Tampereen ammattikorkeakoulu

Tiedonsiirron salaaminen mikrokontrollerilla

Juho Tuominen

OPINNÄYTETYÖ
Lokakuu 2024

Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TUOMINEN, JUHO:
Tiedonsiirron salaaminen mikrokontrollerilla

Opinnäytetyö 30 sivua
Lokakuu 2024

Opinnäytetyön tarkoituksena oli toteuttaa C-kielinen ohjelma, jolla voidaan salata ja purkaa sille annettu viesti käyttäen AES-128-salausalgoritmia. Ohjelman suoritusympäristönä toimii mikrokontrolleri. Työssä käytetään kolmea eri tehoista mikrokontrolleria ja niiden suorituskykyä vertaillaan. Valitut mikrokontrollerit ovat MSP430, STM32-F4 sekä STM32-H7. Opinnäytetyön tavoitteena oli kehittää toimeksiantajan ohjelmistoa.

Opinnäytetyön toimeksiantajana on Patria Aviation Oy. Opinnäytetyö toteutettiin kesällä 2024 toimeksiantajalta saaduilla laitteilla. Työn ohjelmointi suoritettiin käyttämällä Code Composer Studio- ja STM32CubeIDE-ohjelmistoja.

Opinnäytetyön tuloksena saatiin ohjelma, joka hyödyntää Texas Instrumentsin AES-128-kirjastoa. Ohjelmalla voidaan salata ja purkaa ohjelmaan koodattu viesti. Viestin pituudeksi voidaan antaa 16, 32, 48 tai 64 tavua. Saatuja suorituskykytuloksia analysoitiin opinnäytetyön lopussa. Opinnäytetyöstä saatuja tuloksia voidaan hyödyntää toimeksiantajan ohjelmiston mahdollisissa jatkokehityksissä.

Asiasanat: mikrokontrolleri, salausalgoritmi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information Technology
ICT Engineering

TUOMINEN, JUHO:
Data Encryption on Microcontroller

Bachelor's thesis 30 pages
October 2024

The purpose of this thesis was to develop a program that can encrypt and decrypt a given message using the AES-128 encryption algorithm on a microcontroller. The program was developed using the C programming language. Three different microcontrollers with different performance levels were used in the thesis, and their performance was compared. The microcontrollers were MSP430, STM32-F4, and STM32-H7.

This thesis was commissioned by Patria Aviation Oy. Patria Aviation Oy is a company that provides lifecycle support services for airplanes and helicopters. It also offers civilian and military flight training.

As a result of the thesis, a program was developed that uses Texas Instruments' AES-128 library. The program was developed using Code Composer Studio and STM32CubeIDE softwares. The program can encrypt and decrypt a message that is hard coded into the program. The message length can be 16, 32, 48, or 64 bytes.

The results of this thesis can be used in the future development of the client’s software.

Key words: microcontroller, encryption algorithm

SISÄLLYS

ERITYISSANASTO

TAMK	Tampereen ammattikorkeakoulu
op	opintopiste
OS	Operating System, Käyttöjärjestelmä
MCU	Microcontroller, Mikrokontrolleri
IDE	Integrated Development Environment, Kehitysympäristö
AES	Advanced Encryption Standard
DES	Data Encryption Standard
CCS	Code Composer Studio
MHz	Megahertsi
kB	kilobyte, kilotavu
Mbit	Megabit
RAM	Random-access memory, satunnaisjärjestyksessä käytettävä muisti
TI	Texas Instruments
XOR	Exclusive Or, Looginen operaatio

1 JOHDANTO

Tietotekniikan ja digitaalisten järjestelmien kehittyessä turvallisuusnäkökohdat ovat nousseet keskeisiksi tekijöiksi kaikissa sovelluksissa, erityisesti silloin, kun sovelluksissa käsitellään arkaluonteisia tietoja, kuten salasanat tai luottamukselliset viestit. Yksi tärkeimmistä keinoista suojata näitä tietoja on käyttää salausalgoritmeja, jotka salaavat tiedon ulkopuolisilta. Tämä opinnäytetyö keskittyy erityisesti AES-128 (Advanced Encryption Standard 128-bittiä) algoritmin käyttöön mikrokontrollerissa.

Mikrokontrollerit ovat pienikokoisia ja energiatehokkaita tietokoneita, joita käytetään laajasti monenlaisissa sovelluksissa, kuten älykelloissa, kodinkoneissa ja teollisuusautomaatiossa. Niiden rajalliset resurssit, kuten muisti ja laskentateho, asettavat erityisiä haasteita ohjelmistojen toteuttamiselle. AES-128 on tunnettu vahvasta turvallisuudesta ja sitä käytetään laajasti eri ohjelmistoissa.

Tässä työssä tarkastellaan kolmen eri tehoisen mikrokontrollerin suoritustehoa AES-128-salausalgoritmin käytössä. Tarkoituksena on toteuttaa ohjelma, joka salaa ja purkaa eri pituiset viestit näillä mikrokontrollereilla.

Opinnäytetyön tavoitteena on tarjota syvällinen ymmärrys AES-128-salauksen integroimisesta mikrokontrolleriin ja arvioida sen käytännön soveltuvuutta eri tehoisilla kontrollereilla. Opinnäytetyö tuo esille salauksen optimointiin liittyviä haasteita ja tarjoaa käytännön esimerkkejä siitä, miten salaus voidaan toteuttaa tehokkaasti pienitehoisessa ympäristössä.

2 SALAUSALGORITMIT

2.1 Salaamisen ja purkamisen historia ja teoria

Salaamisen perusidea on lähettää tietoa vastaanottajalle niin, että kukaan muu ei pääse lukemaan salattua viestiä. Salaamisella pyritään suojautumaan niitä henkilöitä vastaan, jotka lähettäjän tahtomatta yrittävät päästä käsiksi siirrettävään tietoon. Salauksella muutetaan siirrettävä tieto muotoon, jossa sitä on mahdotonta lukea. Vastaanottajalle salaus puretaan, jotta tieto saadaan taas luettavaan muotoon.

Aikaisimmat tunnetut salaukset ajoittuvat 600 eaa. Spartalaisten aikaan. Spartalaiset käyttivät taisteluiden aikana tietyn paksuista keppiä, johon sidottiin nahkainen hihna. Hihnaan oli kirjoitettu kirjaimia ja oikean paksuiseen keppiin kiedottuna, siitä saatiin selväkielinen viesti. Jos hihnan kietoi väärän paksuiseen keppiin, viestiä ei pystynyt lukemaan.

Kuva 1. Spartalaisten käyttämä ”scytale” keppi, johon on kiedottu nahkainen hihna.

Modernin salauksen ensimmäisiä ja kuuluisimpia laitteita oli Saksalainen Enigma salauslaite, jota Saksan armeija käytti 1920-luvun alusta toisen maailmansodan loppuun asti. Enigman toiminta perustui pyöriviin salauskiekkoihin, jotka vaihtoivat painetun näppäimen merkitsemää kirjainta. Viesti kirjattiin ylös ja lähetettiin salattuna vastaanottajalle, joka identtisesti asetetulla Enigma laitteella sai käännettyä sen takaisin selkokieliseksi. Puolalainen Marian Rejewski oli en-

simmäinen henkilö, joka onnistui purkamaan Enigman salauksen ja edisti näin liittoutuneiden voittoa toisessa maailmansodassa.

1970-luvulla kehitettiin ensimmäinen tietokone pohjainen salausalgoritmi, DES (Data Encryption Standard). Se pysyi käytössä Yhdysvaltojen valtion standardina vuoteen 1997 asti, kunnes se saatiin murrettua. Tämän jälkeen 2000-luvulla Yhdysvallat otti käyttöön nykyisinkin käytössä olevan AES (Advanced Encryption Standard) algoritmin.

Salauksen purkamisen ideana on kääntää salattu teksti takaisin luettavaan muotoon. Modernissa salauksessa käytetään matemaattisesti luotuja avaimia, joilla tieto salataan ja puretaan. Tiedon lähettäjällä ja vastanottajalla tulee olla matemaattisesti yhdistettävä avain, jotta salatun tiedon saa purettua. Väärällä avaimella, tai ilman avainta, moderni salattu tieto on lukukelvotonta ja käytännössä mahdotonta purkaa.

2.2 Erilaiset salausmenetelmät

Modernissa salauksessa käytetään pääosin kahta eri salausmenetelmää. Nämä ovat symmetrinen ja epäsymmetrinen salausmenetelmä. Symmetrinen salaus on kevyempi mutta tietoturvattomampi menetelmä kuin epäsymmetrinen salaus.

2.2.1 Symmetrinen salaus

Symmetrisessä salauksessa lähettäjällä ja vastaanottajalla on identtiset avaimet, joilla tieto pystytään salaamaan ja purkamaan. Menetelmä on hyvin yksinkertainen, sekä turvallinen niin kauan kun avaimen sisältö pysyy salaisena. Tämä onkin symmetrisen salauksen suurin ongelma. Jos kolmas osapuoli saa salauksessa käytettävän avaimen haltuunsa, pystyy hän murtamaan salatun viestin. Tätä vastaan on kehitetty erilaisia tapoja, joissa salausavain muuttuu esimerkiksi päivittäin tai viikoittain.

Yleisimmät käyttökohteet symmetriselle salaukselle ovat tiedostojärjestelmien kuten kovalevyjen salaus, VPN- ja IPsec-protokollat, Wi-Fi-verkkojen WPA-salaukset sekä reaaliaikainen tietoliikenne. Näissä usein halutaan priorisoida nopeaa ja kevyttä algoritmia.

2.2.2 Epäsymmetrinen salaus

Epäsymmetrisessä salauksessa tieto salataan eri avaimella kuin millä se puretaan. Menetelmä käyttää julkista ja salaista avainta, jotka ovat toisistaan riippumattomia. Viesti salataan vastaanottajan julkisella avaimella ja puretaan vastaanottajan salaisella avaimella. Julkinen avain on kaikkien tarvittavien osapuolien nähtävillä, mutta salainen avain on vain vastaanottajan hallussa. Jos siis haluaisin lähettää salatun viestin henkilölle X, tarvitsisin hänen julkisen avaimensa, jolla viestini salataan ja tämän jälkeen sen saisi selville vain ja ainoastaan henkilö X.

Epäsymmetrinen salaus on kuitenkin raskas tapa salata tietoa ja salatun tiedon koko voi kasvaa suureksi. Tämän takia näitä kahta salausmenetelmää joskus yhdistellään modernissa tiedon salaamisessa.

Yleisimmät käyttökohteet epäsymmetriselle salaukselle ovat digitaaliset allekirjoitukset, käyttäjän tunnistaminen ja todentaminen, sähköiset maksujärjestelmät sekä SSL/TLS protokollat. Tunnettuja epäsymmetrisiä algoritmeja ovat muun muassa RSA sekä Diffie-Hellman algoritmit.

2.2.3 Symmetrisen ja epäsymmetrisen salauksen yhdistäminen

Menetelmien yhdistämisessä ideana on salata tieto kertakäyttöisellä symmetrisellä avaimella, joka lähetetään viestin mukana vastaanottajalle. Kertakäyttöinen avain salataan epäsymmetrisesti vastaanottajan julkisella avaimella. Tässä menetelmässä ideana on, että vastaanottajan ei tarvitse purkaa koko viestiä epäsymmetrisesti, vaan ainoastaan kertakäyttöinen avain. Avaimen purkaminen epäsymmetrisesti on kevyempää kuin koko viestin, sekä viestin purkaminen symmetrisellä avaimella on kevyempää kuin epäsymmetrisellä.

2.3 Valitun algoritmin teoria

Opinnäytetyössä käytettäväksi algoritmiksi valittiin AES-128. Tämä algoritmi on laajalti käytetty ja tunnettu sen vahvasta turvallisuudesta ja tehokkuudesta. Texas Instruments tarjoaa avoimen lähdekoodin kirjaston, joka tukee tämän algoritmin käyttöä. Kirjaston avulla voidaan salata ja purkaa viestejä 16 tavun lohkoissa. Tämä kirjasto valittiin sen yksinkertaisuuden ja pienen koon takia, joka mahdollistaa suuremman suorituskyvyn algoritmia käytettäessä mikrokontrollereissa.

2.3.1 Salausprosessi

AES-128 on symmetrinen salausalgoritmi, joka käyttää 128-bittistä avainta salaamiseen ja purkamiseen. Algoritmin turvallisuus perustuu avaimen pituuteen ja toistuvien matemaattisten operaatioiden käyttöön. Algoritmi salaa tiedon 128-bittisinä datapaketteina. Tämä tarkoittaa sitä, että tieto on jaettu 16 tavuisiin lohkoihin, jotka käsitellään salauksen aikana.

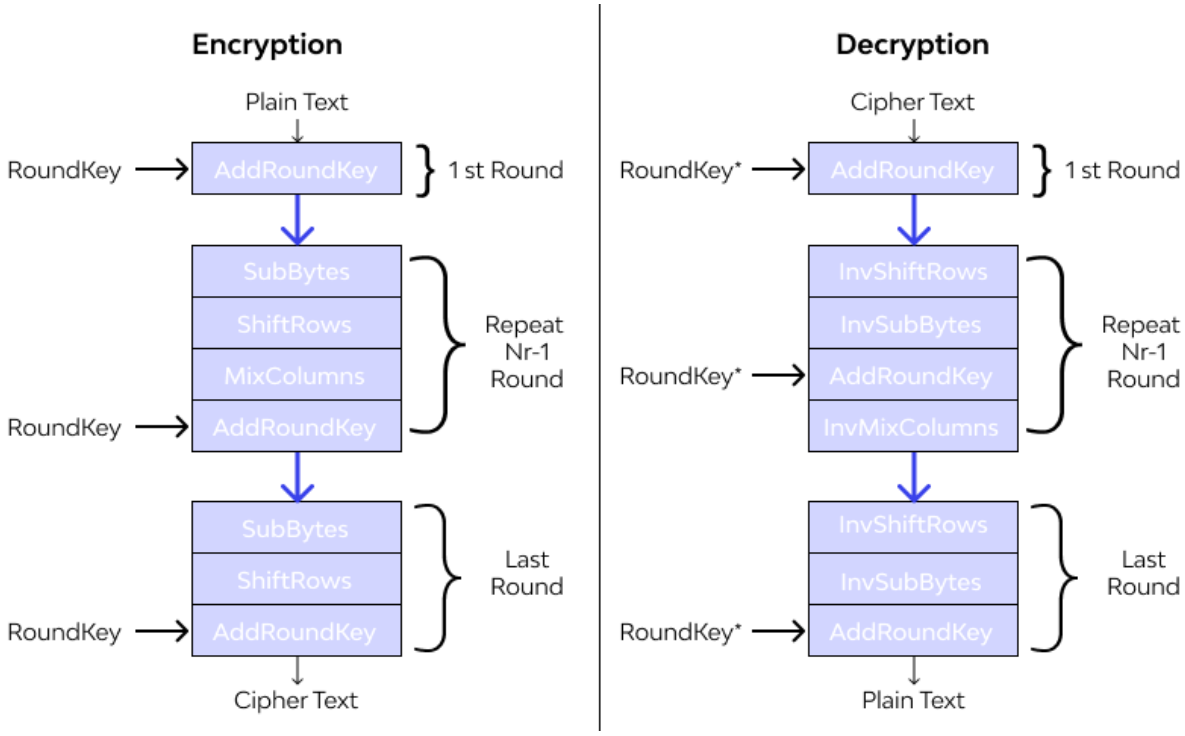
Salausprosessi:

1. Key Expansion: 128-bittinen avain laajennetaan 11 erilliseksi avainlohkoksi.
2. Initial Round: Suoritetaan XOR-operaatio alkuperäisen viestin ja avaimen välillä.
3. Main Rounds: Kymmenen kierrosta, joista jokainen sisältää seuraavat operaatiot:
    - SubBytes: Korvaa jokaisen tavun S-Box-taulukolla.
    - ShiftRows: Siirtää jokaisen rivin tavuja vasemmalle tietyn määrän.
    - MixColumns: Soveltaa matriisikertolaskua jokaiselle lohkon sarakkeelle.
    - AddRoundKey: XOR-operaatio laajennetun avaimen kanssa.
4. Viimeinen kierros: Kuten päävaihe, mutta ilman MixColumns-operaatiota.

Salauksen purkuprosessi

1. Key Expansion:
    - Käytetään samaa avainlaajennusta (Key Expansion) kuin salausprosessissa. 128-bittinen avain laajennetaan 11 erilliseksi avainlohkoksi.
2. Initial Round:
    - Suoritetaan XOR-operaatio salatun viestin ja laajennetun avaimen viimeisen lohkon välillä.
3. Main Rounds:
    - Kymmenen kierrosta, jotka suoritetaan käänteisessä järjestyksessä salausprosessiin verrattuna. Jokainen kierros sisältää seuraavat operaatiot:
        - InvShiftRows: Siirtää jokaisen rivin tavuja oikealle tietyn määrän, joka on käänteinen salauksen ShiftRows-operaatiolle.
        - InvSubBytes: Korvaa jokaisen tavun käänteisellä S-Box-taulukolla.
        - AddRoundKey: Suorittaa XOR-operaation laajennetun avaimen kanssa.
        - InvMixColumns: Soveltaa käänteistä matriisikertolaskua jokaiselle lohkon sarakkeelle.
4. Viimeinen kierros:
    - Kuten päävaihe, mutta ilman InvMixColumns-operaatiota.

S-Box eli substitution box on keskeinen komponentti AES salauksessa. Se toimii salauksen ”SubBytes” -vaiheessa ja sen tehtävänä on vähentää salauksen lineaarisuutta. Jokainen tavu salattavassa lohkossa viedään s-box matriisin läpi. Tässä vaiheessa tavuille suoritetaan kaksi matemaattista muunnosta, käänteisluku- sekä affine-muunnos. Käänteisluku-muunnoksessa jokainen tavu muunnetaan käänteislukuunsa. Affine-muunnos vaiheessa tavuille suoritetaan lineaarinen muunnos XOR-operaatioiden avulla. Tämän jälkeen s-box palauttaa jokaisen tavun uudella arvollaan. Salauksen purussa käytetään käänteistä s-boxia, jotta salauksessa käytetyn s-boxin matemaattiset operaatiot saadaan kumottua.



Kuva 2. AES salaus- ja purkuprosessi.

2.3.2 AES ja DES algoritmit

AES tarjoaa huomattavia parannuksia verrattuna aikaisemmin käytössä olleeseen DES algoritmiin. Suurin eroavaisuus näiden kahden välillä on se, että DES käyttää 56-bittistä salausavainta, kun taas AES käyttää algoritmin tyypin mukaan 128-, 192- tai 256-bittistä avainta. Pidempi salausavain tekee AES:stä turvallisemman ja vaikeammin murrettavan DES verrattuna.

Toinen suuri eroavaisuus algoritmien välillä on niiden tapa salata tieto. Molemmat algoritmit perustuvat lohkosalaukseen, jossa salattava tieto lähetetään tietyn kokoisissa lohkoissa salattavaksi. DES käyttää 8-tavun pituisia lohkoja, kun taas AES käyttää 16-tavuisia tietolohkoja. DES jakaa salattavan tietolohkon kahteen puoliskoon, jonka jälkeen salaus suoritetaan. AES taas tekee koko salattavasta lohkosta matriisin ja salaa sen, jonka takia salaus on nopeampi sekä vaikeampi murtaa. DES:ssä purku ja salaus tehdään samalla algoritmilla, kun taas AES:ssä purku suoritetaan käänteisenä salauksesta.

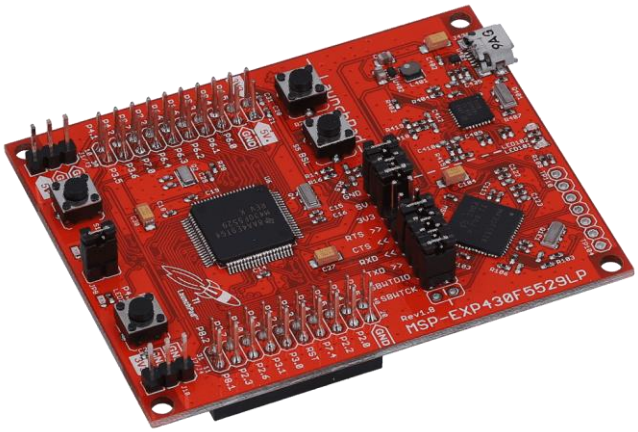
3 MIKROKONTROLLERIT

3.1 Opinnäytetyössä käytettävät mikrokontrollerit

Opinnäytetyössä päädyttiin käyttämään kolmea eri mikrokontrolleria, jotka sisältävät eri tehoiset mikroprosessorit. Valitut kontrollerit ovat MSP430, sekä kaksi STM32 kontrolleria, CortexM4 sekä CortexM7 prosessoreilla. Näillä valituilla kontrollereilla suoritetaan salausalgoritmi sekä salauksenpurkualgoritmi ja niiden tehoja verrataan. Ohjelma kirjoitetaan C-ohjelmointikielellä.

3.1.1 MSP430F5529

Ensimmäinen työssä käytettävä kehitysalusta on MSP430F5529. Se on Texas Instrumentsin suunnittelema ja valmistama alusta, joka sisältää 16-bittisen 25 MHz prosessorin sekä 8 kB RAM-muistia. Se on valituista alustoista heikkotehoisin ja vähiten virtaa käyttävä. MSP430 mikrokontrollerit sopivat hyvin laitteisiin, jotka priorisoivat alhaista virrankulutusta. Esimerkiksi lääketieteelliset laitteet, teollisuuslaitteet ja paristoilla toimivat laitteet.



Kuva 3. MSP430F5529 kehitysalusta

3.1.2 STM32F407G-DISC1

Toiseksi tehokkain työssä käytettävä alusta on STM32F407G-DISC1. DISC liite viittaa STM32 tuotteissa niin sanottuun discovery-alustaan, jossa on paljon erilaisia ominaisuuksia ja ne toimivat hyvin prototyyppien tekemisessä. Käytetty kehitysalusta sisältää 32-bittisen, 168 MHz prosessorin sekä 192 kb RAM-muistia. Se on siis selkeästi MSP430:tä tehokkaampi. Tämänkaltaisia mikrokontrollereita käytetään laajasti automatisoinnissa, moottorien hallinnassa sekä turvallisuusjärjestelmissä.

Kuva 4. STM32F407G-DISC1

3.1.3 STM32H735G-DK

Tehokkain työssä käytetty alusta on STM32H735G-DK. Se sisältää 32-bittisen, 550 MHz prosessorin sekä 128 Mbit RAM-muistia (16000 kB). Se on selkeästi muita käytettyjä alustoja tehokkaampi ja sisältää huomattavan määrän ominaisuuksia verrattuna muihin alustoihin, muun muassa LCD-kosketusnäytön. Tämän tasoisia mikrokontrollereita voidaan käyttää laajasti erilaisissa sulautetuissa järjestelmissä ja alusta tarjoaakin esimerkkiohjelmina erilaisten antureiden ja näyttöjen yhdistelmiä, esimerkiksi lämmönsäätelyä sekä kosketusnäyttöä hyödyntäviä sovelluksia.

Kuva 5. STM32H735G-DK

4 SALAUSOHJELMAN TOTEUTUS

4.1 Kehitykseen käytettävät ohjelmistot

Käytössä on kaksi erilaista mikrokontrolleri tyyppiä, joten työssä käytetään myös kahta erilaista kehitysympäristöä (IDE). MSP430:lle käytetään Code Composer Studio (CCS) ympäristöä, joka tarjoaa ohjelmointiin, debuggaukseen sekä koodin analysointiin tarvittavia työkaluja.

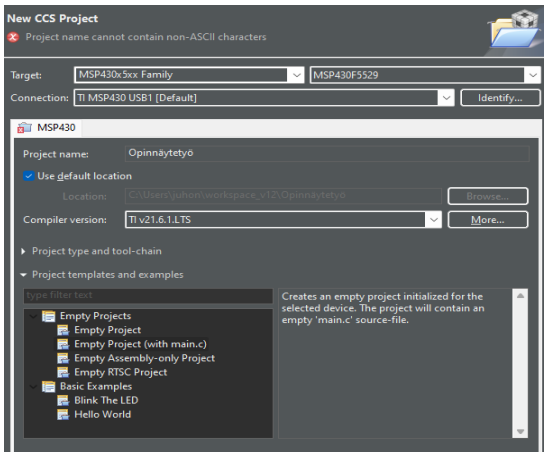
STM32-kontrollereille käytetään työssä STM32CubeIDE ympäristöä, joka tarjoaa samat työkalut kuin CCS, mutta optimoituna STM32:lle.

Molemmat IDE:t ovat ilmaisia ja toimivat niin Windows- kuin Linux-käyttöjärjestelmillä. Tässä työssä ohjelmointiin käytettävä käyttöjärjestelmä on Windows.

Ohjelman toteuttamiseen käytettiin C-ohjelmointikieltä sekä TI:n vapaasti saatavilla olevaa AES128 kirjastoa.

4.1.1 Ohjelmistokehitys Code Composer Studio:lla

Ensimmäisenä tehdään uusi projekti CCS:llä. Projekti nimetään, valitaan tallennuspaikka sekä mille alustalle projekti on tarkoitettu. Näiden vaiheiden jälkeen voidaan kytkeä käytettävä alusta tietokoneeseen.



Kuva 6. Projektin luominen CCS:ssä

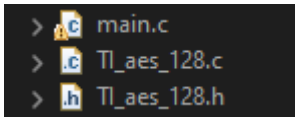
Tämän jälkeen IDE tarjoaa alustetun main-funktion, joka pysäyttää watchdog ajastimen ja päättyy sen jälkeen. Main-funktio on useimmissa koodikielissä funktio, josta ohjelma lähtee liikkeelle. Main suoritetaan siis ensimmäisenä ja sen sisältä päädytään muihin funktioihin ja koodin osiin.

```
#include <msp430.h>


/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer

    return 0;
}

```

Kuva 7. CCS:n alustama main funktio.

Watchdog ajastinta käytetään ohjelman ”vahtimiseen” ja virhetilanteiden estämiseen. Watchdog laskee asetetusta aika-arvosta alaspäin ja kun se saavuttaa nollan, se uudelleenkäynnistää ohjelman. Sen voi nollata eri vaiheissa koodia ja useimmiten se on käytännöllisintä niin sanotuissa loopeissa, joita ohjelma pyörittää useampaan kertaan toimintansa aikana. Jos ohjelma jää jumiin, eikä watchdogin nollausta saavuteta koodissa, watchdog käynnistää ohjelman uudestaan.

Tämän jälkeen lisätään käytettävä AES kirjasto ohjelmakoodin hakemistoon, jotta se on käytettävissä main funktiossa.



Kuva 8. Ohjelmakoodin hakemisto, main lähdekoodi, kirjaston lähdekoodi ja kirjaston ”header” tiedosto, jossa lähdekoodin funktiot, muuttujat, rakenteet ja makrot alustetaan.

Lähdekoodi kirjoitetaan main funktioon, ja siinä käytetään hyväksi AES kirjaston tarjoamaa aes_enc_dec funktiota. Nimi tulee sanoista ”AES encryption decryption”, joka tarkoittaa sitä, että funktiolla voidaan salata ja purkaa viestejä, riippuen sille annetuista parametreistä.

Ensimmäiseksi koodissa nollataan watchdog-ajastin ja alustetaan salattavat viestit. Viestejä on neljä kappaletta ja ne ovat 16 tavua pitkiä. Viestejä on valittu neljä, koska opinnäytetyössä tutkitaan mikroprosessorin suorituskykyä eri pituisten viestien salaamisessa. AES-128-algoritmi pystyy kuitenkin kerrallaan salaamaan tai purkamaan vain 16-tavuisia lohkoja, joten viestit on jaettu jo etukäteen sopiviksi lohkoiksi. Viestit on esitetty heksadesimaaleina.

```
9
10 int main( void )
11 {
12   WDTCTL = WDTPW | WDTHOLD;            // Stop watchdog timer
13   P1DIR |= 0x01;
14
15   unsigned char messages[4][16] = {
16         {0x56, 0x65, 0x72, 0x79, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74, 0x4d, 0x65, 0x73, 0x61, 0x67, 0x65},
17         {0x41, 0x6e, 0x6f, 0x74, 0x68, 0x65, 0x72, 0x4d, 0x65, 0x73, 0x73, 0x61, 0x67, 0x65, 0x54, 0x6f},
18         {0x54, 0x65, 0x6c, 0x6c, 0x59, 0x6f, 0x75, 0x72, 0x4d, 0x6f, 0x6d, 0x49, 0x53, 0x61, 0x69, 0x64},
19         {0x45, 0x6d, 0x62, 0x65, 0x64, 0x64, 0x65, 0x64, 0x50, 0x72, 0x6f, 0x67, 0x72, 0x61, 0x6d, 0x73}
20   };
21
```

Kuva 9. Watchdogin nollaus sekä salattavien viestien taulukon alustus.

Seuraavaksi alustetaan muuttuja salauskierrosten (eli salattavien viestien määrän) valitsemiselle, sekä alustetaan käytettävät avaimet. Tässä ohjelmassa avaimet ovat kovakoodattuna ohjelmaan, mutta oikeassa tilanteessa nämä avaimet tulisivat jotakin ulkoista kautta käytettäväksi ohjelmalle, koska niiden salassapito on avainasemassa viestien salaamisen ja purkamisen kannalta. Avaimet ovat 16 tavuisia ja niiden täytyy olla samanlaiset, jotta symmetrinen salaus voidaan toteuttaa.

```
21
22   const int encryptionRounds = 1; // 1 = 16 byte, 2 = 32 byte, 3 = 48 byte, 4 = 64 byte
23
24   unsigned char key1[]   = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
25                             0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
26
27   unsigned char key2[]   = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
28                             0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
29
```

Kuva 10. Salattavien viestien määrän valitseminen sekä avainten alustus. Muuttujan ”encryptionRounds” jälkeen olevassa kommentissa on esitetty eri arvojen tarkoitus. Kommentti erotetaan ohjelmakoodista kahdella kauttaviivalla.

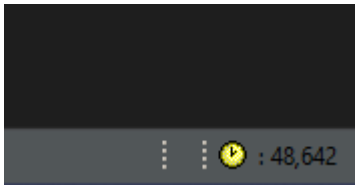
Viimeinen vaihe ohjelmassa on salauksen ja purkamisen toteutus, jotta siihen käytetty aika saadaan mitattua. Indeksinä käytetään muuttujaa ”i”, joka määrittää, kuinka monta kertaa looppi käydään läpi. Yhden kierroksen aikana salataan indeksin mukainen viesti, puretaan se ja lopuksi kopioidaan key2 avaimen arvo

key1:seen. Tämä tehdään siitä syystä, että salauksen aikana myös avaimen arvo muuttuu. Kuitenkin jos halutaan ajaa useampia kierroksia samoilla avaimilla, avaimen arvo täytyy palauttaa samanlaiseksi key2:n kanssa.

```
29
30  unsigned int i;
31
32  for(i = 0; i < encryptionRounds; i++)
33  {
34      aes_enc_dec(messages[i], key1, 0);  // Encrypt
35      aes_enc_dec(messages[i], key2, 1);  // Decrypt
36
37      memcpy(key1, key2, sizeof(key1));
38  }
39
40  return 0;
41 }
```

Kuva 11. Salaus sekä purku tapahtuu for-loopin sisällä, aiemmin valittujen kierrosten määrän mukaan.

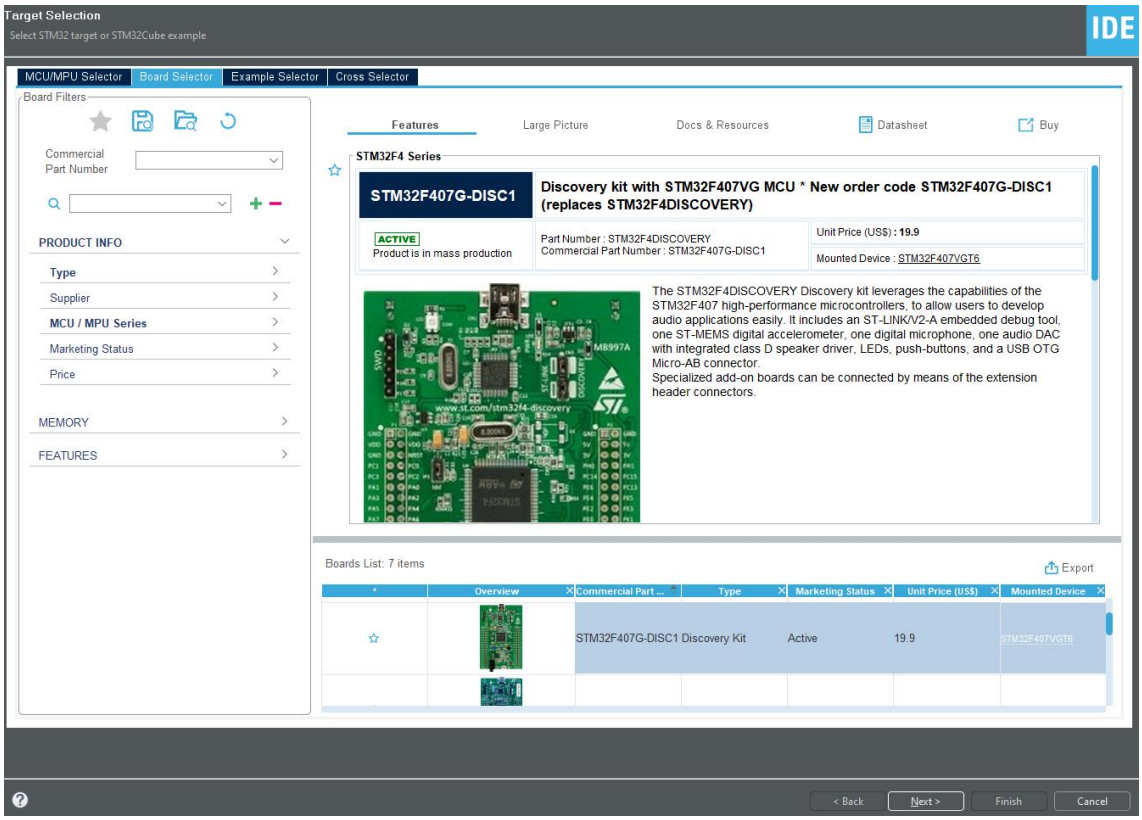
Tämän jälkeen ohjelmaa ajetaan eri kierros arvoilla (1-4) ja debuggerilla tutkitaan ensin salaukseen ja sitten purkamiseen käytetty aika. Tämä saadaan selville laskemalla prosessorin kellojaksot (clock cycle) ja jakamalla se prosessorin nopeudella (nopeus hertseinä). Code Composer Studio näyttää prosessorin kellojaksot debug moodissa, joka helpottaa käytetyn ajan laskemista. Prosessorin nopeus löytyi mikrokontrollerin dokumentaatiosta. Saadut tulokset käydään läpi myöhemmin opinnäytetyössä.



Kuva 12. Kellojaksojen näkymä debug tilassa.

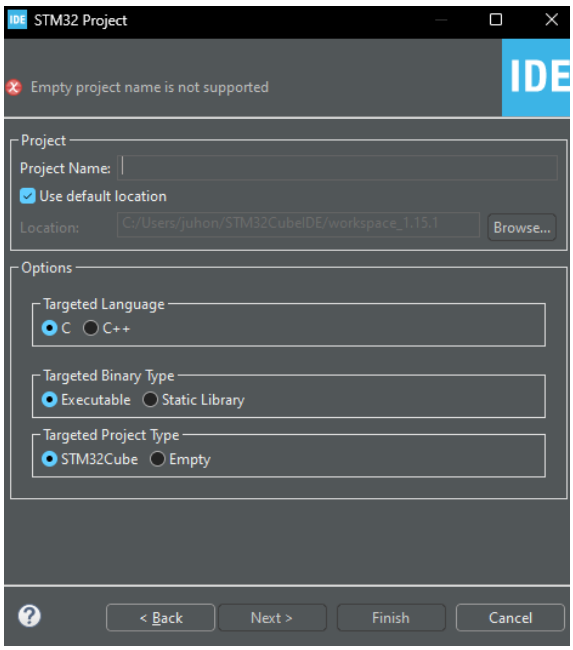
4.1.2 Ohjelmistokehitys STM32CubeIDE:llä

STM32CubeIDE on hyvin samankaltainen IDE kuin Code Composer Studio. Merkittävin ero on käytettävän alustan valinnassa. STM32 mikrokontrollereita on lukuisia erilaisia, joten käytettävän alustan valinta ei ole ihan yhtä simppeliä kuin CCS:ssä. Käytettävä kontrolleri kuitenkin löytyi luettelosta suhteellisen helposti, kun oikeat hakuehdot olivat valittuna. ”Type” valikosta valittiin ”Discovery kit” ja ”MCU/MPU Series” valikosta ”STM32F4” jonka jälkeen listasta löytyi tässä työssä käytettävä STM32F407G kontrolleri.



Kuva 13. Alustan valinta STM32CubeIDE:ssä.

Tämän jälkeen valittiin projektin nimi, käytettävä ohjelmointikieli, ohjelman tyyppi (suoritettava vai kirjasto) sekä projektin tyyppi. Itse ohjelmakoodin kirjoitus tapahtui samaan tapaan kuin CCS:ssä, eikä sitä käydä tässä kohtaa uudestaan läpi.

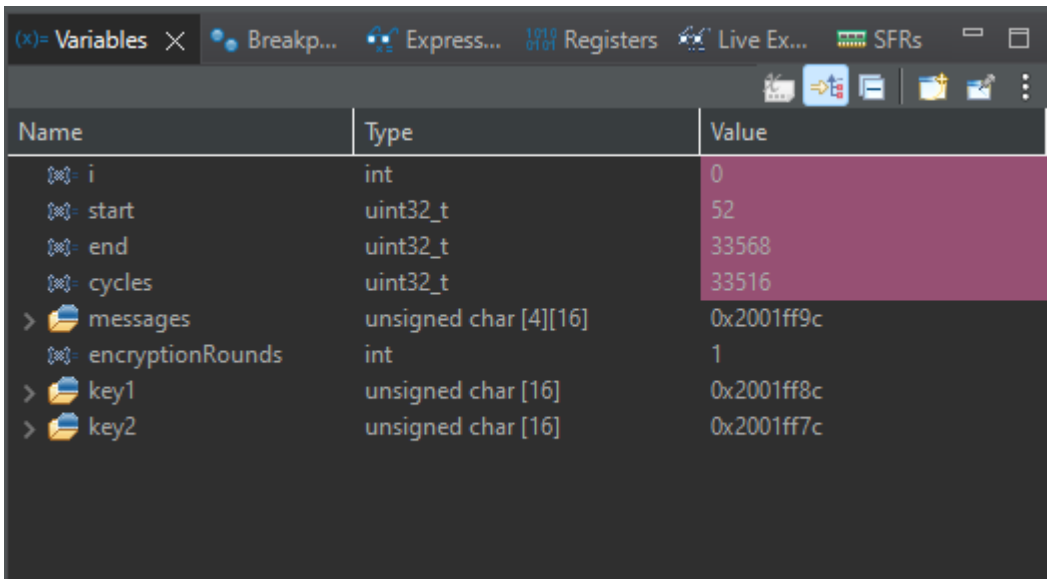


Kuva 14. Projektin alustaminen STM32CubeIDE:ssä.

Ainoa merkittävä ero näiden ohjelmakoodien välillä oli käytettyjen prosessorin kellojaksojen laskemisessa. Siinä missä Code Composer Studio näytti jaksot debug-tilassa, STM32CubeIDE ei sitä tehnyt. Tämä ongelma ratkaistiin käyttämällä ARM Cortex-M-prosessoreista löytyvää "Cycle Counter" -rekisteriä, joka laskee kellojaksojen määrän viimeisimmän nollauksen jälkeen. "Start"-muuttuja ottaa itseensä rekisterin arvon ennen suoritettavan funktion ajamista, "end"-muuttuja taas funktion suorittamisen jälkeen. "Cycles"-muuttuja laskee kuluneet jaksot. Näitä arvoja voidaan tutkia debug-tilassa ja niiden perusteella voidaan laskea käytetty aika kappaleessa 4.2 esitetyllä kaavalla.

```
for(int i = 0; i < encryptionRounds; i++)
{
    start = DWT->CYCCNT;
    aes_enc_dec(messages[i], key1, 0);  // Encrypt

    end = DWT->CYCCNT;
    cycles = end - start;

    start = DWT->CYCCNT;
    aes_enc_dec(messages[i], key2, 1);  // Decrypt

    end = DWT->CYCCNT;
    cycles = end - start;

    memcpy(key1, key2, sizeof(key1));
}
```

Kuva 15. Kellojaksojen laskeminen ohjelmakoodissa.



Kuva 16. Kellojaksojen näyttäminen debug tilassa. Cycles muuttuja näyttää salaukseen käytetyt jaksot.

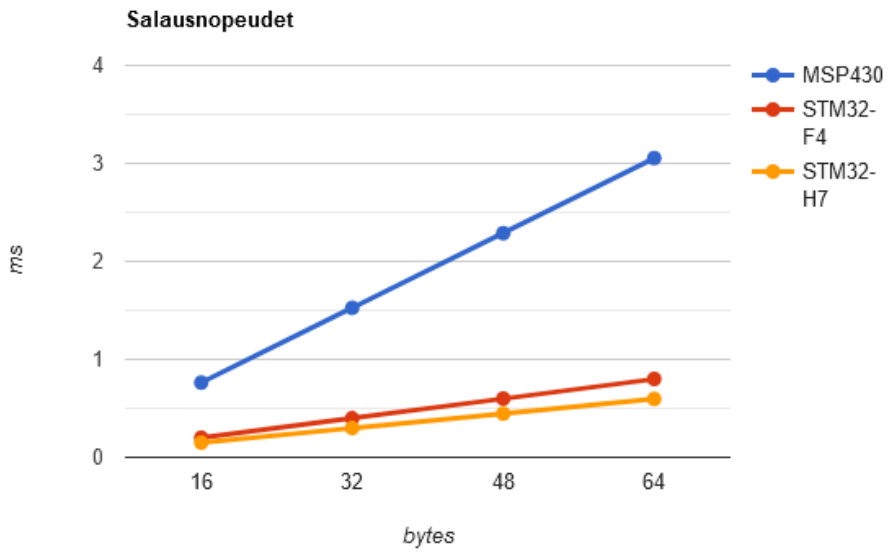
4.2 Saadut tulokset

Tulokset on laskettu kellojaksojen ja prosessorien nopeuksien mukaan kaavalla $\frac{Kellojaksot}{Prosessorinnopeus(Hz)}$. Tulokset on esitetty millisekunteina.

SALAUS	16 Tavua	32 Tavua	48 Tavua	64 Tavua
MSP430	0,763	1,526	2,289	3,053
STM32-F4	0,199	0,399	0,598	0,798
STM32-H7	0,149	0,298	0,446	0,595

PURKU	16 Tavua	32 Tavua	48 Tavua	64 Tavua
MSP430	1,175	2,349	3,524	4,699
STM32-F4	0,306	0,611	0,917	1,222
STM32-H7	0,226	0,452	0,678	0,904

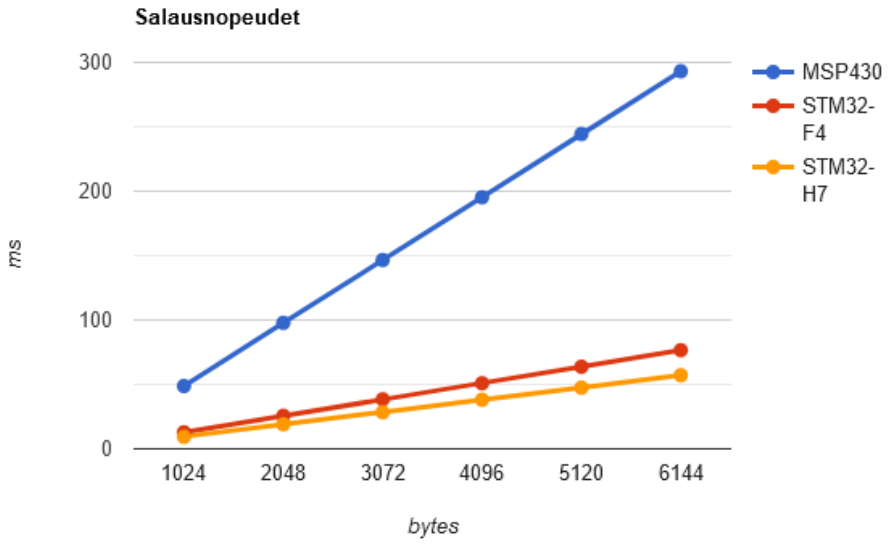
Kuva 17. AES-128 tulokset esitettynä taulukoissa. Tulokset pyöristetty kolmen desimaalin tarkkuudella.



Kuva 18. Salausnopeudet esitettynä viivakaaviossa.

Odotusten mukaisesti voidaan todeta, että käytetty aika kasvaa lineaarisesti salattavan viestin koon mukaan. AES-128 toimii lohkosalaus periaatteella, eli se salaa ja purkaa viestin 16 tavun lohkoissa. Lohkojen lisääminen (eli viestin pituuden muuttaminen) kasvattaa käytettyä aikaa lineaarisesti. Huomataan myös, että salauksen purkaminen vie enemmän aikaa kuin viestin salaaminen.

Salaus suoritettiin vertailun vuoksi myös suuremmilla viestin pituuksilla. Pituuksina käytettiin 1024, 2048, 3072, 4096, 5120 sekä 6144 tavua, eli 1–6 kilotavua. Alla olevasta kuvaajasta huomataan, kuinka MSP430 alkaa olemaan huomattavasti hitaampi suuremmilla data määrillä kuin verrokkinsa. 6 kilotavun viestin salaamiseen kuluu aikaa MSP430:lla lähes 300 ms, kun taas molemmat STM32 mikrokontrollerit suoriutuvat tästä alle 100 ms.



Kuva 19. Suurempien viestien salausnopeudet viivakaaviossa.

	1 kb	2 kb	3 kb	4 kb	5 kb	6 kb
MSP430	48,848	97,696	146,544	195,392	244,24	293,088
STM32-F4	12,768	25,536	38,304	51,072	63,84	76,608
STM32-H7	9,522	19,044	28,566	38,088	47,661	57,132

Kuva 20. Nopeudet taulukkona. Tulokset pyöristetty kolmen desimaalin tarkkuudella ja esitetty millisekunteina.

Saaduista tuloksista voidaan laskea teoreettinen salattu tiedonsiirron nopeus kaavalla $\frac{siirrettävä\ data\ (tavu)}{käytetty\ aika\ (s)}$. Esimerkiksi MSP430 teoreettinen tiedonsiirron nopeus on $\frac{16\ tavua}{0{,}000763\ s} = 20\,969{,}86\ tavua/s$ eli pyöristettynä noin 20,97 kilotavua sekunnissa. Vastaavasti STM32-F4:n siirtonopeus on noin 80,40 kB/s ja STM32-H7:n siirtonopeus on 107,38 kB/s. Tulosten perusteella vodaan todeta, että MSP430 jää selkeästi hitaimmaksi valituista mikrokontrollereista.

	Tiedonsiirto	Vastaanotto
MSP430	20,97 kb/s	13,62 kb/s
STM32-F4	80,40 kb/s	52,29 kb/s
STM32-H7	107,38 kb/s	70,80 kb/s

Kuva 21. Teoreettiset salatun tiedonsiirron ja vastaanottamisen nopeudet.

Huolimatta siitä, että STM32-H7 on nopein vertailussa oleva vaihtoehto viestien salaamiseen ja purkamiseen, tulee kuitenkin sopivaa kontrolleria valittaessa huomioida myös muita riippuvuuksia. Esimerkiksi MSP430 on tunnettu matalasta virrankulutuksestaan ja sen suunnittelussa on erityisesti huomioitu energiatehokkuus. Tämän lisäksi MSP430 on halvempi ja fyysisesti pieni kokoisempi kuin STM32-H7. Myös STM32-F4 suoriutui vertailussa varsin hyvin ja onkin muilta ominaisuuksiltaan kahden edellä mainitun välistä. Kuitenkaan se ei yllä virrankulutuksessa MSP430 tasolle.

Tehokkain työssä käytetty mikrokontrolleri STM32-H7 tukee myös laitteistotason salausta (hardware encryption). Tämä tarkoittaa sitä, että kontrollerilla pystytään salaamaan annettu tieto myös raudalle asennetulla erillisellä moduulilla. Tämän moduulin käyttö on huomattavasti nopeampaa kuin salauksen tuottaminen ohjelmiston avulla. Saaduissa mittauksissa huomataan, että laitteistotason salaus vei aikaa ainoastaan 0,00578 ms, verrattuna ohjelmistotason 0,149 ms. Samaten salauksen purku oli huomattavasti nopeampaa laitteistotasolla, 0,00626 ms verrattuna ohjelmiston 0,226 ms. Laitteistotason salaus on siis noin 25 kertaa nopeampi ja purku noin 36 kertaa nopeampi kuin ohjelmistotason salaus. Tästä voidaan myös laskea teoreettinen siirtonopeus, joka on aiemmin

esitetyn kaavan mukaisesti noin 2786,17 kb/s, eli noin 2,8 Mb/s. Purkunopeus laitteistotasolla on 2555,91 kb/s eli noin 2,6 Mb/s.

Kuitenkin molemmilla salauksilla on omat vahvuutensa ja käyttökohteensa. Ohjelmistotason salaus on helpompi toteuttaa mille tahansa mikrokontrollerille, kun taas laitteistotason salaus on käytettävissä ainoastaan sitä tukevilla mikrokontrollereilla. Esimerkiksi tässä opinnäytetyössä käytetyistä mikrokontrollereista vain yksi tuki laitteistotason salausta.

```
start = DWT->CYCCNT;
if (HAL_CRYP_Encrypt(&hcryp, plaintext, sizeof(plaintext), ciphertext, HAL_MAX_DELAY) != HAL_OK) {
    // Encryption Error
    Error_Handler();
}

end = DWT->CYCCNT;
start = DWT->CYCCNT;

if (HAL_CRYP_Decrypt(&hcryp, ciphertext, sizeof(ciphertext), decryptedtext, HAL_MAX_DELAY) != HAL_OK) {
    // Decryption Error
    Error_Handler();
}

end = DWT->CYCCNT;
```

Kuva 22. Laitteistotason salaus- ja purkufunktiot. Start ja end muuttujia käytetään kellojaksojen laskemiseen.

5 POHDINTA

Opinnäytetyön aiheena oli salausalgoritmin käyttö mikrokontrollereissa ja niiden suorituskykyjen mittaus. Erityisesti opinnäytetyössä keskityttiin AES-128 algoritmiin sekä MSP430, STM32-F4 ja STM32-H7 mikrokontrollereihin. Työhön kuului tausta-aineiston sekä teorian etsiminen ja tutkiminen, käytettäviin mikrokontrollereihin tutustuminen, ohjelmakoodin toteuttaminen sekä tulosten analysointi. Työn tuloksena saatiin ohjelma, joka suorittaa salauksen ja purkamisen eri pituisilla viesteillä mikrokontrollerissa. Ohjelmasta tehtiin kaksi eri versiota, koska työssä käytettiin kahta eri mikrokontrolleri tyyppiä.

Saaduista mittaustuloksista voidaan todeta, että käytetty aika salauksen ja purkamisen yhteydessä kasvaa lineaarisesti salattavan viestin koon mukaan. Tulosten analysoinnista huomataan myös, että viestin purkaminen vie enemmän aikaa kuin sen salaaminen. Tämä on hyvä ottaa huomioon erityisesti sovelluksissa, joissa tiedon vastaanottaminen ja käsittely on keskeisessä osassa.

Teoreettiset siirtonopeudet vahvistavat sen, että MSP430 on suorituskyvyltään selkeästi STM32 kontrollereita alhaisempi. Sen sisältämä prosessori oli myös selkeästi muita heikompi, joten tulos ei ollut yllättävä. Jos tietoa halutaan siirtää ja vastaanottaa paljon, jompikumpi tehokkaammista kontrollereista voisi olla parempi vaihtoehto. Jos taas halutaan mahdollisimman alhaista virrankulutusta, voi MSP430 olla muita vaihtoehtoja parempi valinta. Huomioitavaa on myös se, että vaikka STM32H7:n prosessori oli 22 kertaa nopeampi kuin MSP430:n prosessori, ei salaus ja purku kuitenkaan ollut 22 kertaa nopeampi. Tämä johtuu siitä, että prosessorin nopeuteen vaikuttaa myös muun muassa muistin nopeus.

Yhteenvetona voidaan todeta, että sopivan mikrokontrollerin valinta kryptografisissa sovelluksissa on moniulotteinen prosessi, joka vaatii tarkastelua niin nopeuden kuin virrankulutuksen näkökulmasta. Aikaa kuluu pelkän salauksen lisäksi myös esimerkiksi datan vastaanottoon ja sen lähettämiseen. Kaikki nämä tekijät tulisi ottaa huomioon sopivaa mikrokontrolleria valittaessa. Eri järjestelmien vaatimukset vaikuttavat ratkaisevasti siihen, minkälainen mikrokontrolleri sopii käytettäväksi järjestelmässä.

LÄHTEET

International Journal of Communication Networks and Information Security. 2020. Symmetric Encryption Algorithms: Review and Evaluation study. Viitattu 29.9.2024
https://www.researchgate.net/profile/Haneen-Alabdulrazzaq/publication/349324592_Symmetric_Encryption_Algorithms_Review_and_Evaluation_study/links/602acfa7a6fdcc37a82c0189/Symmetric-Encryption-Algorithms-Review-and-Evaluation-study.pdf

International Journal of Network Security. 2010. Evaluating The Performance of Symmetric Encryption Algorithms. Viitattu 29.9.2024
https://d1wqtxts1xzle7.cloudfront.net/105571997/ijns-2010-v10-n3-p213-219-libre.pdf?1694072034=&response-content-disposition=inline%3B+filename%3DEvaluating_The_Performance_of_Symmetric.pdf&Expires=1727619126&Signature=LmZ0VRfF3kpKJXUBpthkK3JGS7MX3F9ua7yv07-aDKrbNjJWjaEkpBZYeGAd6KsnaXL3Bjvk--qZdLYYKvwkbEhLsPaJ1e9UUjNkFYqnNVpjjrLdCsVGjzBLPJII6KXbZKntOQqNI4v1PysK6y75MKzIt7kQ0ldfxgQ8Ny14c~aySgCuH6XB9WLQ60SITc1VDZDVR2P8UY7vdRf0WhoVbRnYvp443977kbeII0vMq3plUMuLy7D4WA6tDVdXzu5M47v1EvaY1kZJpQ5mlWBHFdWhAkY~B1lSwSpQJjvqe5vk~GHesULJmVIxTNwekjMgpKpAsH3FHRK2hkRLGGN7Cw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

Jyväskylän yliopisto. Tiedonsalaaminen. Viitattu 15.5.2024
https://appro.mit.jyu.fi/doc/tiedonsalaus/

Smart Solutions For Home. 2021. How to program STM32. Viitattu 3.6.2024
https://smartsolutions4home.com/how-to-program-stm32/

SSL Insights. What is AES 128-Bit Encryption?. Viitattu 20.5.2024
https://sslinsights.com/what-is-aes-128-bit-encryption/

STMicroelectronics. STM32F407G. Viitattu 3.6.2024
https://www.st.com/en/evaluation-tools/stm32f4discovery.html

STMicroelectronics. STM32H735G. Viitattu 3.6.2024
https://www.st.com/en/evaluation-tools/stm32h735g-dk.html

Texas Instruments. AES-128 Advanced Encryption Standard. Viitattu 2.6.2024
https://www.ti.com/tool/AES-128

Texas Instruments. MSP430F5529 Datasheet. Viitattu 2.6.2024
https://www.ti.com/product/MSP430F5529

Texas Instruments. MSP430 MCU design & development. Viitattu 2.6.2024
https://www.ti.com/design-development/embedded-development/msp430-mcus.html

Thales. 2023. A Brief History of Encryption. Viitattu 15.5.2024

https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption