Antti Vähälummukka

Sulautettujen laitteiden laitteisto- ja ohjelmistoturvallisuus

Tietotekniikan
pro gradu -tutkielma
16. maaliskuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Kokkolan yliopistokeskus Chydenius

Tekijä: Antti Vähälummukka
Yhteystiedot: vahisa@gmail.com
Puhelinnumero: 040 540 7727
Ohjaaja: Risto T. Honkanen ja Ismo Hakala
Työn nimi: Sulautettujen laitteiden laitteisto- ja ohjelmistoturvallisuus
Title in English: Hardware and Software Security in Embedded devices
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 56 + 11
Tiivistelmä: Tässä työssä selvitetään millaisia laitteisto- ja ohjelmistoturvallisuutta lisääviä ominaisuuksia on käytettävissä nykyaikaisissa sulautettujen laitteiden käyttämissä mikro-ohjaimissa, sekä miten niiden avulla voidaan suojata sulautetun laitteen ohjelmisto ja sen ohjelma- ja työmuistissa olevat suojaamista kaipaavat tiedot. Työssä tutustutaan aiheeseen kirjallisuuskatsauksen sekä valmistajien verkkosivustojen, datalehtien ja ohjelmointioppaiden avulla. Työn tietojen perusteella nykyaikaisissa mikro-ohjaimissa on hyvin paljon turvaominaisuuksia. Ohjelmistokehittäjien tulisikin olla hyvin perillä käyttämänsä mikro-ohjaimen ominaisuuksista, sillä niiden käyttäminen helpottaa turvaominaisuuksien toteuttamista huomattavasti.
Avainsanat: Suojattu käynnistys, suojattu varusohjelman päivitys, tietoturva, mikro-ohjain, mikrokontrolleri
Abstract: This paper investigates what kind of hardware and software security features are available on modern microcontrollers used in embedded systems and how they can be used to secure the firmware and the data stored in FLASH or RAM memory. This paper contains a literature review of the area with some further insight derived from microcontroller manufacturers' web sites, data sheets, reference manuals, and application notes. Modern microcontrollers contain a plethora of security features. It is very important for programmers to recognize the features offered by the hardware in use as the features help the implementation of device's security considerably.
Keywords: Secure boot, secure firmware update, information security, microcontroller

Sanasto

ADC	Analogi-digitaalimuunnin (Analog to Digital Converter), komponentti, jolla analoginen signaali voidaan muuntaa numeeriseksi
AES	Salakirjoitusstandardi (Advanced Encryption Standard), on symmetrinen lohkosalaustekniikka. Sitä nimitetään myös nimellä Rijndael kehittäjänsä mukaan
CMSIS	Arm Ltd:n valmistajariippumaton ohjelmistorajapinta, jota käyttämällä valmistaja- ja piirikohtaiset erot voidaan piilottaa ohjelmoijalta. (Cortex Microcontroller Software Interface Standard)
DAC	Digitaali-analogimuunnin (Digital to Analog Converter), komponentti, jolla numeerinen arvo voidaan muuntaa analogiseksi signaaliksi
DSP	Digitaalinen signaalinkäsittely (Digital Signal Processing) on tekniikka, jossa analogisia signaaleita, kuten ääntä tai radiosignaaleita, käsitellään digitaalisesti, käyttäen soveltuvia algoritmeja
DMA	Muistin suorasaanti (Direct Memory Access) on toiminta, joka siirtää tietoa itsenäisesti muistialueelta toiselle, muistialueelta oheislaitteelle tai oheislaitteelta muistiin
DSI	MIPI-allianssin määrittelemä sarjamuotoinen nopea näyttöliitäntä (Display Serial Interface).
ECC	Virheenkorjauskoodi (Error Correcting Code) on virheenkorjaukseen ja tunnistamiseen käytetty koodi, jonka avulla on mahdollista korjata pieniä virheitä ja havaita useampia virheitä
ECDSA	Elliptisten käyrien salausmenetelmiin perustuva allekirjoitusmenetelmä (Elliptic Curve Digital Signature Algorithm)
FW	Palomuuri (Firewall) STM32 ARM mikro-ohjainten yksikkö, joka muodostaa suojattuja muistialueita
GPU	Grafiikkaprosessori (Graphics Processing Unit) on prosessori, joka on optimoitu grafiikan käsittelyyn ja laskentaan

I2C	Sarjamuotoinen väylä oheislaitekomponenttien liittämiseen (Inter IC Interface), merkitään joskus myös muodossa I^2C tai IIC
IoT	Esineiden internet (Internet of Things)
IP	Immateriaalioikeudet, teollis- ja tekijänoikeudet (Intellectual Property) ovat aineetonta omaisuutta, kuten ideaa tai suunnittelutyötä, koskevia oikeuksia
JTAG	JTAG-liitäntä (Joint Test Action Group) on prosessorin ohjelmointiin ja vianselvitykseen tarkoitettu liitäntä
KMS	Avainten hallintapalvelu (Key Management Service)
MD5	Kryptografinen yksisuuntainen tiivistefunktio (Message Digest 5) on funktio, jolla voidaan laskea tiedosta tiiviste, jota voidaan käyttää tiedon oikeellisuuden tarkastamiseen
MCU	Mikro-ohjain, mikrokontrolleri (Micro Controller Unit)
MIPI	MIPI-allianssi (Mobile Industry Processor Interface)
MISRA	Autoalalla toimivien yritysten yhteenliittymä (Motor Industry Software Reliability Association)
MPU	Muistinsuojausyksikkö (Memory Protection Unit) on prosessorin suojausyksikkö, jolla voidaan suojata muistialueita eri tavoin
NMI	Keskeytystyyppi, jota ei voi ohjelmallisesti estää (Non-Maskable Interrupt). Sen avulla prosessori saadaan suorittamaan keskeytysohjelma heti, kun joku ulkoinen tapahtuma havaitaan
PCROP	STM32 ARM mikro-ohjaimissa oleva muistinsuojausyksikkö (Proprietary Code Read Out Protection), jolla muistialueita voi suojata eri tavoin
PKA	Yksityisen avaimen arkkitehtuuri (Private Key Architecture)
PSA	ARM suorittimien turvallisuusarkkitehtuuri (Platform Security Architecture) on Arm Ltd:n julkaisema parhaisiin käytäntöihin perustuva turvallisuusarkkitehtuuri
RAM	Hajasaantimuisti (Ramdom Access Memory) eli työmuisti on muistityyppi, jonka sisältöä voi sekä lukea että kirjoittaa vapaasti osoitteesta toiseen siirtyen
Reverse Engineering	Takaisinmallinnus, menetelmä, jolla laitteen toiminnallisuus, arkkitehtuuri ja teknologiat selvitetään tutkimalla laitetta.
RDP	Muistin lukusuojaus (Readout Protection)

ROM	Ohjelmamuisti (Read Only Memory), muistityyppi, jonka sisällön voi lukea mutta ei kirjoittaa
RoT	Luotettavuuden varmentava palvelu (Root of Trust), toiminto, jonka tehtävänä on toteuttaa luotettava perusta muille turvatoiminnoille
RTC	Reaaliaikakello (Real Time Clock) on komponentti, joka pitää itsenäisesti yllä kellonaikaa ja päiväystä
SB	Suojattu tai turvallinen käynnistys (Secure Boot)
SFU	Suojattu tai turvallinen varusohjelman päivittäminen (Secure Firmware Update)
SHA	Kryptografinen yksisuuntainen tiivistefunktio (Secure Hash Algorithm) on funktio, jolla voidaan laskea tiedosta sitä huomattavasti lyhyempi tiiviste (Hash), jota voidaan käyttää tiedon oikeellisuuden tarkastamiseen
SPI	Sarjamuotoinen väylä oheislaitekomponenttien liittämiseen (Serial Peripheral Interface)
SWD	Sarjamuotoinen ohjelmointi- ja vianselvitysliitäntä (Serial Wire Debug)
TRNG	Satunnaislukugeneraattori, joka tuottaa todellisia satunnaislukuja (True Random Number Generator). Perustuu usein johonkin todelliseen satunnaisesti esiintyvään ilmiöön, kuten analogiseen kohinaan
UDI	Yksilöllinen sarjanumero (Unique Device Identification)
WD	Vahtikoira (Watchdog) on mikro-ohjaimen yksikkö, jonka avulla voidaan havaita ohjelman suorituksen juuttuminen tai harhautuminen
X OM	Muistityyppi tai muistialue josta ohjelmaa voi suorittaa mutta ei lukea tai kirjoittaa (Execute Only Memory)

Sisältö

1 Johdanto

Sulautettujen laitteiden määrä kasvaa koko ajan ja laitteiden kirjo on valtava. Erilaiset esineiden internetiin (Internet of Things, IoT) liitetyt laitteet ja vähemmän verkotetut sulautetut järjestelmät, kuten esimerkiksi autoissa ja kodinkoneissa olevat, yleistyvät suurta vauhtia. Tuotteiden kopioiminen ja tuotteisiin murtautuminen on kasvava ongelma, koska joidenkin laitteiden taustalla ennen niiden kaupallistamista voi olla hyvinkin suuri tuotekehityspanos. Toisaalta ne saattavat sisältää erittäin arkoja tietoja, kuten salausavaimia, terveystietoja ja muita sen kaltaisia tietoja.

Tässä työssä pyritään selvittämään millaisia laitteistopohjaisia turvallisuusratkaisuja nykyaikaisissa mikro-ohjaimissa on käytettävissä, sekä miten näitä ominaisuuksia käyttäen voi suojata sulautetun järjestelmän sisältämät ohjelmistot, tiedot, algoritmit ja muun aran tiedon. Työ on toteutettu tutustumalla ensin alueeseen liittyvään kirjallisuuteen, mikro-ohjainvalmistajien verkkosivustoihin, datalehtiin sekä ohjelmointioppaisiin ja muodostamalla niistä kokonaiskuva alueesta.

Tulosten perusteella on nähtävissä, että nykyaikaisissa mikro-ohjaimissa on erittäin paljon erilaisia turvaominaisuuksia. Niitä on mukana jopa aivan niin sanotuissa perus mikro-ohjaimissa, mutta tietenkin parhaat turvaominaisuudet löytyvät erityisesti turvallisuutta vaativiin sovelluksiin suunnitelluista mikro-ohjaimista. Ohjelmistokehittäjien tulisikin tuntea käyttämänsä mikro-ohjain ja sen ominaisuudet hyvin, jotta he osaavat käyttää turvaominaisuuksia hyödykseen. Käyttämättä jätetyistä tai väärin konfiguroiduista turvallisuusominaisuuksista on enemmän haittaa kuin hyötyä.

Työn ensimmäinen luku sisältää johdannon työn alueeseen. Toisessa luvussa käsitellään sulautettuja järjestelmiä ja mikro-ohjaimia. Kolmas luku käsittelee sulautettujen järjestelmien tuvallisuutta ja mikro-ohjaimien turvaominaisuuksia. Neljäs luku sisältää työn yhteenvedon. Liite A sisältää selostuksen STMicroelectronicsin julkaiseman suojatun käynnistyksen ja suojatun varusohjelman päivityksen ”Secure Boot and Secure Firmware Update” (SBSFU) ohjelmapaketin kokeilusta.

2 Mikro-ohjaimet ja sulautetut järjestelmät

Kuusikymmentäluvun loppupuoli oli keskuskoneiden aikaa. Tietokoneet olivat fyysisesti suuria, arkkitehtuurit suosivat pitkiä käskysanoja ja ohjelmointi tehtiin reikäkorteilla. Ohjusten ja lentokoneiden ohjausjärjestelmät olivat pääasiassa analogisia. Tämä kaikki muuttui Apollo-ohjelman myötä [39]. USA:n Apollo-kuuohjelmaa varten rakennettiin ohjaustietokone, joka oli ensimmäisiä nykyaikaisia sulautettuja järjestelmiä. Sen suunnitteli Massachusetts Institute of Technology (MIT), joka toteutti sen perusporttipiireillä. Suunnittelussa käytettiin vain kolmea klassisesta porttipiiriä: JA, TAI ja EI-TAI. Vaikka toteutuksessa olisi ollut yksinkertaisempaa käyttää useampia erilaisia porttipiirejä, kuten JA, EI-JA, TAI, EI-TAI, ehdoton-TAI, ehdoton-EI-TAI ja EI -piirejä, MIT päätti luotettavuussyistä käyttää ainoastaan kolmituloisia EI-TAI -portteja loogisten toimintojen toteuttamiseen. Kussakin Apollo-tietokoneessa käytettiin noin 5 000 porttia. Vuoteen 1963 mennessä noin 60 % Yhdysvaltojen logiikkapiirien kokonaistuotannosta käytettiin Apollo-ohjelman ohjaustietokoneen prototyyppeihin. [91]

Ohjaustietokoneen lopullinen kokoonpano sisälsi 36 864 sanaa ohjelmamuistia (Read Only Memory, ROM) ja 2 048 sanaa työmuistia (Random Access Memory, RAM). Sanan pituus oli 16-bittiä. Ohjelmamuisti perustui ferriittirenkaisiin ja työmuisti toteutettiin porttipiireillä. Ohjaustietokone kulutti tehoa vain 55 W, sen käyttöjännite oli 28 volttia ja kellotaajuus oli 2 048 kHz, joka on nykyisiin prosessoreihin verrattuna todella hidas. [91]

Barr ja Massa [17] määrittelevät sulautetun järjestelmän (Embedded System) olevan laitteiston ja ohjelmiston yhdistelmän sekä mahdollisten muiden osien muodostama järjestelmä, joka on suunniteltu toteuttamaan tietty toiminta. Esimerkkinä he mainitsevat mikroaaltouunin; sellainen on lähes joka kodissa, mutta harvat käyttäjät mikroaaltouunilla ruokaa lämmittäessään tiedostavat käyttävänsä samalla sulautettua järjestelmää, eli pientä tietokonetta tai mikro-ohjainta mikroaaltouunin käyttöliittymän kautta.

Karsai ja muut [38] puolestaan listaavat muun muassa seuraavia sulautettuja järjestelmiä: kodin viihdelaitteet, potilaiden seurantajärjestelmät, kulkuneuvot ja teollisuusautomaatio. Hoske [33] taasen määrittelee sulautetun järjestelmän sisältävän

laitteistoa, ohjelmistoa, tiedonsiirtoa, sisäänrakennettua tietoturvaa ja se voi toimia automaattisesti tai käyttäjän ohjaamana.

Muita jokapäiväisiä sulautettuja järjestelmiä sisältäviä laitteita ovat esimerkiksi matkapuhelin, auto, radio, televisio, verenpainemittari, kaukosäädin sekä henkilö- ja talousvaaka. Usein myös älypuhelimen ja kannettavan tietokoneen akussa on erikoistunut latauksen hallintapiiri (Fuel Gauge), joka sisältää myös pienen mikro-ohjaimen. Piirin tarkoitus on pitää kirjaa lataussykleistä, mitata akkuun ladattua ja sieltä purettua energiamäärää sekä tarkkailla akun lämpötilaa, varsinkin ladattaessa, ettei lämpötila nouse liikaa.

Sulautettu järjestelmä voi olla myös esimerkiksi sensorinoodi, joka mittaa veden laatua bioterrorin varalta. Sillä tulee tällöin olla monia keinoja estää sekä ohjelmiston että laitteistoon kajoaminen, jotta hyökkääjä ei pääse ohittamaan valvonnan turvatoimia ja pilaamaan vedenottamon vettä. [34]

Mikro-ohjain on keskeisin sulautetun järjestelmän rakenneosa. Se on ohjelmoitava komponentti, josta muodostuu laitteen äly. Mikro-ohjain sisältää paljon liitäntöjä, muistia ja muita tarpeellisia toimintalohkoja.

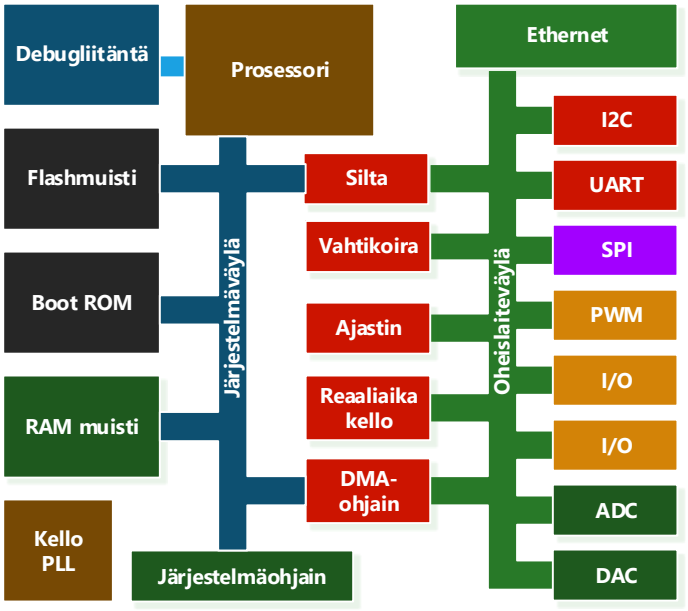
2.1 Mikro-ohjaimien ominaisuuksia

Mikro-ohjaimet (Microcontroller Unit, MCU) ovat yksi sulautettujen järjestelmien tärkeimpiä osia. Ne sisältävät laitteen älyn eli suorittimen lisäksi ohjelma- ja käyttömuistin sekä lukuisia liitäntöjä ja väyliä. Niiden tuotantomäärät ovat todella suuria, sillä nykyisin toteutetaan mitä yksinkertaisimmatkin kuluttajalaitteet mikro-ohjainten avulla. Vuonna 2017 maailmassa tuotettiin noin 26 miljardia mikro-ohjainta, vuonna 2018 noin 28 miljardia. Vuoden 2019 ennuste näyttää jo hieman laskevaa trendiä noin 27 miljardilla mikro-ohjaimelle, mutta vuonna 2020 tuotannon odotetaan taas nousevan noin 29 miljardiin kappaleeseen. [35].

Mikro-ohjaimen ympärille rakentuvat laitteen ulkoiset liitännät, kuten näyttö, painikkeet, merkkivalot, näppäimet ja anturit. Mikro-ohjaimia on hyvin monen tasoisia ja tehoisia. Vaikka mikro-ohjainten esi-isä, Intelin 4004, olikin vain nelibittinen [27], ovat nykyisin yksinkertaisimmat mikro-ohjaimet yleensä vähintään kahdeksanbittisiä. Näiden pienimpien mikro-ohjainten käyttökohteita voivat olla esimerkiksi sähköhammasharjan ohjaus ja käyttöliittymä yhdellä napilla.

Mikro-ohjaimet ovat digitaalisia ohjaimia, joissa on prosessorin lisäksi yhteen rakennettuna tulo- ja lähtöliitäntöjä, käyttö- ja ohjelmamuistia, laskureita ja ajas-

timia, joita on esitelty kuvassa 2.1 [16]. Yleiskäyttöinen prosessori on puolestaan nimensä mukaisesti pelkkä prosessori; se tarvitsee toimiakseen siihen erikseen liitettävät käyttö- ja ohjelmamuistit, tulo- ja lähtöliitäntäpiirit sekä ajastin- ja laskuripiirit. Sellaisenaan se ei vielä pysty toimimaan [23]. Mikro-ohjaimien ja yleiskäyttöisten prosessoreiden suurin ero on sinä, että mikro-ohjaimessa itse prosessori vie vain pienen osan piipinta-alasta. Suurin osa mikro-ohjainpiirien piipinta-alasta kuluu muistien toteuttamiseen (käyttö- ja ohjelmamuisti), kellosignaalien muodostamiseen ja jakeluun, järjestelmäväylän toteuttamiseen ja oheislaitteiden logiikkaan (liitäntäpinnit, analogi-digitaalimuuntimet, kommunikaatioväylät, ajastimet, yms.) [93]. Tyypillinen mikro-ohjaimen rakenne toimintalohkoineen on esitetty kuvasta 2.1.



Kuva 2.1: Tyypillinen mikro-ohjaimen rakenne [93]

Tärkein komponentti on itse prosessori, sen tehtävänä on suorittaa ohjelmoijan sille kirjoittamaa ohjelmaa. Se hyödyntää muita mikro-ohjaimen osia järjestelmäväylän kautta, jolle puolestaan liittyvät ohjelmamuisti (Read Only Memory, ROM, FLASH) ja käyttömuisti (Random Access Memory, RAM). Oheislaiteväylän kautta prosessori pääsee ohjaamaan erilaisia liitäntäpiirejä, joilla liitytään ympäröivään maailmaan.

Debugliitäntä liittyy suoraan prosessorin ytimeen ja järjestelmäväylään, sen kautta on mahdollista ohjelmoida ohjelmakoodi ohjelmamuistiin tai tutkia prosessorin

rekistereitä ja tilaa sekä muistin sisältöä jopa ohjelman suorituksen aikana. Sen avulla voidaan ohjelma keskeyttää milloin tahansa ja seurata ohjelman suoritusta vaikka rivi riviltä ohjelmankehitysympäristön osalla, jota kutsutaan debuggeriksi. Analogi-digitaalimuuntimen (Analog to Digital Converter, ADC) avulla prosessori voi lukea analogisia signaaleita ja digitaali-analogimuuntimen (Digital to Analog Converter, DAC) avulla puolestaan ohjata analogisten lähtöjen jännitteitä.

Vahtikoiran (Watchdog) tehtävänä on huomata ohjelman suorituksen jumiutuminen. Sen toiminta perustuu siihen, että ohjelmasta käydään määrävälein päivittämässä vahtikoiraa. Jos asetettuun aikaan ei ole tehty vahtikoiran päivitystä se käynnistää prosessorin uudestaan, eli resetoi sen. Tällöin ohjelman suoritus alkaa aivan alusta samaan tapaan kuin laitteen käynnistyessä virtojen kytkemisen jälkeen. Käynnistyksen yhteydessä on mahdollista tarkistaa käynnistyksen syy ja toimia eri tavalla eri syiden kohdalla [58]. Vahtikoiran suorittaman uudelleenkäynnistyksen jälkeen on mahdollista siirtyä diagnostiikkaohjelmaan tarkistamaan, onko viallisia komponentteja tunnistettavissa.

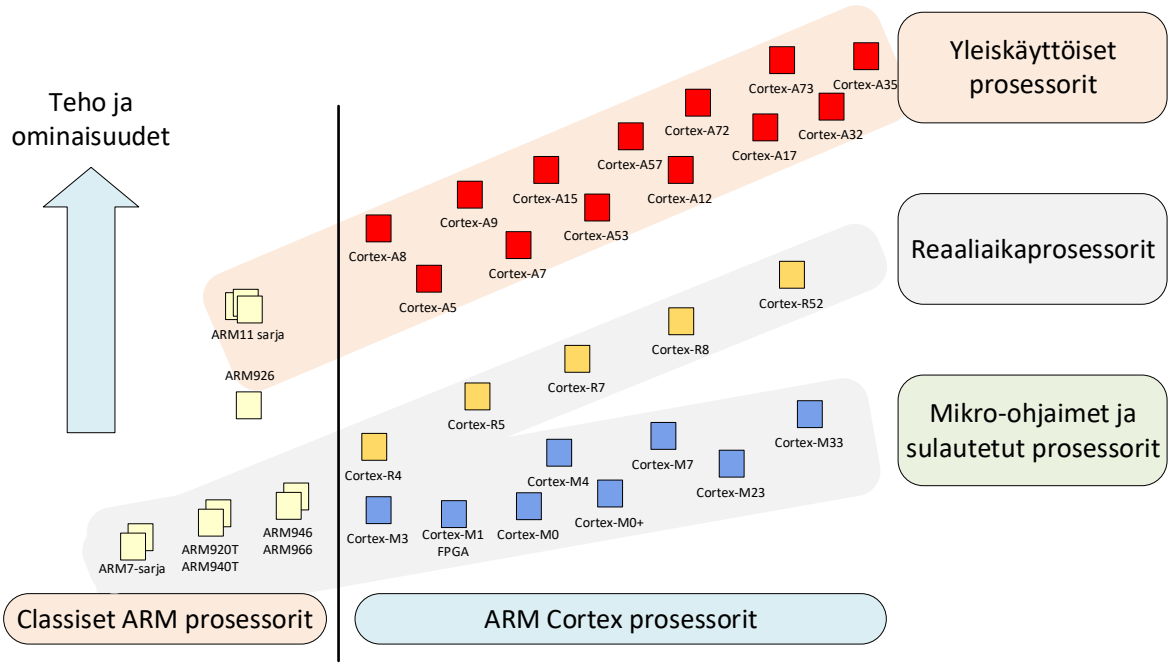
SPI (Serial Peripheral Interface) ja I^2C tai I2C (Inter Integrated Circuit) ovat sarjamuotoisia synkronisia väyliä, joihin voi liittää erilaisia antureita, näyttöjä ja muisteja, kuten Flash-, EEPROM- ja RAM-muistipiirejä. Niiden avulla voi myös useampi mikro-ohjain kommunikoida keskenään. UART-liitäntä (Universal Asyncronous Receiver Transmitter) on asynkroninen sarjaliitäntä. Sen alkuperäinen tarkoitus on ollut toimia liitäntänä käyttäjän tekstipohjaiselle päätteelle tai tulostimelle. Nykyisin UART-liitäntää käytetään usein ohjelmakehityksen aikana erilaisten testitulostusten tulostamiseen ohjelmoijan avuksi pääteohjelman avulla tarkasteltavaksi.

DMA-ohjain (Direct Memory Access) on yksikkö, joka osaa siirtää tietoa muistialueelta toiselle, muistialueelta oheislaitteelle tai oheislaitteelta muistiin ilman prosessorin apua. Reaaliaikakello (Real Time Clock, RTC) on komponentti, joka pitää itsenäisesti yllä kellonaikaa ja päiväystä myös virtakatkojen aikana. Sieltä sulautettu järjestelmä voi lukea luotettavan ajan aina halutessaan, jolloin ohjelmiston ei tarvitse yrittää pitää yllä tarkkaa aikaa.

Suurimmat mikro-ohjaimeksi luokiteltavat kontrollerit ovat 32-bittisiä ja sisältävät ohjelmamuistia satoja megatavuja ja työmuistia satoja kilotavuja. Sellainen voi löytyä vaikkapa ohjaamasta television toimintoja.

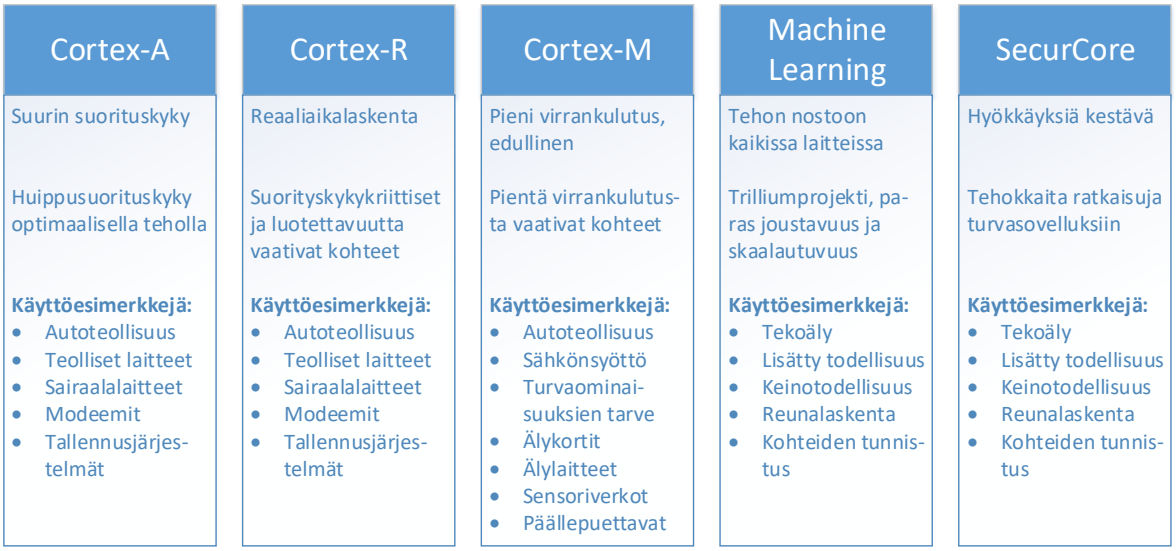
2.1.1 ARM-mikro-ohjaimet

Esimerkkinä suosituista mikro-ohjaimista tarkastellaan ARM -mikro-ohjaimia. ARM Ltd on yritys, joka on luonut ARM-arkkitehtuurin ja kehittää sitä ja uusia prosessorityyppejä edelleen. Se lisensoi suunnittelemiaan mikro-ohjaimia ja prosessoreja muille yrityksille [93] eli se myy immateriaalioikeudet (Intellectual/Immaterial Property, IP) suunnitelmiinsa ja ostaja toteuttaa saamansa suunnitelman ympärille liityntöjä, turvaominaisuuksia ja muita toimintoja. ARM prosessorit ovat saavuttaneet lähes monopolin mm. matkapuhelinten prosessoreina. Niissä tosin ei enää käytetä mikro-ohjainta, vaan niin sanottua yleiskäyttöistä prosessoria, joissa ohjelma- ja käyttömuisti ovat piirin ulkopuolisia komponentteja. ARM-prosessoriperheet voidaan jakaa klassisiin ARM prosessoreihin ja ARM-Cortex -prosessoreihin, kuten kuvassa 2.2. Kuvassa ne on jaettu myös käyttötarkoituksen mukaisesti kolmeen luokkaan: mikro-ohjaimet, reaaliaikaprosessorit ja yleiskäyttöiset prosessorit.



Kuva 2.2: ARM-prosessorien luokittelu [9]

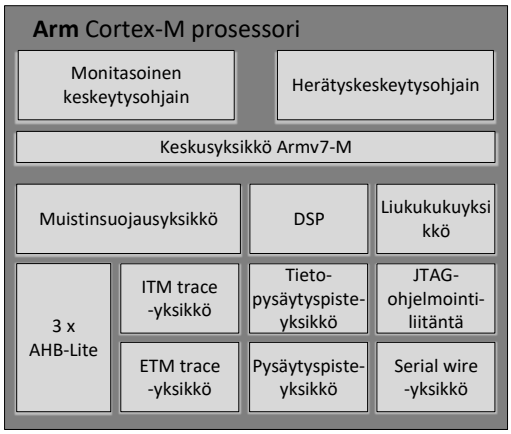
Nykyisin käytössä olevat ARM-suorittimet jaetaan kuvan 2.3 mukaisesti viiteen luokkaan. Niistä Cortex-M -sarja on sulautetuissa laitteissa suosituin. A-sarjan prosessorit ovat lähinnä yleiskäyttöisiä prosessoreja, kun taas M- ja R-sarjan prosessorit ovat mikro-ohjaimia. [7]



Kuva 2.3: ARM-prosessoriperheet [7]

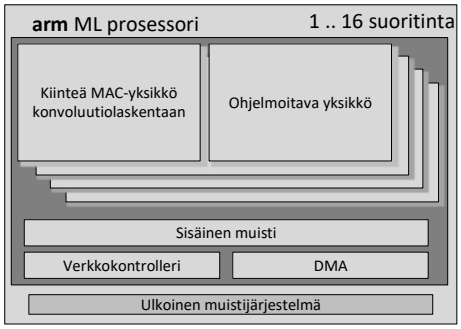
Cortex-A -perhe sisältää isoimpia ARM-prosessoreita. Niitä käytetään paljon muun muassa älypuhelimissa ja tablettitietokoneissa, mutta myöskin paljon tehoa vaativissa sulautetuissa järjestelmissä. Tällaisessa käytössä niissä käytetään yleisesti joko jotain reaaliaikakäyttöjärjestelmää (RTOS, Real Time Operating System) tai jotain Linux-jakelua tai Androidia, joka sekin vaatii alleen Linux-käyttöjärjestelmän. Perheeseen kuuluu lukuisia eri versioita alkaen Cortex-A5 prosessorista, jota saa yhdestä neljään ytimisenä, ja päätyen 64-bittiseen Cortex-A76 prosessoriin, joka tukee neljän ytimen klusterointia. [2]

Cortex-R -perhe on suunniteltu erityisesti nopeaan ja deterministiseen prosessointiin. Perheeseen kuuluu tehojärjestyksessä (pienimmästä suurempaan): Cortex-R4, Cortex-R5, Cortex-R7 ja Cortex-R8. Perheeseen kuuluu myös Cortex-R52, joka sisältää uusimmat ja parhaimmat turvallisuusominaisuudet. [4]



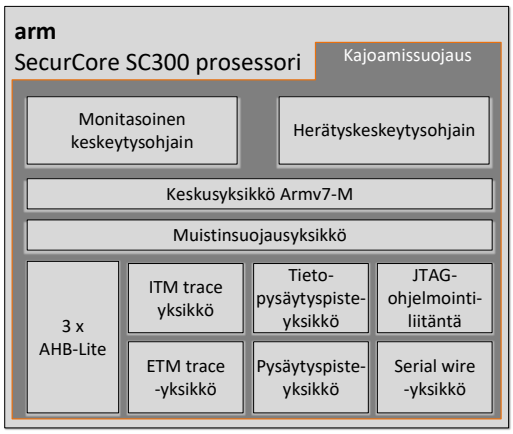
Kuva 2.4: ARM Cortex-M prosessori [8]

Kuvassa 2.4 esitelty Cortex-M -perhe on suunniteltu kohteisiin, jossa tarvitaan pientä virrankulutusta ja edullista hintaa. Perheeseen kuuluu tehojärjestyksessä (pienimmästä suurimpaan): Cortex-M0, Cortex-M0+, Cortex-M1, Cortex-M3, Cortex-M4 ja Cortex-M7, joka on perheen tehokkain prosessori. Pienimmissä prosessoreissa ei ole liukulukuyksikköä, joten ne eivät sovellu kohteisiin, joissa tehdään paljon liukulukulaskentaa. [3]



Kuva 2.5: ARM ML prosessori [5]

Kuvassa 2.5 esitelty ARM ML -perhe on suunniteltu koneoppimis-, neuroverkko- ja tekoälysovelluksiin, erityisesti mobiili- ja kamerasovelluksiin, joissa se antaa moninkertaisen suorituskyvyn verrattuna tavallisiin prosessoreihin, signaalinkäsittelyprosessoreihin (Digital Signal Processing, DSP) ja grafiikkaprosessoreihin (Graphics Processing Unit, GPU). Ne on suunniteltu alusta lähtien tukemaan erityisen nopeaa koneoppimista. [5]

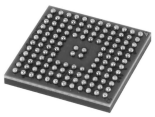
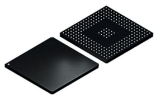


Kuva 2.6: ARM SecurCore SC300 prosessori [12]

Kuvan 2.6 mukainen SecurCore -perhe on suunniteltu korkean suorituskyvyn ja suurien tuotantomäärien älykortteihin ja sulautettuihin turvasovelluksiin. Perheeseen kuuluu kaksi prosessoria, SC000 ja SC300, joista ensimmäinen on suunniteltu suurivolyymisiin tuotteisiin ja jälkimmäinen suurempaa suorituskykyä vaativiin tuotteisiin. [12]

Mikro-ohjaimia on olemassa todella monen kokoisia ja tehoisia. Taulukossa 2.1 on esitelty kolme eritehoista STMicroelectronicsin (STM) valmistamaa ARM-mikro-ohjainta. Niistä pienin ja halvin on pienitehoinen (32MHz) ja melko vähävirtainen sisältäen vain 11 IO-liityntää, suurin taasen sisältää todella paljon muistia ja on nopea, sen kellotaajuus on 400MHz ja siinä on yli 160 IO-liitäntä.

Taulukko 2.1: STM:n ARM-mikro-ohjaimia

	STM32L011	STM32L486	STM32H743
Kuvaus	Pienivirtainen, pienitehoinen pikkuprosessori	Erittäin pienivirtainen, keskitehoinen	Suuritehoinen
Ydin	M0+	Cortex M4	Cortex M7
Ohjelmamuisti (Flash)	8 kB	1024 kB	2048 kB
Käyttömuisti (Staattinen RAM)	2 kB	128 kB	1024 kB
Kellotaajuus	32 MHz	80 MHz	400 MHz
Kotelo ja mitat (muitakin kotelo vaihtoehtoja on)	6,4mm x 5,0mm x 1,0mm TSSOP 14 nastainen kotelo	7mm x 7mm x 0,6mm UFBGA (Ultra Fine-Pitch Ball Grid Array) 132-nastainen kotelo	14mm x 14mm x 0,8mm TFBGA240+25
			
IO-liitäntöjä	11	109	168
Käyttöjännite	1,65 - 3,6 V	1,71 - 3,6 V	1,62 - 3,6 V
Virrankulutus	lepotilassa 0,23μA aktiivisena 87μA/MHz	lepotilassa 0,03μA aktiivisena 100μA/MHz	lepotilassa 7μA aktiivisena 278μA/MHz
Hintaluokka 1000 kpl erässä	0,62€[28]	5,56€[60]	16,45€[29]
Ominaisuuksia	USART, SPI, I2C, 12-bit AD-muuntimia, ajastimia, Systick tuki, reaaliaikakello, 2 x vahtikoira	USB OTG, USART, SPI, I2C, 12-bit AD-muuntimia, ajastimia, 2 kpl vahtikoiria, reaaliaikakello, Systick tuki, AES-kiihdytin, TRNG satunnaislukugeneraattori	USB OTG, 2 kpl vahtikoiria, USART, SPI, I2C, 12-bit AD-muuntimia, ajastimia, reaaliaikakello, Systick tuki, AES-kiihdytin, TRNG satunnaislukugeneraattori

2.2 Sulautettujen järjestelmien ohjelmointi

Sulautettujen järjestelmien ohjelmointi eroaa tavanomaisesta sovellusohjelmoinnista usealla eri tavalla. Sulautetun järjestelmän ohjelmoijan tulee tuntea laitteisto, johon ohjelmaa kehitetään. Suuri osa ohjelmoijan ajasta voi mennä siihen, että saadaan vaikkapa painonäppäimen palveluohjelman ja muiden keskeytysohjelmien palveluajat sopivaan balanssiin tai muistinsuojausyksikön konfiguraatio suunnitelman mukaiseksi. Laitteistoläheisten asioiden miettiminen vie siis suuren osan ohjelmoijan ajasta ja laitteisto, johon ohjelmaa kehitetään, vaihtuu projektista toiseen. Vianselvitys voi vaatia vianselvitysohjelman (debugger) lisäksi oskilloskoopin ja logiikka-analysaattorin käyttämistä [88]. Sulautettujen järjestelmien ohjelmointi vaatii ennen kaikkea hyvää ymmärrystä laitteistosta ja sen komponenteista [41].

Kaikki reaalimaailmaan liittyvät suureet, kuten lämpötila, painikkeiden painamisen tunnistaminen, näytön käsittely yms. pitää ohjelmoida mikro-ohjaimen liitäntä-

nastoihin liitettyjen kytkinten tai oheispiirien avulla. Tämä tarkoittaa sitä, että suuri osa ohjelmointia on erilaisten oheispiirien käsittelyä, joka puolestaan on ns. matalan tason ohjelmointia tai laiteläheistä ohjelmointia, jossa käsitellään paljon yksittäisiä bittejä tai bittiryhmiä loogisten operaattoreiden avulla.

Moderneissa mikro-ohjaimissa on suuri joukko ominaisuuksia, jotka auttavat rakentamaan vakaan ja luotettavan järjestelmän. Tällaisia komponentteja ovat muun muassa keskeytysohjain, systeemikello, ajastimet, I/O-liitännät ja vahtikoira (Watch Dog) [41].

Sulautettua järjestelmää ohjelmoitaessa on koko ajan muistettava sekä se mitä ollaan tekemässä että sulautettujen järjestelmien rajoitukset, kuten [41]:

- Rajallinen ohjelmamuisti. Kaikki halutut ominaisuudet eivät välttämättä mahdu mukaan.
- Rajallinen käyttömuisti. Kaikki puskurit ja tietorakenteet on suunniteltava hyvin ja muistia säästäen.
- Rajallinen laskentateho. Kaikkea haluttua tietoa ei pysty käsittelemään reaaliajassa.
- Paristo- tai akkukäyttöisyys. Tehonkulutus on pidettävä yleensä mahdollisemman alhaisena.

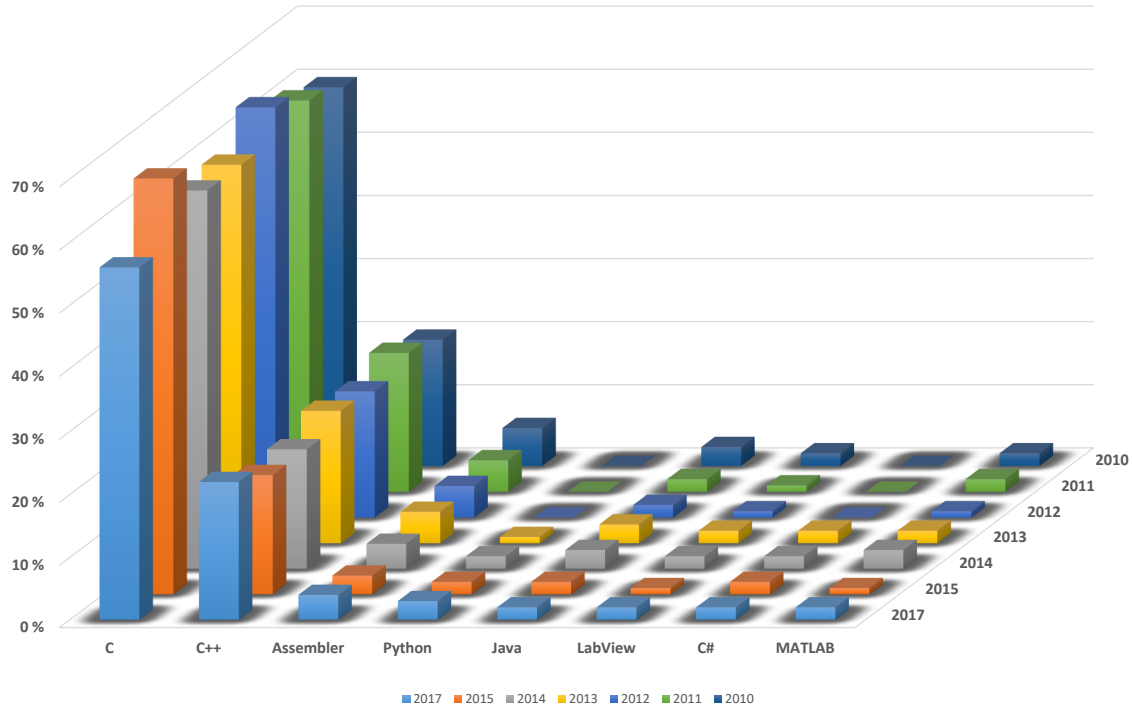
2.3 Sulautettujen järjestelmien ohjelmointiin käytettyjä kieliä

Sulautettujen järjestelmien ohjelmointiin käytettävän kielen tulee olla ennen kaikkea riittävän joustava, jotta sillä voidaan ohjelmoida laitteistoläheisesti, eli lukea ja kirjoittaa helposti mikro-ohjaimen rekistereitä ja muistia, sillä useissa mikro-ohjaimissa kaikki liityntäpiirit, ajastimet ja kommunikaatioväylät näkyvät ohjelmoijalle muistipaikkoina, joihin on pystyttävä kirjoittamaan ja joita on pystyttävä lukemaan ohjelmasta mahdollisimman helposti ja tehokkaasti. Tuotetun koodin tulee käyttää mahdollisimman vähän työmuistia ja ohjelmamuistia [48].

Yksi suosituimmista ohjelmointikielistä sulautettujen järjestelmien projekteissa on C-kieli [89]. Se toteuttaa edellä mainitut vaatimukset erinomaisesti. Ohjelmankehitystyössä tarvittavien työkalujen (ristikääntäjä, linkkeri, debuggeri, yms.) ominaisuuksien tunteminen ja työnkulun ymmärtäminen on onnistuneelle ohjelmistoprojektille tärkeää [44]. Muita tarpeeksi joustavia ja tehokkaita kieliä ovat mm. C++ ja Assembler, eli symbolinen konekieli.

Arrow Electronicsin omistama embedded.com sivusto tekee vuosittain sulautettu-

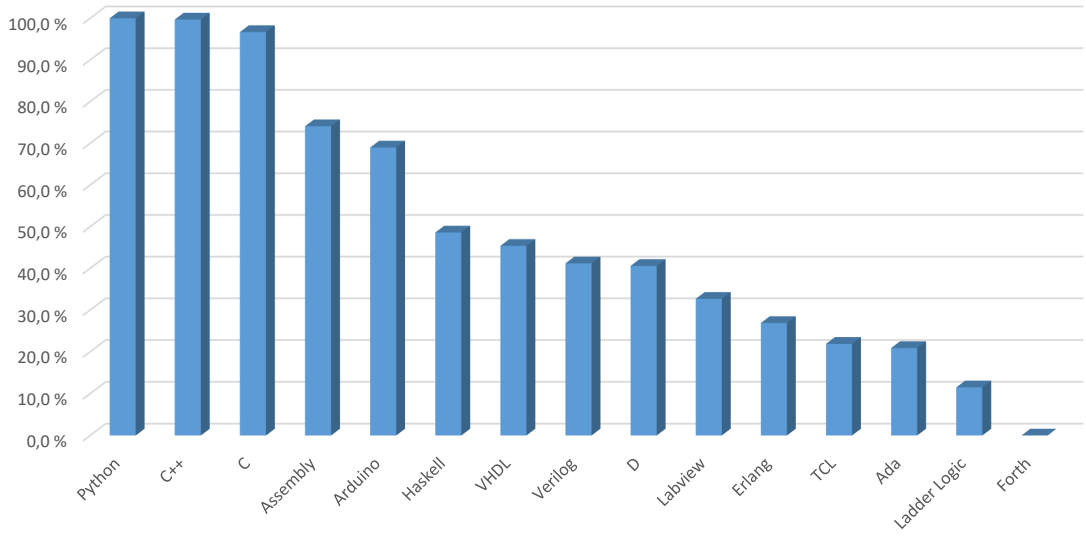
jen järjestelmien markkinatutkimuksen [26]. Sen mukaan käytetyin ohjelmointikieli sulautettujen ohjelmoinnissa on edelleen C-kieli. Kuvassa 2.7 esitetään sulautetuissa projekteissa käytettyjen kielten yleisyys vuodesta 2010 lähtien. Siihen on koottu tiedot useamman vuoden tutkimuksista ja siitä on nähtävissä, että noin 60 - 70 % projekteista toteutetaan C-kielellä, C++ -kielen osuus on noin 20 % ja symbolisen konekielien osuus on noin 4 %. Muiden kielten osuus on lähes merkityksettömän pieni. Myöskään mitään suurta siirtymää pois C-kielen käytöstä ei ole havaittavissa. Ehkä kuitenkin osa C-kielen käyttäjistä on siirtynyt C++ -kieleen.



Kuva 2.7: Ohjelmointikielten osuus sulautetuissa projekteissa[24, 25, 26]

Vastaavanlaisia tutkimuksia on muitakin, kuten IEEE Spectrumin "The 2018 Top Programming Languages" [20]. Kuvassa 2.8 on esitetty kyseisen tutkimuksen tulos sulautettujen järjestelmien ohjelmoinnissa. Tätä tutkimusta on arvosteltu muun muassa EETimes-sivustolla [69] siitä, että se antaa väärän kuvan, ja että Python ei voi olla niin yleisessä käytössä sulautettujen järjestelmien ohjelmoinnissa kuin siinä annetaan ymmärtää. Python on luonteeltaan tulkattava kieli, joskin löytyy myös kääntäjiä, jotka kääntävät Pythonia suoraan suoritettavaksi kieleksi. Yksi selitys Pythonin ylikorostuneisuudelle tutkimuksessa on se, että hyvin monissa sulautetuissa

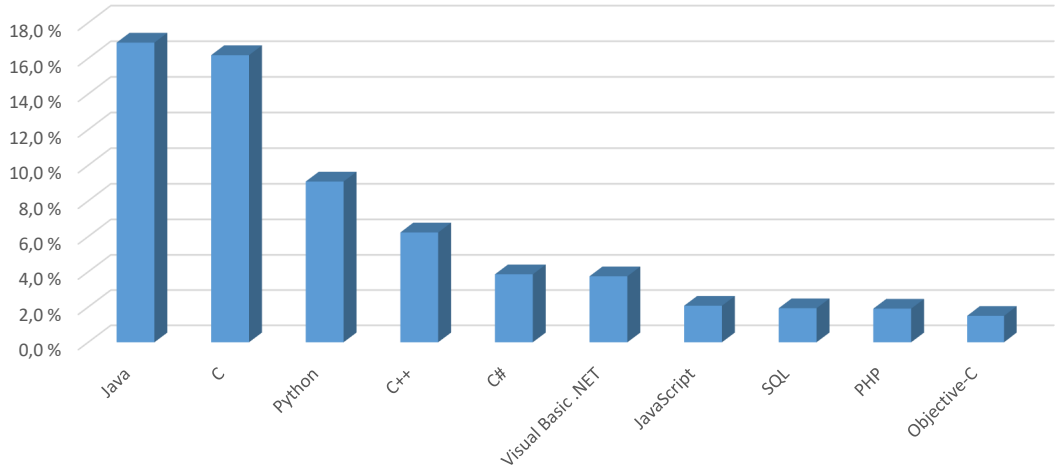
projekteissa Pythonia käytetään jossain roolissa, esimerkiksi käännösympäristön apukielenä tai muuna skriptauskielenä avustamassa ohjelmistopaketin valmistusta. Linux-pohjaisissa sulautetuissa sovelluksissa osa toiminnoista voi hyvinkin olla toteutettu Pythonilla, mutta koko muu järjestelmä C- tai C++ -kielellä.



Kuva 2.8: Ohjelmointikielten osuus sulautetuissa projekteissa IEEE Spectrumin mukaan [20]

TIOBE-indeksin mukainen ohjelmointikielten yleisyys esitellään kuvassa 2.9. TIOBE indeksi lasketaan useiden eri hakukoneiden antamista tuloksista kyselystä +"<*language*> programming", jossa <*language*> on jokainen tutkimukseen hyväksytty ohjelmointikieli vuorollaan. [86]

Indeksi ei mittaa ohjelmointikielten hyvyyttä eikä niiden soveltuvuutta tiettyyn tarkoitukseen, vaan se mittaa sitä, kuinka paljon hakukoneista saadaan hakutuloksia kyseisestä kielestä. Tähän tietenkin sisältyvät GitHubin kaltaiset versionhallintajärjestelmät sekä kaikki lukuisten, muun muassa ohjelmointipainotteisten, keskustelupalstojen keskustelut kielestä.



Kuva 2.9: Ohjelmointikielten osuus projekteissa yleisesti TIOBE:n mukaan [86]

C-kieli

C-kieli on yleiskäyttöinen ohjelmointikieli, jonka Dennis Ritchie kehitti 1970-luvulla PDP-11 tietokoneelle Unix käyttöjärjestelmän kehitystyötä varten. Se pohjautuu B-kieleen, joka puolestaan pohjautuu BPCL-kieleen. Nämä molemmat olivat tyypittömiä kieliä. C-kielen muuttujat varustettiin tyypillä, jotta kääntäjä tietäisi, kuinka monta tavua muistia muuttujalle piti varata. C-kieli on suhteellisen matalan tason kieli siinä mielessä, että se sisältää melko pienen joukon varattuja sanoa ja rakenteita. Sitä on helppo käyttää laiteläheisessä ohjelmoinnissa sen sisältämien osoitinmekanismien vuoksi, sekä myös siksi, että se ei rajoita ohjelmoijaa tekemällä rajoittavia tarkistuksia [67]. Tätä ominaisuutta pidetään usein myös yhtenä C-kielen pahimmista ongelmista, sillä kääntäjä tekee juuri sen, mitä ohjelmoija pyytää.

C-kielen standardointi aloitettiin vuonna 1983 kun American National Standards Institute (ANSI) muodosti komitean X3J11 määrittelemään C-kielen spesifikaation. Standardi pohjautui sen aikaiseen C-kielen Unix toteutukseen. Unix-sidonnaiset osat eriytettiin IEEE:n työryhmän 1003 käsiteltäviksi. Niistä muodostui myöhemmin Portable Operating System Interface (POSIX) -standardin perusta. Vuonna 1989 C-standardi vahvistettiin, sen nimeksi tuli ANSI X3.159-1989 "Programming Language C". Tätä versiota kutsutaan usein nimellä ANSI C tai standardi C, joskus myös C89.

Vuonna 1990 International Organization for Standardization (ISO) vahvisti ANSI C:n spesifikaation ja nimesi sen nimellä ISO/IEC 9899:1990. Tästä johtuen C89 ja C90 tarkoittavat itse asiassa samaa standardia. C-kielen standardointia jatkaa nykyisin

ISO/IEC komitea JTC1/SC22/WG14. Standardia päivitetään viiden vuoden välein [36]. Vuonna 1999 standardia päivitettiin, uuden standardin nimeksi tuli ISO/IEC 9899:1999, se tunnetaan yleensä nimellä C99.

Viimeisin standardi on vuonna 2018 julkaistu ISO/IEC 9899:2018, joka tunnetaan sekä nimellä C17 että C18. Se ei tuonut tullessaan uusia ominaisuuksia, sillä siinä lähinnä korjataan C11:ta raportoituja virheitä [37].

C++ -kieli

C++ -kielen kehitti Bjarne Stroustrup laajentamaan C-kieltä AT&T:n Bell Laboratoriossa 1980-luvun alussa. Sen suurimmat muutokset verrattuna C-kieleen ovat tietoabstraktion, olio-ohjelmoinnin ja geneerisen ohjelmoinnin tuki. Lisäksi se tuo mukanaan parannuksia ja laajennuksia C-kieleen. [45, 78] C-kieli on C++ -kielen osajoukko, eli C++ -kääntäjät ymmärtävät myös C-kielellä kirjoitettua lähdekoodia. Bjarne Stroustrup kuvaileekin C++ -kieltä sanoin: "A better C", eli "parempi C-kieli"[79].

C++ on kehittynyt 1980-luvulta lähtien pikkuhiljaa. Nykyisen nimensä C++ sai vuonna 1983 Rick Mascittin ehdotuksesta, sitä ennen sitä kutsuttiin nimellä "C with Classes"[80]. Ensimmäinen virallinen standardi oli C++ ISO/IEC 14882:1998, joka tunnetaan nimellä C++98. Vuonna 2003 julkaistiin C++03, jossa korjattiin useita aiemman standardin ongelmia. Vuonna 2005 standardointikomitea julkaisi taas uuden suunnitelman muutoksista ja lisäyksistä, kuten laajan standardikirjaston (Standard Template Library, STL). Uudelle versiolle annettiin nimi C++0x, koska se piti julkaista ennen vuosikymmenen vaihdetta. Se valmistui kuitenkin vasta vuoden 2011 keskivaiheilla. Uusi standardiversio sai siis nimekseen kuitenkin C++11. [22]

Uusimmat standardit ovat C++14 ja C++17, niistä käytetään usein nimitystä "Modern C++". Uusi standardi julkaistaan nykyisin kolmen vuoden välein, joten seuraava julkaisu tulee vuonna 2020 ja on nimeltään C++20.

Esimerkki C-kielen soveltuvuudesta laitteistoläheiseen ohjelmointiin

Listauksessa 2.1 on esitetty pelkistetty katkelma ohjelmasta, joka toistuvasti kirjoittaa IO-portin A bittiin viisi ylätilaa ja alatilaa. Se pyrkii kuvaamaan C- ja C++ -kielien erinomaisesta soveltuvuutta laiteläheiseen ohjelmointiin. Koodi on karsittu versio tarvittavista määrittelyistä, joilla voidaan lukea ja kirjoittaa muistiavaruuteen (osoitteeseen 0x4000 0000) sijoitettua liitäntäporttia. Riveiltä 25 ja 27 voidaan havaita,

että alkumäärittelyjen jälkeen muistiin sijoitettuun IO-porttiin on helppo kirjoittaa haluttuja arvoja. Lukeminen on yhtä helppoa, kuten riviltä 24 voi havaita.

```
1  #include <stdint.h>
2  // STM32-series ARM MCU IO-port register descriptions
3  typedef struct
4  { //                name           description                offset
5    volatile uint32_t MODER;      // mode                       0x00
6    volatile uint32_t OTYPER;     // output type reg            0x04
7    volatile uint32_t OSPEEDR;    // output speed reg           0x08
8    volatile uint32_t PUPDR;      // pull-up/pull-down reg      0x0C
9    volatile uint32_t IDR;        // input data reg             0x10
10   volatile uint32_t ODR;        // output data reg            0x14
11   volatile uint32_t BSRR;       // bit set/reset reg          0x18
12   volatile uint32_t LCKR;       // configuration lock reg     0x1C
13   volatile uint32_t AFR[2];     // alternate function reg     0x20-0x24
14 } GPIO_TypeDef;
15
16 #define GPIOA_BASE   (0x40000000UL)
17 #define GPIO_PIN_5   ((uint16_t)0x0020)
18 #define GPIO_PIN_6   ((uint16_t)0x0040)
19 #define PA           ((GPIO_Typedef *)GPIOA_BASE)
20
21 int main() {
22   // 'Mirror' port A pin 6 state to pin 5 as fast as possible
23   while (1) {
24     if ( PA->IDR & GPIO_PIN_6 ) {    // if Port A pin 6 is 1
25          PA->BSRR = GPIO_PIN_5;      // set Pin 5 of GPIO port A to 1
26     } else {
27          PA->BSRR = (uint32_t)GPIO_PIN_5 << 16; // Set it to 0
28     }
29   }
30 }
```

Listaus 2.1: IO-porttiin kirjoittaminen

2.4 Ohjelmointi ja tietoturva

Ohjelmoija ja ohjelmistosuunnittelija ovat vastuussa tuottamansa ohjelman tietoturvan tasosta. Siihen he voivat vaikuttaa huomioimalla tietoturvan koko ohjelman kehityskaaren kaikkien vaiheiden aikana. Käytettiinpä ohjelmistoprojektissa ja ohjelmoinnissa mitä tahansa lähestymistapaa, on tietoturva otettava koko ajan huomioon.

Suunnitteluvaiheessa tehdään linjauksia, joissa tietoturvan perusta lyödään lukkoon. Ohjelmointivaiheessa puolestaan voidaan hyvätkin suunnitelmat tehdä tyhjiksi oikomalla tai tekemällä vähemmän turvallisia ratkaisuja. Tässä vaiheessa kannattaa käyttää jotain ohjelmointikäytäntöä (Coding Standard), kuten MISRA C tai CERT-C. Niiden ohjeet ulottuvat laajoista kokonaisuuksista hyvinkin pieniin yksityiskohtiin. Näiden molempien perusajatuksena on antaa ohjelmoijille ohjeita, joita noudattamalla syntyy turvallisempaa ohjelmakoodia.

Ohjeista ei ole mitään hyötyä, jos niitä ei noudateta, joten yleensä on syytä käyt-

tää koodin analysointityökaluja, jotka tarkistavat ohje ohjeelta ja sääntö säännöltä, onko tuotettu koodi ohjeiden mukaista. Myös koodikatselmuksissa on seurattava ohjeidenmukaisuutta.

Ohjelmoijan vastuulle ei pitäisi jättää tietoturvan syvällistä tuntemista vaativia päätöksiä. Ei ole järkevää olettaa, että tietoturvaa toteuttavien ja käyttävien ohjelmoijien tulisi olla tietoturvan ammattilaisia ja tuntea kaikki alan ongelmat. Ohjelmoijille tulisi tarjota valmiiksi mietityt ratkaisut ja alustat ja antaa ohjeistus niiden käytöstä. Niinpä jotkin uudet ja nousevat IoT-alustat, kuten aliluvussa 3.4.1 esiteltävä turvallisuusarkkitehtuuri PSA, ottavat muun muassa tietoturvan vastuulleen ja antavat ohjelmoijien keskittyä sovelluslogiikan toteuttamiseen. [68]

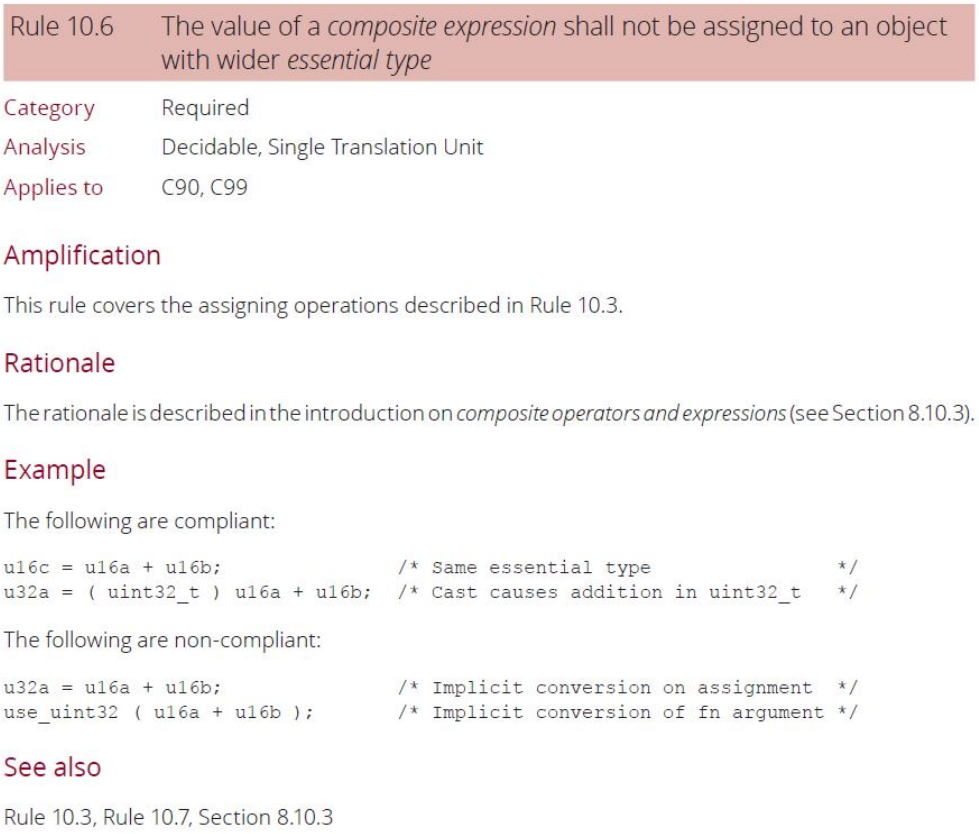
MISRA C -ohjeistus ja ohjelmointikäytänteet

Kun 1990-luvulla autoalalla havaittiin, että elektroniikan merkitys autoissa kasvaa koko ajan, perustettiin MISRA (Motor Industry Software Reliability Association), joka on autoalalla toimivien yritysten yhteenliittymä. Sen päätehtävänä on antaa ohjelmistoturvallisuuteen liittyviä ohjeita ja suosituksia. Niillä on suuri merkitys, ei vain koko autoalalle, vaan myös yksittäisille auton käyttäjille parantuneena turvallisuutena. [64]

MISRA C -ohjeistuksesta on julkaistu useita eri versioita, kuten MISRA C:1998, MISRA C:2004, MISRA C:2008 ja MISRA C:2012, josta on julkaistu useita painoksia. Kirjoitushetkellä uusin on MISRA C:2012 painos 3, revisio 1. MISRA C koostuu direktiiveistä ja vaatimuksista, jotka kummatkin on luokiteltu kolmeen luokkaan: Pakollinen (mandatory) on ohje, jota on noudatettava poikkeuksetta. Vaadittu (required) on ohje, josta voidaan poiketa, jos siihen on painava syy. Ohjeellinen (advisory) on suositus, josta voi poiketa jos niin halutaan.

On myös oltava määriteltynä prosessi, jonka mukaan toimitaan, kun ohjeesta joudutaan poikkeamaan. Poikkeamisen syy on myös dokumentoitava prosessin mukaisesti. Ohjeellinen (advisory) ohje on nimensä mukaisesti vähemmän tärkeä, mutta niitä suositellaan noudatettavaksi.

Ohjeilla pyritään huomioimaan muun muassa seuraavia asioita: Kääntäjien erilaisten ominaisuuksien aiheuttamat ongelmat, ongelmallisten rakenteiden ja kirjastofunktioiden käyttämisongelmat, koodin ylläpidettävyys- ja vianselvitysongelmat, hyväksi havaittujen käytäntöjen käyttäminen ja symbolien näkyvyyteen liittyvät ongelmat.



Rule 10.6	The value of a *composite expression* shall not be assigned to an object with wider *essential type*
Category	Required
Analysis	Decidable, Single Translation Unit
Applies to	C90, C99

Amplification

This rule covers the assigning operations described in Rule 10.3.

Rationale

The rationale is described in the introduction on *composite operators and expressions* (see Section 8.10.3).

Example

The following are compliant:

```
u16c = u16a + u16b;               /* Same essential type               */
u32a = ( uint32_t ) u16a + u16b;  /* Cast causes addition in uint32_t   */
```

The following are non-compliant:

```
u32a = u16a + u16b;               /* Implicit conversion on assignment  */
use_uint32 ( u16a + u16b );       /* Implicit conversion of fn argument */
```

See also

Rule 10.3, Rule 10.7, Section 8.10.3

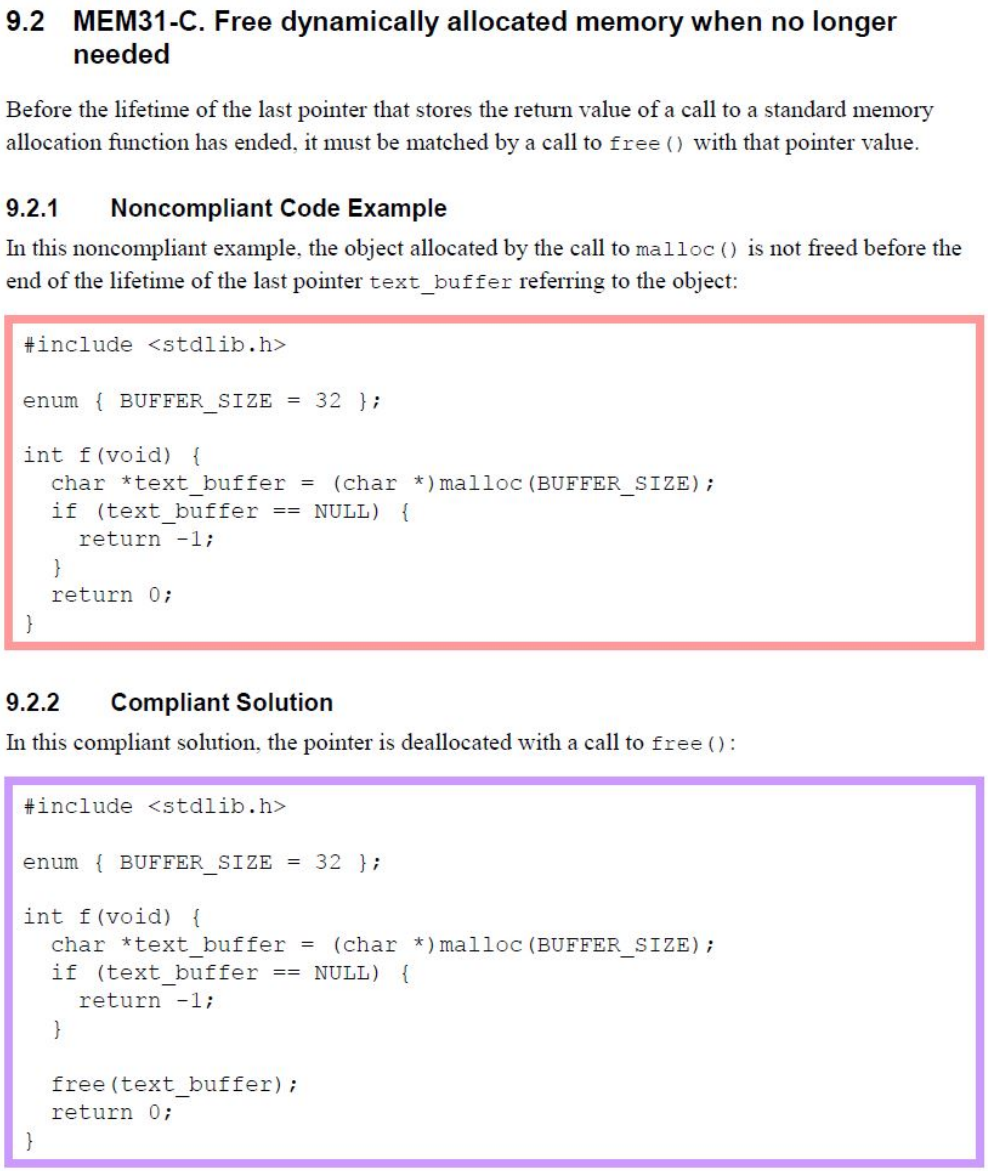
Kuva 2.10: Esimerkki MISRA C:2012 säännöstä [59]

MISRA C:2012 ohjeistus antaa myös ohjeita siitä, kuinka MISRA C integroidaan ohjelmakehitysprosessiin ja millaisia aktiviteetteja prosessin on sisällytettävä sekä millaisia dokumentteja prosessista tulee syntyä. Vaadituista ohjeista poikkeamiset on dokumentoitava perusteluineen.

Kuvassa 2.10 on esimerkki MISRA C:2012 säännöstä 10.6. Säännön olennainen sanoma on se, että kapeampaa tietotyyppiä olevaa arvoa ei tule sijoittaa leveämpään muuttujaan. Syynä on tietotyypin muunnokseen liittyvät mahdolliset ongelmat. Tyypin muuntaminen tulee kirjoittaa koodiin näkyviin, jotta koodia lukiessa voidaan olettaa kirjoittajan tienneen, että tyyppi muuttuu ja ottaneen mahdolliset sivuvaikutukset huomioon. [59] MISRA C:2012 sisältää kaikkiaan 17 direktiiviä ja 156 vaatimusta. Sivuja ohjeistuksessa on noin 250.

CERT-C -ohjeistus ja ohjelmointikäytänteet

Computer Emergency Response Team Coordination Center (CERT/CC) on voittoa tavoittelemattoman Yhdysvaltalaisen Software Engineering Instituten (SEI) osa, jonka tavoitteena on tutkia ohjelmistovirheitä ja internetin turvallisuutta. Se julkaisee tutkimuksiensa tuloksia ja pyrkii edistämään yhdessä yritysten ja hallituksen kanssa ohjelmistoturvallisuutta ja internetin turvallisuutta.



9.2 MEM31-C. Free dynamically allocated memory when no longer needed

Before the lifetime of the last pointer that stores the return value of a call to a standard memory allocation function has ended, it must be matched by a call to `free()` with that pointer value.

9.2.1 Noncompliant Code Example

In this noncompliant example, the object allocated by the call to `malloc()` is not freed before the end of the lifetime of the last pointer `text_buffer` referring to the object:

```
#include <stdlib.h>

enum { BUFFER_SIZE = 32 };

int f(void) {
  char *text_buffer = (char *)malloc(BUFFER_SIZE);
  if (text_buffer == NULL) {
    return -1;
  }
  return 0;
}
```

9.2.2 Compliant Solution

In this compliant solution, the pointer is deallocated with a call to `free()`:

```
#include <stdlib.h>

enum { BUFFER_SIZE = 32 };

int f(void) {
  char *text_buffer = (char *)malloc(BUFFER_SIZE);
  if (text_buffer == NULL) {
    return -1;
  }

  free(text_buffer);
  return 0;
}
```

Kuva 2.11: Esimerkki CERT-C säännöstä MEM31-C[73]

CERT C Secure Coding Standard sisältää 98 ohjetta tai sääntöä turvallisten, luotta-

vien ja tietoturvallisten ohjelmien kirjoittamiseen C-kielellä. Jokainen ohje koostuu otsikosta, kuvauksesta sekä esimerkeistä, joissa sama koodi on toteutettu ohjetta noudattaen sekä ohjeen ohjeita noudattamatta. CERT C ei sisällä koodin ulkoasuun liittyviä ohjeita.

Ohjeistus sai alkunsa vuonna 2006 C kielen standardointielimen ISO/IEC JTC1/SG22/WG14 työryhmän kokouksessa, kun Thomas Plum ehdotti Robert Seacordille, että CERT:n tulisi määritellä oma turvallisen ohjelmoinnin ohjeistus. Perusteluna oli se, että MISRA C oli suunnattu turvakriittisiin sovelluksiin ja oli siten suunnattu hyvin erilaiseen käyttöön kuin CERT-C -ohjeistuksen olisi tarkoitus suuntautua. [70, 73]

CERT-C -ohjeistuksen tarkoituksena on ottaa käyttöön tunnettuja ja hyväksi havaittuja käytänteitä, sekä uusia vähemmän tunnettuja käytänteitä, joita tarvitaan, kun vanhat eivät ole riittäviä. Ennen kaikkea ohjeistus tukee C-kielen version C11 käyttöä. [71]

Kuvassa 2.11 on yksi CERT C sääntö, MEM31-C. Se liittyy dynaamisen muistin käsittelyyn ja sen sisältö on pääpiirteittäin se, että jos varataan muistia dynaamisesti, on muisti myös vapautettava, kun sitä ei enää käytetä. Sanomaa vahvistamassa on kaksi esimerkkiä, oikea ja väärä tapa käyttää dynaamista muistia.

Staattinen koodianalyysi

Staattisella koodianalyysilla tarkoitetaan analyysiä, jossa ohjelmakoodia tutkitaan käynnistämättä sitä missään laitteessa. Ensimmäinen staattinen koodianalyysityökalu, Lint, kehitettiin 1970-luvulla Bell Laboratoriossa löytämään virheitä ohjelmakoodista, joka kääntyi virheittä, mutta ei kuitenkaan toiminut niin kuin piti tai kaatui yllättäen jossain tilanteissa. Se toimii etsimällä ohjelmakoodista tunnettuja virheitä tai vääriä ohjelmointitapoja, kuten alustamattomia muuttujia, ja raportoimalla niistä käyttäjälle. [46, 92]

MISRA C ja CERT-C ovat ohjeistuksia, joissa on määritelty tällaisia tunnettuja virheitä, mutta eivät itsessään ole analyysityökaluja. Monet kaupalliset staattiset koodianalyysityökalut käyttävät näitä säännöstöjä hyväkseen. Staattinen koodianalyysi suoritetaan yleensä ohjelman kääntämisen jälkeen, jos käännös onnistuu virheittä. Analyysin avulla löydetään yleensä vielä korjattavaa ja koodin laatu pysyy parempana kuin ilman staattista koodianalyysiä.

Dynaaminen koodianalyysi

Dynaamisella koodianalyysillä tarkoitetaan koodin analysoimista sen ollessa käynnissä ajoympäristössään. Tällöin voidaan tarkkailla esimerkiksi pinon riittävyyttä ja muistialueiden täyttöasteita. Ohjelmaa analysoidaan ajoympäristössään, joten ohjelmalle on jotenkin järjestettävä ne syötteet, joita ohjelma saisi normaalistikin sitä ajettaessa. Tämä voidaan järjestää esimerkiksi niin sanotuilla syötetiedostoilla tai laitteistolla, joka simuloi normaalikäytössä laitteelle saapuvia syötteitä ja signaaleja.

Ajon aikana kerätään jäljitystiedostoon (Trace File) kaikki tieto siitä mitä ohjelma teki. Tällainen tiedosto voi olla hyvin suuri. Se sisältää koko ohjelman suorituksen ajalta kaikki käskyt, jotka ohjelma suoritti ja mihin muistiosoitteisiin se viittasi. Sitä voidaan myöhemmin analysoida sitä varten kehitetyillä analysointityökaluilla, jolloin saadaan tietoa muun muassa siitä, mitä ohjelma oikeasti teki eri syötteiden saapuessa ja paljonko pinoa oli käytössä ohjelman eri vaiheissa. [19]

Yksikkötestit

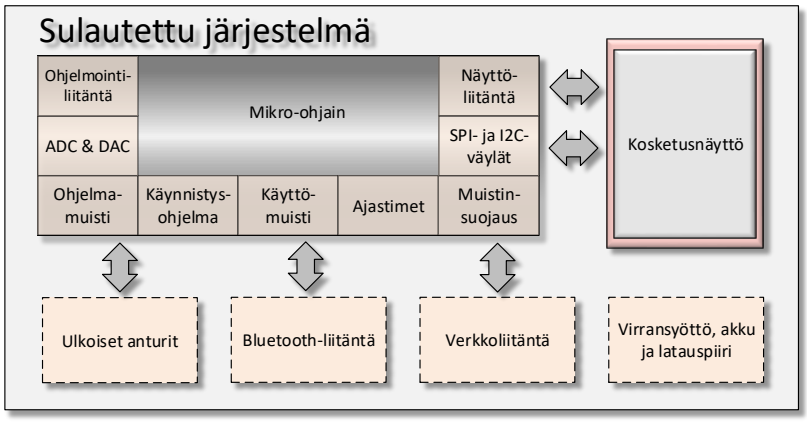
Yksikkötestauksessa ohjelmakoodia testataan esimerkiksi funktio kerrallaan erilaisilla syötteillä niin, että koodin kaikkia mahdollisia osia saadaan testatuksi. Yksikkötestauskin on dynaamista koodianalyysiä, sillä ohjelmaa testataan sen ollessa käynnissä.

Työn kulku yksikkötestausta käytettäessä on hieman erilainen kuin normaalisti. Ohjelmakoodin osan valmistuttua ajokuntoon se ensin käännetään, sitten suoritetaan kyseiselle toiminnallisuudelle suunnitellut testitapaukset ja katsotaan, toimiiko ohjelma niin kuin oli suunniteltu. Ellei, sitä korjataan ja ajetaan testit uudestaan. [32]

2.5 Mikro-ohjaimien käyttö sulautetuissa järjestelmissä

Kuvassa 2.12 esitellään erään kuvitteellisen sulautetun järjestelmän ylätason rakennekaavio. Siitä ilmenee muun muassa se, että mikro-ohjain sisältää huomattavan määrän toimintoja, joiden lisäksi tarvitaan vain muutamia lisäkomponentteja (kuvassa katkoviivalla) mikro-ohjaimen tarjoamien toimintojen lisäksi. Näyttö liittyy hitaammissa laitteissa jollain yleiskäyttöisellä sarjamuotoisella liitännällä, kuten SPI- tai I2C-väylällä.

Sovelluksesta riippuen voi suurehko näytön päivitysnopeus olla tarpeen, jos on tarpeen näyttää esimerkiksi liikkuvaa kuvaa. Tällöin usein käytetään joko yleis-



Kuva 2.12: Eräs sulautettu järjestelmä

käyttöisillä liitäntänastoilla toteutettua rinnakkaismuotoista liitäntää tai erityisesti pienelle näytöille suunniteltua DSI-liitäntää (Display Serial Interface, DSI), joka on MIPI allianssin (Mobile Industry Processor Interface) standardoima. Tällä liitännällä varustettuja näyttöjä käytetään usein mobiililaitteissa. Kosketusnäytön ohjainpiiri liitetään mikro-ohjaimeen yleensä I2C-väylällä.

Sopivan mikro-ohjaimen valinta

Laitteeseen sopivan mikro-ohjaimen valinta voi olla joskus vaikea tehtävä, sillä huomioon otettavia seikkoja on paljon. Usein eräs tärkeimmistä valintakriteereistä on hinta, varsinkin jos kyseessä on massatuote. Tällöin jokainen säästetty sentti komponenttikuluissa voi vaikuttaa huomattavasti tuotteen kokonaistuottoon. Mahdollinen 20 sentin säästö laitteen mikro-ohjaimessa (tai missä tahansa käytetyssä komponentissa), kun arvioitu myyntimäärä on 100 000 kappaletta, vaikuttaa tuottoon 20 000 €. Tämä kertautuu, kun jokainen käytetty komponentti valitaan niin, että voidaan käyttää halvinta mahdollista, kuitenkaan laadusta tinkimättä. Tietenkin jokainen komponentti, jonka voi jättää kokonaan pois, vaikuttaa tuottoon myös ratkaisevasti ja pienentää valmistuskustannuksia.

Mikro-ohjaimen valintaan vaikuttavia tekijöitä ovat muun muassa vaadittu toimintalämpötila-alue, haluttu käyttöjännite, piirin saatavuus ja odotettavissa oleva piirin saatavuusaika, piirin kotelointi, virrankulutus, kosteuden kesto, tarvittavien liitäntänastojen määrä, haluttu suorituskyky, tarvittavien muiden liitäntöjen määrä,

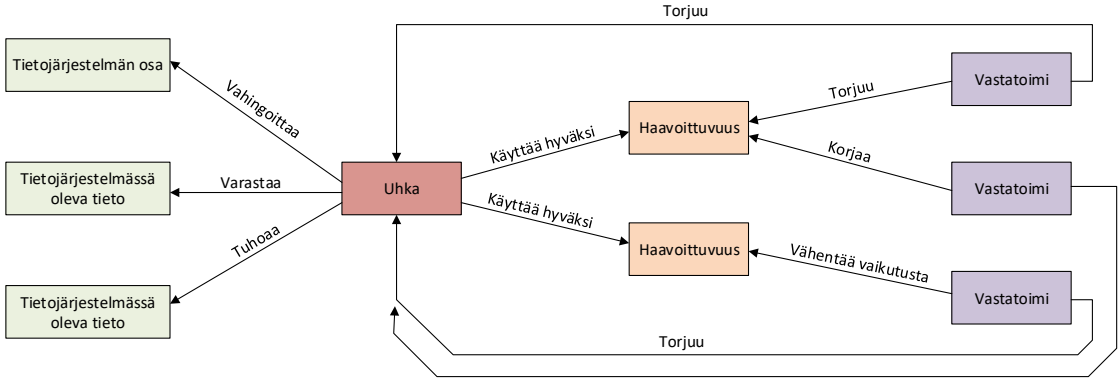
piirin vaatima oheiskomponenttien määrä, käyttö- ja ohjelmamuistin määrä, halutaanko integroituja langattomia toimintoja ja muut vastaavat seikat.

3 Sulautettujen järjestelmien tietoturva

Tietoturvaa voi lähestyä esimerkiksi uhka-analyysien avulla. Uhka-analyysien tekemisessä käytetyssä metodissa, PTA (Practical Threat Analysis), esiintyy seuraavia toimintoja [65]:

- Uhka (Threat)
- Laite, ohjelma tai tieto (Asset)
- Haavoittuvuus (Vulnerability)
- Vastatoimi (Countermeasure)

Kuvassa 3.1 uhka aiheuttaa vaurion laitteelle tai tiedolle. Uhka käyttää hyväksi haavoittuvuutta. Haavoittuvuus voidaan torjua (tai sen tehoa vähentää) vastatoimella. Nämä määritelmät kuvaavat hyvin myös sulautettujen järjestelmien tietoturvaa, joskin melko korkealla tasolla.



Kuva 3.1: Uhkat [65]

Sulautettujen laitteiden tietoturvaratkaisujen luonne poikkeaa perinteisten laitteiden tietoturvaratkaisuista, koska niissä käytettävissä olevat resurssit ovat selkeästi pienemmät ja rajoittuneemmat laitteen laskentatehon ja muistin määrän takia [34].

Sulautettujen järjestelmien suunnittelijat käsittävät usein tietoturvan olevan valmiiseen tuotteeseen jälkeenpäin lisättävä ominaisuus, joka muodostuu kryptografisista algoritmeista ja tietoturvaprotokollista. Todellisuudessa tietoturva on sulau-

tettujen järjestelmien suunnittelulle ulottuvuus, joka on otettava huomioon koko suunnittelun ja valmistusprosessin ajan. [66]

Ohjelmistoarkkitehtejä ja ohjelmoijia on opetettu suunnittelemaan ja toteuttamaan kaikki sulautetut laitteet minimikustannuksin. Heidän päätöksensä pohjautuvat omaan kokemuspohjaan ja edellisiin projekteihin. Usein tehdään päätöksiä, jotka vähättelevät tietoturvaa varsinkin pitkällä tähtäimellä. Kustannusten nousua, joka aiheutuisi tietoturvaa ymmärtävistä suunnittelijoista, pidetään liian isona. Tämä johtuu suurelta osin myös siitä, että tietoturvaosaajista on huutava pula, sulautettuihin järjestelmiin erikoistuneista tietoturvaosaajista on vielä suurempi pula. Tämä on johtanut palkkioiden nousuun. [68]

Ravi et al. [66] esittävät, että sulautettujen järjestelmien tietoturvassa on juuri sulautetuille järjestelmille ominaisia haasteita. Ne vaativat uusia lähestymistapoja järjestelmien suunnitteluun, alkaen arkkitehtuurista ja päättyen toteutukseen. He listaavat myös useita haasteita sulautettujen järjestelmien tietoturvassa, kuten laskentakyky eli resurssit, energia, liian suuri määrä protokollia ja standardeja, kajoaminen, luotettavuus ja kustannukset.

Whang et al. [34] puolestaan kuvailevat sulautetun tietoturvan suunnittelun erityispiirteitä sanomalla, että sulautetut järjestelmät ovat resurssi- ja energiarajoitteisia, ja että perinteiset työasemien tai palvelimien tietoturva-arkkitehtuurit eivät sovellu, vaikka olisikin hukuttelevaa käyttää suoraan niitä.

Arm Ltd:n kyselytutkimuksen [1] tuloksista selviää, että kuluttajien huoli tietoturvasta on suuri. Vain 21 % vastaajista ei ollut huolestunut tietoturvasta ja vain 10 % sanoi, että tietoturva ei vaikuta ostopäätökseen. Mielenkiintoista tutkimuksessa on myös se, että 32 % sanoo, ettei osaa tunnistaa tai erottaa luotettavia laitteita turvattomista.

3.1 Sulautettujen järjestelmien tietoturvavaatimuksia

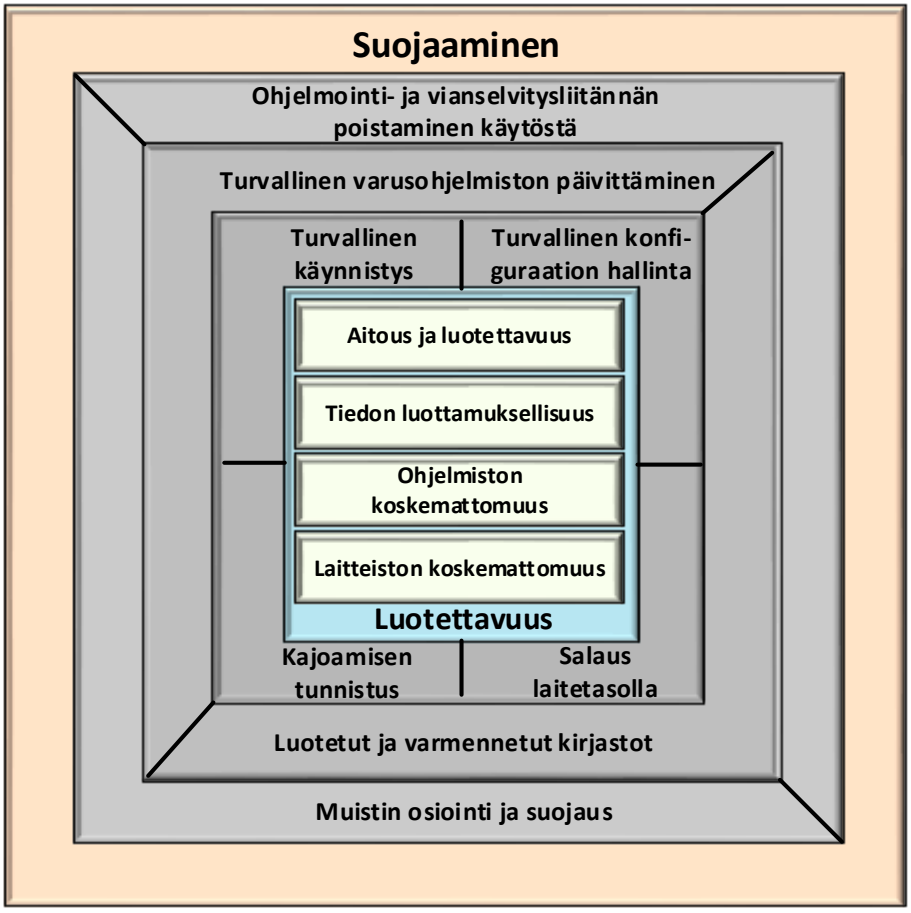
Kuvassa 3.2 esitetään tietoturvan kerroksellinen malli. Keskellä kuvaa sijaitseva osa, luotettavuus, sisältää luotettavuuden perusrakenneosat, jotka ovat:

- Aitous ja luotettavuus, joka tarkoittaa tunnistamisen yksikäsitteisyyttä ja muuttumatonta identiteettiä, väärentämisen ja kopioimisen estämistä sekä varmenteiden suojaamista [90, 47].

- Tiedon luottamuksellisuus, joka sisältää monia asioita, kuten avainten suojaa-

minen, käyttäjätietojen suojaaminen ja tiedon suojaaminen [90].

- Ohjelmiston koskemattomuus, jossa tarkastellaan muun muassa ohjelmakoodin eheyttä, turvallista tapaa siirtää tietoa ja turvallisten ja riskialttiiden ohjelman osien eristämistä toisistaan [90].
- Laitteiston koskemattomuus, joka tarkoittaa sitä, että laitteistoon kajoaminen, kuten laitteen purkaminen, on estetty tai se havaitaan ja siihen voidaan reagoida [90].



Kuva 3.2: Kerroksellinen tietoturva [90]

Edellä kuvattua luotettavuusosiota ympäröivät sitä tukevat laite- ja ohjelmistopalvelut, jotka ovat:

- Kajoamisen tunnistava laitteisto, joka saa tiedon luvattomasta muistiosoitteiden lukemisesta tai kirjoittamisesta laitteen kotelon avaamisesta tai muusta vastaavasta, ja voi laitteistotasolla tyhjentää muistin tai lukita koko mikro-ohjaimen.
- Salausalgoritmeja toteuttavat laitelohko, joka salaa ja purkaa tietoa standardialgoritmeilla ohjelmallisia toteutuksia nopeammin.
- Turvallinen käynnistysohjelmisto (Secure Boot) huolehtii siitä, että mitään vierasta ohjelmaa ei päästetä käynnistymään, kun piiri käynnistää toimintansa.
- Turvallinen konfiguraation hallinta pitää huolen siitä, että mitään laitteen asetustietoja ei voida luvattomasti muuttaa. Asetustietoihin sisältyy tyypillisesti salausavaimia ja salasanoja, joten niitä on suojeltava mahdollisimman hyvin.

Edellisiä ympäröi ohjelmistokerros, joka huolehtii siitä, että varusohjelmisto voidaan päivittää vain luotetuista lähteistä (Secure Firmware Update, SFU). Tähän käytetään tyypillisesti tarkistussummia, tiivisteitä (Secure Hash Algorithm, SHA) ja yksityisen avaimen arkkitehtuuria (Private Key Architecture, PKA), jossa varusohjelmisto tarkistussummineen salataan valmistajan yksityisellä avaimella ja puretaan piirille tallennetulla valmistajan julkisella avaimella, kun varusohjelmiston päivitys suoritetaan. Jos salausta ei saada purettua tai puretun varusohjelmiston tarkistussumma tai tiiviste ei täsmää puretusta varusohjelmasta laskettuun, päivittämistä ei suoriteta loppuun.

Uloimmalla kerroksella olevat ohjelmointi- ja vianselvitysporttien käytöstä poistaminen ja muistin osiointi ja suojaaminen tukevat sisempien kerrosten tuomaa turvaa. Piirin normaali ohjelmointiliitäntä, kuten JTAG tai SWI liitäntä, on poistettava käytöstä pysyvästi. Muutoin piirin ohjelma on mahdollista korvata millä tahansa uudella ohjelmalla ohjelmointiliitännän kautta ja käyttää sitä tallennettujen tietojen, kuten salausavainten ja salasanojen lukemiseen. Muistissa olevien arkojen tietojen suojaamiseksi on monissa mikro-ohjaimissa mahdollisuus osioida muisti erilaisiin alueisiin. Näistä osan voi määritellä sellaisiksi, joista ei voi tietoa lukea, mutta siellä olevaa koodia voi suorittaa. Tällaisen muistialueen avulla on mahdollista toteuttaa monenlaisia suojauksia.

3.2 Uhkat ja ongelmat

Sulautettujen järjestelmien kenttä kasvaa nopeasti muun muassa sellaisilla laitteilla kuin matkapuhelimet, älykortit, puettavat tietokoneet ja erilaiset verkotetut sensorit. Niitä halutaan suojata monestakin syystä. Yksi syy on pitää laitteen toiminta sellaisena, kuin sen valmistaja sen halusi olevan. Ulkopuolisen tahon muuttama ohjelma voi olla laitteen käyttäjälle vaarallinen tai turvaton. Se voi myös toimittaa laitteeseen tallennettua tietoa ulkopuolisille tai tehdä asioita, joita alkuperäinen ohjelmisto ei esimerkiksi turvallisuuden takia tehnyt. Etenkin ihmisen elintoimintoja ylläpitävän laitteen, kuten sydämen tahdistimen, ohjelmiston vakaus ja asiallinen toiminta pitää turvata estämällä asiaton ohjelmiston ja siihen vaikuttavan konfigurointitiedon muuttaminen niin hyvin kuin mahdollista [34].

Sulautettu järjestelmä voi olla myös esimerkiksi sensori, joka vedenottamossa mittaa veden laatua bioterrorin varalta. Sillä tulee tällöin olla useita keinoja estää sekä ohjelmistoon että laitteistoon kajoaminen, jotta hyökkääjä ei pääse ohittamaan turvatoimia ja pilaamaan vedenottamon vettä. [34]

3.2.1 Kopioiminen

Sekä laitteiston että ohjelmiston kopioiminen on uhka monellakin tavalla. Kopioiminen uhkaa kehitystyön tehnyttä yritystä taloudellisesti, koska mahdollisesti suuretkin tuotekehityskulut jäävät toteutumatta. Tämän takia kopioija voi myydä tuotettaan halvemmalla ja säästyy tuotekehityskuluilta.

Toinen potentiaalisesti hyvinkin vakava uhka koituu käyttäjille, sillä tuoteturvallisuus voi olla kopiotuotteessa huonompi, etenkin jos komponentit eivät ole juuri samoja kuin alkuperäisessä tuotteessa.

Kolmas uhka syntyy laitteiden sisältämien tietojen paljastumisesta kopioinnin yhteydessä. Laite voi sisältää salausavaimia tai arvokkaita algoritmeja.

3.2.2 Toiminnan muuttaminen

Jos kopioija lisäksi muuttaa laitteen toimintaa joko ohjelmiston tai laitteiston muutoksin, voivat seuraukset olla arvaamattomat. Laitteen toiminta voi muuttuu huonommaksi ja ehkä myös paremmaksi, huonossa tapauksessa (laitteesta riippuen) jopa haitalliseksi tai vaaralliseksi. Muuttaminen voi myös aiheuttaa arkojen tietojen paljastumista, sillä muutettu ohjelma pääsee käsiksi laitteen sisältämään arkaan tai

salaiseen tietoon ja välittää sen mahdollisesti eteenpäin.

Laitteen rikkoutuminen on muutosten seurauksena mahdollista, sillä laitteen jännitteiden ja virtojen hallinta on usein ohjelmiston hallinnassa. Laitteen toiminta voi muutoksien seurauksena muuttua epävakaaksi tai jopa loppua kokonaan, jos esimerkiksi kajoamisen hallinta huomaa muutokset ja poistaa kaikki laitteen tiedot ja ohjelmiston.

3.2.3 Takaisinmallinnus

Takaisinmallinnus (Reverse Engineering) määritellään eri lähteissä eri tavoin, esimerkiksi Chikofsky ja Cross määrittelevät sen artikkelissaan [21] olevan prosessi, jossa tunnistetaan järjestelmän komponentit ja niiden keskinäiset suhteet sekä luodaan järjestelmän kuvaus alkuperäisestä poikkeavassa muodossa tai korkeammalla abstraktiotasolla. McLoughlinin [52] mukaan taas takaisinmallinnus on prosessi, jossa laitteen toiminnallisuus, arkkitehtuuri ja käytetty teknologia analysoidaan siten, että on mahdollista uudelleenkäyttää niitä vastaavanlaisen laitteen toteutuksessa. Toisessa julkaisussa McLoughlin [53] kuvailee kuluttajalaitteen takaisinmallinnuksen olevan systeemin toiminnan, arkkitehtuurin ja tekniikka kartoittamista niin, että on mahdollista monistaa alkuperäisen tuotteen arkkitehtuuri tai tekniikka.

Laitevalmistajat uhraavat paljon rahaa tutkimukseen ja tuotekehitykseen suunnitellessaan uusia sulautettuja laitteita. Rahat oletetaan saatavan takaisin, kun laite aikanaan tulee markkinoille mahdollisesti ainoana laatuaan [53]. Jos joku tuo markkinoille samanlaisen laitteen halvemmalla ennen kuin tuotekehitys- ja tutkimusrahat on ansaittu takaisin, voivat tappiot muodostua hyvinkin suuriksi. Kilpailija on voinut tehdä samanlaisen laitteen itsekin, mutta mahdollisesti myös kopioimalla tuotteen käyttämällä takaisinmallinnusta.

Jos kilpailija saa tehtyä laitteen huomattavasti pienemmillä tuotekehityskuluilla, voi se määritellä hinnan huomattavasti halvemmaksi kuin alkuperäisellä tuotteella oli ja siten vallata suuren markkinaosuuden. Tähän se on voinut päästä käyttämällä takaisinmallinnusta aidon tuotekehitysprojektin sijaan. [53]

Useimmiten takaisinmallinnuksen motiivina on taloudellinen hyöty; saatua tietoa voidaan hyödyntää lyhentämään kehitysaikaa, samalla lyhentäen aikaa, joka kuluu suunnittelun alusta siihen, kun tuote pääsee markkinoille [53]. Takaisinmallinnuksen tavoitteena voi olla myös lisätä ymmärrystä järjestelmästä, jotta sitä olisi helpompi pitää yllä ja kehittää [21]. Selvitettäviksi asioiksi lähteet [21, 52, 53, 61, 87] listaavat muun muassa seuraavia:

- Toiminnan selvittäminen
- Järjestelmän tai sen osan monistaminen tai kopioiminen
- Järjestelmän toiminnan muuttamisen mahdollistaminen
- Liikesalaisuuksien paljastaminen
- Tiedonsaanti ja oppiminen
- Järjestelmän validointi tai verifiointi
- Järjestelmän turvallisuuden tarkastaminen, esimerkiksi kansallisen turvallisuuden näkökulmasta
- Kadonneen tiedon takaisin saaminen toimimattomista järjestelmistä
- Mahdollisten sivuvaikutusten analysointi
- Uudelleenkäytettävien ohjelmiston osien tunnistaminen

Takaisinmallinnus voidaan aloittaa esimerkiksi selvittämällä laitteen toiminta käyttämällä sitä mahdollisimman monipuolisesti, tutkimalla käyttöohjeita, huolto-ohjekirjoja, tuote-esitteitä ja kaikkia muita mahdollisia tietolähteitä. Tuloksena on lista tuotteen toiminnallisuuksista. Sitä käytetään myöhemmin tarkistuslistana, kun verrataan kopiolaitteen toimintaa alkuperäiseen. [52]

Laite puretaan ja osien sijainti ja järjestys dokumentoidaan vaikkapa valokuvin. Laitteen purkaja merkitsee muistiin kaikki purkamisen aikana syntyvät huomiot. Laitteesta tehdään mekaniikka- ja asennuskuvat. Laitteesta laaditaan osaluettelo, joka kertoo, mistä komponenteista se koostuu tai on valmistettu [53]. Osa käytetyistä komponenteista voi olla vaikea tunnistaa, sillä laite saattaa sisältää laitetta varten suunniteltuja ja valmistettuja mikropiirejä. Passiiviset komponentit ja tavanomaiset puolijohteet on helpohko tunnistaa vaikkakin pintaliitososien rajallinen pinta-ala on johtanut siihen, että merkintöjä niiden pintaan ei juuri enää mahdu.

Järjestelmän arkkitehtuurin analysointi paljastaa eri toimintoja vastaavien alijärjestelmien karkean rakenteen. Tässä vaiheessa on tärkeätä tunnistaa mahdollinen ohjelmointi- tai debug-liitäntä, jota voidaan hyödyntää ohjelmiston kopiointiin ja muuttamiseen [42].

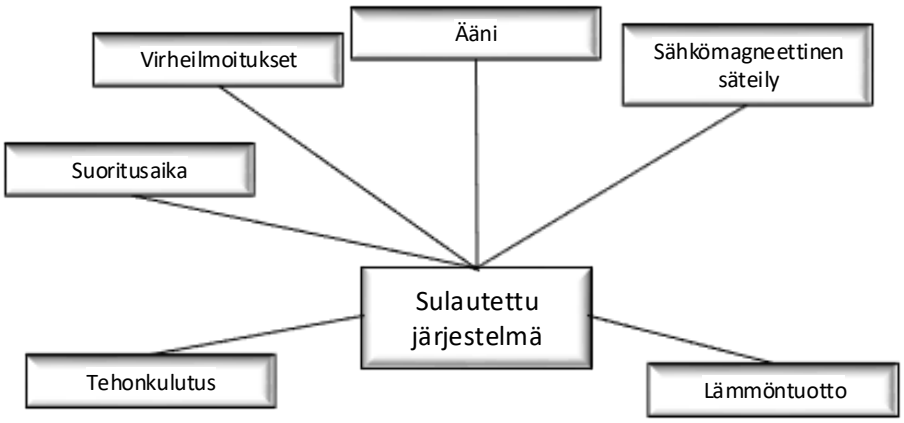
Piirilevyn rakenteen selvittäminen tehdään järjestyksessä alkaen ulkopuolelta ja päätyen sisimpiin osiin. Komponentit poistetaan ja reikien paikat merkitään ylös. Piirilevyn rakenne hajotetaan erottamalla eri kerrokset toisistaan kuorimalla kerrokset irti toisistaan. Näin saadaan lähes täydellisesti piirilevyn johdotukset dokumentoitua. Myös röntgenkuvausta voidaan hyödyntää piirilevyn rakenteen selvittämiseen [15]. Kytkentäkaavio saadaan tehtyä piirilevydokumentaation ja komponenttien datalehtien avulla, joista yleensä löytyy esimerkkikytkentä tyypilliseen käyttöön [53].

Laitteen varusohjelman voi saada laitteen haihtumattomasta muistista, joka sisältää yleensä sekä mikrokontrollerin että muiden ohjelmakoodia vaativien komponenttien koodin. Jos erillistä haihtumatonta muistipiiriä ei ole, on todennäköistä, että koodi on mikrokontrollerin sisäisessä muistissa. Tällöin koodi voi olla suojattu monin eri tavoin lukemiselta ja koodin selvittäminen voi olla huomattavan työlästä ja hankalaa. Kun koodi on saatu, siitä voidaan muuttaa halutut kohdat, kuten valmistajan tunnisteet ja sarjanumerot. [61]

Näiden vaiheiden jälkeen on laite saatu kopioitua täydellisesti. Prosessi voi olla hyvinkin helppo, mutta myös erittäin vaikea, jos laitteen valmistaja on suojannut laitteen monipuolisesti.

3.2.4 Sivukanava-analyysi

Sivukanava-analyysissä pyritään selvittämään laitteen toimintaa tutkimalla muun muassa laitteen aiheuttamaa sähkömagneettista säteilyä, tehonkulutusta sekä toimintojen suoritusaikoja ja selvittämällä niitä hyväksi käyttäen arkaluonteisia tietoja, kuten salausavaimia ja viestejä [72]. Kuvassa 3.3 on esitettynä muutamia mahdollisia sivukanavia. Sivukanavaa hyödyntämällä hyökkääjän ei tarvitse kajota laitteeseen ollenkaan vaan voi saada selville arkaluonteista tietoa havainnoimalla ilmiötä laitteen ulkopuolelta. [30]



Kuva 3.3: Sivukanavat [31]

Sulautettu järjestelmä, kuten muutkin järjestelmät, säteilevät sähkömagneettista sä-

teilyä. Mikro-ohjaimen sisäisten porttipiirien tilan muutokset aiheuttavat lyhyitä virtapiikkejä, ne puolestaan aiheuttavat sähkö- ja magneettikentän muutoksen eli sähkömagneettisen kentän muutoksen. Tällaisia kenttiä voidaan havainnoida lähikenttäantureilla (Near-Field Electromagnetic Probe). [31, 43]

Tehoanalyysihyökkäys perustuu siihen, että laitteen virrankulutus on laitteen sisäisten kytkentätoimenpiteiden funktio ja on siten riippuvainen käsiteltävästä tiedosta. Kun suuri määrä jatkuvia virrankulutusmittauksia on käytettävissä, on mahdollista saada selville luottamuksellista tietoa analysoimalla tätä tietoa. [72]

Koodin samanlaisuuden testaaminen voidaan suorittaa vertaamalla kahden laitteiston virrankulutusta, kun ne suorittavat samaa tehtävää. Tällä tavalla voidaan havaita, onko suoritettava koodi kokonaan tai osittain samaa, esimerkiksi luvatta kopioitu alkuperäisestä laitteesta. [43]

Laitteen suorittamaa koodia voidaan muuttaa laskemalla sen käyttöjännitettä juuri sopivassa kohdassa käskyn suoritusta. Tällöin käskyn suoritus häiriintyy ja sen muuttunut tulos käsitellään seuraavassa suoritettavassa käskyssä. [43]

3.3 Suojautuminen ja toipuminen

Edellisessä luvussa esitettiin uhkia ja ongelmia, tässä luvussa esitellään tapoja suojautua tai toipua niiltä.

3.3.1 Kopioinnilta suojautuminen

Kopioinnilta ei ole mahdollista suojautua täysin, mutta se on mahdollista tehdä vaikeammaksi, hitaammaksi ja riittävän kalliiksi toteuttaa, jotta kopiointi ei olisi kannattavaa. Myös suojautumiseen liittyvät toimenpiteet maksavat, joten on löydettävä tasapaino mahdollisien kopioinnista aiheutuneiden tulojen menetysten ja suojautumiseen käytetyn panoksen välillä.

Vianselvitys- ja ohjelmointiliitäntöjen poistaminen käytöstä sekä laitteen käynnistämisen estäminen muualta kuin sisäiseltä ohjelmamuistilta tekee laitteen ohjelmiston kopioinnin vaikeaksi. Kajoamisen tunnistamismekanismeilla voidaan ohjelmistoa myös suojella niin, että kajoamisen havaittuaan laite tyhjentää ohjelmansa pysyvästi ja peruuttamattomasti. [40]

Käytössä olevat ulkoiset kommunikointiväylät pitää suojata niin, että siellä kulkevaa tietoa ei ole mahdollista ulkopuolisten lukea, eikä uutta tietoa sinne kirjoittaa

vaikkapa käyttämällä jotain salausalgoritmia liikenteen salaamiseen. Tällaisen väylän kautta saattaa muutoin olla mahdollista myöskin päivittää laitteen ohjelmisto.

Piirilevyn ja laitteen kytkentäkaavion suojaaminen on vaikeampaa. Kopiointia voi vaikeuttaa esimerkiksi valamalla laitteen sisälle jotain kovaa eristävää massaa, kuten valumuovia, ja suunnittelemalla piirilevy niin, että mahdollisimman suuri osa vedoista kulkee piirilevyn sisäkerroksissa [81, 53].

3.3.2 Toiminnan muuttamiselta suojautuminen

Toiminnan muuttamiseksi on laitteen ohjelmaa päästävä muuttamaan, joten suojautuminen edellyttää laitteiston suojaamista niin, että ei ole mahdollista käynnistää laitetta muulta kuin sisäiseltä muistilta, sekä ohjelmiston eheyden ja muuttumattomuuden valvontaa salaus- ja tarkistussumma-algoritmeja käyttämällä.

Hyvin tärkeää on myös suojata laite ohjelman päivittämiseltä muulla tavoin kuin valmistajan avaimilla allekirjoitetuilla ohjelmistoilla. Tietenkin myös laitteen ohjelmointiliitännät on ohjelmoitava pysyvästi pois käytöstä. Myös kajoamisen esto- ja tunnistus auttavat ohjelmiston muuttamisen ehkäisyssä.

3.3.3 Takaisinmallinnukselta suojautuminen

Takaisinmallinnukselta voidaan suojautua laitetasolla esimerkiksi jättämällä piirilevyn ylä- ja alapinnan painatukset (Silk Screen) pois. Kun käytetään BGA-koteloisia (Ball Grid Array) mikropiirejä on piirilevyn johdotuksia vaikeampi seurata. Piirilevyvetojen reitittäminen kokonaan välikerroksilla ja ylä- ja alapintojen varaaminen pelkästään jännite- ja maatasoiksi suojaa piirilevyn suunnittelua ja kytkentäkaaviota ja ne muodostavat kuparisen 'näkösuojan'. Kun vielä käytetään haudattuja läpivientejä (Buried Vias), otetaan JTAG- ja muut vianselvitys- ja ohjelmointiliitännät kokonaan pois käytöstä ja poistetaan komponenteista kaikki merkinnät, on takaisinmallintajan työtä vaikeutettu jo merkittävästi [53].

Piirilevyn takaisinmallinnusta röntgenkuvaamalla voi yrittää haitata käyttämällä materiaaleja ja komponentteja, jotka eivät kestä röntgensäteilylle altistamista tai käyttämällä säteilyn tunnistavia komponentteja ja käynnistämällä jonkinlainen itsetuhomekanismi, kun tietty säteilyannos on ylitetty [15].

3.3.4 Sivukanava-analyysin vaikeuttaminen

Tuotepiratismista ja IP-oikeuksien loukkauksista on tullut suuri huolenaihe monille teollisuuden aloille, niin myös sulautettujen järjestelmien valmistajille. Niitä vastaan yritetään suojautua eri keinoin, mutta myös niiden tunnistaminen on yhä tärkeämpää.

Sivukanavailmiöitä voidaan hyödyntää tunnistamaan ohjelmistokopiointi käyttämällä niin sanottua ohjelmistovesileimaa. Se on ohjelmiston toiminta, joka voidaan havaita sivukanavan kautta ohjelmistoa käytettäessä ja siten todeta laitteeseen kajoamatta, onko kopioksi epäilty laite samanlainen kuin laite, jonka kopioksi epäiltyä laitetta luullaan [18].

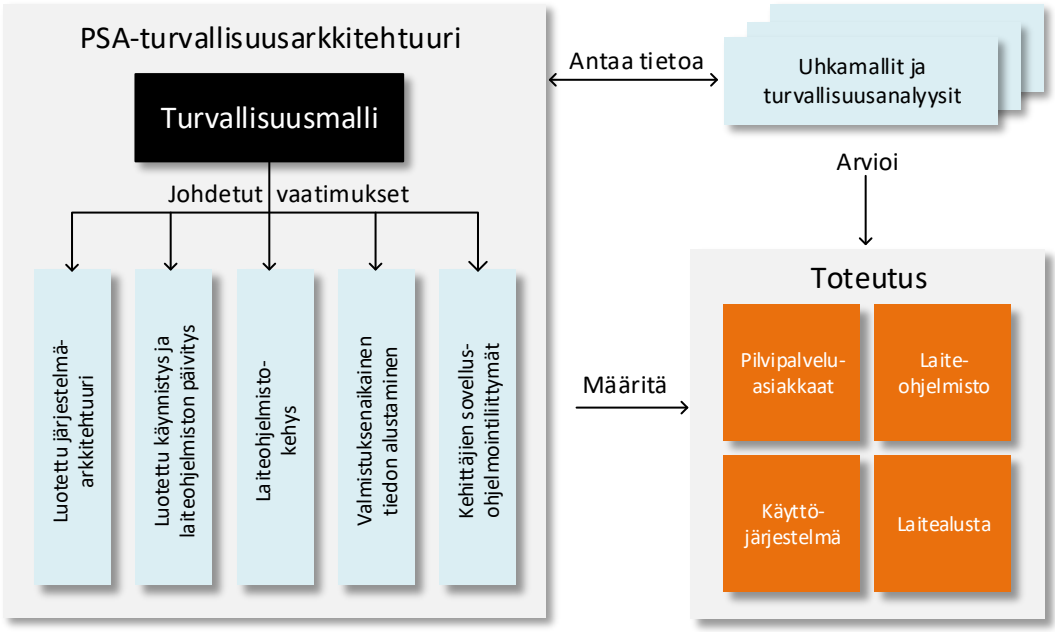
3.4 ARM Ltd:n tarjoamia turvaominaisuuksia

Arm Ltd tarjoaa piirinvalmistajille erilaisin turvaominaisuuksin varustettuja ytimiä. Niitä on ovat muun muassa Cryptocell-300 ja Cryptocell-700 perheet, Arm Crypto Island perhe, Cortex-M35P, SecureCore SC300 ja SC000 perheet sekä tekniikkoina tarjottavat TrustZone Technology for Processor IP ja TrustZone Security System IP [10].

Tavanomaisiin ytimiin, kuten esimerkiksi Cortex-M4 -piireissä käytettäviin ytimiin, on Arm Ltd valmiiksi määritellyt vain muistin suojausyksikön (Memory Protection Unit, MPU). Muut turvaominaisuudet ovat valmistajakohtaisia ja siten eri valmistajilla on hyvinkin erilainen turvaominaisuuksien tarjonta.

3.4.1 PSA turvallisuusarkkitehtuuri

Turvallisuuden toteuttaminen voi olla kallista koko laitteen elinkaari huomioiden. Hyvin suojattuja laitteita on myös vaikea hallita isossa mittakaavassa. Turvallisuusasiantuntijat ovat kalliita ja heistä on pulaa. Tämä vaikeuttaa erityisesti pienien ja aloittavien yrityksien toimintaa. Turvallisuus on kokoajan liikkuva maali ja ala kehittyy jatkuvasti, uusia haavoittuvuuksia syntyy koko ajan.



Kuva 3.4: PSA-turvallisuusarkkitehtuuri [6]

Arm Ltd:n PSA-turvallisuusarkkitehtuuri (Platform Security Architecture), jota on havainnollistettu kuvassa 3.4, tekee tietoturvallisen laitteen suunnittelun nopeammaksi, helpommaksi ja halvemmaksi. Se perustuu alan parhaisiin käytäntöihin, ja siinä määritellään yhteiset suunnittelua ohjaavat turvallisuusvaatimukset. PSA:ssa turvallisuussuunnittelu tehdään neljässä vaiheessa, jotka ovat analysointi, arkkitehtuuri, toteutus ja sertifiointi. [6]

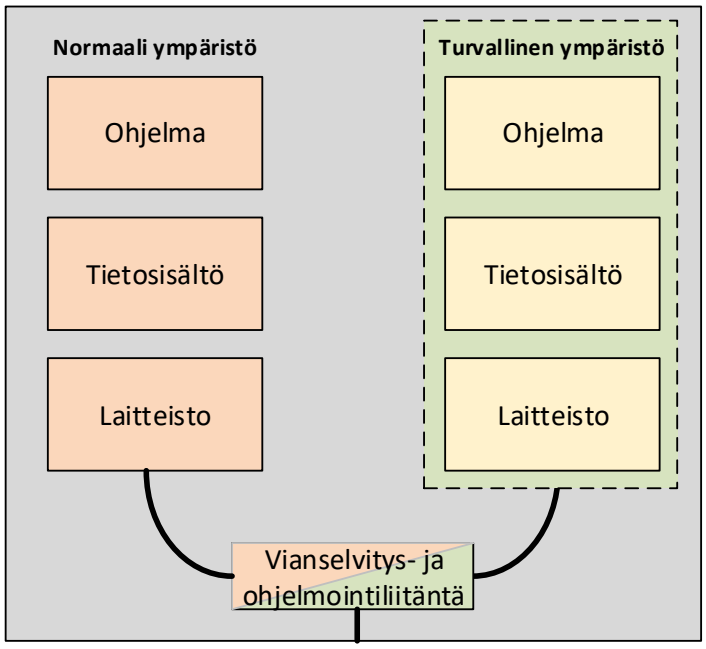
Analysointivaiheessa laitteen omaisuuksia arvioidaan ja se on koko turvallisuussuunnittelun lähtökohta. Siinä arvioidaan suojattavat resurssit, kaikki mahdolliset uhkat, uhkien laajuus ja vakavuus, hyökkäystyypit ja -menetelmät, joita niissä voidaan käyttää haavoittuvuuksien hyödyntämiseen. Laitteeseen kohdistuvien uhkien arviointi on välttämätöntä, ja sen avulla voidaan määrittää selkeät laitekohtaiset turvallisuusvaatimukset.

Toteutusvaiheessa käytetään pohjana avoimen lähdekoodin referenssitoteutusta, joka täyttää arkkitehtuurivaiheessa asetetut vaatimukset. Laitteistosuunnittelijat voivat käyttää luotettavaa laiteohjelmistoa (Trusted Firmware-M) luotettavana lähtökohtana, josta lähteä liikkeelle. Sertifiointi tehdään sertifiointijärjestelmän mukaisesti. Se tarjoaa monitasoisesti takeet laitteille analysointivaiheessa vahvistettujen turvalli-

suusvaatimusten toteutumisesta. [11]

3.4.2 ARMv8-M TrustZone arkkitehtuuri

Kuvassa 3.5 esitellään uudemmissa ARMv8-M arkkitehtuurin mikro-ohjaimissa, kuten Cortex-M23, Cortex-M33, Cortex-M35 ja Cortex-M55, käytettävissä olevan TrustZone arkkitehtuuri, jossa mikro-ohjaimen ytimeen on lisätty toinen rekisteriryhmä, jota käytetään turvallisessa ympäristössä, ja muita tietoturvaa avustavia yksiköitä niin, että 'turvallinen' ja 'normaali' toiminnallisuus voidaan erottaa täysin toisistaan [14, 13]. Vianselvitys- ja ohjelmointiliitäntä ovat yhteisiä molemmille ympäristöille.



Kuva 3.5: TrustZone arkkitehtuuri [85]

TrustZone-ympäristössä voidaan ajaa yhtaikaa turvallista ohjelmistoa, kun tavallisessa ympäristössä ajetaan jotain muuta sovellusta. Ympäristöt ovat toisistaan kokonaan erillisiä, ja ne eivät pääse käsiksi toistensa tietoihin, ellei sitä erikseen sallita. [85]

Muistin voi jakaa näiden toimintojen kesken ja ne eivät pysty lukemaan toistensa muistialueita. Kommunikointi toimintojen välillä tehdään tätä tarkoitusta varten lisätyllä käskyllä SCM (Secure Monitor Call).

3.5 STMicroelectronicsin STM32-perheen turvaominaisuuksia

Tämän aliluvun tiedot perustuvat STMicroelectronicsin datalehtiin, teknisiin raportteihin ja seminaarimateriaaleihin [74, 90]. STMicroelectronicsin ARM-mikro-ohjaimissa on mallista riippuen monipuolisesti erilaisia turvaominaisuuksia. Ne luokitellaan joko staattisiksi tai dynaamisiksi. Staattiset ominaisuudet kytketään päälle optiotavujen (Option Bytes) bittejä ohjelmoimalla. Niihin kuuluvat muistin lukusuojausyksikkö (Readout Protection, RDP), muistinsuojausyksikkö (Proprietary Code Read Out Protection, PCROP), muistin kirjoitussuojausyksikkö (Write Protection, WRP), alijännitteeltä suojaava yksikkö (Brownout Reset, BOR) ja turvapiilo (Secure Hide Protection, SHP).

Dynaamisia eli ajonaikaisesti päällekytkettäviä turvaominaisuuksia ovat muistinsuojausyksikkö (Memory Protection Unit, MPU), kajoamisen tunnistusyksikkö (Tamper Detection) ja palomuuri (Firewall, FW). Ne on kytkettävä päälle joka kerta laitteen käynnistyessä, eikä mikro-ohjain muista niiden edellistä tilaa.

3.5.1 STM32L4-perheen turvamekanismeja

Seuraavassa esitellään STMicroelectronicsin STM32L4 mikro-ohjaimen tietoturvaan liittyviä ominaisuuksia. L4-sarja on 80MHz:n M4 ytimellä varustettu mikro-ohjainperhe, jossa on suuri määrä turvaominaisuuksia.

Muistinsuojausyksikkö (Memory Protection Unit, MPU) jakaa muistin jopa kahdeksaan erilliseen alueeseen, joille voidaan erikseen määritellä suojaustaso. Suojaustaso voi olla: Ei pääsyä (No Access), luku (Read Only, RO), kirjoitus (Write), suoritus (Execute) tai näiden sopiva yhdistelmä. Jos suojattua aluetta yritetään lukea, kirjoittaa tai suorittaa määrittelyjen vastaisesti, tapahtuu joko niin sanottu kova virhe (Hard Fault) tai ydin lukitaan kokonaan.

Palomuuri (Firewall, FW) muodostaa luotettuja muistialueita (Trust Area), jotka ovat suojattuja kaikista muista ohjelman alueista. Suojatun muistialueen käsittely muista ohjelmista aiheuttaa mikro-ohjaimen uudelleenalustuksen eli resetin. Kun alue on luotu, se on aktiivinen seuraavaan uudelleenkäynnistykseen asti. Se on käytettävissä sekä työ- että ohjelmamuistille.

Muisti voidaan suojata (Readout Protection, RDP) kolmella eri tasolla: Tasolla nolla suojausta ei ole, joten muistin luku, kirjoitus ja tyhjennys ovat mahdollisia prosessorin ulkopuolisia liitäntöjä käyttäen. Tasolla yksi sekä ohjelma- että työmuistin lukeminen, kirjoittaminen ja tyhjennys on estetty. Tasolla kaksi kaikki tason yksi

rajoitukset ovat voimassa ja lisäksi käynnistäminen ei ole mahdollista muualta kuin ohjelmamuistista. JTAG-ohjelmointiliitäntä on poistettu käytöstä, joten muistia ja prosessorin muitakaan tietoja ei voi lukea prosessorin ulkopuolta. Tasolta kaksi ei ole mahdollista palata takaisin vähemmän suojatuille tasoille.

Ohjelmamuistin (FLASH) osa on mahdollista suojata (Proprietary Code Read Out Protection, PCROP) niin, että sinne sijoitettua koodia voidaan suorittaa, mutta ei lukea eikä kirjoittaa. Työmuistia ei voi suojata samalla lailla. Virheenkorjaava muisti (Error Code Correction, ECC) on muistityyppi, jossa lasketaan kahdeksanbittinen virheenkorjauskoodin jokaista 64 bittiä kohti. Sen avulla on mahdollista korjata yksittäisen bitin virhe tai havaita kaksi virhettä. Korjatusta ja havaitusta virheestä voidaan asettaa tapahtumaan keskeytys. Havaittu virhe aiheuttaa ei-estettävän keskeytyksen (Non Maskable Interrupt, NMI).

Jokaisella prosessorilla on yksilöllinen 96-bittinen tehtaalla asetettu sarjanumero (Unique Device Identifier, UDI) joka ei toistu useisiin vuosiin prosessoreiden valmistuksessa. Sitä voidaan käyttää muun muassa laitteen tunnistamiseen (Authentication) tai osana salausalgoritmien avainta. AES-salakirjoitusyksikkö on yksikkö, joka osaa salata ja purkaa AES (Advanced Encryption Standard) algoritmin mukaisesti 128- ja 256-bittisillä avaimilla.

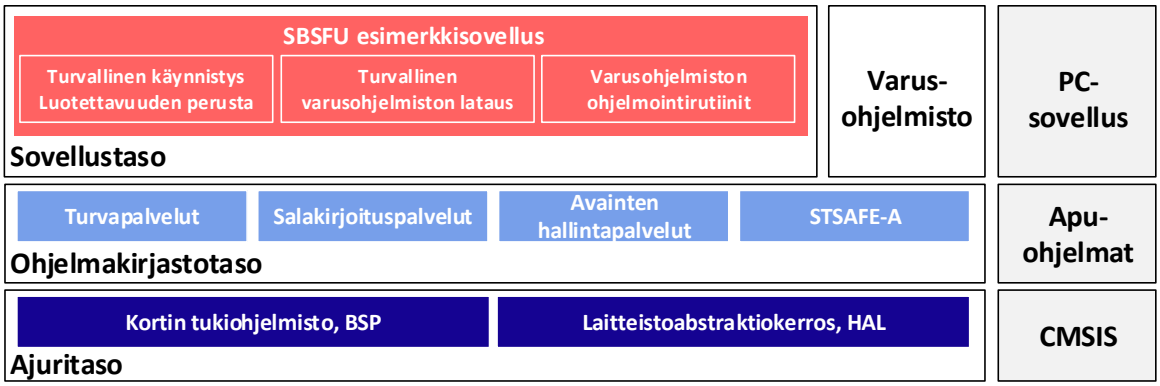
SHA-tiivisteiden laskentayksikkö on yksikkö, joka osaa laskea SHA-1, SHA-224, SHA-256 ja MD5 -tiivisteitä. Yksikköä voi käyttää myös yhdessä DMA:n kanssa. Satunnaislukugeneraattori (True Random Number Generator, TRNG) puolestaan on yksikkö, joka tuottaa 32-bittisiä satunnaislukuja. Yksikkö tuottaa satunnaislukuja analogisen kohinalähteen tuottaman kohinan satunnaisuuteen tukeutuen. Kajoamisen tunnistusliitäntä on liitäntäyksikkö, joka pystyy tyhjentämään haihtumattoman varmuuskopiomuistin (Backup Domain), joka on eräs reaaliaikakellon paristovarmennetuista muistialueista. Sinne kirjoitetut tiedot säilyvät, vaikka laite on uudelleenkäynnistetty tai virta katkaistu niin kauan kuin reaalikellon paristossa riittää virtaa, eli tyypillisesti vuosia.

3.5.2 Suojattu käynnistysohjelma ja suojattu varusohjelman päivitys

STMicroelectronics on julkaissut ohjelmistopaketin, nimeltään *'X-CUBE-SBSFU Secure Boot and Secure Firmware Update'*. Se on tarkoitettu referenssitoteutukseksi, jonka pohjalta laitteiden suunnittelijat voivat lähteä kehittämään omia suojausratkaisuaan. Sellaisenaankin se on käyttökelpoinen turvaominaisuuksien kokeiluun ja testaamisen. Se muodostaa alustan, jolla voidaan toteuttaa turvallinen käynnistys, turvallinen

varusohjelman päivitys, tietojen salaus ja avainten hallinta. Ohjelmapaketin voi ladata STMicroelectronicsin web-sivuilta [77]. Suojatun käynnistyksen ja suojatun varusohjelman päivittämisen ohjelmistopaketin toimintaa on kuvailtu liitteessä A.

Ohjelmistopaketti sisältää tietoturvallisen laitteen rakentamiseen tarvittavia toimintoja sekä joukon esimerkkiprojekteja STMicroelectronicsin kokeilukorteille. Ohjelmapaketin yleiskuvaus on nähtävissä kuvassa 3.6. Ylimpänä siinä esitetään sovellustaso, jolla sijaitsee SBSFU esimerkkisovellus. Tämä koostuu kolmesta osasta, joista ensimmäinen on 'Turvallinen käynnistys', joka on samalla luotettavuuden perusta (Root of Trust). Toinen osa on 'Turvallinen varusohjelman lataus' ja kolmas on 'Varusohjelman ohjelmointirutiinit'.



Kuva 3.6: X-CUBE-SBSFU ohjelmistopaketin yleiskuvaus [77]

Näiden lisäksi tarvitaan myös varsinainen sovellus, eli varusohjelmisto, jonka SBSFU:n turvalliset lataus- ja ohjelmointirutiinit kirjoittavat piirin muistille. Tätä kokonaisuutta STMicroelectronics nimittää sovellustasoksi.

Kuvassa keskimmäisenä sijaitsee sovellustaso. Se käyttää hyväkseen ohjelmakirjastotasoa, joka sisältää turvapalvelun (Secure Engine, SE), salakirjoituspalvelun (Cryptography), avainten hallintapalvelun (Key Management Services, KMS) ja tuen STSAFE-A100 -piirille, joka tarjoaa autentikointi- ja allekirjoitupalveluja.

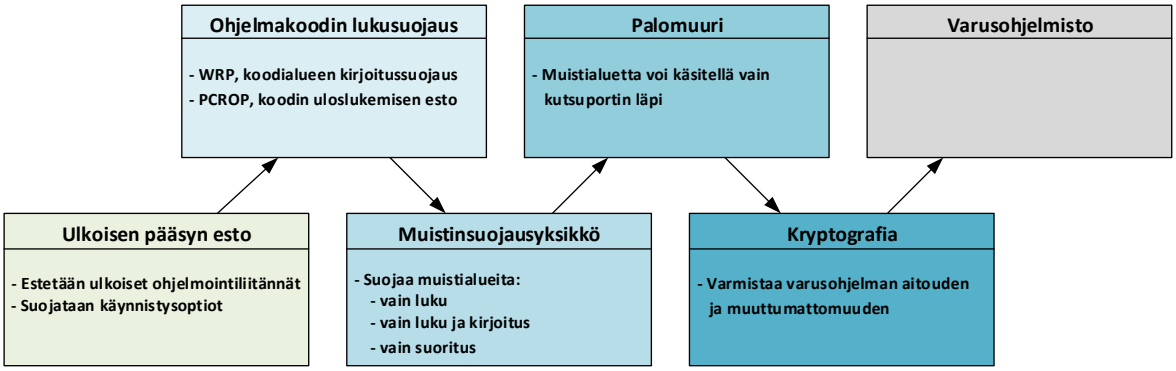
Alimpana sijaitsee ajurikerros, jolla sijaitsevat kortin tukiohjelmisto (Board Support Package, BSP) ja laitteistoabstraktiokerros (Hardware Abstraction Layer, HAL). BSP tarjoaa käytössä olevalle laitteistolle sopivan ohjelmointirajapinnan, HAL puolestaan toimii ohjelmiston ja BSP:n välissä kerroksena, joka yhtenäistää eri laitteiden käyttämiseen tarvittavat rajapintakutsut. Ajurikerrokseen kuuluvaksi voidaan lukea myös CMSIS (Cortex Microcontroller Software Interface Standard), joka on ARM

Ltd:n määrittelemä valmistaja- ja piirikohtaisia eroja poistava ohjelmakirjasto, jonka kautta erilaisia ARM-prosessoreita voidaan ohjelmoida yhteisen rajapinnan kautta.

Seuraavissa aliluvuissa käsitellään tarkemmin STMicrolectronicsin SBSFU:ta käyttöohjeen [75] tietoihin pohjautuen. Käsittelyssä keskitytään suojattuun ohjelmistopäivitykseen ja käynnistykseen, arkojen tietojen suojaamiseen ja ohjelmointiliitäntöjen käytöstä poistamiseen.

Suojattu käynnistysohjelma

Suojattu käynnistysohjelma (Secure Boot, SB) takaa käynnistettävän laitteen käynnistysohjelman koskemattomuuden ja aitouden sekä luo samalla kuvassa 3.7 esitetyn luotettavuusketjun juuren eli luotettavuuden perustan (Root of Trust, RoT), jonka päälle kaikki muut turvakomponentit rakentavat lisää turvallisuutta ja luotettavuutta.



Kuva 3.7: Luotettavuusketju

Suojattu käynnistysohjelma tehdään mikro-ohjaimen ainoaksi mahdolliseksi käynnistystavaksi. Se käynnistyy aina kun mikro-ohjain uudelleenkäynnistetään, eikä sitä voi käynnistää muulla tavoin. Sen koodi on muuttamatonta (Immutable) ja sillä on hallittu pääsy arkoihin tietoihin kuten allekirjoitusavaimiin.

Suojattu käynnistysohjelma tarkistaa laitteen käynnistyessä mikro-ohjaimen turvallisuuteen ja suojamiseen liittyvät asetukset, kuten palomuurin (FW) ja muistinsuojausyksikön (MPU) ja tarvittaessa korjaa väärin olevat asetukset oikeiksi. Myös varusohjelmiston allekirjoituksen oikeellisuus tarkistetaan.

Suojattu ohjelmistopäivitys tarjoaa turvallisen tavan päivittää laitteen ohjelmisto

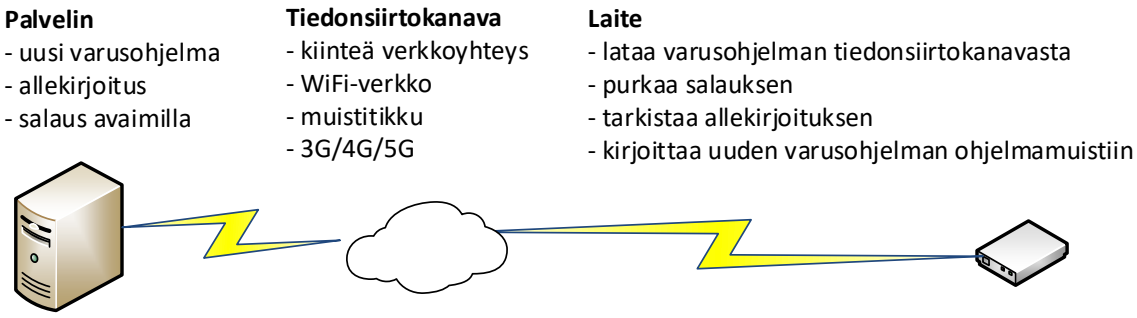
myös kentällä. Myös langaton päivitys, esimerkiksi miehittämättömissä sijoituspaikoissa, on mahdollinen, kuten myös tavanomaiset päivitysmenetelmät käyttäjän käynnistäminä.

Suojattu varusohjelmiston päivitys

Laitteen suojaaminen ohjelmiston asiattomalta muuttamiselta on SBSFU:ssa toteutettu siten, että varusohjelman päivittäminen mahdollistetaan vain tarkastetulla sisällöllä (Firmware Integrity) ja vain tunnetuista lähteistä (Firmware Authenticity). Lisäksi mikro-ohjaimen ohjelmointi on estetty JTAG- tai SWD-ohjelmointiliitännän kautta, jolloin ainoa mahdollisuus päivittää varusohjelmisto on tehdä päivitys SBSFU:n tarjoaman turvallisen päivitystavan avulla. Varusohjelmisto koskemattomuus varmistetaan päivitystapahtuman aikana, jotta voidaan varmistua siitä, että suoritettava ohjelma ei ole vioittunut tai sitä ei ole muutettu. Se on toteutettu laskemalla varusohjelmasta tarkistussumma tai tiiviste. Tarkistussumma ja varusohjelman binääri lisätään ohjelmiston päivityspakettiin ja mikro-ohjaimessa sijaitseva suojatun ohjelmistopäivityksen toteutus hylkää päivityspaketin, jos summa tai tiiviste ei täsmää.

Aitouden varmentaminen pyrkii varmistamaan sen, että varusohjelma on peräisin tunnetusta ja luotetusta lähteestä. Näin luvattomien ohjelmien asentaminen ja suorittaminen voidaan estää. Tämä on toteutettu niin, että tarkistussummalla tai tiivisteellä varustettu päivityspaketti salataan jollain käyttötarkoitukseen riittävän hyvällä algoritmilla, kuten yksityisen ja julkisen avaimen algoritmilla. Mikro-ohjaimessa sijaitseva suojatun ohjelmistopäivityksen toteutus hylkää päivityspaketin, jos salauksen purku oikealla avaimella ei onnistu. Purkuun tarvittava julkinen avain täytyy olla myös suojatun käynnistys- ja päivitysohjelman tiedossa.

Kuvassa 3.8 esitellään eräs mahdollinen varusohjelmiston päivitystapa. Siihen osallistuu tyypillisesti kaksi osapuolta: Valmistajan palvelin, joka allekirjoittaa ja salaa ohjelmiston sekä kentällä oleva laite, joka sisältää SBSFU-toiminnot ja tarvittavat avaimet salatun ohjelmiston purkamiseen ja allekirjoituksen tarkastamiseen. Näiden kahden osapuolen välillä on jokin tiedonsiirtokanava, kuten verkko tai vaikkapa muistikortti tai USB-tikku, josta laite voi lukea ohjelman. Tämän tiedonsiirtokanavan ei tarvitse olla millään tavalla erityinen, suojattu tai luotettava.



Kuva 3.8: Suojattu ohjelmistopäivitys [75]

Kuvassa vasemmalla sijaitseva palvelin on laitteen valmistajan hallussa oleva palvelin, jonka avulla uusi varusohjelma paketoidaan tietoturvalliseen päivityspakettiin. Tällöin lähdekielisestä versiosta käännetty varusohjelman binääriversio paketoidaan laitteen suojatun käynnistys- ja päivitysohjelman ymmärtämään muotoon.

Arkojen tietojen suojaaminen

STM32-perheen mikro-ohjaimissa on mahdollisuus suojata muistialueita eri tavoin. Sopivan tavan tai tapojen valinta riippuu käytetystä mikro-ohjaimesta sekä tietenkin myös halutusta suojaustasosta. SBSFU:ssa avaimet ja muu suojausta vaativa tieto talletetaan ohjelmamuistiin, joka sitten suojataan muistin lukusuojauksen ja palomuurin avulla.

Ohjelmointiliitäntöjen poistaminen käytöstä

Ohjelmointiliitäntöjen poistaminen käytöstä on laitteen suojauksen kannalta välttämätöntä, sillä ohjelmointi- ja vianselvitysliitännät tarjoavat muuten todella helpon tavan korvata laitteen ohjelmisto uudella tai tutkia muistin sisältöä. Se tuo mukanaan myös käytännön ongelmia, sillä piiriä ei tämän jälkeen voi ohjelmoida uudestaan muutoin, kuin sille itse ohjelmoidulla käynnistys- tai päivitysohjelmalla.

3.6 Muiden valmistajien ARM-mikro-ohjaimet

Jokaisella ARM prosessoreiden tai mikro-ohjaimien valmistajalla on ARM Ltd:ltä lisensoidun ja keskeisessä asemassa olevan ARM ytimen ympärillä omat lisäyksensä, niin liityntöjen kuin turvaominaisuuksienkin osalta.

Taulukko 3.1: Mikro-ohjaimien turvaominaisuuksia [49, 82, 83, 62, 57, 74, 90]

	Muistinsuojausyksikkö	Lukusuojausyksikkö	Kajoamisentunnistus	Vain suoritus muistialueelta	Turvapiilo	Alijännitesuojaus	Palomuuri	Ohjelmointiliitännän esto	Virheenkorjaava muisti	Yksilöllinen tunniste	Todelliset satunnaisluvut	SHA	MD5	AES	RSA	CRC	Eheyden valvonta	TrustZone
STMicro-electronics	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√		√
Maxim Integ-rated Products Inc.	√	√	√	√		√	√	√		√	√	√		√			√	√
Texas Ins-truments	√	√	√	√		√		√		√		√		√	√	√		√
Nuvoton Techno-logy Corporation	√		√	√		√		√	√	√	√	√		√		√		√
Microchip Techno-logy Inc.	√		√					√			√		√		√	√	√	√

Taulukossa 3.1 esitetään yhteenveto joidenkin ARM-mikro-ohjainvalmistajien Cortex-M mikro-ohjaimien turvaominausuuksista. Seuraavissa aliluvuissa käsitellään esimerkinomaisesti taulukossa mainittujen valmistajien ratkaisuja kuitenkin keskittyen lähinnä turvaominaisuuksiin.

3.6.1 Maxim Integrated Products Inc.

Maxim Integrated Products Inc. on perustettu vuonna 1983 Yhdysvalloissa. Se valmistaa analogia- ja sekasignaalipiirejä, antureita, tehopuolijohteita, radiotaajuuspiirejä sekä ARM Cortex perheen mikro-ohjaimia. Sen liikevaihto ylitti 2,5 miljardia dollaria eli noin 2,3 miljardia euroa vuonna 2018 [49].

Pienivirtaiset mikro-ohjaimet

Maximin mikro-ohjaintarjonnassa on kymmenkunta pienen virrankulutuksen omaavaa Cortex-M4 -pohjaista mikro-ohjainta joista pienin, MAX32660, on saatavissa myös 16-nastaisessa WLF (Wafer Level Packaging) kotelossa, jonka mitat ovat vain 1,6 x 1,6

mm. Tässä piirissä ei ole mitään turvaominaisuuksia. Muissa sarjan mikro-ohjaimissa on joitain kryptografisia ominaisuuksia, kuten AES-128, AES-192, AES-256, DSA/RSA 2048-bitin avainpituuteen asti, todelliset satunnaisluvut (TRNG) ja suojattu käynnistys.

Turvaominaisuuksilla varustetut mikro-ohjaimet

Maximilla on turvaominaisuuksilla varustettuja ARM-mikro-ohjaimia Cortex-M3, Cortex-M4 ja ARM 9 Deep Cover -perheissä. Niissä olevia turvaominaisuuskia ovat muun muassa: Julkisen avaimen infrastuktuurin suojattu käynnistysohjelma, haihtumaton AES avainten säilytys, AES, DES ja SHA laitteistopohjaiset kiihdyttimet, laitteistopohjainen todellisia satunnaislukuja generoiva yksikkö (TRNG), kajoamisentunnistus ja reaaliaikainen ulkoisen muistin salaus ja eheyden valvonta [51].

ChipDNA Physically Unclonable Function -teknologia

Maxin Integrated on kehittänyt ChipDNA nimisen tekniikan, joka tunnetaan myös nimellä Physically Unclonable Function eli PUF. Se perustuu CMOS-piirien luonnollisesti tapahtuviin satunnaisiin ilmiöihin, joita käytetään luomaan jokaiselle piirille yksilöllinen kryptografinen avain. PUF-avain on riippumaton käyttöjännitteestä, lämpötilasta ja muista piirin käyttöympäristön muutoksista eikä muutu piirin vanhetessa. ChipDNA:n perustavaa laatua oleva ero muihin avainten säilytystekniikoihin nähden on se, että avainta ei ole olemassa missään kohti ohjelma- tai työmuistia selväkielisenä eikä salattuna, vaan se muodostetaan CMOS-piirien rakenteellisiin eroihin perustuen aina tarvittaessa ja on joka piirille yksilöllinen. [50]

3.6.2 Texas Instruments

Texas Instruments on Yhdysvaltalainen 1930-luvulla perustettu monialayritys, joka nykyään valmistaa muun muassa taskulaskimia, puolijohteita ja elektroniikan komponentteja. Vuonna 2018 sen liikevaihto oli 15,7 miljardia dollaria, eli noin 14,3 miljardia euroa, josta noin 3,55 miljardin dollarin osuus tuli sulautettujen prosessoreiden myynnistä. [84]

Se valmistaa joitain ARM Cortex-M3 mikro-ohjaimia ja melko suurta määrää erilaisia ARM Cortex-M4 mikro-ohjaimia. Parhaiten varustelluissa Texas Instrumentsin Cortex-M4 mikro-ohjaimissa on turvallisuuteen ja salakirjoitukseen liittyviä omi-

naisuuksia, kuten: Muistinsuojausyksikkö (MPU), tarkistussummalaskuri (CRC), AES-128, AES-192, AES-256 ja DES salakirjoitusalgoritmien tuki sekä SHA ja MD5 tiivisteiden laskennan tuki. Ohjelmointiliitännän voi lukita salasanalla, niissä on yksilöllinen sarjanumero, ohjelmamuisti voidaan suojata tyhjentämiseltä tai ohjelmoinnilta ja sen voi myös suojata niin, että vain koodin suoritus on sallittu. Kaikki muu muistin käsittely, kuten lukeminen, on kielletty. Niissä on myös suojatun käynnistyksen ja suojatun varusohjelmiston päivityksen tuki. [82, 83]

3.6.3 Nuvoton Technology Corporation

Nuvoton Technology Corporation on Taiwanilainen, vuonna 2008 Winbondista irtautunut puolijohdevalmistaja, joka valmistaa ARM Cortex-M0, ARM Cortex-M4 ja Cortex-M23 mikro-ohjaimia. Lisäksi se valmistaa mikro-ohjainkortteja, sekasignaalipiiriratkaisuja sekä ison joukon monipuolisia 8051-perheen mikro-ohjaimia. Nuvotonin liikevaihto oli vuonna 2018 10,04 miljardia uutta Taiwanin dollaria (NT$), eli noin 297 miljoonaa euroa [63].

Yrityksen valmistamat ARM Cortex-M23 piirit sisältävät ARMv8-M Trustzone teknologian ja perustuvat ARM Ltd:n PSA turvallisuusarkkitehtuuriin. Niissä on myös suojattu käynnistysohjelma ja laittestotason salauksen tuki, mukaan lukien elliptisten käyrien salausmenetelmät. Niiden ohjelmamuisti voidaan suojata salasanalla, niissä on todellisia satunnaislukuja generoiva yksikkö (TRNG) ja niiden ohjelmamuistia voi suojata määrittelemällä alueita, joista koodia voi vain suorittaa (Execute Only Memory, XOM). Niiden ohjelmointiliitännän voi poistaa käytöstä ja ne tukevat kajoamisen tunnistustoimintoja. [62]

3.6.4 Microchip Technology Inc.

Microchip Technology Inc. on Yhdysvaltalainen analogiapiirien, digitaalipiirien ja mikro-ohjainten valmistaja. Vuonna 2016 se osti toisen ison mikro-ohjainvalmistajan, Atmel Corporationin, jonka liikevaihto vuonna 2015 oli 1,17 miljardia dollaria eli noin 1,06 miljardia euroa. Microhipin liikevaihto vuonna 2018 oli 3,98 miljardia dollaria, eli noin 3,6 miljardia euroa. Atmelkaupan myötä Microchip siis kasvoi liki kolmanneksella. [55]

Microhipin mikro-ohjainvalikoimaan tulivat Atmelin hankinnan yhteydessä myös ARM Cortex-M0, Cortex-M4, Cortex-M7 ja Cortex-M23 mikro-ohjaimet. Cortex-M23 ohjaimet sisältävät ARM TrustZone teknologian mukaiset ominaisuudet. TrustZone-

ominaisuuksien lisäksi Microchipin ARM mikro-ohjaimissa on suuri määrä turvaominaisuuksia kuten kajoamisen tunnistus, suojattu käynnistys, suojattu varusohjelmiston päivitys, AES salakirjoitus, ohjelma- ja työmuistin lukusuojaus, todellisia satunnaislukuja tuottava satunnaislukugeneraattori (TRNG) ja suojattu vianselvitysliitäntä. [56, 54]

Cortex-M7 arkkitehtuurissa on lisäksi eheyden valvonta toiminto (Integrity Check Monitor, ICM), joka käyttää hyväkseen muistin suorasaantiyksikköä sekä SHA-tiivisteiden laskentayksikköä. Tätä voidaan käyttää hyväksi muun muassa suojatussa käynnistysohjelmassa laskemaan ohjelman tiiviste, jolla voidaan todentaa ettei ohjelmaa ole muutettu. [57]

4 Yhteenveto

Työn tarkoituksena oli selvittää minkälaisia laitteistopohjaisia turvallisuusratkaisuja nykyaikaisissa mikro-ohjaimissa on, sekä selvittää, miten niitä voidaan käyttää tiedon ja ohjelmiston suojaamiseen. Työssä keskityttiin mikro-ohjainten laitteistopohjaisiin turvaominaisuuksiin, joita voi käyttää ohjelmiston, tiedon ja algoritmien suojaamiseen. Käsittelyssä sivuttiin myös turvallisuusratkaisujen vaikutusta ohjelmistokehitysprosessiin.

Käsittelyn perusteella voi päätellä, että nykyaikaisissa mikro-ohjaimissa on paljon monipuolisia turvaominaisuuksia. Niiden tunteminen laitteistotasolla sekä niiden käytön osaaminen ohjelmistokehitysprojekteissa nouseekin merkittäväksi tekijäksi tietoturvan kannalta, sillä turvaominaisuudet, joita ei ole aktivoitu, tai joita ei osata hyödyntää eivät tee laitteesta yhtään turvallisempia, vaan saattavat tehdä siitä jopa haavoittuvamman.

Saadut tulokset tulevat vaikuttamaan ainakin omassa työssäni lisääntyneen tietoturvaosaamisen ja STMicroelectronicsin mikro-ohjainten tietoturvamahdollisuuksien tuntemisen kautta. Erityisen hyödyllistä oli tutustua suojattuun käynnistysohjelmaan ja suojattuun varusohjelman päivittämiseen. Niiden tuntemisesta tulee olemaan paljon hyötyä omassa työssäni.

Tutkimusta voisi laajentaa siten, että siihen otettaisiin mukaan enemmän mikro-ohjaimia ja mikro-ohjainvalmistajia. Tällöin saataisiin laajempi kuva markkinoilla olevien mikro-ohjainten turvallisuusratkaisuista. Seuraava askel olisi ottaa mikro-ohjainta suurempien ja tehokkaampien suorittimien turvaratkaisut mukaan selvitykseen. Tästä todennäköisesti aiheutuisi työmäärän huomattava kasvu alueen laajuuden vuoksi.

Lähteet

[1] ARM LTD. 2019 TECHNOLOGY SURVEY & PREDICTIONS. URL http://info.arm.com/b0000jA8ZS91XdDh010rSQi, viitattu 20.12.2018.

[2] ARM LTD. Arm Cortex-A Series Processors. URL https://developer.arm.com/ip-products/processors/cortex-a, viitattu 9.4.2019.

[3] ARM LTD. Arm Cortex-M Series Processors. URL https://developer.arm.com/ip-products/processors/cortex-m, viitattu 9.4.2019.

[4] ARM LTD. Arm Cortex-R Series Processors. URL https://developer.arm.com/ip-products/processors/cortex-r, viitattu 9.4.2019.

[5] ARM LTD. Arm Machine Learning Processor. URL https://developer.arm.com/ip-products/processors/machine-learning/arm-ml-processor, viitattu 10.4.2019.

[6] ARM LTD. Arm Platform Security Architecture - Overview white paper. URL https://pages.arm.com/PSA-Building-a-secure-IoT.html, viitattu 11.11.2019.

[7] ARM LTD. Arm processors for the widest range of devices - from sensors to servers. URL https://www.arm.com/products/silicon-ip-cpu, viitattu 17.12.2018.

[8] ARM LTD. Cortex-M. URL https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4, viitattu 23.5.2019.

[9] ARM LTD. Cortex-M for Beginners - An overview of the Arm Cortex-M processor family and comparison (2017). URL https://community.arm.com/cfs-file/__key/telligent-evolution-components-attachments/01-2057-00-00-00-01-28-35/Cortex_2D00_M-for-Beginners-_2D00_-2017_5F00_EN_5F00_v2.pdf, viitattu 1.3.2020.

[10] ARM LTD. Layered Security for Your Next SoC. URL https://www.arm.com/products/silicon-ip-security, viitattu 1.3.2020.

[11] ARM LTD. Platform Security Architecture. URL https://www.arm.com/why-arm/architecture/platform-security-architecture, viitattu 24.4.2019.

[12] ARM LTD. SecurCore. URL https://developer.arm.com/ip-products/processors/securcore, viitattu 10.4.2019.

[13] ARM LTD. The Arm Cortex-M4 processor is a highly-efficient embedded processor. URL https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4, viitattu 3.3.2020.

[14] ARM LTD. Trustzopne For Cortex-M. URL https://www.arm.com/why-arm/technologies/trustzone-for-cortex-m, viitattu 10.11.2019.

[15] ASADIZANJANI, N., TEHRANIPOOR, M., JA FORTE, D. Pcb reverse engineering using nondestructive x-ray tomography and advanced image processing. *IEEE Transactions on Components, Packaging and Manufacturing Technology 7*, 2 (Feb 2017), 292–299.

[16] BAL, G., BEKIROGLU, E., BAYINDIR, R., JA UZEL, H. Microcontroller based digitally controlled ultrasonic motor drive system. *Journal of Electroceramics 20*, 3-4 (2008), 265–270.

[17] BARR, M., JA MASSA, A. *Programming Embedded Systems*. O'Reilly Media, Sebastopol, USA, 2009.

[18] BECKER, G. T., BURLESON, W., JA PAAR, C. Side-channel watermarks for embedded software. Julkaisusarjassa *IEEE 9th International New Circuits and systems conference* (Bordeaux, Ranska, June 2011), 478–481.

[19] BINKLEY, D. Source code analysis: A road map. Julkaisusarjassa *Future of Software Engineering (FOSE '07)* (Washington DC, USA, May 2007), 104–119.

[20] CASS, S. The 2018 top programming languages. URL https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018, viitattu 6.10.2019.

[21] CHIKOFSKY, E. J., JA CROSS, J. H. Reverse engineering and design recovery: A taxonomy. *IEEE software 7*, 1 (1990), 13–17.

[22] CPLUSPLUS.COM. History of C++. URL http://www.cplusplus.com/info/history/, viitattu 5.10.2019.

[23] CRISP, J. *Introduction to microprocessors and microcontrollers*. Elsevier, Burlington MA, USA, 2003.

[24] EMBEDDED. 2012 Embedded Market Survey. URL https://www.embedded.com/electronics-blogs/embedded-market-surveys/4405646/2012-Embedded-Market-Survey, viitattu 1.10.2019.

[25] EMBEDDED. 2014 Embedded Market Survey. URL https://www.embedded.com/electronics-blogs/embedded-market-surveys/4440709/2014-Embedded-Market-Survey, viitattu 1.10.2019.

[26] EMBEDDED. 2017 Embedded Market Survey. URL https://www.embedded.com/electronics-blogs/embedded-market-surveys/4458724/2017-Embedded-Market-Survey, viitattu 1.10.2019.

[27] FAGGIN, F., HOFF, M. E., MAZOR, S., JA SHIMA, M. The history of the 4004. *IEEE Micro 16*, 6 (Dec 1996), 10–20.

[28] FARNELL ELEMENT14. URL https://fi.farnell.com/stmicroelectronics/stm32l011d3p6/mcu-arm-cortex-m0-32mhz-tssop/dp/2851003?st=STM32L0, viitattu 12.2.2019.

[29] FARNELL ELEMENT14. URL https://fi.farnell.com/stmicroelectronics/stm32h743xih6/mcu-arm-cortex-m7-400mhz-tfbga/dp/2820777?st=STM32H743, viitattu 12.2.2019.

[30] FOURNARIS, A. P., POCERO FRAILE, L., JA KOUFOPAVLOU, O. Exploiting hardware vulnerabilities to attack embedded system devices: a survey of potent microarchitectural attacks. *Electronics 6*, 3 (2017).

[31] HE, B., WANG, J., HE, D., JA CHENAND, W. Study on EM-side channel test for embedded system. Julkaisusarjassa *Proceedings of 2011 International Conference on Computer Science and Network Technology* (Harbin, Kiina, Dec 2011), vol. 4, 2368–2371.

[32] HOFFMAN, T. Embedded C/C++ Unit Testing Basics. URL https://interrupt.memfault.com/blog/unit-testing-basics, viitattu 4.11.2019.

[33] HOSKE, M. T. What is an embedded system? URL https://www.controleng.com/articles/what-is-an-embedded-system/, viitattu 27.11.2018.

[34] HWANG, D. D., SCHAUMONT, P., TIRI, K., JA VERBAUWHEDE, I. Securing embedded systems. *IEEE Security & Privacy 4*, 2 (2006), 40–49.

[35] IC INSIGHTS. Microcontrollers will regain growth after 2019 slump, 2019. URL http://www.icinsights.com/news/bulletins/Microcontrollers-Will-Regain-Growth-After-2019-Slump/, viitattu 25.8.2019.

[36] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. ISO/IEC 9899:1999 Programming languages - C. URL https://www.iso.org/standard/29237.html, viitattu 22.10.2019.

[37] ISO/IEC. The official home of ISO/IEC JTC1/SC22/WG14 - C. URL http://www.open-std.org/JTC1/SC22/WG14/, viitattu 22.10.2019.

[38] KARSAI, G., MASSACCI, F., OSTERWEIL, L., JA SCHIEFERDECKER, I. Evolving embedded systems. *Computer 43*, 5 (May 2010), 34–40.

[39] KIDWELL, P. Journey to the moon: the history of the apollo guidance computer. *IEEE Annals of the History of Computing 21*, 1 (Jan 1999), 78–79.

[40] KIPERBERG, M. *Preventing reverse engineering of native and managed programs*. PhD thesis, University of Jyväskylä, 2015. URL http://urn.fi/URN:ISBN:978-951-39-6437-5, viitattu 12.11.2019.

[41] LACAMERA, D. *Embedded Systems Architecture*. Packt Publishing Ltd., Birmingham, UK, 2018.

[42] LEE, K., LEE, Y., LEE, H., JA YIM, K. A brief review on jtag security. Julkaisusarjassa *10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)* (Fukuoka, Japani, July 2016), 486–490.

[43] LEMKE-RUST, K., JA SAMARIN, P. Exploring embedded software with side channels and fault analysis. Julkaisusarjassa *12th European Workshop on Microelectronics Education (EWME)* (Braunschweig, Saksa, Sep. 2018), 67–70.

[44] LING MING, SHI LONGXING, JA TANG YONGMING. An advanced C programming course for the embedded systems development. Julkaisusarjassa *Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (Hongkong, Kiina, Aug 2012), H2A–1–H2A–3.

[45] LIPPMAN, S. B. *C++ Primer*. Addison-Wesley Publishing Company, Reading, Massachusetss, USA, 1990.

[46] LOURIDAS, P. Static code analysis. *IEEE Software 23*, 4 (July 2006), 58–61.

[47] LUNDGREN, B. Defining information security. *Science and Engineering Ethics 25*, 2 (2019), 419.

[48] MAURER, S. S. A survey of embedded systems programming languages. *IEEE Potentials 21*, 2 (April 2002), 30–34.

[49] MAXIM INTEGRATED PRODUCTS INC. Annual Report 2019. URL `https://s21.q4cdn.com/176677460/files/doc_financials/Annual-Proxy/2019/2019-Annual-Report.pdf`, viitattu 6.3.2020.

[50] MAXIM INTEGRATED PRODUCTS INC. DeepCover Embedded Security Solution Guide. URL `http://pdfserv.maximintegrated.com/en/sg/DeepCover-Embedded-Security-Solutions.pdf`, viitattu 3.3.2020.

[51] MAXIM INTEGRATED PRODUCTS INC. Secure Microcontroller. URL `https://www.maximintegrated.com/en/products/embedded-security/secure-microcontrollers.html`, viitattu 6.3.2020.

[52] MCLOUGHLIN, I. Secure embedded systems: The threat of reverse engineering. Julkaisusarjassa *14th IEEE International Conference on Parallel and Distributed Systems* (Melbourne, Australia, Dec 2008), 729–736.

[53] MCLOUGHLIN, I. Reverse engineering of embedded consumer electronic systems. Julkaisusarjassa *IEEE 15th International Symposium on Consumer Electronics (ISCE)* (Nanyang, Singapore, June 2011), 352–356.

[54] MICROCHIP TECHNOLOGY INCORPORATED. 32-bit Embedded Security Solutions. URL `https://www.microchip.com/design-centers/32-bit/security-solutions-with-32-bit-microcontrollers`, viitattu 8.3.2020.

[55] MICROCHIP TECHNOLOGY INCORPORATED. Annual Report 2018. URL `https://www.microchip.com/sec/annual/FY18/2018%20Proxy%20Statement%20and%20Annual%20Report.pdf`, viitattu 8.3.2020.

[56] MICROCHIP TECHNOLOGY INCORPORATED. SAM L Microcontrollers. URL https://www.microchip.com/design-centers/32-bit/sam-32-bit-mcus/sam-l-mcus, viitattu 8.3.2020.

[57] MICROCHIP TECHNOLOGY INCORPORATED. What is ICM? How to Use the ICM For Cryptography? URL https://www.microchip.com/mymicrochip/filehandler.aspx?ddocname=en601538, viitattu 8.3.2020.

[58] MICROMOUSE ONLINE. Determine STM32 reset source. URL http://www.micromouseonline.com/2012/03/29/stm32-reset-source/, viitattu 23.4.2019.

[59] MISRA C WORKING GROUP. *MISRA C:2012, Guidelines for the use of the C language in critical systems*, 3 ed. HORIBA MIRA, Ltd, Nuneaton, Iso-Britannia, February 2019.

[60] MOUSER ELECTRONICS. URL https://www.mouser.fi/Search/Refine?Keyword=STM32L486QG, viitattu 12.2.2019.

[61] MÜLLER, H. A., JAHNKE, J. H., SMITH, D. B., STOREY, M.-A., TILLEY, S. R., JA WONG, K. Reverse engineering: A roadmap. Julkaisusarjassa *Proceedings of the Conference on The Future of Software Engineering* (New York, NY, USA, 2000), ICSE '00, ACM, 47–60.

[62] NUVOTON TECHNOLOGY CORPORATION. NuMicro M2351 Series - a TrustZone empowered microcontroller series focusing on IoT security. URL https://www.nuvoton.com/products/microcontrollers/arm-cortex-m23-mcus/m2351-series/, viitattu 3.3.2020.

[63] NUVOTON TECHNOLOGY CORPORATION. Nuvoton Technology Corp. Annual Report 2018. URL https://www.nuvoton.com/export/sites/nuvoton/files/EN_107_Annualreport.pdf, viitattu 6.3.2020.

[64] PERFORCE SOFTWARE INC. What is MISRA? An overview of the MISRA standard. URL https://www.perforce.com/resources/qac/what-misra-overview-misra-standard, viitattu 27.10.2019.

[65] PTATECHNOLOGIES. Practical Threat Analysis in-depth. URL http://www.ptatechnologies.com/PTA3.htm, viitattu 14.11.2018.

[66] RAVI, S., RAGHUNATHAN, A., KOCHER, P., JA HATTANGADY, S. Security in embedded systems: Design challenges. *ACM Trans. Embed. Comput. Syst. 3*, 3 (Aug. 2004), 461–491.

[67] RITCHIE, D. M., JOHNSON, S. C., LESK, M. E., JA KERNIGHAN, B. W. Unix time-sharing system: The c programming language. *The Bell System Technical Journal 57*, 6 (July 1978), 1991–2019.

[68] ROGERS, D. *IoT-Securing by Design*. Tekninen raportti, Copper Horse. URL `https://learn.arm.com/rs/714-XIJ-402/images/IoT_Whitepaper_Secure_By_Design.pdf`, viitattu 28.2.2020.

[69] ROWE, M. IEEE survey ranks programming languages. URL `https://www.eetimes.com/document.asp?doc_id=1333572`, viitattu 6.10.2019.

[70] SEACORD, R. C. C Secure Coding Rules: Past, Present, and Future. URL `http://www.informit.com/articles/article.aspx?p=2088511`, viitattu 1.11.2019.

[71] SEACORD, R. C. *The CERT C Coding Standard*. Addison-Wesley Publishing Company, Reading, Massachusetss, USA, 2014.

[72] SEYYEDI, S. A., KAMAL, M., NOORI, H., JA SAFARI, S. Securing embedded processors against power analysis based side channel attacks using reconfigurable architecture. Julkaisusarjassa *IFIP 9th International Conference on Embedded and Ubiquitous Computing* (Melbourne, Australia, Oct 2011), 255–260.

[73] SOFTWARE ENGINEERING INSTITUTE. SEI CERT C Coding Standard, Rules for Developing Safe, Reliable, and Secure Systems. URL `https://resources.sei.cmu.edu/downloads/secure-coding/assets/sei-cert-c-coding-standard-2016-v01.pdf`, viitattu 4.11.2019.

[74] STMICROELECTRONICS. *Application note: Introduction to STM32 microcontrollers security*. Tekninen raportti AN5156, STMicroelectronics. URL `https://www.st.com/content/ccc/resource/technical/document/application_note/group1/9f/0b/e4/b6/75/15/4f/e2/DM00493651/files/DM00493651.pdf/jcr:content/translations/en.DM00493651.pdf`, viitattu 6.2.2019.

[75] STMICROELECTRONICS. *Getting started with the X-CUBE-SBSFU STM32Cube Expansion Package*. Tekninen raportti UM2262, STMicroelectronics. URL

https://my.st.com/resource/en/user_manual/dm00414687.pdf, viitattu 6.12.2019.

[76] STMICROELECTRONICS. System Workbench for STM32. URL https://www.openstm32.org/System%2BWorkbench%2Bfor%2BSTM32, viitattu 13.2.2020.

[77] STMICROELECTRONICS. X-CUBE-SBSFU, Secure boot & secure firmware update software expansion for STM32Cube. URL https://www.st.com/content/st_com/en/products/embedded-software/mcu-mpu-embedded-software/stm32-embedded-software/stm32cube-expansion-packages/x-cube-sbsfu.html, viitattu 11.12.2019.

[78] STROUSTRUP, B. Evolving a language in and for the real world: C++ 1991-2006. URL http://stroustrup.com/hopl-almost-final.pdf, viitattu 5.10.2019.

[79] STROUSTRUP, B. The C++ Programming Language. URL http://www.stroustrup.com/C++.html, viitattu 6.10.2019.

[80] STROUSTRUP, B. *The Design and Evolution of C++*. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, 1994.

[81] STUDNIA, I., NICOMETTE, V., ALATA, E., DESWARTE, Y., KAÂNICHE, M., JA LAAROUCHI, Y. Survey on security threats and protection mechanisms in embedded automotive networks. Julkaisusarjassa *43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)* (Budapest, Unkari, June 2013), 1–12.

[82] TEXAS INSTRUMENTS. Microcontrollers (MCU). URL http://www.ti.com/microcontrollers/other-mcus/products.html#p887=ARM-Cortex-M3;ARM%20Cortex-M4F, viitattu 6.3.2020.

[83] TEXAS INSTRUMENTS. Security. URL http://www.ti.com/technologies/security/overview.html?keyMatch=SECURITY&tisearch=Search-EN-everything#portfolio, viitattu 8.3.2020.

[84] TEXAS INSTRUMENTS. Texas Instruments 2018 Annual Report. URL https://investor.ti.com/static-files/c442d3c4-74a7-4f93-8c7b-58159bca07ef, viitattu 6.3.2020.

[85] THORNTON, S. Arm TrustZone explained. URL https://www.microcontrollertips.com/embedded-security-brief-arm-trustzone-explained/, viitattu 10.11.2019.

[86] TIOBE SOFTWARE BV. Tiobe index for october 2019. URL https://www.tiobe.com/tiobe-index/, viitattu 9.10.2019.

[87] TONELLA, P., TORCHIANO, M., DU BOIS, B., JA SYSTÄ, T. Empirical studies in reverse engineering: state of the art and future trends. *Empirical Software Engineering 12*, 5 (Oct 2007), 551–571.

[88] VEERABAHU, M. 5 differences between embedded software engineer and software developer. URL https://www.linkedin.com/pulse/5-differences-between-embedded-maharajan/, viitattu 27.3.2019.

[89] WANG, G. Modeling C-Based Embedded System Using UML Design. Julkaisusarjassa *2009 International Conference on Mechatronics and Automation* (Changchun, Kiina, Aug 2009), 2973–2977.

[90] WASKIEWICZ, B. Platform level security for iot devices. URL https://www.st.com/content/dam/technology-tour-2017/session-2_track-2_platform-security-iot.pdf, viitattu 6.2.2019.

[91] WILLIAMSON, M. Aiming for the moon: the engineering challenge of apollo. *Engineering Science and Education Journal 11*, 5 (Oct 2002), 164–172.

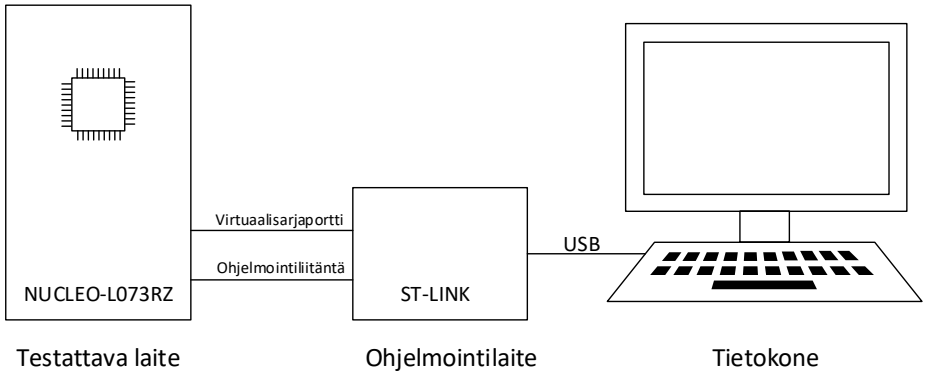
[92] WONG, B. The latest static and dynamic code analysis tools. *Electronic Design 57*, 11 (Jun 11 2009), 40.

[93] YIU, J. *The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors*. Newnes, Amsterdam, Hollanti, 2013.

A Suojattu käynnistysohjelma ja suojattu varusohjelman päivitys

A.1 Testausmenetelmä

Myöhemmin liitteen luvussa A.2 määritellyissä testitapauksissa käytetään kuvan A.1 mukaista koejärjestelyä. Testattavana laitteena on STMicroelectronicsin valmistama kokeilukortti NUCLEO-L073RZ ja ohjelmointilaitteena samaan korttiin integroitu ST-LINK V2 -ohjelmointilaite, jota on saatavana myös erillisenä laitteena.



Kuva A.1: Koejärjestely

Testattava SBSFU-ohjelmisto on noudettavissa STMicroelectronicsin sivuilta [77], ohjelmistopaketin lataaminen on ilmaista, mutta vaatii rekisteröitymisen ja kirjautumisen. Testeissä käytetyn ohjelmiston tunniste on STM32CubeExpansion_SBSFU_V2.2.0. Tässä ohjelmapaketissa on valmiit ohjelmaprojektit useille käännösympäristölle. Tässä työssä testit on käännetty käyttäen yhtä tuetuista käännösympäristöistä, ilmaista Eclipse Neon -pohjaista "System Workbench for STM32"(SW4STM32), joka on saatavilla verkosta [76]. Ohjelmistopaketti sisältää suojatun käynnistysohjelman ja suojatun varusohjelman päivityksen. Testattava ohjelmistopaketti on NUCLEO-STM32L07RZ3-kokeilukortille mukautettu SBSFU-ohjelmiston versio, jonka nimi on "NUCLEO-L073RZ_2_Images_SBSFU".

Ohjelmapaketti sisältää kolme aliprojektia: SECoreBin, SBSFU ja UserApp. SECoreBin sisältää 'turvamoottorin' (Secure Engine, SE) sekä kaiken luotetun koodin, SBSFU sisältää suojatun käynnistyksen (SB) sekä suojatun varusohjelman päivityksen (SFU) ja UserApp sisältää esimerkkisovelluksen, joka ladataan kokeilukortille SFU:ta käyttäen. Se sisältää testitapauksissa käytetyt turvatestit, jotka käyttävät kokeilukortin sarjaporttia konsoliyhteytenä. ST-LINK-ohjelmointilaitteen virtuaalisarjaportti on kytketty mikro-ohjaimen sarjaporttiin, joten tietokoneessa ei tarvita erillistä sarjaliitäntää, yksi USB-liitäntä riittää.

Testaamisessa käytetään edellä kuvattua SBSFU:n esimerkkitoteutusta, joka tarjoaa tekstipohjaisen käyttöliittymän laitteen sarjaporttikonsolin kautta. Konsolilla esitetään menuja, joiden kautta eri testejä voidaan valita sekä myös testien tuloksia. Käynnistyessä laite tulostaa suojaustasoon liittyviä ilmoituksia. Testien tulokset dokumentoidaan liittämällä sarjakonsoliyhteyden tulosteista asiaan liittyvät osat testitapausten tuloksien käsittelyn yhteyteen. Tarkoituksena on testata, miten SBSFU:n avulla saadaan mikro-ohjain ja sen varusohjelma suojattua valtuuttamattomalta muuttamiselta sekä eritasoisia muistialueiden suojauksia.

A.2 Testikuvaukset

A.2.1 Ohjelmointiliitännän poistaminen käytöstä

Testissä kytketään ohjelmointilaite tutkittavan laitteen ohjelmointiliittimeen ja yritetään suorittaa joku toimenpide ohjelmointilaitteen kautta, esimerkiksi muistialueen lukeminen tai pelkästään liitynnän alustus. Mikäli toimenpide onnistuu, on testi epäonnistunut, sillä turvallinen käynnistysohjelma (SB) on ensimmäistä kertaa käynnistyessään poistanut ohjelmointiliitännän käytöstä.

A.2.2 Ohjelmakoodialueen suojaaminen palomuurilla

Testi yrittää lukea palomuurilla suojatulla ohjelmamuistialueella sijaitsevaa avainta, jonka arvo on **OEM_KEY_COMPANY1**, kutsumalla pientä assemblerohjelmaa, joka onnistuessaan palauttaa 16-merkkisen avaimen arvon. Mikäli lukeminen onnistuu, on testi epäonnistunut ja se tulostaa saadun avaimen konsolille. Tällöin ohjelmakoodialue ei ole suojattu. Onnistunut testi puolestaan aiheuttaa laitteen uudelleenkäynnistymisen suojausmekanismin toiminnan mukaisesti. Uudelleenkäynnistyksen yhteydessä konsolille tulostuu uudelleenkäynnistyksen syy.

A.2.3 Työmuistialueen suojaaminen palomuurilla

Testi yrittää lukea palomuurilla suojattua tietoa työmuistialueelta **SRAM1** suoralla muistinosoituksella. Mikäli lukeminen onnistuu, on testi epäonnistunut ja se tulostaa ruudulle lukemansa tiedon. Tällöin työmuistialue ei ole suojattu. Onnistunut testi puolestaan aiheuttaa laitteen uudelleenkäynnistymisen suojausmekanismin toiminnan mukaisesti. Uudelleenkäynnistyksen yhteydessä konsolille tulostuu uudelleenkäynnistyksen syy.

A.2.4 Ohjelmakoodin suojaaminen kirjoittamiselta

Testi lukee lukusuojaamattomasta osasta ohjelmamuistia 2048 tavua taulukkoon, jonka jälkeen se pyyhkii kirjoitusta varten saman muistialueen ja yrittää kirjoittaa lukemansa tiedon takaisin samaan muistialueeseen. Mikäli pyyhkiminen tai kirjoittaminen onnistuu, on testi epäonnistunut, eli ohjelmamuistialue ei ole kirjoitussuojattu. Onnistunut testi huomaa kirjoituksen epäonnistuvan ja päättää testin.

A.2.5 Muuttuneen ohjelmakoodin tunnistaminen

Testi poistaa ensin käytöstä muistinsuojausyksikön (MPU), jotta ohjelmamuistiin voidaan kirjoittaa. Seuraavaksi testi muuttaa ohjelmamuistin sisältöä nollaamalla varusohjelman alusta 512 tavua. Onnistuneen kirjoituksen lopuksi testi uudelleenkäynnistää laitteen. Testin epäonnistuminen huomataan siitä, että uudelleenkäynnistyksessä turvallinen käynnistysohjelma ei huomaa muuttunutta varusohjelmaa. Onnistuminen tunnistetaan siitä, että turvallinen käynnistysohjelma huomaa varusohjelmiston tarkistussumman olevan nyt väärin, eikä suostu käynnistämään sitä.

A.2.6 Kajoamisen tunnistussisääntulon toiminta

Testi pyytää käyttäjää kytkemään alatilan kajoamisentunnistusnastaan PA0 ja odottaa 10 sekuntia. Mikäli käyttäjä muuttaa PA0:n tilan alatilaksi ja laite jatkaa toimintaansa on testi epäonnistunut. Onnistuneen testin huomaa siitä, että laite käynnistyy uudestaan ja käynnistyessään kertoo käynnistyssyyksi kajoamisentunnistuksen.

A.2.7 Vahtikoiran toiminta

Testi pysäyttää ohjelman normaalin suorituksen suorittamalla 10 sekunnin viiverutiinin. Mikäli 10 sekunnin viiveen jälkeen ohjelman suoritus jatkuu, on testi epäonnistunut, eli vahtikoira ei ole toiminnassa. Onnistuneen testin tunnistaa siitä, että laite käynnistyy uudestaan ja käynnistyessään kertoo käynnistyssyyksi vahtikoiran.

A.2.8 Varusohjelmiston päivitys

Testissä ladataan kortille uusi varusohjelmisto päivityspaketilla, jossa varusohjelman sisältävä osa on vioittunut. Jos varusohjelman päivitys siitä huolimatta onnistuu, on testi epäonnistunut. Jos varusohjelman päivitys epäonnistuu, on testi onnistunut.

A.3 Testaustulokset

Testien tekemistä varten SBSFU sisältää varusohjelman, jolla suojauksia ja varusohjelman lataamista voi kokeilla. Konsolitulostuksessa A.1 näkyy ohjelman valikkorakenne. Riveillä 8 - 18 näkyy käynnistyksen aikaisten testien tuloksia, kuten rivillä 12 edellisen uudelleenkäynnistyksen syy. Rivillä 32 olevan kehotteen perässä näkyy tehty valinta hakasuluissa, tässä **2**, jolloin ohjelma siirtyy valikkoon "Test menu", jonka tulostukset ovat nähtävissä riviltä 34 alkaen.

Seuraavissa aliluvuissa konsolitulostuksista on poimittu mukaan vain testien suorittamisen ja niiden tuloksien kannalta välttämättömät konsolitulosteet. Mahdollisesti pois jätettyjä osia kuvaamaan on lisätty merkintä ∼∼∼∼, ja jos rivin sisältö on jouduttu jakamaan kahdelle riville merkitään jatkoriviä ↪ etuliitteellä.

```
 1 ================================================================
 2 =              (C) COPYRIGHT 2017 STMicroelectronics            =
 3 =                                                               =
 4 =              Secure Boot and Secure Firmware Update           =
 5 ================================================================
 6
 7
 8 = [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
 9 = [SBOOT] STATE: CHECK STATUS ON RESET
10           INFO: A Reboot has been triggered by a Hardware reset!
11           Consecutive Boot on error counter = 0
12           INFO: Last execution detected error was:No error. Success.
13 = [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
14 = [SBOOT] STATE: CHECK USER FW STATUS
15           A valid FW is installed in the active slot - version: 1
16 = [SBOOT] STATE: VERIFY USER FW SIGNATURE
17 = [SBOOT] STATE: EXECUTE USER FIRMWARE
18 ================================================================
19 =              (C) COPYRIGHT 2017 STMicroelectronics            =
20 =                                                               =
21 =                          User App #A                          =
```

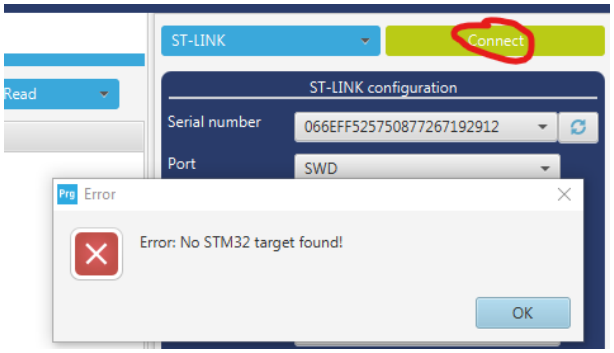
```
22 ================================================================================
23
24
25 ================== Main Menu ============================
26
27   Download a new Fw Image ------------------------------ 1
28   Test Protections ------------------------------------- 2
29   Test SE User Code ------------------------------------ 3
30
31   Selection : [2]
32
33 ================== Test Menu ============================
34
35   Test Protection: Firewall - CODE --------------------- 1
36   Test Protection: Firewall - VDATA -------------------- 2
37   Test Protection: PCROP ------------------------------- 3
38   Test Protection: WRP --------------------------------- 4
39   Test Protection: IWDG -------------------------------- 5
40   Test Protection: TAMPER ------------------------------ 6
41   Test : CORRUPT ACTIVE IMAGE -------------------------- 7
42   Previous Menu ---------------------------------------- x
43
44   Selection : [1]
45 ~~~~
```

Konsolitulostus A.1: Varusohjelman valikkorakenne

A.3.1 Testi ”Ohjelmointiliitännän poistaminen käytöstä”

Testissä kytketään ST-LINK-ohjelmointilaiteen JTAG- tai SWD-liitäntä mikro-ohjaimen ohjelmointiliittimeen ja kytketään ohjelmointilaite tietokoneen USB-liitäntään. Ohjelmointilaitteen käyttöliittymänä on tässä käytetty STMicroelectronicsin ilmaista STM32CubeProgrammer-ohjelmaa.

Kun ohjelmointilaitteen kautta yritetään päästä käsiksi mikro-ohjaimen ohjelmointiliitäntään, saadaan kuvassa A.2 näkyvä virheilmoitus, joka kertoo, että ST-LINK-ohjelmointilaite on tunnistettu mutta "Connect-painikkeen painamisen jälkeen saadaan virheilmoitus "Error: No STM32 target found". Tämä johtuu siitä, että turvallinen käynnistysohjelma on poistanut ohjelmointiliitännän käytöstä. Tämän virheilmoituksen saaminen todentaa sen, että ohjelmointiliitäntä on pois käytöstä ja testi on siis onnistunut.



Kuva A.2: Virheilmoitus STM32CubeProgrammer ohjelmasta

A.3.2 Testi ”Ohjelmakoodialueen suojaaminen palomuurilla”

Testissä ohjelmamuistin suojatulla alueella olevaa avainta yritetään lukea. Testin käynnistys ja tulostukset näkyvät konsolitulostuksessa A.2. Rivillä 3 tapahtuu avaimen lukeminen ja riviltä 5 alkaen näkyy lukemisen aiheuttama laitteen uudelleenkäynnistys. Riveillä 10 - 12 näkyy käynnistyksen syy sekä laskuri, joka kertoo monesko uudelleenkäynnistys tämä oli.

Kuten riviltä 12 näkyy, oli käynnistyksen syynä palomuurivirhe, joten testin suoritus onnistui, eli avaimen lukeminen epäonnistui, juuri niin kuin kuuluikin. Rivillä 16 turvallinen käynnistysohjelma kertoo varusohjelman tarkastustuloksen olevan kunnossa, joten se voidaan käynnistää. Tämä tapahtuu rivillä 18.

```
1  ====== Test Protection: Firewall - CODE =================
2
3    -- Reading Key
4
5  = [SBOOT] System Security Check successfully passed. Starting...
6  = [FWIMG] Slot #0 @: 8020000 / Slot #1 @: 800f000 / Swap @: 8029000
7  ~~~~
8  = [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
9  = [SBOOT] STATE: CHECK STATUS ON RESET
10           WARNING: A Reboot has been triggered by a Firewall reset!
11           Consecutive Boot on error counter = 1
12           INFO: Last execution detected error was:Firewall error.
13 = [EXCPT] FIREWALL RESET FAULT!
14 = [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
15 = [SBOOT] STATE: CHECK USER FW STATUS
16           A valid FW is installed in the active slot - version: 1
17 = [SBOOT] STATE: VERIFY USER FW SIGNATURE
18 = [SBOOT] STATE: EXECUTE USER FIRMWARE
19 ~~~~
```

Konsolitulostus A.2: Ohjelmakoodialueen suojaaminen palomuurilla

A.3.3 Testi ”Työmuistialueen suojaaminen palomuurilla”

Tässä testissä työmuistin suojatulla alueella **SRAM1** olevaa tietoa yritetään lukea. Testin käynnistys ja tulostukset näkyvät konsolitulostuksessa A.3. Rivillä 3 tapahtuu tiedon lukeminen ja riviltä 5 alkaen näkyy lukemisen aiheuttama laitteen uudelleenkäynnistys. Riveillä 10 - 12 näkyy käynnistyksen syy sekä laskuri, joka kertoo monesko uudelleenkäynnistys tämä oli.

Kuten riviltä 12 näkyy, oli uudelleen käynnistyksen syynä palomuurivirhe, joten testin suoritus onnistui, eli työmuistialueen lukeminen epäonnistui, juuri niin kuin pitikin. Rivillä 16 turvallinen käynnistysohjelma kertoo varusohjelman olevan kunnossa, joten se voidaan käynnistää. Tämä tapahtuu rivillä 18.

```
1  ====== Test Protection: Firewall - VDATA ================
2
3    -- Reading address: 0x20000000
4
5  = [SBOOT] System Security Check successfully passed. Starting...
6  = [FWIMG] Slot #0 @: 8020000 / Slot #1 @: 800f000 / Swap @: 8029000
7  ~~~~
8  = [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
9  = [SBOOT] STATE: CHECK STATUS ON RESET
10           WARNING: A Reboot has been triggered by a Firewall reset!
11           Consecutive Boot on error counter = 1
12           INFO: Last execution detected error was:Firewall error.
13 = [EXCPT] FIREWALL RESET FAULT!
14 = [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
15 = [SBOOT] STATE: CHECK USER FW STATUS
16           A valid FW is installed in the active slot - version: 1
17 = [SBOOT] STATE: VERIFY USER FW SIGNATURE
18 = [SBOOT] STATE: EXECUTE USER FIRMWARE
19 ~~~~
```

Konsolitulostus A.3: Työmuistialueen suojaaminen palomuurilla

A.3.4 Testi ”Ohjelmakoodin suojaaminen kirjoittamiselta”

Tässä testissä ohjelmamuistin kirjoitussuojatulla alueella olevaa tietoa yritetään muuttaa. Testin käynnistys ja tulostukset näkyvät konsolitulostuksessa A.4. Rivillä 3 tapahtuu tiedon lukeminen ohjelmamuistista ja rivillä 5 tyhjennetään ohjelmamuistialue kirjoitusta varten. Rivillä 7 testi ilmoittaa, että muistialue on kirjoitussuojattu. Tämä kertoo onnistuneesta testistä, joten testin suoritus onnistui, eli ohjelmamuistialueen muuttaminen epäonnistui, juuri niin kuin kuuluikin.

```
1  ====== Test Protection: Firewall - VDATA ================
2
3    -- Reading address: 0x20000000
4
5  = [SBOOT] System Security Check successfully passed. Starting...
6  = [FWIMG] Slot #0 @: 8020000 / Slot #1 @: 800f000 / Swap @: 8029000
7  ~~~~
8  = [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
```

```
 9  = [SBOOT] STATE: CHECK STATUS ON RESET
10            WARNING: A Reboot has been triggered by a Firewall reset!
11            Consecutive Boot on error counter = 1
12            INFO: Last execution detected error was:Firewall error.
13  = [EXCPT] FIREWALL RESET FAULT!
14  = [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
15  = [SBOOT] STATE: CHECK USER FW STATUS A valid FW is installed in the
      ↪ active slot - version: 1
16  = [SBOOT] STATE: VERIFY USER FW SIGNATURE
17  = [SBOOT] STATE: EXECUTE USER FIRMWARE
18  ~~~~
```

Konsolitulostus A.4: Ohjelmakoodin suojaaminen kirjoittamiselta

A.3.5 Testi ”Muuttuneen ohjelmakoodin tunnistaminen”

Tässä testissä poistetaan muistinsuojausyksikkö käytöstä, jotta ohjelmamuistiin voidaan kirjoittaa. Testi muuttaa ohjelmamuistin sisältöä nollaamalla varusohjelman alusta 512 tavua. Testin käynnistys ja tulostukset näkyvät konsolitulostuksessa A.5. Rivillä 3 tapahtuu ohjelmamuistin alkuosan tyhjentäminen ja rivillä 5 ohjelma kertoo mitä uudelleenkäynnistyksessä pitäisi tapahtua. Rivillä 20 testi ilmoittaa, että kunnossa olevaa varusohjelmaa ei löydy ja rivillä 24 ohjelma odottaa, että käyttäjä päivittäisi laitteeseen uuden varusohjelman. Tämä on odotettu tulos, joten testin suoritus onnistui, eli ohjelmamuistialueen muuttaminen tunnistettiin, eikä laite käynnisty muuttuneella varusohjelmalla.

```
 1  ====== Test: CORRUPT ACTIVE IMAGE ======================
 2    -- Disable MPU protection to be able to erase
 3    -- Erasing 0x200 bytes at address: 0x8020200
 4    -- Waiting watchdog reset ...
 5    -- At next boot Signature Verification will fail. Download a new FW
       ↪ to restore UserApp !!
 6
 7
 8  = [SBOOT] System Security Check successfully passed. Starting...
 9  = [FWIMG] Slot #0 @: 8020000 / Slot #1 @: 800f000 / Swap @: 8029000
10  ~~~~
11  = [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
12  = [SBOOT] STATE: CHECK STATUS ON RESET
13            WARNING: A Reboot has been triggered by a Watchdog reset!
14            Consecutive Boot on error counter = 1
15            INFO: Last execution detected error was:Watchdog error.
16  = [EXCPT] WATCHDOG RESET FAULT!
17  = [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
18  = [SBOOT] STATE: CHECK USER FW STATUS
19            Slot #0 not empty : erasing ...
20            No valid FW found in the active slot nor new encrypted
21            FW found in the UserApp download area
22            Waiting for the local download to start...
23  = [SBOOT] STATE: DOWNLOAD NEW USER FIRMWARE
24            File> Transfer> YMODEM> Send ........
25  ~~~~
```

Konsolitulostus A.5: Muuttuneen ohjelmakoodin tunnistaminen

A.3.6 Testi ”Kajoamisen tunnistussisääntulon toiminta”

Tämä testi pyytää käyttäjää kytkemään alatilan kajoamisentunnistusnastaan PA0 ja odottamaan sen jälkeen 10 sekuntia. Laitteen kajoamisenestotoiminnan pitäisi havaita nastan muuttunut tila ja pyyhkiä muisti. Testin käynnistys ja tulostukset näkyvät konsolitulostuksessa A.6. Rivillä 2 ohjelma pyytää käyttäjää kytkemään liityntänasta PA0 alatilaan. Rivillä 5 ohjelma antaa käyttäjälle 10 sekuntia aikaa tehdä kytkentä. Kytkentä tehtiin reilusti alle annetun ajan. Rivillä 6 ohjelma ilmoittaa, että testi epäonnistui. Testin onnistuessa olisi laitteen pitänyt tehdä muistin tyhjennys ja uudelleenkäynnistys eikä sitä tässä tapahtunut, testi siis epäonnistui.

```
1  ====== Test Protection: TAMPER ========================
2    -- Pull PA0 (CN8.1) to GND
3    -- -- Note: sometimes it may be enough to put your finger close to
       ↪ PA0 (CN8.1)
4    -- Should reset if TAMPER is enabled.
5    Waiting for 10 seconds...
6    -- Waited 10 seconds, if you have connected TAMPER pin to GND it
       ↪ means TAMPER protection is NOT ENABLED !!
7  ~~~~
```

Konsolitulostus A.6: Kajoamisen tunnistussisääntulon toiminta

A.3.7 Testi ”Vahtikoiran toiminta”

Tämä testi pysäyttää ohjelman normaalin toiminnan suorittamalla 10 sekunnin viiverutiinin päivittämättä sen aikana vahtikoiraa. Testin käynnistys näkyy konsolitulostuksessa A.7 rivillä 2. Rivistä 3 alkaen näkyy laitteen uudelleenkäynnistys, joten testi on onnistunut. Rivit 8 ja 10 kertovat uudelleenkäynnistyksen syyksi vahtikoiran.

```
 1  ====== Test Protection: IWDG ===========================
 2    -- Waiting 10000 (ms). Should reset if IWDG is enabled.
 3  = [SBOOT] System Security Check successfully passed. Starting...
 4  = [FWIMG] Slot #0 @: 8020000 / Slot #1 @: 800f000 / Swap @: 8029000
 5  ~~~~
 6  = [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
 7  = [SBOOT] STATE: CHECK STATUS ON RESET
 8            WARNING: A Reboot has been triggered by a Watchdog reset!
 9            Consecutive Boot on error counter = 1
10            INFO: Last execution detected error was:Watchdog error.
11  = [EXCPT] WATCHDOG RESET FAULT!
12  = [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
13  = [SBOOT] STATE: CHECK USER FW STATUS
14            A valid FW is installed in the active slot - version: 1
15  = [SBOOT] STATE: VERIFY USER FW SIGNATURE
16  = [SBOOT] STATE: EXECUTE USER FIRMWARE
17  ~~~~
```

Konsolitulostus A.7: Vahtikoiran toiminta

A.3.8 Testi ”Varusohjelmiston päivitys”

Tässä testissä ladataan kortille uusi varusohjelmisto päivityspaketilla, jonka sisältö on vioittunut. Viallinen paketti on tehty niin, että toimivasta paketista on otettu kopio, jonka sisältöä on muokattu heksaeditorilla. Ohjelmanpakettia on muutettu yhden tavun osalta osoitteessa 0x0200.

Konsolitulostuksesta A.8 on nähtävissä testin kulku. Rivillä 4 on lähetetty uusi varusohjelma pääteohjelman tiedonsiirto toiminnolla. Tiedonsiirtoon käytetään YMODEM-protokollaa. Rivillä 7 tiedonsiirto on onnistuneesti päättynyt ja laite käynnistää itsensä uudestaan. Riviltä 19 on nähtävissä, että SFU on huomannut uuden salatun varusohjelman löytyvän muistista ja se tulee purkaa latausta varten. Rivillä 21 ja 22 SFU on huomannut, että allekirjoitus ei täsmää, joten se käynnistää laitteen uudestaan. Riviltä 25 alkaen näkyy uudelleenkäynnistyksen tulostuksia, rivillä 29 näkyy uudelleenkäynnistyksen syy, joka tässä on salauksen purun epäonnistuminen (Decrypt failure). Rivillä 32 SB huomaa, että alkuperäinen varusohjelmabinaari on kunnossa ja käynnistää sen uuden ladatun ja viallisen sijaan. Testi on näin ollen onnistuneesti suoritettu.

```
 1  ================ New Fw Download ========================
 2    -- Send Firmware
 3    -- -- Erasing download area ...
 4    -- -- File> Transfer> YMODEM> Send ..
 5    -- -- Programming Completed Successfully!
 6    -- -- Bytes: 19328
 7    -- Image correctly downloaded - reboot
 8
 9  = [SBOOT] System Security Check successfully passed. Starting...
10  = [FWIMG] Slot #0 @: 8020000 / Slot #1 @: 800f000 / Swap @: 8029000
11  ...
12  = [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
13  = [SBOOT] STATE: CHECK STATUS ON RESET
14            INFO: A Reboot has been triggered by a Software reset!
15            Consecutive Boot on error counter = 0
16            INFO: Last execution detected error was:No error. Success.
17  = [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
18  = [SBOOT] STATE: CHECK USER FW STATUS
19            New Fw Encrypted, to be decrypted
20  = [SBOOT] STATE: INSTALL NEW USER FIRMWARE
21  = [SBOOT] STATE: HANDLE CRITICAL FAILURE
22  = [EXCPT] SIGNATURE CHECK FAILED!
23  = [SBOOT] STATE: REBOOT STATE MACHINE
24  ...
25  = [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
26  = [SBOOT] STATE: CHECK STATUS ON RESET
27            INFO: A Reboot has been triggered by a Software reset!
28            Consecutive Boot on error counter = 0
29            INFO: Last execution detected error was:Decrypt failure.
30  = [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
31  = [SBOOT] STATE: CHECK USER FW STATUS
32            A valid FW is installed in the active slot - version: 1
33  = [SBOOT] STATE: VERIFY USER FW SIGNATURE
34  = [SBOOT] STATE: EXECUTE USER FIRMWARE
35  ====================================================================
36  =              (C) COPYRIGHT 2017 STMicroelectronics                =
37  =                                                                   =
38  =                          User App #D                              =
39  ====================================================================
40  ========= End of Execution ==========
41  ...
```

Konsolitulostus A.8: Varusohjelmiston päivitys

A.4 Tulokset

Edellisen aliluvun kahdeksasta testistä onnistui seitsemän. Ainoastaan testi ”Kajoamisen tunnistussisääntulon toiminta” epäonnistui. Epäonnistumisen syynä on todennäköisesti ohjelmistopaketin konfiguraatiossa oleva virhe, jonka takia kajoamisenestotoimintoja ei ole otettu oikein mukaan turvamoottorin (SE) ominaisuuksiin.

Onnistuneiden testien tuloksista voi päätellä, että suhteellisen pienessä ja halvassa mikro-ohjaimessakin on monipuolisia mahdollisuuksia tehdä toteutettavasta laitteesta tietoturvallinen, kunhan suunnittelussa otetaan tietoturva huomioon ja mieluiten heti suunnittelun alkuvaiheessa.