läpi. (8, s. 19-20.)

Kierroksen lopussa asetetaan sen *palaset*-taulukon, jonka indeksi on *kierros*, arvoksi palanen samasta *palaset*-taulukosta, jonka indeksi on *kierros* − *palakoko,* ja palasella tehdään XOR-operaatio *pala-apu* -muuttujan kanssa. Lopuksi lisätään *kierroksen* arvoa yhdellä. (8, s. 20.)

Palaset-taulukko muutetaan matriisiksi, jossa jokainen 4 tavua järjestyksessä muodostavaa yhden sarakkeen. Avainprosessia tehdään niin monta kertaa, että jokaiselle salauksen kierrokselle (10, 12 tai 14) on oma avainmatriisi. (8, s. 14, 20.)

3.3 Selvätekstin prosessi

Salausprosessin sisällä käytetään kaksiulotteista taulukkoa nimeltä *State*. Prosessin alussa selvätekstilohko jaetaan tavuihin ja laitetaan järjestykseen sarake kerrallaan kuvan 9 tavalla, jossa yhden elementin sisältö on kooltaan 8 bittiä (Kuva 9). Näin saadaan *State*-taulukon sisältö. *State*-taulukon sisällöllä tehdään vielä XOR-operaatio avaimen kanssa. (8, s. 14, 15.)



KUVA 9. Selvätekstin lajittelu State-taulukkoon (8, s. 9)

Tästä eteenpäin tehdään kierroksia, joissa ensimmäisenä DES:n tyylisesti katsotaan *State*-taulukon bitti ja sille vastaava arvo *substitution*-taulukosta (kuva 10).

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

KUVA 10. AES:n S-taulukko (8, s. 16)

Taulukon arvot on laskettu valmiiksi aiemmin mainitulla Galois'n kunnalla, sillä mahdollisuuksia on vain 2^8 eli 256. Tämä tapahtuu seuraavasti:

Lasketaan kunnan elementin *a* modulaariaritmetiikan käänteisluku *b*, jotta niiden kertolasku yhteen on 1, kun käytetään moduloa supistamaan arvo kunnan sisälle. Tämä voidaan esittää seuraavasti: $a(x) \cdot b(x) = 1 \; mod \; (x^8 + x^4 + x^3 + x + 1)$. Itse käänteisluku saadaan laskettua laajennetulla Eukleideen algoritmilla, tai esimerkiksi Fermat'n pienen lauseen tavalla $a^{2^8-2} \; mod \; (2^8)$ (9, s. 2). Koska arvolle 0 ei ole käänteislukua, on sen näennäiseksi käänteisluvun arvoksi asetettu 0. Tämän jälkeen saadulle luvulle tehdään affiini muunnos kaavalla 1. (8, s. 12, 13, 15, 16.)

$$b'_i = b_i \oplus b_{(i+4) mod \; 8} \oplus b_{(i+5) mod \; 8} \oplus b_{(i+6) mod \; 8} \oplus b_{(i+7) mod \; 8} \oplus c \qquad \textit{KAAVA 1}$$

i = indeksiarvo ja $0 \leq i \leq 8$
b_i = indeksin i bitti b
c = binääriyhdistelmä 01100011

Tämä voidaan esittää matriisioperaationa asian selventämiseksi (Kuva 11).

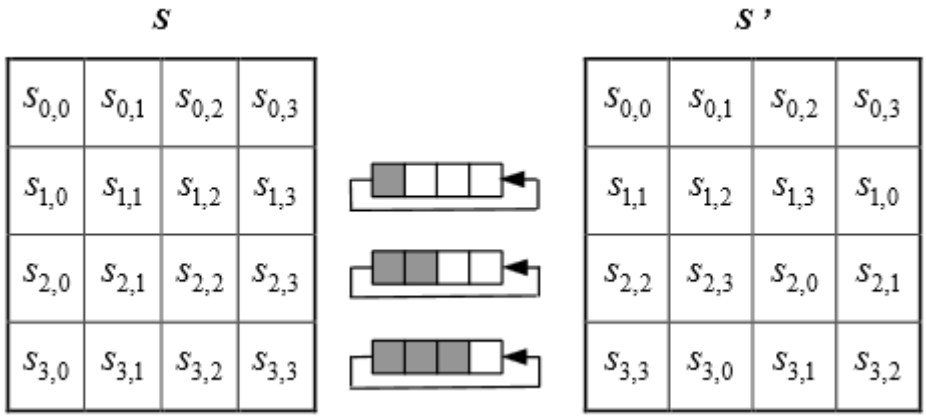
Kuitenkin riippuen järjestelmästä voi taulukon pitäminen muistissa olla huonompi vaihtoehto kuin laskea uuden bitin arvo ohjelmaa ajettaessa, sillä taulukko vie tilaa noin 2 kbit, koska $256\ mahdollisuutta * 8\ bittiä = 2048\ bittiä$.

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

KUVA 11. Matriisioperaatio S-taulukon laskemiseksi (8, s. 16)

Esimerkkinä bittikombinaatio 0101 1110 muutetaan heksadesimaalimuotoon, joka on 5E. Ensimmäinen luku kertoo taulukon rivin numeron ja toinen sarakkeen. Tässä tapauksessa saadaan ulostuloarvoksi 58_{hex} , joka vastaa lukua 01011000_{bin}.

Seuraavaksi siirretään *State*-taulukon rivit, paitsi ensimmäinen, kolme kertaa siten, että rivin numero kertoo, kuinka monta tavua siirretään vasemmalle. Ylivuotavat tavut siirretään alkuun. (Kuva 12.)



KUVA 12. Bittisiirto havainnollistettuna (8, s. 17)

Tämän jälkeen jokainen *State*-taulukon sarake sekoitetaan valmiiksi lasketulla matriisilla (kuva 13). Matriisin arvot saadaan polynomilla $3x^3 + x^2 + x + 2$. *State*-taulukon sarakkeen arvo saadaan kertomalla arvon rivi matriisista vastaavan rivin arvolla kaavalla 2. Kertolasku tehdään Galois'n kunnan kertolaskulla käyttämällä moduluksena polynomia $x^4 + 1$. (8, s. 17, 18.)

$$S_{r,c} = (S_{0,c} \cdot \alpha_{,0}) \oplus (S_{1,c} \cdot \alpha_{r,1}) \oplus (S_{2,c} \cdot \alpha_{r,2}) \oplus (S_{3,c} \cdot \alpha_{r,3}) \qquad \textit{KAAVA 2}$$

c = sekoitettavan *State*-sarakkeen numero
α = sekoitusmatriisi
r = rivinumero

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

KUVA 13. Sekoitusmatriisi (8, s. 18)

Viimeiseksi salausprosessin kierroksella tehdään avaimen laajennusprosessin ulostulon ja *State*-lohkon yhdistys XOR-operaatiolla, jossa avainmatriisin, eli *palaset*-taulukon, arvo katsotaan *State*-lohkon sarakkeesta ja rivistä tavalla $S_{r,c} \rightarrow A_{r,c}$, jossa *A* on avainmatriisi. (8, s. 15, 18.)

Tämä prosessi tehdään 9, 11 tai 13 kertaa, riippuen avaimen koosta, kunnes viimeisellä kerralla tehdään kerran *substitution*- ja siirtofunktiot. Koko prosessi päätetään viimeisellä avainlohkon ja *State*-lohkon yhdistämisellä, ja sen jälkeen syötetään ulos salattu teksti. (8, s. 15.)

3.4 AES:n purku

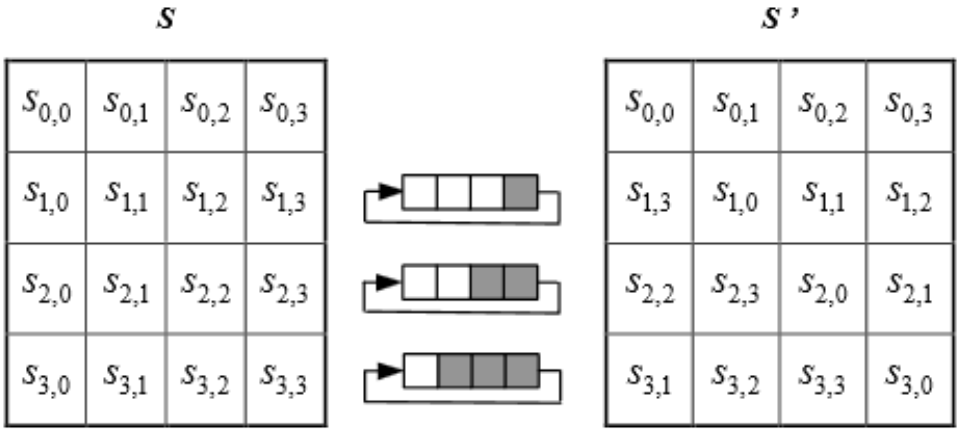
Koska AES ei ole samalla tavalla symmetrinen kuin DES, salaustekstin avausprosessissa tehdään monet tapahtumat hieman eri tavalla. Kuitenkin avaimen laajennus tehdään myös purkamisessa ja siinä se on täysin

samanlainen, sillä laajennuksen ulostulolle tehdään XOR-operaatio molemmissa tapauksissa. (8, s. 23.)

Alussa siis tehdään samanlainen avainprosessi kuin salauksessa ja laitetaan salattu teksti *state*-taulukkoon samalla tavalla. Sen jälkeen avainprosessin ulostulon ja *state*-taulukon sisällöllä tehdään XOR-operaatio, mutta avainmatriisina on viimeisen kierroksen matriisi. (8, s. 21.)

Tästä eteenpäin voidaan käydä kierroksia läpi esimerkkimuuttujalla *kierros*, joka saa alustavaksi arvoksi $kierrosten\ määrä - 1$, ja jokaisen kierroksen lopussa vähennetään arvoa yhdellä. Kierroksia tehdään niin monta kertaa, kunnes *kierros*-muuttujan arvo on 1. Näin viimeinen kierros on silmukan ulkopuolella. (8, s. 21.)

Ensimmäisenä kierroksella tehdään invertoitu bittisiirto kuvan 14 tavalla, jossa myöskään salausprosessin tapaan ei ensimmäisellä rivillä tehdä mitään, mutta toisella siirretään viimeinen tavu eteen, kolmannella rivillä siirretään kaksi ja neljännellä kolme tavua. (8, s. 21, 22.)



KUVA 14. Invertoitu bittisiirto havainnollistettuna (8, s.22)

Seuraavaksi viedään *state*-taulukko oman *substitution*-taulukon läpi kuvan 1 tavalla, jossa matriisin koordinaatit vastaavat salauksen *substitution*-laatikon ulostulon koordinaatteja (8, s. 21, 22).

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

KUVA 15. Invertointiversio substitution-taulukosta (8, s.22)

Uusi *state*-taulukko käydään XOR-operaation läpi avainmatriisin avulla, jonka indeksinumero on *kierros*-muuttujan arvo (8, s. 21).

Tämän jälkeen tehdään invertoitu sekoitus *state*-taulukolle. Sekoitusmatriisin arvot saadaan samalla tavalla kuin salauksen matriisin, mutta käytetään polynomia $0Bx^3 + 0Dx^2 + 09x + 0E$. (Kuva 16.) Samoin myös kertolaskussa käytetään modulo $x^4 + 1$. (8, s. 21, 23.)

Sekoitus tehdään kaavalla 3 (8, s. 23).

$S_{r,c} = (S_{0,c} \cdot \alpha_{,0}) \oplus (S_{1,c} \cdot \alpha_{r,1}) \oplus (S_{2,c} \cdot \alpha_{r,2}) \oplus (S_{3,c} \cdot \alpha_{r,3})$ *KAAVA 3*

c = sekoitettavan *State*-sarakkeen numero
α = sekoitusmatriisi
r = rivinumero.

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

KUVA 16. Invertointisekoituksen taulukko (8, s.23)

Kierrosten päätyttyä tehdään viimeisenä invertoitu siirto *state*-taulukolle, invertoitu *substituutio* ja XOR-operaatio avainmatriisin kanssa, jonka indeksinumero on 0, eli ensimmäisellä matriisilla. Näin lopputuloksena on alkuperäinen selväteksti. (8, s. 21.)

4 RSA -SALAUS

RSA perustuu alkulukuihin ja julkisen avaimen kryptografiaan, jossa viestin salaajalla ja vastaanottajalla on kaksi avainta, yksi julkinen ja yksi salainen. Alkulukuja käytetään avainten generoimiseen ja ne ovat suuria lukuja. Koska alkulukujen jakaminen alkutekijöihin on työlästä, varsinkin suurilla alkuluvuilla, vie salauksen purkaminen kauan. (10, s. 1, 2.)

Salausalgoritmia käytetään nykyään tietoliikenteen salaamisessa, kuten SSH-, OpenPGP-, S/MIME-, SSL- ja TLS-protokollissa sekä digitaalisessa allekirjoituksessa (11). RSA-algoritmia voidaan myös käyttää toisten salausten kanssa generoimaan avaimia tai lähettämään yhteinen salainen avain RSA:lla viestinä (12).

Salauksen ideoimisen aloitti englantilainen kryptoloogikko James H. Ellis. Hän kehitteli ideaa käyttää julkisia avaimia jo iäti käytettyjen salaisten avainten sijaan. Ajatuksena oli käyttää julkista lukitusta, jolla toinen osapuoli voisi salata viestinsä omalla salaisella avaimella ja vastaanottaja voisi purkaa salauksen käyttämällä omaa salaista avainta. Hän ei kuitenkaan keksinyt tapaa tehdä ideasta toimivaa kokonaisuutta. (13.)

Ellisin kolleega Clifford Cocks jatkoi idean kehittämistä ja yritti keksiä tapaa, jolla salaaminen on helppo tehdä, mutta purkaminen vaikeaa ellei, toinen osapuoli tiedä salaista aineosaa. Hän ratkaisi arvoituksen käyttämällä kahden alkuluvun tuloa. Korottamalla jokin luku alkulukujen tulolla ja käyttämällä moduloa on mahdollista selvittää alkuperäinen luku vain jos tiedossa ovat funktion tekijät. (13.)

Vuonna 1977 Ron Rivest, Adi Shamir ja Leonard Adle Massachusetts Institute of Technologystä jatkoivat epäsymmetrisen salauksen idean kehitystä toteutukseen asti, vaikka he eivät tienneet Clifford Cocksin keksinnöstä sen ollessa salattua tietoa (13). He patentoivat salauksen vuonna 1983, ja lopulta

vuonna 2000 he julkaisivat salauksensa nimellä RSA, joka tulee keksijöiden sukunimistä (14, s. 2).

Vuonna 1982 he perustivat RSA Data Security -nimisen yrityksen, joka hallinnoi epäsymmetristen salausten standardeja (15, linkit Company -> About Us). Myöhemmin yrityksen nimi vaihtui RSA Securityksi ja se julkaisi ensimmäisen standardin salauksesta vuonna 1993 nimellä PKCS#1 (16, s. 1).

Salauksessa käytettävä avain saadaan yhdistämällä yhden osapuolen salainen avain toisen osapuolen julkisen avaimen kanssa. Näin molemmat saavat rakennettua salaisen avaimen ulkopuolisen tahon näkyvistä. (16, s. 6, 7.)

Salauksen alussa jompikumpi osapuoli tekee julkisen avaimen, jota käytetään läpi salauksen. Tämä tapahtuu valitsemalla kaksi mielivaltaista alkulukua *p* ja *q*. Näiden kahden luvun kertolaskusta saadaan arvo *n*. Mitä suurempia luvut *p ja q* ovat, sitä työläämpiä ne on laskea (1, s. 148). Tästä syystä luku *n* tai julkinen ja salainen avain on valmiiksi laskettu tai poltettu laitteeseen, jotta sitä ei tarvitse laskea (17, kohta 57:50).

Arvosta *n* lasketaan pienin yhteinen jaettava positiivinen kokonaisluku (*LCM*), joka merkitään $\lambda(n) = LCM(p-1, q-1)$. Ennen PKCS#1:tä laskettiin Eulerin φ-funktio $\varphi(n) = (p-1)(q-1)$. (18, s. 6.)

Seuraavaksi valitaan julkiinen avain *e*, joka kuuluu joukkoon {1,2,...,λ(n)-1} ja on keskenään jaoton luvun λ(n) kanssa (16, s. 6). Yleensä *e:*ksi valitaan luku 65537 (1, s.147).

Valitun luvun avulla lasketaan salainen avain *d*, joka on modulaariaritmetiikan käänteisluku luvulle *e*. Se voidaan laskea laajennetulla Eukleideen algoritmilla $gcd\big(e, \lambda(n)\big) = ex + \lambda(n)y$, jossa *x* on *d*. Näin lauseke $e \cdot d \equiv 1 \ (mod\ \lambda(n))$ on tosi. (16, s. 6, 7.)

Julkinen avain on muotoa (*n*,*e*) ja salainen avain on luku *d*. Se osapuoli, joka käyttää julkista avainta, lähettää sen toiselle osapuolelle. Luvut *p, q* ja λ(n) on

syytä tuhota, sillä ne eivät saa paljastua ja niitä ei tästä lähtien enää käytetä. (1, s. 142–143.)

Avainten avulla voidaan tehdä salaus (kaava 4) ja purku (kaava 5). Salaaminen tapahtuu yleensä muuttamalla selvätekstin binääriluvut desimaaliluvuiksi ja tekemällä sen avulla salaus- ja purkufunktiot. (16, s. 10, 11.)

$$c = m^e \ mod \ n$$ *KAAVA 4*

c = salattu teksti
e = julkinen avain
m = selväteksti
n = aiemmin laskettu lukujen *p* ja *q* tulo

$$m = c^d \ mod \ n$$ *KAAVA 5*

c = salattu teksti
d = salainen avain
m = selväteksti
n = aiemmin laskettu lukujen *p* ja *q* tulo

5 AES-128 TOTEUTUS ARDUINO UNOLLA

Arduino Uno on kehitysalusta, joka on rakennettu Atmega328P-mikrokontrollerin ympärille. Kehitysalustaa ohjelmoidaan Arduino Software -ohjelmointiympäristöä käyttämällä. (19, linkit Products → Arduino → Arduino Uno.)

Valitsin kyseisen laitteen toteutuksen tekemiseen, koska olin käyttänyt sitä aiemmin opinnoissani ja näin kiinnostavana lähtökohtana rakentaa sellainen ohjelma, joka veisi mahdollisimman vähän muistia ja prosessointitehoa. Tämän toteutuksen tarkoituksena oli yrittää luoda sellainen ohjelma, joka olisi mahdollista ajaa muissakin kehitysialustoissa.

Kiitokset käyttäjälle Kokke GitHubissa, jonka luoma Tiny AES128 in C -ohjelma auttoi ratkaisemaan kertolaskun toteutuksen salauksessa käytettävän Galois'n kunnan sisällä (20). Ohjelman koodi on liitteenä tässä työssä (liite 1).

Koodin alussa määriteltiin arvot *Nb*:lle, *Nk*:lle ja *Nr*:lle, jotka ovat 4, 4 ja 10. Näiden jälkeen luotiin globaaleja muuttujia. Ensimmäisenä luotiin 8 bitin alkiokokoinen, 16 alkion levyinen taulukko *state*. Sen jälkeen luotiin 16-bittinen *aika*-niminen kokonaismuuttuja, jonka avulla voitiinn tarkastella salaus- ja purkuprosessin kestoa millisekunteina. Näiden jälkeen luotiin 8-bitin alkiokokoiset, 256 alkion levyiset taulukot *substitution*-taulukoita vastaamaan ja yksi 8-bitin alkiokokoinen taulukko *rcon*-taulukolle, jonka sisältö on *rcon*-taulukon vasemmanpuoleiset tavut. Seuraavaksi luotiin ja alustettiin 8-bitin alkiokokoiset taulukot avaimelle (*key*), sisäänsyötettävälle viestille (*in*) kokoa määrittelemätön taulukko ja avaimen laajennusprosessin ulostulolle (*KeyWord*) 176 levyinen taulukko. Avain- ja viestitauluko alustettiin standardissa esitellyn esimerkin tapaiseksi (8, s. 33).

5.1 Avaimen laajennusprosessin toteutus

Näiden jälkeen luotiin funktio avaimen laajennusprosessille, jota voitiinn myöhemmin kutsua salausprosessin alussa. Olisi myös ollut mahdollista kutsua sitä vain kerran *setup*-funktiossa, mutta koodin laajennusta ajatellen, koska avain voidaan rakentaa muuttuvaksi, muuttuu silloin myös avaimen laajennusprosessin ulostulo.

Laajennusprosessin alussa luodaan kaksi sisäistä muuttujaa indeksille ja väliaikaiselle taulukolle, joka tulivat pitämään sisällään väliaikaisia laajennettuja arvoja avaimelle. Indeksimuuttuja *i* on 16-bitin kokoinen kokonaisluku ja väliaikainen taulukko *temp* on 8-bitin kokoinen ja 4 alkion levyinen. Seuraavaksi täytettiin ensimmäiset alkiot *KeyWord*-taulukosta avaimen arvoilla *while*-loopin avulla. Tämän jälkeen alustettiin *i*:n olevan *Nk*:n arvo, joka on 128-bittisellä avaimella 4. Aloitettiin uusi *while*-loop, jossa tapahtumia tehdään niin kauan kun *i* on vähemmän kuin *Nb* kertaa *Nr:*ään lisätty 1. Toisin sanoen, niin kauan kunnes *i* saa arvon 44.

Itse *while*-loopin sisällä tehtiin seuraavasti. Alussa käytiin täyttämässä *temp*-taulukko *KeyWord*-arvoilla *for*-loopin avulla, jossa käytettiin avustavana muuttujana kokonaislukua *j*. Toimintaa tehtiin niin kauan, kunnes *j* on 4, ja *j*:n arvoa kasvatettiin yhdellä jokaisella kierroksella. *KeyWord*-taulukon indeksit olivat muotoa $(i-1)*4+j$. Tämän jälkeen käytiin ehtolausekkeita läpi. Jos *i* jaettuna *Nk* ei anna jakojäämää, siirrettiin *temp*-taulukon sisältö yhdellä vasemmalle, vietiin saatu *temp*-taulukon sisältö *substitution*-taulukon läpi ja siitä saadulle arvolle tehtiin vielä XOR-operaatio *rcon*-arvon kanssa. Koska XOR-operaatio *rcon*-arvon kanssa *temp*-taulukolle muuttaa vain *temp*-taulukon ensimmäistä arvoa, oli *rcon*-taulukko lyhennetty vastaamaan muutoksen aiheuttavia arvoja. Palaten ehtolausekkeisiin, jos *i* jaettuna *Nk* antaa jakojäämän ja *Nk* on suurempi kuin kuusi ja *i* jaettuna *Nk* antaa jakojäämäksi 4, vietiin *temp*-taulukko *substitution*-taulukon läpi. Ehtolausekkeiden jälkeen asetettiin uudet arvot *KeyWord*-taulukolle *for*-loopin avulla, jossa iteroitiin *j*-kokonaismuuttujan avulla sitä aina lisäten yhdellä, kunnes *j* on 4. *KeyWord*-

taulukon alkion indeksi $(i * 4) + j$ sai uuden arvon XOR-operaatiolla, jossa parametreina olivat *temp*-taulukon indeksi *j* ja *KeyWord*-taulukon indeksi $((i - Nk) * 4) + j$. While-loopin lopussa kasvatettiin *i*:n arvoa yhdellä.

5.2 Salausprosessin toteutus

Avaimen laajennusprosessifunktion määrittelyn jälkeen tehtiin määrittelyt salauksessa käytettäviin tapahtumiin. Ensimmäisenä luotiin *xtime*-niminen funktio, joka otti parametrina 8-bittisen muuttujan *x* ja palautti uuden arvon, jolle oli tehty modulaariaritmetiikan kertolasku (19, linkit aes.c -> rivi 280). Seuraavaksi luotiin funktio *MixColumns*, joka sekoitti *state*-taulukon sisällön aiemmin selitetyllä tavalla käyttäen kertolaskuun *xtime*-funktiota (19, linkit aes.c -> rivi 303). Viimeiseksi ennen salausprosessin määrittelyä luotiin funktio *ShiftRows*, joka toteutti *state*-alkioiden siirron standardissa määritellyllä tavalla (8, s. 17).

Salausprosessin funktio oli nimeltään *Cipher* ja se ei palauttanut mitään, sillä se vain muutti globaalin *state*-taulukkomuuttujan sisältöä ja itse salattu teksti tulostettiin myöhemmin. Salauksen alussa kutsuttiin avaimen laajennusprosessia, luotiin avustava indeksimuuttuja *i*, täytettiin *state*-taulukko *in*-taulukon sisällöllä, josta löytyivät salattavan tekstin kirjaimet heksadesimaaleina, ja tehtiin alustava lisäys *state*- ja *KeyWord*-taulukon välillä XOR-operaatiolla.

Tästä eteenpäin jatkettiin *for*-silmukalla, jossa käytettiin aiemmin luotua muuttujaa *i* alustamalla sen arvoksi 1 ja tekemään kierroksia, kunnes *i* on yhtä suuri kuin *Nr*, aina kasvattaen *i*:n arvoa yhdellä jokaisella kierroksella. Kierroksen alussa vietiin *state*-taulukko *substitution*-taulukon läpi antamalla *substitution*-taulukon indeksiksi *state*-taulukon alkion arvon. Seuraavaksi kutsuttiin *ShiftRows*- ja *MixColumns*-funktioita, joiden jälkeen tehtiin kahden *for*-silmukan avulla *state*-taulukon ja kierroksesta riippuvan *KeyWord*-taulukon alkioiden välinen XOR-operaatio, josta saatu arvo sijoitettiin *state*-taulukon käsiteltävään alkioon. Syy kahteen *for*-silmukkaan on *state*-taulukon

yksiulotteisuus, jolloin sen indeksit olivat nollasta viitentoista eivätkä kahden numeron yhdistelmiä.

Itse *for*-silmukat olivat rakennettu siten, että ensimmäinen silmukka teki kierroksia *j*-muuttujalla arvosta 0 lisäten itseään yhdellä jokaisella kierroksella ja silmukka loppui, kun *j* sai arvon 4. Seuraava *for*-silmukka pyöri edellä mainitun sisällä ja se toimi samalla tavalla, mutta muuttujana oli *k*. Indeksien käsittely tehtiin kaavalla 6.

$$state[(j * 4) + k] \text{ ^ } = KeyWord[(i * Nb * 4) + (j * Nb) + k]$$ *KAAVA 6*

i = ylemmän tason *for*-silmukan muuttuja ts. kierroksen numero
j = ensimmäisen *for*-silmukan muuttuja
k = toisen *for*-silmukan muuttuja
Nb = aiemmin mainittu pysyvä arvo 4

Ylimmän *for*-silmukan käytyä itsensä läpi tehtiin vielä *state*-taulukolle *substitution*-taulukon läpi vienti, *ShiftRows*-funktion kutsu ja edellä mainittu *state*- ja *KeyWord*-taulukon välinen XOR-operaatio.

5.3 Salauksen purkamisprosessin toteutus

Ennen salausfunktion määrittelyä luotiin funktio käänteiselle *MixColumns*-operaatiolle, toinen funktio kertolaskulle kahden polynomin välillä GF(2^8):n sisällä (19, linkit aes.c -> rivit 303-340) ja funktio *InvShiftRows* purkamisen alkioiden siirrolle standardin mukaisesti (8, s. 21).

Näiden jälkeen tehtiin salauksen purkamiseen funktio nimeltä *InvCipher*. Sen alussa lisättiin *state*-taulukon sisältö yhteen *KeyWord*-taulukon viimeisien alkioiden kanssa XOR-operaatiolla. Tästä eteenpäin käytiin kierroksia läpi *for*-silmukassa, jossa luotiin kierrosmuuttuja *i* ja alustettiin sen arvoksi $Nr - 1$. Muuttujaa vähennettiin jokaisella kierroksella yhdellä ja kierroksia tehtiin niin kauan, kunnes *i* on nolla.

For-silmukan alussa kutsuttiin *InvShiftRows*-funktiota, jonka jälkeen vietiin *state*-taulukon sisältö purkamiseen käytettävän *substitution*-laatikon läpi. Saatu arvo käsiteltiin XOR-operaatiolla kuten salausprosessin tavalla, kahdella *for*-silmukalla, mutta indeksien käsittely tehtiin kaavalla 7.

$$state[(j * 4) + k] \text{ ^ } = KeyWord[(i * Nb * 4) + (j * Nb) + k] \qquad \textit{KAAVA 7}$$

i = ylemmän tason *for*-silmukan muuttuja ts. kierroksen numero
j = ensimmäisen *for*-silmukan muuttuja
k = toisen *for*-silmukan muuttuja
Nb = aiemmin mainittu pysyvä arvo 4

Viimeiseksi kierroksella kutsuttiin *InvMixColumns*-funktiota. Kierrosten jälkeen kutsuttiin vielä kerran *InvShiftRows*-funktiota, vietiin *state*-taulukko purkuprosessin *substitution*-taulukon läpi ja lisättiin saatu arvo yhteen XOR-operaatiolla *KeyWord*-taulukon alkioiden kanssa kuten aiemmin, mutta indeksien käsittely tapahtui kaavalla 8.

$$state[(j * 4) + k] \text{ ^ } = KeyWord[(0 * Nb * 4) + (j * Nb) + k] \qquad \textit{KAAVA 8}$$

i = ylemmän tason *for*-silmukan muuttuja ts. kierroksen numero
j = ensimmäisen *for*-silmukan muuttuja
k = toisen *for*-silmukan muuttuja
Nb = aiemmin mainittu pysyvä arvo 4

5.4 Salauksen toiminta Arduinossa

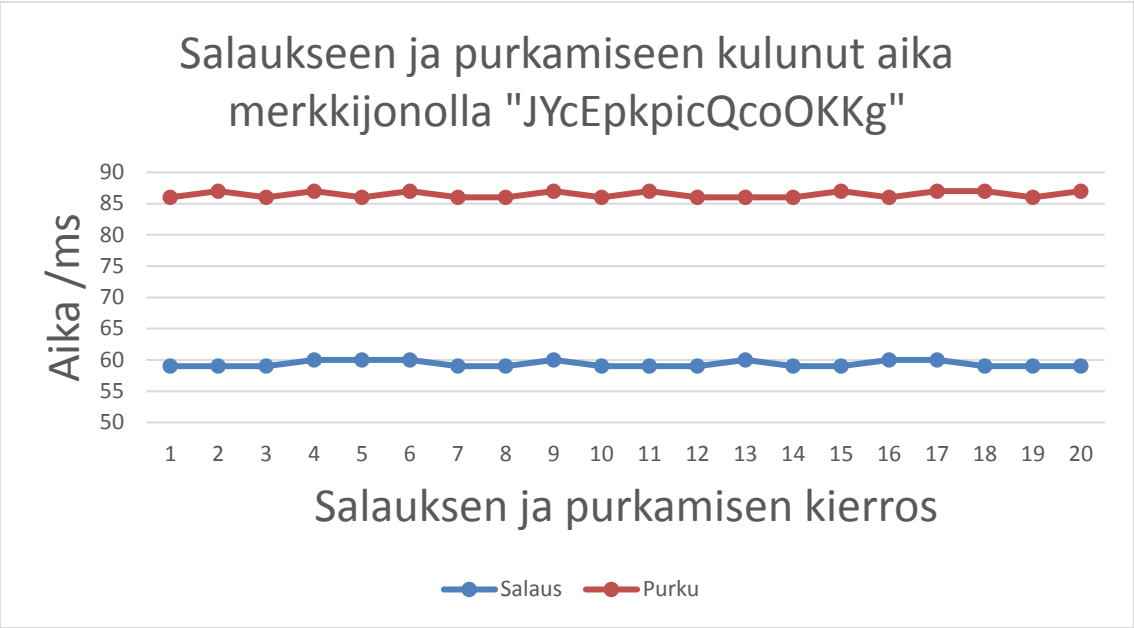
Arduinon ohjelman *setup*-funktiossa käynnistettiin sarjaliikenneväylä käyttämällä symbolinopeutena arvoa 9600 ja lähettämällä väylälle viesti, jossa kehotettiin antamaan salattava selväteksti.

Tämän jälkeen ohjelmaa jatkettiin *loop*-funktiossa, jossa ohjelma tulee pyörimään, kunnes se sammutetaan. Funktion alussa tarkasteltiin, oliko väylällä merkkejä, joita voitiin lähettää salattavaksi. Jos vapaita merkkejä oli yli 15, voitiin aloittaa salausprosessi. Heti alkuun otettiin *aika*-muuttujaan tämän hetkinen aika millisekunteina ja luotiin *char*-tyyppinen *inputArray*-taulukko-muuttuja, johon *for*-loopin avulla sijoitettiin sarjaväylältä luettu merkki ja *in-*

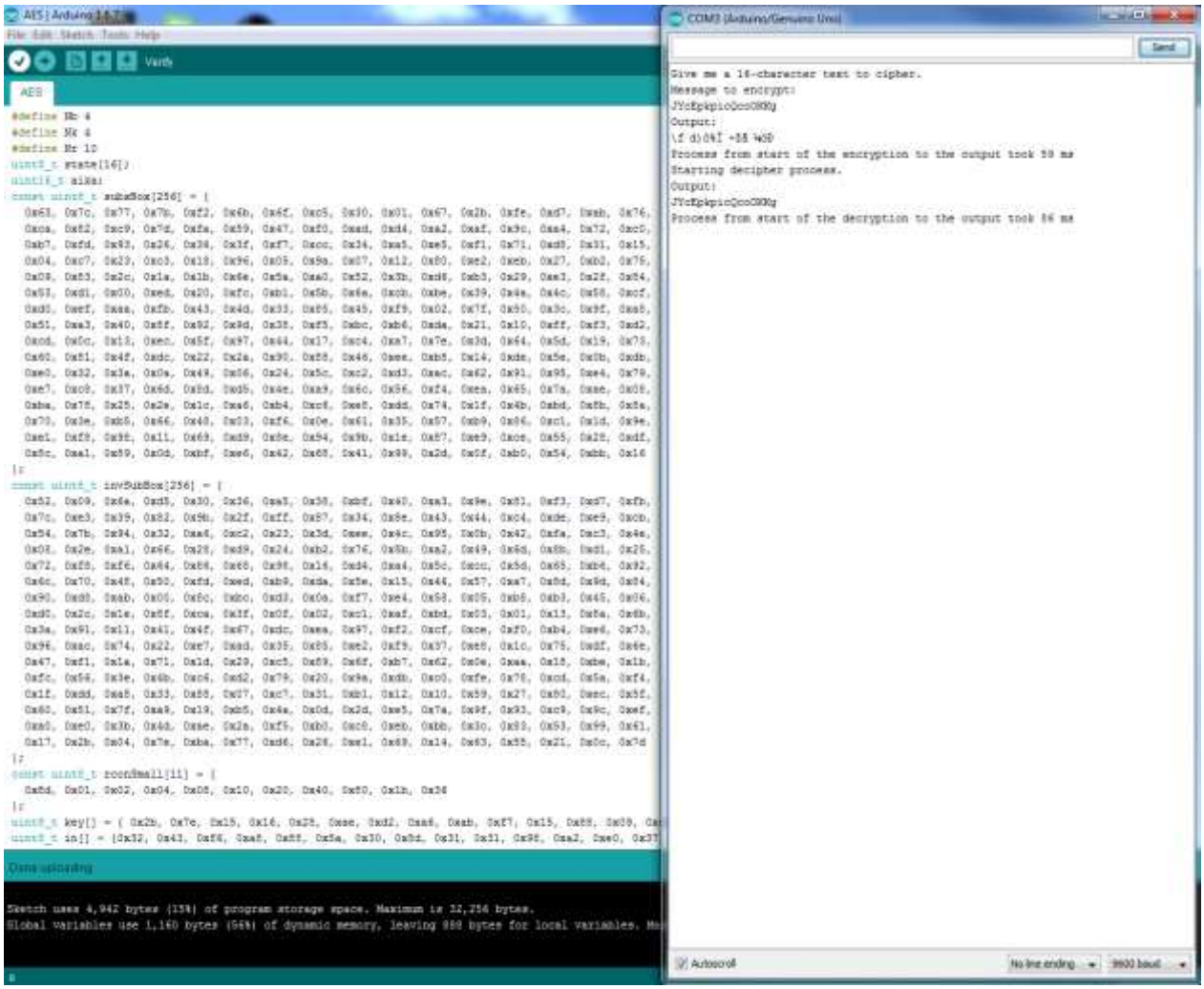
taulukkoon sijoitettiin 8-bitin notaatio kyseisestä merkistä. Seuraavaksi tulostettiin sarjaporttiin *inputArray*:n sisältö, jotta voitiin varmistaa lähetettävän datan validiteetti, jonka jälkeen kutsuttiin *Cipher*-funktiota tekemään salausprosessi. Salausprosessin valmistuttua tulostettiin sarjaporttiin *state*-taulukon sisältö *char*-tyyppisinä merkkeinä, kerrottiin salauksen valmistuneen ja tulostettiin vielä nykyisen ajan millisekunnit vähennettynä *aika*-muuttujalla, jolloin saatiin prosessiin kulunut aika.

Salauksen jälkeen voitiin aloittaa salatun tekstin purkaminen. Tulostettiin sarjaporttiin ilmoitus purkamisen aloittamisesta, otettiin uusi aikaleima ylös *aika*-muuttujaan ja kutsuttiin *InvCipher*-funktiota. Koska *state*-taulukko oli globaalimuuttuja, ohjelman ei tarvinnut käsitellä sarjaporttiin tulostetun sarjan sisältöä takaisin *in*-taulukkoon. Lisäksi koska ohjelma oli muutenkin rakennettu esitystarkoitukseen, voitiin *state*-taulukkoa käyttää viestin purkamisen esittämiseen. *InvCipher*-funktion tehtyä tehtävänsä tulostettiin *char*-tyyppisinä merkkeinä uusi *state*-taulukko, ilmoitettiin purkuprosessin valmistuneen ja kerrottiin kulunut aika, kuten aiemmin *aika*-muuttujan avulla millisekunteina. Lopussa vielä tyhjennettiin sarjaportti ja asetettiin *aika*-muuttujan arvo nollaksi seuraavaa viestiä varten.

Salaus kesti keskimäärin noin 59 millisekuntia ja purkaminen noin 86 millisekuntia (kuva 17). Ohjelma vei staattista muistia noin 5 kt:n verran ja dynaamista muistia 1,2 kt:n verran (kuva 18).



KUVA 17. Salaukseen ja purkamiseen kestävän ajan mittaus



KUVA 18. Toimiva toteutus Arduinolla AES128-salauksesta, jossa näkyvät kulunut aika ja käytetyn muistin määrä

6 LOPPUSANAT

Työn päätarkoituksena oli oppia nykyajan tietoliikenteessä käytettävien salausmenetelmien historiasta ja toiminnasta. Lisättynä tarkoituksena oli tehdä toteutus jostain näistä menetelmistä Arduino-laitteella.

Salausmenetelmiin paneutumalla olen oppinut paljon salauksien toiminnasta yleisesti ja olen myös oppinut arvostamaan matemaattista työtä niiden takana. Tarkemmin paneutumalla kolmeen menetelmään ymmärrän nyt niiden ominaisuudet ja rakenteet. Opinnäytetyötä tehdessä opin myös ensimmäistä kertaa ryhmäteoriasta ja laajentanut lukuteorian osaamista. Suurena apuna matematiikan asioissa on ollut OAMK:n yliopettaja Jarkko Hurme, joka avusti minua ymmärtämään ryhmäteoriaa paremmin.

Toteutusta tehdessä olen huomannut, kuinka vaikeita jotkin matemaattiset operaatiot ovat kääntää konekielelle, kun pidetään huoli laitteen muistin käsittelystä ja prosessin nopeudesta. Samalla olen huomannut omien taitojeni puutteellisuuden mikrokontrollerien ohjelmoinnissa, josta toimivana esimerkkinä on toteutuksessa käytettävien modulaariaritmetiikan kertolaskujen toteutus käyttäjätunnus Kokken koodilla.

Lopputulokseen olen tyytyväinen, sillä opin itseäni kiinnostavien salausmenetelmien historiasta, niiden toiminnasta ja niiden käyttötarkoituksista sekä sain tehtyä toimivan toteutuksen itseäni kiinnostavasta menetelmästä.

Haluaisin kiittää ohjaajaani Tuula Hopeavuorta hänen avustaan tekstin oikoluvussa ja hänen palautteestaan eri vaiheista työn aikana. Lisäksi haluaisin kiittää toista ohjaajaani, Veijo Väisästä, hänen avustaan opinnäytetyön toteuttamisessa ja seuraavien työtehtävien miettimisessä. Ilman heitä olisi työni ollut paljon vaikeampi ja lopputulos olisi varmasti ollut suppeampi.

LÄHTEET

1. Järvinen, Petteri 2003. Salausmenetelmät. Jyväskylä: Docendo.

2. Rouse, Margaret 2014. Data Encryption Standard (DES). TechTarget. Saatavissa: http://searchsecurity.techtarget.com/definition/Data-Encryption-Standard. Hakupäivä 5.3.2016.

3. Data Encryption Standard. 1999. National Institute of Standards and Technology. Saatavissa: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf. Hakupäivä 15.1.2016.

4. "EFF DES Cracker" machine brings honesty to crypto debate. 1998. Electronic Frontier Foundation. Saatavissa: https://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_descracker_pressrel.html. Hakupäivä 22.1.2016.

5. National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information. 2003. National Institute of Standards and Technology. Saatavissa: http://csrc.nist.gov/groups/ST/toolkit/documents/aes/CNSS15FS.pdf. Hakupäivä 5.3.2016.

6. Villanueva, John Carl 2015. What AES Encryption Is And How It's Used To Secure File Transfers. JScape. Saatavissa: http://www.jscape.com/blog/aes-encryption. Hakupäivä 5.3.2016.

7. Rouse, Margaret 2014. Advanced Encryption Standard (AES). TechTarget. Saatavissa: http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard. Hakupäivä 5.3.2016.

8. Advanced Encryption Standard. 2001. National Institute of Standards and Technology. Saatavissa: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. Hakupäivä 3.2.2016.

9. Abdulah Abdulah, Zadeh 2012. Division and Inversion Over Finite Fields. Teoksessa Dr. Jaydip, Sen (toim.). Cryptography and Security in Computing. Rijeka, Kroatia: InTech Europe. S. 117-130. Saatavissa: http://cdn.intechopen.com/pdfs/29704.pdf. Hakupäivä 3.3.2016.

10. Kleinjung, Thorsten - Kazumaro, Aoki -Franke, Jens - Lenstra, Arjen K. - Thomé, Emmanuel - Bos, Joppe W. - Gaudry, Pierrick - Kruppa, Alexander - Montgomery, Peter L. - Osvik, Dag Arne - te Riele, Herman - Timofeev, Andrey - Zimmerman, Paul 2010. Factorization of a 768-bit RSA modulus. Saatavissa: http://eprint.iacr.org/2010/006.pdf. Hakupäivä 25.2.2016.

11. Rouse, Margaret 2014. RSA algorithm (Rivest-Shamir-Adleman). TechTarget. Saatavissa: http://searchsecurity.techtarget.com/definition/RSA. Hakupäivä 5.3.2016.

12. RSA Laboratories, Standards Initiatives. 3.1.7 How is the RSA algorithm used for privacy in practice?. EMC. Saatavissa: http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/rsa-algorithm-used-for-privacy-in-practice.htm. Hakupäivä 5.3.2016.

13. Espiner, Tom 2010. GCHQ pioneers on birth of public key crypto. ZDNet. Saatavissa: http://www.zdnet.com/article/gchq-pioneers-on-birth-of-public-key-crypto/. Hakupäivä 5.3.2016.

14. Robinson, Sara 2003. Still Guarding Secrets after Years of Attacks, RSA Earns Accolades for its Founders. SIAM News vol. 36, nro 5. Saatavissa: http://www.msri.org/people/members/sara/articles/rsa.pdf. Hakupäivä 5.3.2016.

15. RSA Security. Saatavissa: https://www.rsa.com/en-us/. Hakupäivä 5.3.2016.

16. Kalinski, B. 1998. PKCS #1: RSA Encryption Version 1.5. Saatavissa: http://www.rfc-base.org/txt/rfc-2313.txt. Hakupäivä 5.3.2016.

17. Paar, Christof 2014. Lecture 12: The RSA Cryptosystem and Efficient Exponentiation by Christof Paar. Video. Saatavissa: https://www.youtube.com/watch?v=QSlWzKNbKrU. Hakupäivä 27.2.2016.

18. RSA Cryptography Standard. 2012. RSA Laboratories. Saatavissa: https://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf. Hakupäivä 15.2.2016.

19. Arduino. Saatavissa: http://www.arduino.cc/. Hakupäivä 1.4.2016.

20. Kokke 2015. Tiny AES128 in C. GitHub. Saatavissa: https://github.com/kokke/tiny-AES128-C. Hakupäivä 17.3.2016.

Ohjelmakoodi AES128-salauksesta Arduinolla

```
#define Nb 4
#define Nk 4
#define Nr 10
uint8_t state[16];
uint16_t aika;
const uint8_t subsBox[256] = {
  0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67,
0x2b, 0xfe, 0xd7, 0xab, 0x76,
  0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2,
0xaf, 0x9c, 0xa4, 0x72, 0xc0,
  0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5,
0xf1, 0x71, 0xd8, 0x31, 0x15,
  0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80,
0xe2, 0xeb, 0x27, 0xb2, 0x75,
  0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6,
0xb3, 0x29, 0xe3, 0x2f, 0x84,
  0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe,
0x39, 0x4a, 0x4c, 0x58, 0xcf,
  0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02,
0x7f, 0x50, 0x3c, 0x9f, 0xa8,
  0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda,
0x21, 0x10, 0xff, 0xf3, 0xd2,
  0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e,
0x3d, 0x64, 0x5d, 0x19, 0x73,
  0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8,
0x14, 0xde, 0x5e, 0x0b, 0xdb,
  0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac,
0x62, 0x91, 0x95, 0xe4, 0x79,
  0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4,
0xea, 0x65, 0x7a, 0xae, 0x08,
  0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74,
0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
  0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57,
0xb9, 0x86, 0xc1, 0x1d, 0x9e,
  0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87,
0xe9, 0xce, 0x55, 0x28, 0xdf,
  0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d,
0x0f, 0xb0, 0x54, 0xbb, 0x16
};
const uint8_t invSubBox[256] = {
  0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3,
0x9e, 0x81, 0xf3, 0xd7, 0xfb,
  0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43,
0x44, 0xc4, 0xde, 0xe9, 0xcb,
  0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95,
0x0b, 0x42, 0xfa, 0xc3, 0x4e,
  0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2,
0x49, 0x6d, 0x8b, 0xd1, 0x25,
  0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c,
0xcc, 0x5d, 0x65, 0xb6, 0x92,
  0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46,
0x57, 0xa7, 0x8d, 0x9d, 0x84,
  0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58,
0x05, 0xb8, 0xb3, 0x45, 0x06,
```

```
  0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd,
0x03, 0x01, 0x13, 0x8a, 0x6b,
  0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf,
0xce, 0xf0, 0xb4, 0xe6, 0x73,
  0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37,
0xe8, 0x1c, 0x75, 0xdf, 0x6e,
  0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62,
0x0e, 0xaa, 0x18, 0xbe, 0x1b,
  0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0,
0xfe, 0x78, 0xcd, 0x5a, 0xf4,
  0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10,
0x59, 0x27, 0x80, 0xec, 0x5f,
  0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a,
0x9f, 0x93, 0xc9, 0x9c, 0xef,
  0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb,
0x3c, 0x83, 0x53, 0x99, 0x61,
  0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14,
0x63, 0x55, 0x21, 0x0c, 0x7d
};
const uint8_t rconSmall[11] = {
  0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36
};
uint8_t key[] = { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c };
uint8_t in[] = {0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a, 0x30, 0x8d, 0x31,
0x31, 0x98, 0xa2, 0xe0, 0x37, 0x07, 0x34};
uint8_t KeyWord[176];
void KeyExpansion() {
  uint16_t i = 0;
  uint8_t temp[4];
  while (i < Nk) {
    KeyWord[(i * 4)] = key[(i * 4)];
    KeyWord[(i * 4) + 1] = key[(i * 4) + 1];
    KeyWord[(i * 4) + 2] = key[(i * 4) + 2];
    KeyWord[(i * 4) + 3] = key[(i * 4) + 3];
    i++;
  }

  i = Nk;
  while (i < (Nb * (Nr + 1))) {
    for (int j = 0; j < 4; j++) {
      temp[j] = KeyWord[(i - 1) * 4 + j];
    }
    if (i % Nk == 0) {
      //RotKeyWord schedule
      uint8_t apu = temp[0];
      for (int j = 0; j < 3; j++) {
        temp[j] = temp[j + 1];
      }
      temp[3] = apu;


      //SubKeyWord schedule
      for (int j = 0; j < 4; j++) {
        temp[j] = subsBox[temp[j]];
      }

      //XOR w/ Rcon
      temp[0] ^= rconSmall[i / Nk];

    }
```

```
    else if ((Nk > 6) && (i % Nk == 4)) {
      //SubKeyWord schedule
      for (int j = 0; j < 4; j++) {
        temp[j] = subsBox[temp[j]];
      }
    }
    for (int j = 0; j < 4; j++) {
      KeyWord[(i * 4) + j] =  KeyWord[((i - Nk) * 4) + j] ^ temp[j];
    }
    i++;
  }
}
uint8_t xtime(uint8_t x)
{
  return ((x << 1) ^ (((x >> 7) & 1) * 0x1b));
}
void MixColumns() {
  uint8_t Tmp, Tm, t;
  for (int a = 0; a < 4; a++) {
    t = state[(a * 4) + 0];
    Tmp = state[(a * 4) + 0] ^ state[(a * 4) + 1] ^ state[(a * 4) + 2]
^ state[(a * 4) + 3];
    Tm = state[(a * 4) + 0] ^ state[(a * 4) + 1]; Tm = xtime(Tm);
state[(a * 4) + 0] ^= Tm ^ Tmp;
    Tm = state[(a * 4) + 1] ^ state[(a * 4) + 2]; Tm = xtime(Tm);
state[(a * 4) + 1] ^= Tm ^ Tmp;
    Tm = state[(a * 4) + 2] ^ state[(a * 4) + 3]; Tm = xtime(Tm);
state[(a * 4) + 2] ^= Tm ^ Tmp;
    Tm = state[(a * 4) + 3] ^ t; Tm = xtime(Tm); state[(a * 4) + 3] ^=
Tm ^ Tmp;
  }
}
void ShiftRows() {
  uint8_t apu, apu1;
  //second row
  apu = state[1];
  state[1] = state[5];
  state[5] = state[9];
  state[9] = state[13];
  state[13] = apu;
  //third row
  apu = state[2];
  apu1 = state[6];
  state[2] = state[10];
  state[6] = state[14];
  state[10] = apu;
  state[14] = apu1;
  //fourth row
  apu = state[3];
  state[3] = state[15];
  state[15] = state[11];
  state[11] = state[7];
  state[7] = apu;
}
void Cipher() {
  KeyExpansion();
  uint8_t i;
  for (int a = 0; a < 16; a++) {
    state[a] = in[a];
  }
  //First AddRoundKey
```

```
  for (int a = 0; a < 16; a++) {
    state[a] ^= KeyWord[a];
  }
  //start of for-loop
  for (i = 1; i < Nr; i++) {
    //SubBytes routine
    for (int j = 0; j < 16; j++) {
      state[j] = subsBox[state[j]];
    }

          //ShiftRows routine
    ShiftRows();

    //MixColumns routine
    MixColumns();

    //AddRoundKey routine
    for (int j = 0; j < 4; j++) {
      for (int k = 0; k < 4; k++) {
        state[(j * 4) + k] ^= KeyWord[(i * Nb * 4) + (j * Nb) + k];
      }
    }
  } //end of for-loop and start of last round

  //SubBytes routine
  for (int a = 0; a < 16; a++) {
    state[a] = subsBox[state[a]];
  }
  //ShiftRows routine
  ShiftRows();
  //AddRoundKey
  for (int j = 0; j < 4; j++) {
    for (int k = 0; k < 4; k++) {
      state[(j * 4) + k] ^= KeyWord[(i * Nb * 4) + (j * Nb) + k];
    }
  }
}
void InvMixColumns() {
  uint8_t a, b, c, d;
  for (int i = 0; i < 4; ++i)
  {
    a = state[(i * 4) + 0];
    b = state[(i * 4) + 1];
    c = state[(i * 4) + 2];
    d = state[(i * 4) + 3];

    state[(i * 4) + 0] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Mul-
tiply(c, 0x0d) ^ Multiply(d, 0x09);
    state[(i * 4) + 1] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Mul-
tiply(c, 0x0b) ^ Multiply(d, 0x0d);
    state[(i * 4) + 2] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Mul-
tiply(c, 0x0e) ^ Multiply(d, 0x0b);
    state[(i * 4) + 3] = Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Mul-
tiply(c, 0x09) ^ Multiply(d, 0x0e);
  }
}
uint8_t Multiply(uint8_t x, uint8_t y)
{
  return (((y & 1) * x) ^
          ((y >> 1 & 1) * xtime(x)) ^
          ((y >> 2 & 1) * xtime(xtime(x))) ^
```

```
          ((y >> 3 & 1) * xtime(xtime(xtime(x)))) ^
          ((y >> 4 & 1) * xtime(xtime(xtime(xtime(x))))));
}
void InvShiftRows() {
  uint8_t apu;
  //second row
  apu = state[13];
  state[13] = state[9];
  state[9] = state[5];
  state[5] = state[1];
  state[1] = apu;
  //third row
  apu = state[2];
  state[2] = state[10];
  state[10] = apu;
  apu = state[6];
  state[6] = state[14];
  state[14] = apu;
  //fourth row
  apu = state[3];
  state[3] = state[7];
  state[7] = state[11];
  state[11] = state[15];
  state[15] = apu;
}
void InvCipher() {
  //AddRoundKey before loop
  for (int j = 0; j < 4; j++) {
    for (int k = 0; k < 4; k++) {
      state[(j * 4) + k] ^= KeyWord[(Nr * Nb * 4) + (j * Nb) + k];
    }
  }
  //start of for loop
  for (int i = (Nr - 1); i > 0; i--) {
    //InvShiftRows
    InvShiftRows();
    //InvSubBytes
    for (int i = 0; i < 16; i++) {
      state[i] = invSubBox[state[i]];
    }
    //AddRoundKey
    for (int j = 0; j < 4; j++) {
      for (int k = 0; k < 4; k++) {
        state[(j * 4) + k] ^= KeyWord[(i * Nb * 4) + (j * Nb) + k];
      }
    }
    //InvMixColumns
    InvMixColumns();
  }//end of for loop

  //InvShiftRows
  InvShiftRows();
  //InvSubBytes
  for (int i = 0; i < 16; i++) {
    state[i] = invSubBox[state[i]];
  }
  //AddRoundKey
  for (int j = 0; j < 4; j++) {
    for (int k = 0; k < 4; k++) {
      state[(j * 4) + k] ^= KeyWord[(0 * Nb * 4) + (j * Nb) + k];
    }
```

```
  }
}
void setup() {
  Serial.begin(9600);
  Serial.println("Give me a 16-character text to cipher.");
}
void loop() {
  if (Serial.available() > 15) {
    aika = millis();
    char inputArray[16];
    for (int i = 0; i < 16; i++) {
      inputArray[i] = Serial.read();
      in[i] = (uint8_t)inputArray[i];
    }
    Serial.println("Message to encrypt: ");
    for (int i = 0; i < 16; i++) {
      Serial.print(inputArray[i]);
    }
    Serial.print("\n");
    Cipher();
    Serial.println("Output: ");
    for (int k = 0; k < 16; k++) {
      Serial.print((char)state[k]);
    }
    Serial.print("\n");
    Serial.print("Process from start of the encryption to the output
took "); Serial.print((millis() - aika)); Serial.println(" ms");
    Serial.println("Starting decipher process.");
    aika = millis();
    InvCipher();
    Serial.println("Output: ");
    for (int k = 0; k < 16; k++) {
      Serial.print((char)state[k]);
    }
    Serial.print("\n");
    Serial.print("Process from start of the decryption to the output
took "); Serial.print((millis() - aika)); Serial.println(" ms");
    Serial.print("\n");
    Serial.flush();
    aika = 0;
  }
}
```

LIITE 2

Saku Rautio

SULAUTETTUJEN OHJELMISTOJEN KEHITYSTYÖKALUJEN EVALUOINTI

SULAUTETTUJEN OHJELMISTOJEN KEHITYSTYÖKALUJEN EVALUOINTI

Saku Rautio
Opinnäytetyö, osa 2
Kevät 2018
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä: Saku Rautio
Opinnäytetyön nimi: Sulautettujen ohjelmistojen kehitystyökalujen evaluointi
Työn ohjaaja: Veijo Väisänen, Petri Soininen
Työn valmistumislukukausi ja -vuosi: kevät 2018
Sivumäärä: 33

RD Velho oy etsii sulautetun ohjelmiston kehitykseen parempia työkaluja, joilla voitaisiin helpottaa ohjelmistokehitystä, validoida ohjelmiston toimivuutta, jäljittää sulautetun ohjelmiston virheitä ja yhtenäistää yrityksen sisällä käytettäviä työkaluja.

Tehtävänä oli evaluoida kahta mahdollista työkalupakettia sekä vapaasti levitettäviä ohjelmistovaihtoehtoja ja saada selville, mikä niistä olisi paras RD Velho -yrityksen käyttöön.

Evaluointi tapahtui esittelemällä vaihtoehdot koodin kirjoittamiselle, kääntäjät, virheenjäljityksen mahdollisuudet sekä lisenssit. Lopuksi tehtiin vertailu työkalupakettien välillä ja johtopäätökset siitä, mitä tehdä seuraavaksi.

Johtopäätöksenä on odottaa vielä yhden työkalun esilletuloa ja tällä hetkellä käsiteltävien työkalujen kypsymistä.

Asiasanat: sulautettu ohjelmisto, ohjelmistotyökalut, evaluointi, vapaasti levitettävä ohjelmisto, virheenjäljitysohjelma

ABSTRACT

Oulu University of Applied Sciences
Degree Program of Information Technology, software development

Author: Saku Rautio
Title of thesis: Sulautettujen ohjelmistojen kehitystyökalujen evaluointi
Supervisor: Veijo Väisänen, Petri Soininen
Term and year when the thesis was submitted: spring 2018
Pages: 33

RD Velho oy is looking for better tools for embedded software development, which would help software development, validating the software, debugging of the software and to standardize the tools used inside the company.

The task was to evaluate two possible tools and free software options, and to find out which of these would be the best choice for use to the RD Velho company.

The evaluation was done by showcasing the options to write code, the compilers, the possibilities of debugging and the licensing. In the end, a comparison was made between the tools and a conclusion was made on what to do next.

As the conclusion, it is advisable to wait until the publication of one more tool and to wait for the tools evaluated to become more mature.

Keywords: embedded software, software tools, evaluation, free software, debugging

ALKULAUSE

Haluan kiittää ohjaajaani lehtori Veijo Väisästä hänen tuestaan ja kärsivällisyydestään opinnäytetyötäni varten.

Haluan kiittää työnantajaani RD Velhoa, joka antoi minulle aiheen opinnäytetyöhön ja tuen sen tekemistä varten. Erityisesti kiitän työtovereitani Senior Software Designer Petri Soinista, Senior Software Designer Jukka Luomajokea sekä Software Designer Toni Heikkistä.

12.4.2018

Saku Rautio

SISÄLLYS

1 JOHDANTO

RD Velho on suunnittelutoimisto, jonka osaamisalueina ovat ohjelmisto-, elektroniikka-, muotoilu-, pakkaus- sekä mekaniikkasunnittelu. Tällä hetkellä RD Velhon ohjelmistosuunniteluosasto on kasvamassa nopeasti ja on tullut ilmi tarve yhtenäistää työkalut sekä ottaa käyttöön parempia työkaluja. Tämän takia on nyt haluttu selvittää, mitä työkaluja on saatavilla sulautetun ohjelmiston kehitykseen.

Tämän työn tarkoituksena on tutkia, mitkä työkalut olisivat parhaimpia ja sopivimpia RD Velholle sulautettujen ohjelmistojen kehittämiseen. Tarkasteltavina asioina ovat kehitysympäristöt, kääntäjät, tuki C- ja C++-kielille, virheenjäljityksen laajuus sekä lisensointi.

Työtä tehtiin tutustumalla työkalujen toimintoihin ja etsimällä löytyvätkö virheenjäljitykseen koodin purkaminen, keskeytyskohdat, stack trace ja muuttujien seuranta. Näitä ominaisuuksia tutkittiin samalla ohjelmalla työkalujen kesken.

Lopputuloksena on vertailu ilmaisten työkalujen, ARM:n MDK-ARM v5:n sekä IAR:n Embedded Workbenchin välillä. Tämän työn lopputulos on apuna käytettävän työkalun valinnassa.

Koska osa työstä on arkaluonteista ja ei suotavaa julkistettavaksi, nämä osat ovat vain RD Velhon käytössä.

2 VAPAASTI LEVITETTÄVÄT TYÖKALUT

Yhtenä tarkastelukohteena on vapaasti levitettävien työkalujen käyttöönotto. Nimellisesti tunnetuimpia työkaluja ovat esimerkiksi GCC, Clang, Eclipse, NetBeans ja GDB.

Vapaasti levitettävät työkalut ovat yrityksille ohjelmistokehityksessä hyviä siinä mielessä, että ne ovat ilmaisia, ne skaalautuvat hyvin ja niitä käytetään monessa paikassa. Näin ollen niihin saatava tuki ja integroituminen muihin järjestelmiin on joissain tilanteissa parempi kuin kaupallisissa vaihtoehdoissa (1).

2.1 IDE

Sulautetun ohjelmiston kehitykseen on monta IDE:ä, Integrated Desktop Environmentia, joista moni on kuitenkin muunnoksia olemassaolevista IDE:istä. Esimerkiksi Texas Instrumentsin Code Composer Studio on tehty Eclipsen päälle (2), kuten myös STM:n ja AC6:n yhteistyössä tehty SW4STM32 (3). Koska RD Velholla on aiemmin tehty kehitystä Eclipsen kanssa, on se järkevin ottaa käsiteltäväksi IDE:ksi.

Eclipsen käyttö koodia kirjoitettaessa on kätevää ja tarvituimmat toiminnot ovat vain muutaman näppäinyhdistelmän takana. Hiiren vieminen muuttujan tai funktion kohdalle avaa ponnahdusikkunan, jossa näkyy kyseisen koodiobjektin sisältö. F3-näppäimen painaminen vie käyttäjän funktion määrittelyyn tai objektin deklaraatioon, kun kursori on funktion tai objektin kohdalla. Eclipsellä koodia kirjoitettaessa Ctrl + välilyönti -yhdistelmällä saadaan ennakoituja vaihtoehtoja, mitä käyttäjä haluaa kirjoittaa.

Eclipse CDT -versiossa mukana tulee natiivi tuki C- ja C++-kielille. Se tukee C++-kieltä C++14-standardiin asti (vaikkakin C++14-tuki on kesken) (4). C-kielen tuesta ei ole mainintaa, mutta luultavammin sen pitäisi tukea C11-versioon asti. Kääntäjän tuesta riippuen ohjelma sitten kääntyy standardien mukaisesti tai ei.

Eclipseen tuodaan ja asennetaan ns. plug-ineja, eli lisäosia, Eclipsen sisäänrakennetun Marketplace-nimisen palvelun kautta. Esimerkiksi GNU MCU Eclipse -lisäosa tuo tuen ARM-prosessoreille.

2.2 Kääntäjät

2.2.1 GCC

GCC on yleisin kääntäjä C- ja C++-kielen ohjelmille ja se onkin ollut RD Velholla yleisin käytetty kääntäjä sulautettujen ohjelmistojen kehityksessä. GCC tukee sekä uusinta C-standardia että C++14-standardia. C++17:n tuki on vielä kesken (5).

GCC:tä on yleensä vielä muokattu puolijohdevalmistajan mukaan, jotta sillä voidaan kääntää prosessorille ymmärrettävää ohjelmaa. Näin esimerkiksi toimii ARM:n GNU Arm Embedded Toolchain (6) ja Atmelin AVR GNU Toolchain (7). Monissa työkaluissa GCC:n käyttäminen on sisällytetty korkeamman tason työkaluihin, jolloin GCC:n käytöstä ei tarvitse huolehtia tai sitä ei tarvitse ylläpitää.

2.2.2 Clang

Clang on C-, C++-, Objective C- ja Objective C++ -kielille tarkoitettu frontend LLVM-kääntäjäarkkitehtuurin päälle. Clangin tarkoitus on olla kääntäjä, joka on nopea ja käyttää mahdollisimman vähän muistia. (8.)

Clang tukee C-kieltä uusimpaan standardiin asti (9) sekä C++-kieltä täysin C++14-standardiin asti (10).

Clangia ei ole tuettu tällä hetkellä alustoille ja prosessoreille, joita RD Velho käyttää, mutta kääntäjän kutsumiskomentoja muuttamalla manuaalisesti voidaan ohjelma saada käännettyä (11). Tämä kuitenkin tekee ohjelmistoprojektin ylläpidosta, dokumentaatiosta ja tuotannosta vaikeaa, jolloin Clang ei ole varteenotettava vaihtoehto.

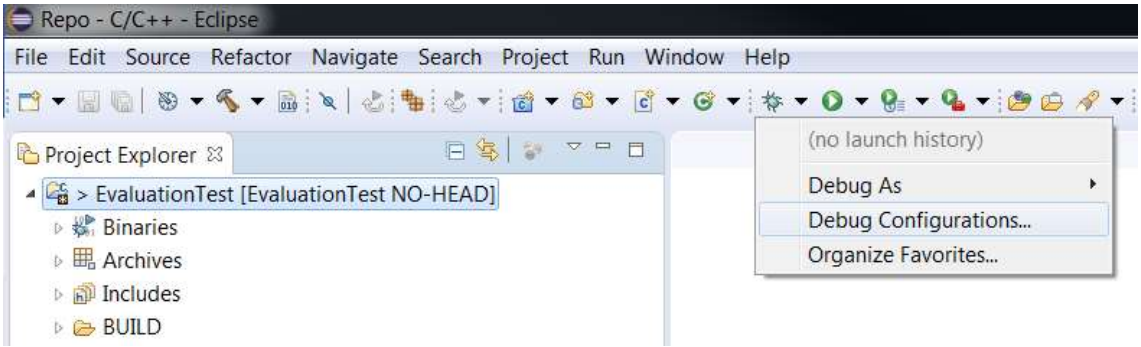
2.3 Virheenjäljittäjä

Open On-Chip Debugger (OpenOCD) on ilmainen virheenjäljittäjä ja ohjelmoija ARM-prosessoreille. OpenOCD toimii GNU Debuggerin päällä ja toimii JTAG-/SWD-rajapinnassa laitteiden kanssa (12).

OpenOCD voidaan integroida helposti Eclipseen lisäosalla. Integrointi vaatii GNU MCU Eclipse OpenOCD -lisäosan, GDB-virheenjäljittäjän (joka tulee osana GNU Arm Embedded Toolchainia), OpenOCD-asennuksen sekä ajurit itse virheenjäljittäjäsovittimille. (13).

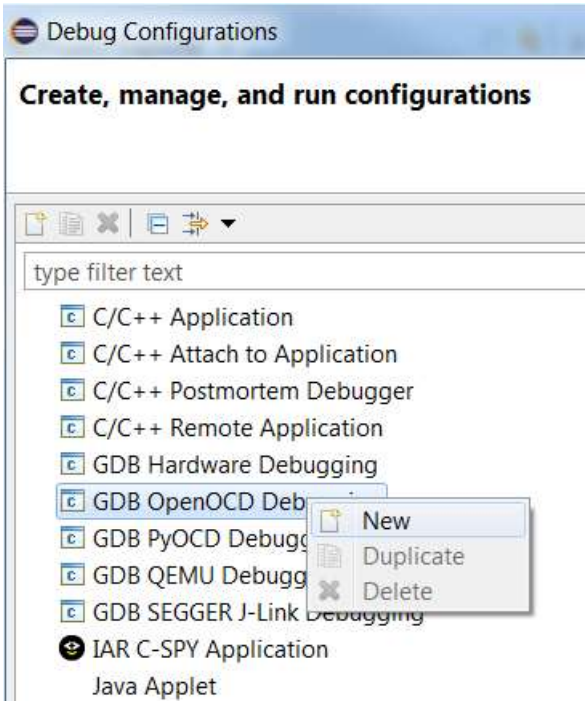
OpenOCD:n käyttö Eclipsen projektissa tapahtuu tekemällä sille virheenjäljityskonfiguraatio. Virheenjäljityskonfiguraatiolla kerrotaan, mitä ohjelmaa käytetään virheenjäljityssession käynnistämiseen, mikä on kohdealusta ja mitä ohjelmaa tutkitaan.

Valitaan virheenjäljityskuvakkeen alavalikko auki ja sieltä valitaan Debug Configurations… (kuva 1).



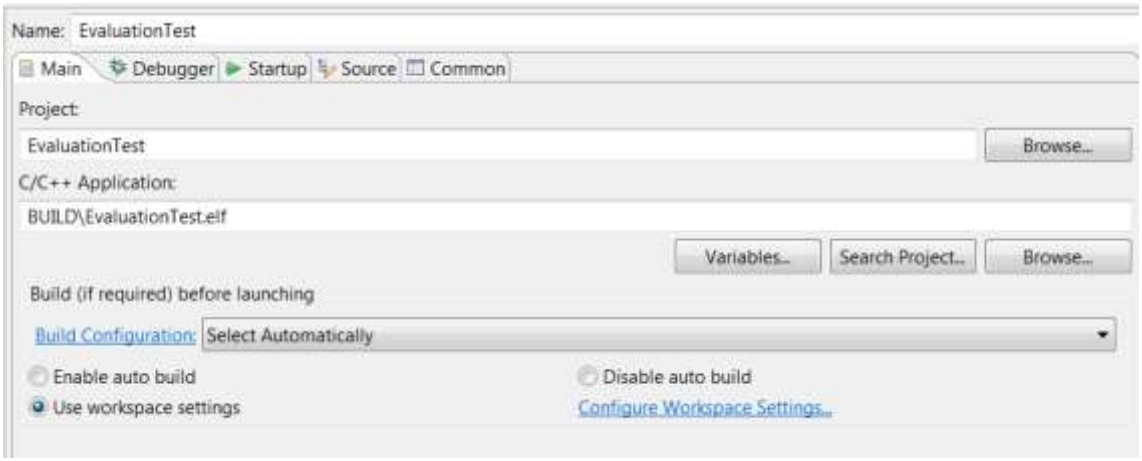
KUVA 1. Eclipsen Debug Configurations…-valikko

Sitten valitaan GDB OpenOCD Debugging ja oikealla hiirenklikkauksella saadaan valikko, josta valitaan New (kuva 2).

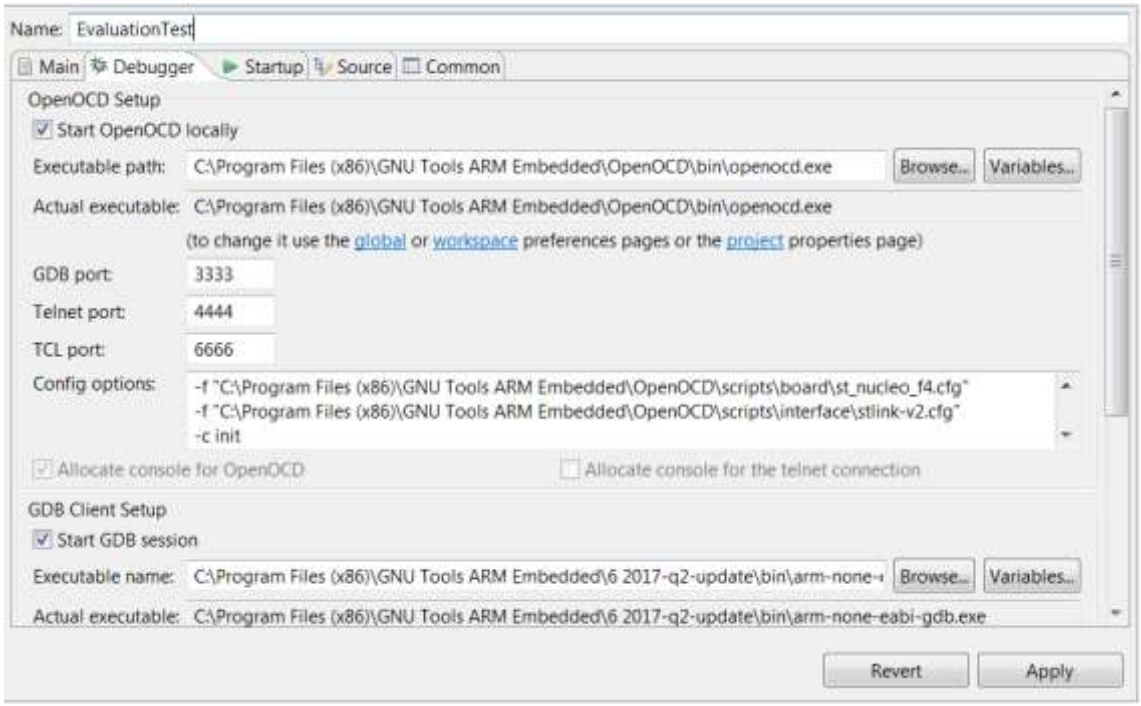


KUVA 2. Uuden OpenOCD Debugging -konfiguraation luonti

Eclipsen OpenOCD -lisäosa generoi tarvittavat tiedot, kuten mitä ohjelmaa virheenjäljitetään (kuva 3), mistä löytyy OpenOCD:n GDB-serveri, miten se pitää konfiguroida ja mistä löytyy ARM GNU Embedded Toolchainin GDB client (kuva 4).

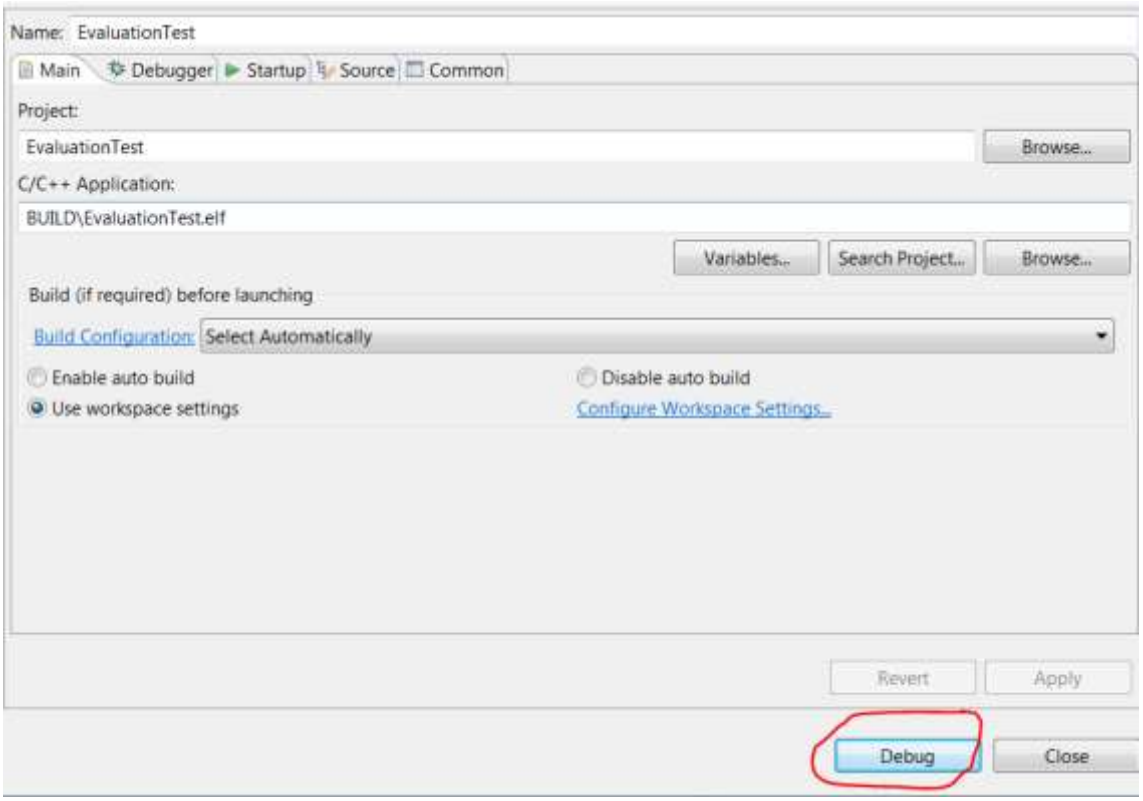


KUVA 3. Luotu OpenOCD:n virheenjäljityskonfiguraatio

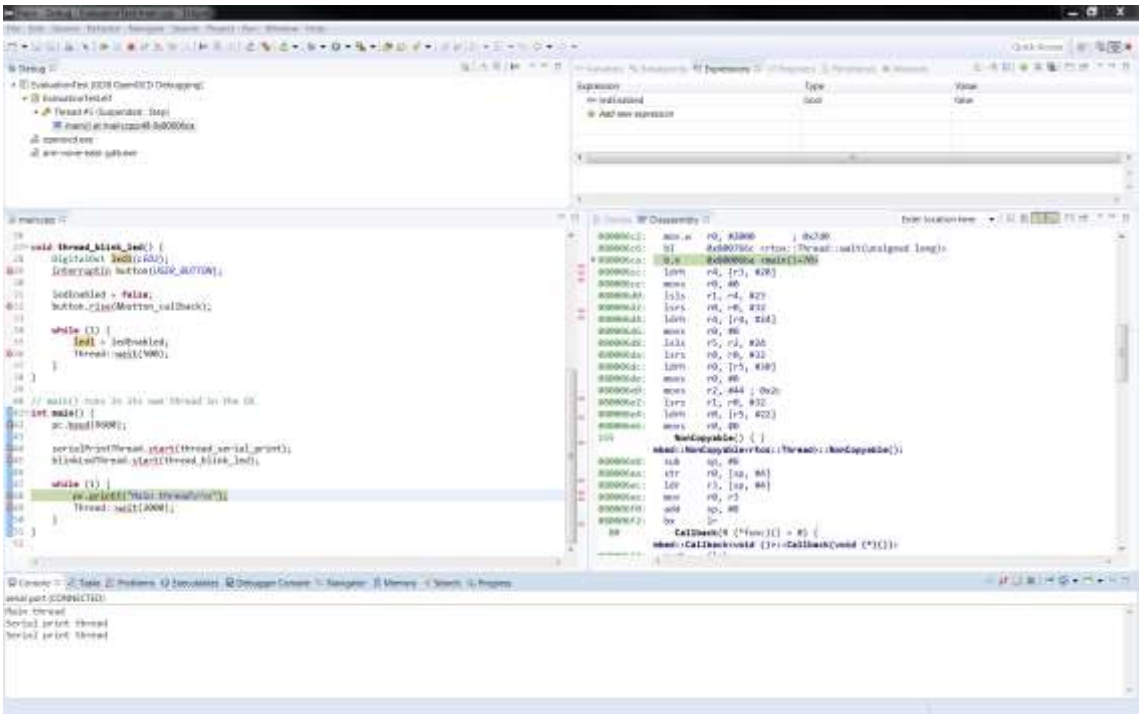


KUVA 4. OpenOCD:n virheenjäljityskonfiguraation konfigurointi

Virheenjäljitys käynnistetään painamalla Debug-nappia (kuva 5), jolla saadaan esille virheenjäljitysnäkymä (kuva 6).

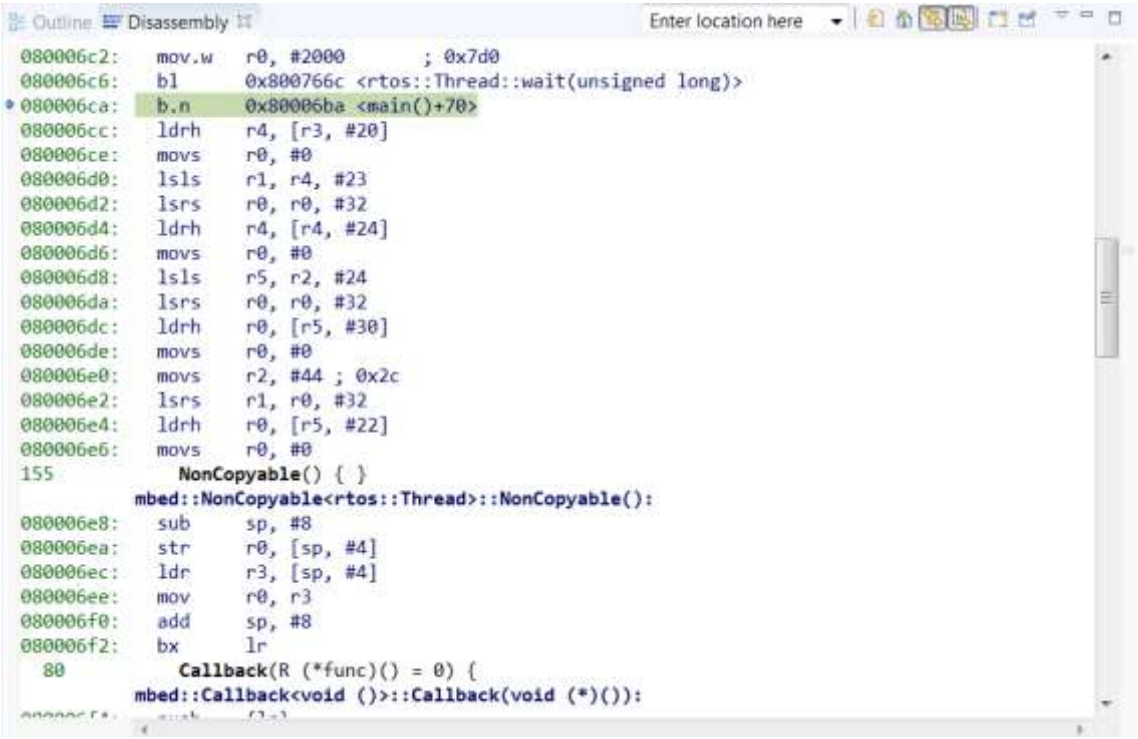


KUVA 5. OpenOCD:n virheenjäljityskonfiguraation käynnistys

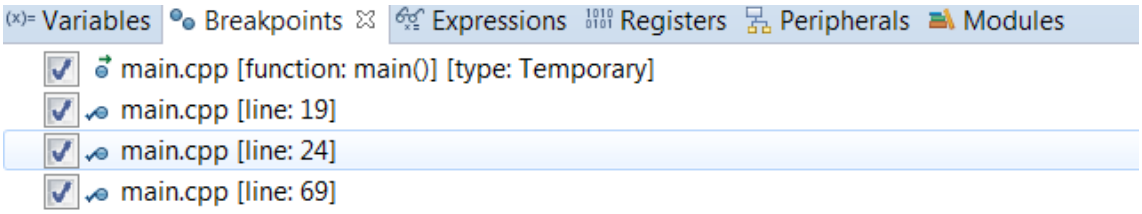


KUVA 6. Virheenjäljitysnäkymä OpenOCD:lla

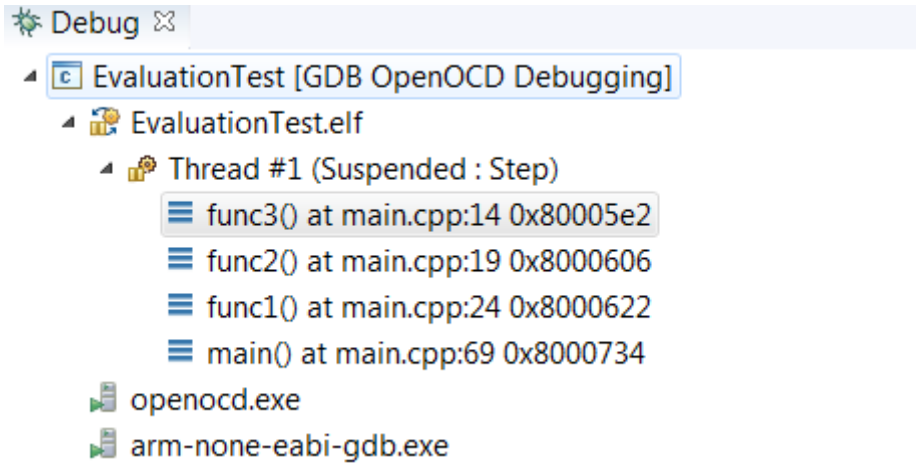
OpenOCD:lla saadaan näkyviin koodin purku virheenjäljitettävästä ohjelmasta (kuva 7), keskeytyskohdat (kuva 8), stack trace (kuva 9), muuttujien seuranta (kuva 10) sekä koodiobjektin sisällön tutkiminen (kuva 11).



KUVA 7. Koodin purku OpenOCD:lla

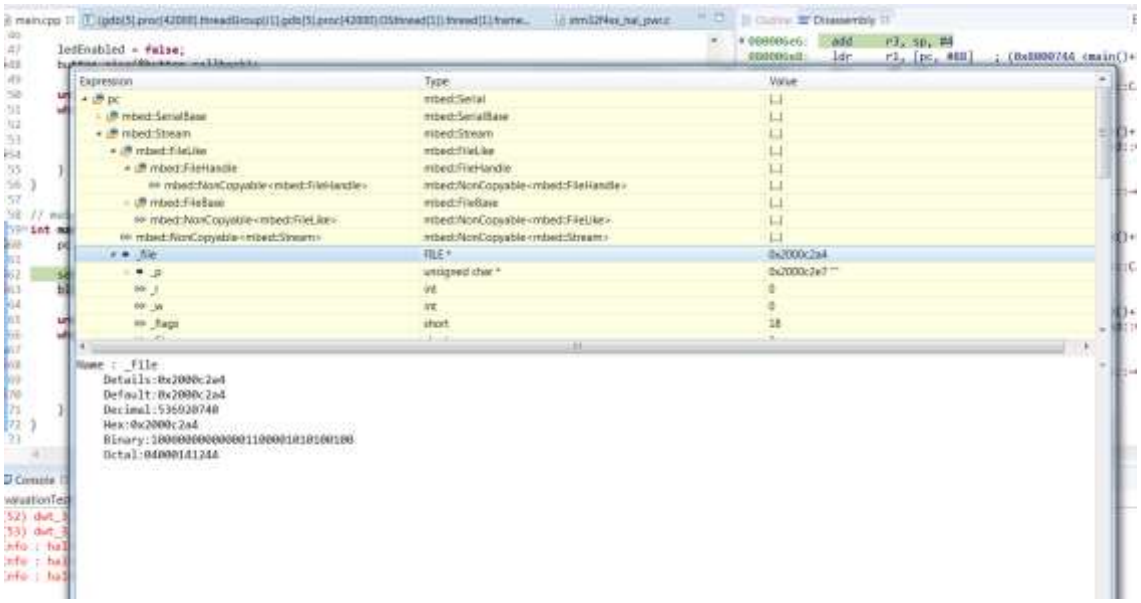


KUVA 8. Keskeytyskohdat OpenOCD:lla



KUVA 9. Stack trace OpenOCD:lla

KUVA 10. Muuttujien seuranta OpenOCD:lla.



KUVA 11. OpenOCD:lla koodiobjektin sisällön tutkiminen

2.4 Lisenssit

Eclipsen käyttö on Eclipse Public License -lisenssin alla. Sen tarkoitus on olla ilmainen ja yritysympäristöystävällinen. Sen käyttäjä saa käyttää, muuttaa, kopioida ja jakaa sen alla olevia ohjelmia ja niiden muutoksia. Joissain tapauksissa lisenssin käyttäjän täytyy julkistaa tehdyt muutokset. (14.) Lisenssin alla olevaa ohjelmistoa ei voida pitää tavaramerkin alla. Tästä ei kuitenkaan ole haittaa, kun siitä pohjautuva ohjelmisto voidaan ali-lisenssöidä ja ohjelmistokehityksessä Eclipsen lähdekoodista ei tehdä uutta ohjelmaa, vaan siitä käännettyä ohjelmaa käytetään uuden ohjelmiston tekemiseen.

GCC on GNU General Public License version 3 -lisenssin alla (15). Lisenssi antaa mahdollisuuden käyttää ohjelmistoa, muokata ja jakaa sitä vapaasti (16).

Clang on BSD-lisenssin alla (17). Lisenssi antaa mahdollisuuden käyttää ohjelmistoa, muokata ja jakaa sitä vapaasti (18).

OpenOCD on GNU General Public License -lisenssin alla (19) ja antaa näin samat mahdollisuudet ja rajoitukset kuin GCC:n lisenssi.

Näillä tiedoilla voidaan todeta, että tässä kappaleessa esiteltyjä vapaasti levitettäviä työkaluja voidaan käyttää vapaasti ja huoletta niin kauan, kunnes tulee tarve muokata jotain ohjelmistoa RD Velhon käyttöön sopivaksi.

3 ARM:N TYÖKALUT

3.1 IDE

ARM:n tarjoama IDE on nimeltään uVision, joka kuuluu kehitysympäristöpakettiin nimeltä MDK-ARM (20).

uVision on hieman karheampi ja monimutkaisempi kuin Eclipse, mutta sen käyttö on silti suhteellisen luontevaa ja helppoa. Ohjelma voidaan kääntää projektin valikosta tai build-pikanäppäimellä. Ohjelma voidaan ladata laitteelle myös helposti Flash-valikosta tai pikanäppäimellä. Virheenjäljitys aloitetaan Debug-valikosta tai myös yhdellä pikanäppäimellä.

uVisionissa on mukana toiminnallisuudet kuten automaattinen täydennys ja koodiobjektin määritykseen hyppääminen. Siinä ei kuitenkaan ole mukana refaktorointiin tarkoitettuja ominaisuuksia, kuten koodiobjektin uudelleennimeäminen.

Pack installer -työkalun avulla voidaan asentaa uVisioniin tuki tarvittaville prosessoreille tai ohjelmistokomponenteille (kuten reaaliaikakäyttöjärjestelmille), sekä sen avulla voidaan ladata esimerkkejä, kuinka käyttää joko prosessorin tai IDE:n toimintoja.

Manage run-time environment -komponentin avulla voidaan antaa esimerkiksi virheenjäljitykseen tuki jollekin reaaliaikajärjestelmälle tai jäljitykseen jotakin virheenjäljittäjää käyttämällä.

3.2 Kääntäjä

Kääntäjänä toimii ARM:n oma kääntäjä *ARM Compiler*. ARM Compiler tulee ARM MDK:n mukana ja toimii lisenssin alaisena. Kääntäjää voidaan käyttää komentoriviltä sellaisenaan tai helpommin uVisionin kanssa, jolloin uVision käsittelee ja käskyttää kääntäjää automaattisesti projektin konfiguraatioiden mukaisesti.

ARM Compiler v5 tukee C-kieltä C99-standardin mukaisesti (21) ja C++-kieltä C+11-standardin mukaisesti suurimmalta osin (22).

3.3 Virheenjäljittäjä

ARM:n uVisionilla ja ULink Pro -virheenjäljittäjällä voidaan saada stack trace vaikka ohjelman alusta niin pitkälle kuin halutaan (kunhan tietokoneella on tarpeeksi muistia lokitiedostolle), voidaan muuttaa muuttujien arvoa lennosta, saada tietoa threadeistä ja niiden ajoituksista sekä tehdä kaikki se, mitä vapaasti levitettävillä työkaluilla voidaan tehdä: asettaa keskeytyskohtia (kuva 12), tutkia koodiobjektin sisältö (kuva 13), tehdä koodin purku (kuva 14) ja nähdä stack trace (kuva 15).

Virheenjäljitys konfiguroidaan menemällä projektin asetuksiin. Projektin päältä voidaan painaa hiiren oikealla ja valita Options for Target… -valinta (kuva 16).

```
54          Thread::wait(500);
55      }
56  }
57
58  // main() runs in its own thread in the OS
59  int main() {
60      pc.baud(9600);
61
62      serialPrintThread.start(thread_serial_print);
63      blinkLedThread.start(thread_blink_led);
64
65      unsigned int counterMainThread = 0;
66      while (1) {
67          counterMainThread++;
68          pc.printf("Main thread\r\n");
69          func1();
70          Thread::wait(2000);
```

KUVA 12. Keskeytyskohta uVisionilla

Watch 1

Name	Value	Type
counterMainThread	0x00000002	unsigned int
pc	0x20000B18 &pc	struct Serial
SerialBase	0x20000B18 &pc	struct SerialBase
Stream	0x20000BD4	struct Stream
FileLike	0x20000BD4	struct FileLike
FileHandle	0x20000BD4	struct FileHandle
FileBase	0x20000BD8	struct FileBase
NonCopyable	0x20000BD4	NonCopyable<struct FileLike>

KUVA 13. Koodiobjektien sisällön tutkiminen uVisionilla

Disassembly

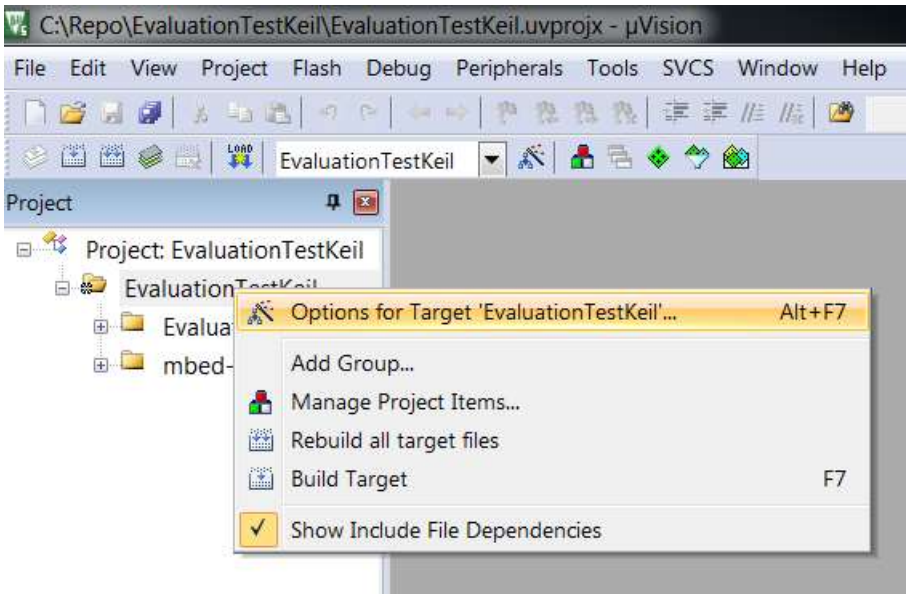
```
    65:     unsigned int counterMainThread = 0;
0x08007568 2400      MOVS          r4,#0x00
    66:     while (1) {
0x0800756A E00B      B             0x08007584
    67:         counterMainThread++;
0x0800756C 1C64      ADDS          r4,r4,#1
    68:         pc.printf("Main thread\r\n");
0x0800756E A10B      ADR           r1,{pc}+0x30  ; @0x0800759C
0x08007570 4805      LDR           r0,[pc,#20]  ; @0x08007588
0x08007572 30BC      ADDS          r0,r0,#0xBC
0x08007574 F7FEF8AF  BL.W          Stream::printf (0x080056D6)
    69:         func1();
0x08007578 F7FDFD92  BL.W          func1 (0x080050A0)
    70:         Thread::wait(2000);
0x0800757C F44F60FA  MOV           r0,#0x7D0
0x08007580 F7FEFB2C  BL.W          wait (0x08005BDC)
    66:     while (1) {
0x08007584 E7F2      B             0x0800756C
0x08007586 0000      DCW           0x0000
```

KUVA 14. Koodin purku uVisionilla

Call Stack + Locals

Name	Location/Value	Type
osRtxTimerThread : 0x200004A8	0x0800861D	Task
osRtxIdleThread : 0x20000460	0x08007C55	Task

KUVA 15. Stack trace uVisionilla



KUVA 16. Projektin asetuksien löytäminen uVisionilla

uVisionilla saatiin stack trace näkyviin, mutta se ei näyttänyt kuin mbed-os:n Timer- ja Idle-threadit (joita ei edes voitu avata), eikä nykyisen kontekstin stack trace:a. Näin ollen mbed-os:n kanssa stack tracen tutkiminen ei näytä mahdolliselta.

RTX5-reaaliaikakäyttöjärjestelmälle ei valitettavasti ole tukea uusimmassa uVisionin mbed-lisäosassa sekä mbed-os:n käyttämä RTX5-versio on muokkaus alkuperäisestä ja tätä muokattua versiota MDK-ARM -työkalut eivät tue. Näin ollen mbed-os:n threadauksen profilointi ei onnistu tässä evaluoinnissa.

3.4 Lisenssi

Tarjottavana ja todennäköisesti käytettävänä MDK-ARM-versiona olisi Essential Edition ja vielä kelluvalla lisenssillä. Tähän kuuluisi mukaan ARM C/C++ Compilation toolchain, uVision5 IDE/Debugger/Simulator, Device support: Cortex-M, RTX RTOS source code, Floating license sekä 12 kuukauden huolto sisältäen tekniset päivitykset.

Tarjottavina virheenjäljittijänä olisivat ULINK Pro sekä ULINK2. ULINK Pro tukee JTAG-virheenjäljitystä USB:n kautta (50 MHz), flashin ohjelmointia (32 kt/s), data tracea (100 Mt/s) ja instruction tracea (800 Mbit/s). ULINK2 tukee

JTAG-virheenjäljitystä USB:n kautta (10 MHz), flashin ohjelmointia (25 kt/s) ja data trace:a (1 Mt/s). Näiden kahden virheenjäljittäjän välillä on siis erona ULINK2:n kanssa olematon tuki instruction trace:lle sekä kaikkien toimintojen pienempi tehokkuus.

4 IAR:N TYÖKALUT

4.1 IDE

IAR:n tarjoaman IDE:n nimi on IAR Embedded Workbench. Tällä IDE:llä on mahdollista luoda, kehittää ja virheenjäljittää ohjelmia.

IDE:llä on pitkä historia ja sitä mainostetaankin työkaluna, joka on pysynyt samanlaisena monien vuosien jälkeen. Tästä syystä IDE näyttää ja käyttäytyy vanhanmallisesti – moderniin koodin kirjoitukseen se ei sovellu. IDE:ssä on kuitenkin mukana ns. automaattinen täydennys ja mahdollisuus hypätä funktion tai muuttujan määrittelyyn. Siinä ei ole mukana kuitenkaan refaktorointiin toiminnallisuuksia tai Eclipsen tapaista mahdollisuutta nähdä funktion sisältö viemällä kursori sen päälle.

Virheenjäljitys tapahtuu luontevasti ja asetuksien konfiguroimisen jälkeen toiminta on helppoa. Virheenjäljityksen aloitettua IDE muuntautuu erilaiseksi myötäytymään virheenjäljitykseen. Esille tulevat ikkunat keskeytyskohdille, stack trace:lle ja muille hyödylliselle näkymälle. Lisää näkymiä, kuten jäljitykset ja ajoitukset, saadaan virheenjäljittäjän alavalikosta.

4.2 Kääntäjä

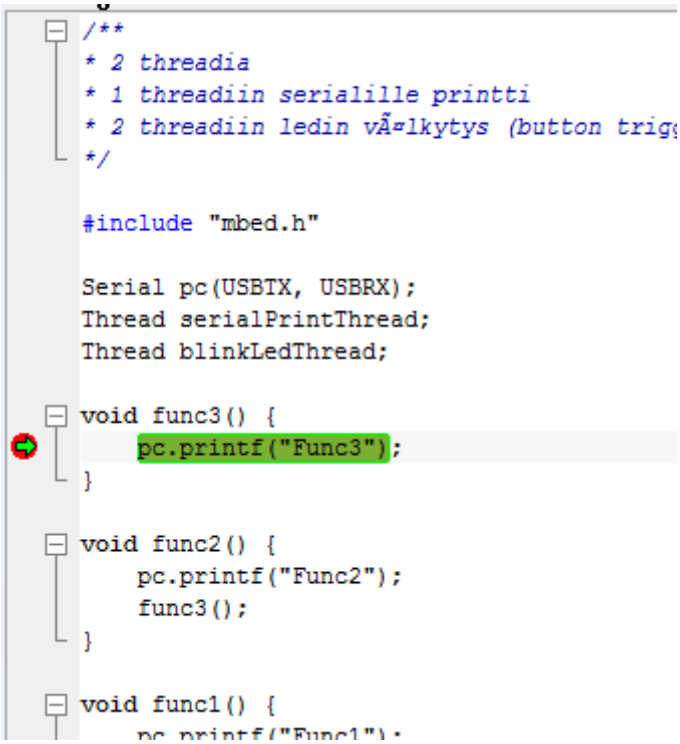
IAR:n kääntäjä tukee kääntäjän konfiguroinnin mahdollisuuksien perusteella C11- ja C++14-standardia. Kääntäjän konfigurointi tapahtuu IDE:n kautta. Esimerkiksi voidaan konfiguroida optimoinnin taso GCC:n tapaan, kertoa, onko char signed vai unsigned, sekä määritellä symboleita.

Kääntäjä tulee IAR Embedded Workbenchin asennuksen mukana ja toimii lisenssin alla; käännöksessä otetaan lisenssi käyttöön ja käännöksen valmistuttua lisenssi vapautetaan.

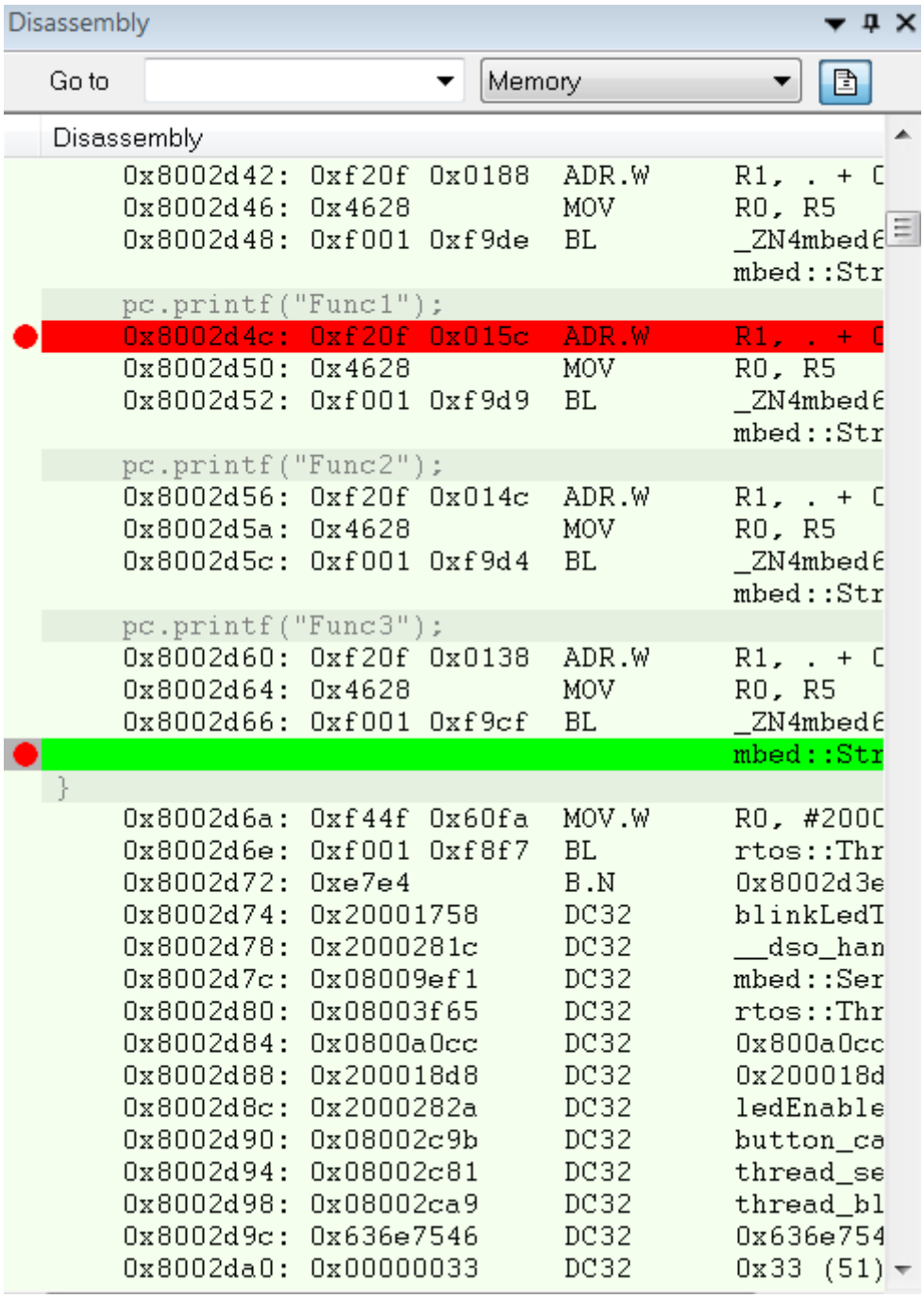
4.3 Virheenjäljittäjä

IAR tarjoaa moninaisia virheenjäljittäjiä, joilla voidaan ohjelmoida kohteita ja virheenjäljittää niitä. Lisäksi niillä voidaan saada trace-dataa. IAR Embedded

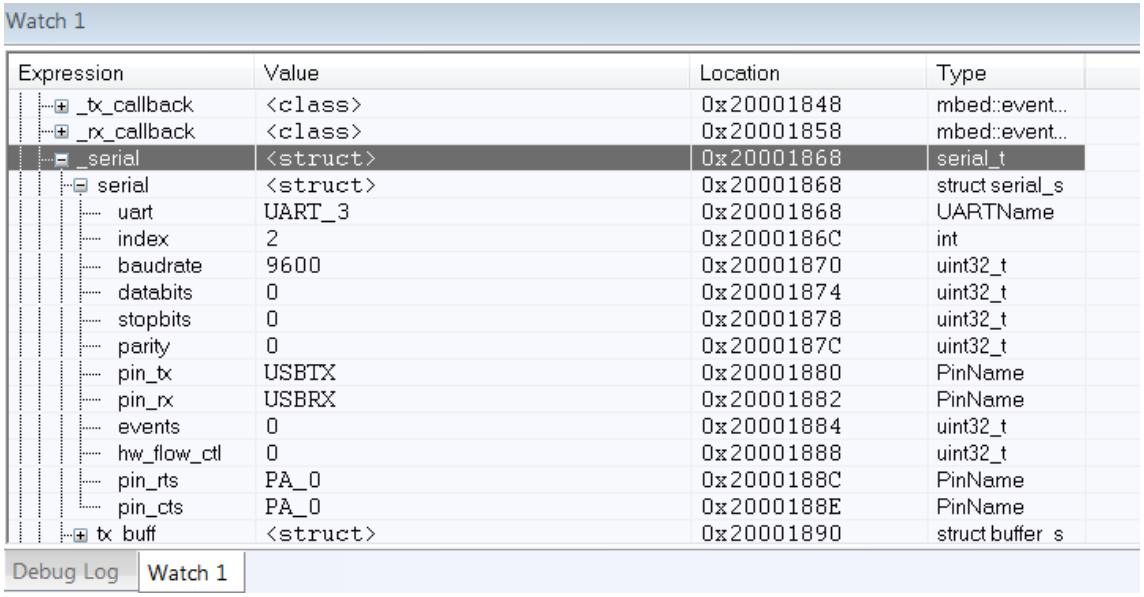
Workbenchin avulla voidaan virheenjäljityksessä asettaa keskeytyskohtia (kuva 17), purkaa koodi (kuva 18), seurata muuttujia, tutkia koodiobjektin sisältö (kuva 19) ja nähdä stack trace (kuva 20). Näin ollen näillä virheenjäljittäjillä voidaan tehdä myös kaikki se, mitä vapaasti levitettävillä työkaluilla voidaan tehdä, ja vielä enemmän.



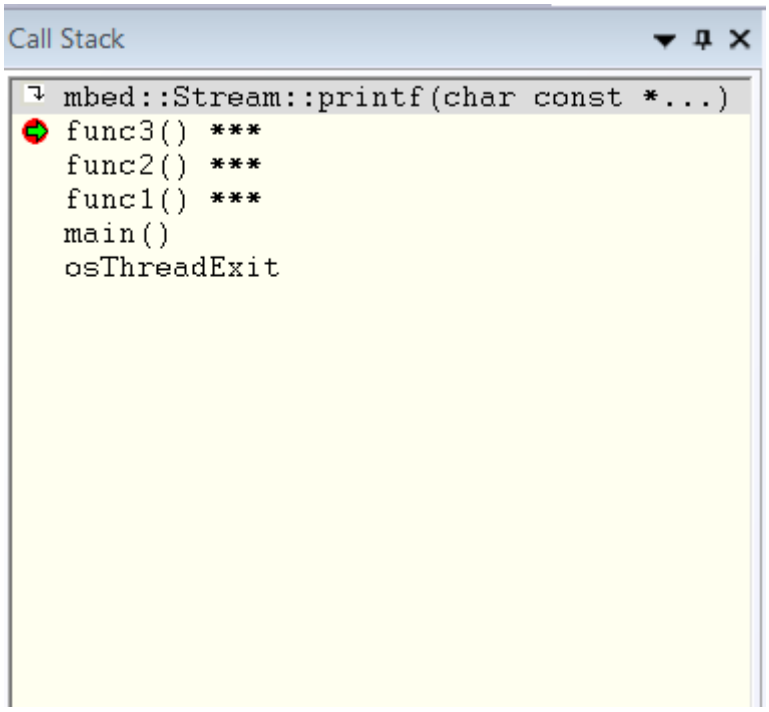
KUVA 17. IAR Embedded Workbenchin keskeytyskohtien käyttäminen



KUVA 18. IAR Embedded Workbenchillä koodin purku



KUVA 19. IAR Embedded Workbenchin koodiobjektin sisällön tutkiminen



KUVA 20. IAR Embedded Workbenchin stack trace

Evaluointiin ja yleiseen käyttöön sopivaksi on ottaa tarkasteluun I-Jet Trace CM L- ja XL-mallit. Näillä malleilla voidaan virheenjäljittää ja ohjelmoida Cortex-M-laitteita, muttei sitä vaativampia. Koska tarkoituksena onkin hankkia työkaluja Cortex-M-laitteille, muita I-Jetejä ei tarvitse ottaa tarkasteluun. Ero L- ja XL-

mallien välillä on niiden sisäinen muisti; L-mallissa on 64 Mt ja XL-mallissa 256 Mt.

Valitettavasti IAR:n tuki RTX5-reaaliaikakäyttöjärjestelmälle oli vanhentunut mbed-lisäosassa ja näin evaluointia mbed-os:n thredaukselle ei voitu tehdä. Tämän korvaava päivitys tulisi versioon 8.5.1.

4.4 Lisenssi

IAR mahdollistaa ns. Network-lisenssin käytön, jolla voi olla yksi lisensoitu käyttäjä useamman yrityksen toimipisteen kesken. Tätä lisensoitua käyttäjää käytetään vain, kun halutaan joko ajaa IAR Embedded Workbenchin kääntäjää, virheenjäljittäjää tai C-RUN/C-STAT-työkaluja.

Volume License Programin avulla voidaan monen lisenssin hankkimisesta saada alennusta. Arvioitu määrä tarvittaville lisensseille olisi noin viisi. Viisi olisikin juuri se alaraja ensimmäiselle asteelle. Vaikka tältä ensimmäiseltä asteelta ei saakaan alennusta, seuraavista lisensseistä saataisiin 5%:n alennus.

5 ARM:N, IAR:N JA VAPAASTI LEVITETTÄVIEN OHJELMISTOJEN VERTAILU

5.1 IDE:t

Vapaasti levitettävien työkalujen IDE:issä keskipiste on ollut selvästi koodin kirjoittamisessa eikä niinkään virheenjäljitys- tai kääntämisympäristöissä. Eclipsen ominaisuudet, kuten funktion sisällön katsominen kursorilla tai refaktorointi, eivät ole uVisionissa tai IAR Embedded Workbenchissa tuettuja. Niissä on kuitenkin mukana automaattinen täydennys ja koodiobjektien määrityksiin hyppääminen. Kääntöpuolena vapaasti levitettävien työkalujen IDE:issä ei ole kunnollista virheenjäljitys- tai kääntämistukea, koska niiden pitäisi tukea monia eri alustoja ja prosessoriarkkitehtuureja, joiden tukemiseen ei ole todennäköisesti vapaita ohjeita tai työkaluja.

5.2 Kääntäjät

IAR:n ja ARM:n työkaluissa kääntäjän asettaminen ja konfigurointi on optimoitua ja suuresti konfiguroitavissa. Lisäksi niiden asentaminen ja käyttöönotto tapahtuu IDE:iden asennuksen yhteydessä.

Vapaasti levitettävien työkalujen IDE:issä tarvitaan kääntäjien ja kääntämiseen liittyvien työkalujen erillistä asentamista ja konfiguroimista. Myös näiden työkalujen ylläpitäminen täytyy tehdä erilleen.

Kääntäjällä tuotettavaa ajettavaa ohjelmaa voidaan vertailla kahdella tavalla: käännetyn ohjelman koolla ja käännetyn ohjelman suoritusnopeudella. Mitä pienempi kooltaan käännetty ohjelma on ja mitä nopeampi ohjelma suoritetaan, sitä parempi.

Vertailuohjelmaksi tehtiin ohjelma, joka ajaa Whetstone-aliohjelman sekä tulostaa sarjaporttiin aliohjelman suoriutumisen keston millisekunteina. Ohjelman alusta generoitiin STM:n CubeMX-ohjelmalla. Lähdekoodi ja kohdealusta GCC:lle sekä ARM:n ja IAR:n työkaluille on sama. Näin saadaan eliminoitua koodin ja alustan tuomat erilaisuudet ja voidaan keskittyä kääntäjiin

ja kääntämisen työkalujen eroihin. Ohjelma ajettiin STM32F437V6T6-mikrokontrollerilla, joka ajoi ohjelmaa 16 MHz:n nopeudella, ja työkalujen tuloksia verrattiin (taulukko 1).

TAULUKKO 1. Vertailuohjelman tulokset eri kääntäjillä

Ohjelman optimisointitaso	**GCC**	**IAR**	**ARM**
O0	Kesto (ms): 115 938 Koko (kt): 28	Kesto (ms): 97 461 Koko (kt): 22	Kesto (ms): 253 716 Koko (kt): 21
O1	Kesto (ms): 104 867 Koko (kt): 25	Kesto (ms): 96 682 Koko (kt): 22	Kesto (ms): 247 817 Koko (kt): 19
O2	Kesto (ms): 76 998 Koko (kt): 25	Kesto (ms): 94 217 Koko (kt): 21	Kesto (ms): 247 287 Koko (kt): 18
O3	Kesto (ms): 77 110 Koko (kt): 26	Kesto (ms): 82 814 Koko (kt): 20	Kesto (ms): 246 856 Koko (kt): 18

Huomataan, että IAR:n työkalut tuottavat ohjelman kokoon nähden parhaimman tuloksen. GCC:llä saadaan generoitua nopeampaa ohjelmaa, mutta ohjelma vie myös eniten muistia. ARM:n työkaluilla tuotettu ohjelma vie vähiten muistia, mutta tuotetun ohjelman ajaminen vie kaikista eniten.

5.3 Virheenjäljittäjät

Vapaasti levitettävillä työkaluilla voidaan tietää yleisten rekisterien tilat, purkaa koodi, seurata muuttujia tai koodiobjekteja, käyttää keskeytyskohtia, lukea stack trace ja ohjelmoida myös kohdelaite. Näillä työkaluilla voidaan siis tehdä kaikki välttämätön, mutta niillä ei voida tutkia threadien tiloja tai nähdä ajoituksia.

ARM:n työkaluilla voidaan tehdä samat jutut kuin vapaasti levitettävillä työkaluilla, mutta lisänä on data trace ja ULINK Prolla instruction trace. Näillä saataisiin tarkempaa tietoa prosessorin toiminnasta. Threadien tiloja ja ajoituksia ei voitu tarkastella, koska ne eivät olleet tuettuja evaluoinnin aikana, joten niiden toimintaa ei voida kommentoida.

IAR:n työkaluilla voidaan myös tehdä samat, kuin vapaasti levitettävillä työkaluilla. Lisänä on instruction trace ja mukana pitäisi olla threadien tilojen ja ajoituksien tutkiminen. Threadien tilojen ja ajoituksien tutkiminen eivät kuitenkaan olleet tuettuna evaluoinnin aikana, joten niiden toimintaa ei voida kommentoida.

5.4 Lisenssit

Tutkaillut vapaasti levitettävät työkalut ovat sellaisten lisenssin alla, että niitä voidaan käyttää ilmaiseksi ja tarvittaessa muokata halutunlaiseksi. Niiden käyttöä ei ole esimerkiksi rajoitettu siten, että vain tietty määrä käyttäjiä saa niitä käyttää, tai niin, että niitä saa käyttää vain tietyn ajan.

6 YHTEENVETO

Tämän työn aiheena oli tutkia, mitkä työkalut olisivat parhaimpia ja sopivimpia RD Velholle sulautettujen ohjelmistojen kehittämiseen. Tarkasteltavina asioina olivat kehitysympäristöt, kääntäjät, tuki C- ja C++-kielille, virheenjäljityksen laajuus sekä lisensointi.

Näillä tiedoilla näyttää siltä, että paras vaihtoehto on käyttää koodin kirjoittamiseen Eclipseä, koska siinä on mukana code completion, refaktorointi ja muut mukavuudet, joita ei ole mukana muissa työkaluissa.

Kääntämiseen paras vaihtoehto olisi käyttää IAR:n työkaluja, koska sillä saadaan luodun ohjelman kokoon nähden paras nopeus. Jos ohjelman koko ei ole ongelma, näyttää GCC:n käyttäminen olevan paras vaihtoehto.

Virheenjäljitykseen IAR ja ARM ovat aika lähellä toisiaan. ARM:n työkaluilla mbed-os:n (luultavin tulevaisuudessa käytettävä reaaliaikakäyttöjärjestelmä) threadien profilointi on hankalaa konfiguraatiotiedoston tekemisen takia. IAR:n työkaluilla mbed-os:n threadien profilointi ei ollut edes mahdollista, mutta jos asiat toimivat, kuten niiden pitää uusimmassa IAR Embedded Workbenchin versiossa, se tulisi olemaan paras vaihtoehto virheenjäljitykselle. Kaikilla työkaluilla voidaan tehdä vähimmäinen tarvittava (stack trace, keskeytyskohtien asettaminen, muuttujien seuranta, objektien sisältöjen tutkiminen ja koodin purku).

ARM:ilta on tulossa uusi työkaluohjelmisto Mbed Studio mbed-os:ää varten, jonka pitäisi ratkaista kyseiset ongelmat (23). Olisi siis parasta odottaa, kunnes Mbed Studio julkaistaan, ja ottaa uusi evaluointikierros, jossa se olisi mukana.

LÄHTEET

1. Wheeler, David A. 2015. Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers! Saatavissa: https://www.dwheeler.com/oss_fs_why.html. Hakupäivä 9.3.2018.
2. Code Composer Studio (CCS) Integrated Development Environment (IDE). Texas Instruments. Saatavissa: http://www.ti.com/tool/CCSTUDIO. Hakupäivä 9.3.2018.
3. HomePage. OpenSTM32 Community. Saatavissa: http://www.openstm32.org/HomePage. Hakupäivä 9.3.2018.
4. Ridge, Nathan 2017. C++ Language Support in Eclipse CDT. Eclipse. Saatavissa: https://www.eclipse.org/community/eclipse_newsletter/2017/april/article3.php. Hakupäivä 9.3.2018.
5. C++ Standards Support in GCC. GCC, the GNU Compiler Collection. Saatavissa: https://gcc.gnu.org/projects/cxx-status.html. Hakupäivä 9.3.2018.
6. GNU Arm Embedded Toolchain. Arm Developer. Saatavissa: https://developer.arm.com/open-source/gnu-toolchain/gnu-rm. Hakupäivä 9.3.2018.
7. AVR and ARM Toolchains (C Compilers). Microchip. Saatavissa: https://www.microchip.com/avr-support/avr-and-arm-toolchains-%28c-compilers%29. Hakupäivä 9.3.2018.
8. Clang: a C language family frontend for LLVM. The LLVM Compiler Infrastructure. Saatavissa: https://clang.llvm.org/. Hakupäivä 9.3.2018.
9. Language Compability. The LLVM Compiler Infrastructure. Saatavissa: https://clang.llvm.org/compatibility.html. Hakupäivä 9.3.2018.
10. C++ Support in Clang. The LLVM Compiler Infrastructure. Saatavissa: https://clang.llvm.org/cxx_status.html. Hakupäivä 9.3.2018.
11. Cross-compilation using Clang. Clang 7 documentation. Saatavissa: https://clang.llvm.org/docs/CrossCompilation.html. Hakupäivä 9.3.2018.
12. About. OpenOCD. Saatavissa: http://openocd.org/doc-release/html/About.html#What-is-OpenOCD_003f. Hakupäivä 16.3.2018.

13. The OpenOCD debugging Eclipse plug-in. 2018. GNU MCU Eclipse. Saatavissa: https://gnu-mcu-eclipse.github.io/debug/openocd/. Hakupäivä 16.3.2018.
14. Eclipse Public License 1.0 (EPL-1.0). tl;drLegal. Saatavissa: https://tldrlegal.com/license/eclipse-public-license-1.0-%28epl-1.0%29. Hakupäivä 16.3.2018.
15. License. GNU Compiler Collection. Saatavissa: https://gcc.gnu.org/onlinedocs/libstdc++/manual/license.html. Hakupäivä 16.3.2018.
16. GNU General Public License v3 (GPL-3). tl;drLegal. Saatavissa: https://tldrlegal.com/license/gnu-general-public-license-v3-%28gpl-3%29. Hakupäivä 16.3.2018.
17. Clang - Features and Goals. The LLVM Compiler Infrastructure. Saatavissa: https://clang.llvm.org/features.html#license. Hakupäivä 16.3.2018.
18. BSD 3-Clause License (Revised). tl;drLegal. Saatavissa: https://tldrlegal.com/license/bsd-3-clause-license-%28revised%29. Hakupäivä 16.3.2018.
19. The GNU MCU Eclipse tools licenses. 2017. GNU MCU Eclipse. Saatavissa: https://gnu-mcu-eclipse.github.io/licenses/tools/#openocd. Hakupäivä 16.3.2018.
20. MDK-ARM Microcontroller Development Kit. Arm ARM. Saatavissa: http://www.keil.com/products/arm/mdk.asp. Hakupäivä 5.4.2018.
21. The compiler. Arm ARM. Saatavissa: http://www.keil.com/support/man/docs/armcc/armcc_chr1359124192377.htm. Hakupäivä 16.3.2018.
22. C++11 supported features. Arm ARM. Saatavissa: http://www.keil.com/support/man/docs/armcc/armcc_chr1407404265784.htm. Hakupäivä 16.3.2018.
23. Alderson, Joe 2018. Introducing Mbed Studio. Arm Mbed. Saatavissa: https://os.mbed.com/blog/entry/Introducing-Mbed-Studio/. Hakupäivä 16.3.2018