Opinnäytetyö (AMK)

Tietotekniikan koulutusohjelma

Sulautetut ohjelmistot

2017

Jero Nikkanen

AUTENTIKOINTI- JA SALAUSALGORITMIEN SUORITUSMITTAUKSET PIIREILLÄ STM32F407 JA STM32F100

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Sulautetut ohjelmistot

2017

Jari-Pekka Paalassalo | Marko Vilola

Jero Nikkanen

AUTENTIKOINTI- JA SALAUS ALGORITMIEN SUORITUSMITTAUKSET PIIREILLÄ STM32F407 JA STM32F100

Opinnäytetyön tarkoituksena oli tutkia eri autentikointi ja salausalgoritmeja sekä valita algoritmien suoritusarvoja tutkimalla parhaat vaihtoehdot Telesten laitteille. Työssä käytettiin testialustana kahta mikropiiriä, joiden prosessoreita Teleste käyttää HFC-verkon laitteissaan.

Opinnäytetyön tuloksena syntyi Excel-taulukko, joka sisältää yleisimpien autentikointi- ja salausalgoritmien mittaustulokset sekä STM32F407 että STM32F100-mikropiireillä mitattuna. Mittaukset sisältävät prosessorien suoritusajat, Flash ja RAM-muistin käytön sekä pinon käytön ohjelman pahimpina ruuhka aikoina.

ASIASANAT:

algoritmi, taulukko, autentikointi, salaus, prosessori, Flash, RAM, pino

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme

Completion year of the thesis | Total number of pages

Instructor(s)

Author(s)

AUTHENTICATION AND ENCRYPTION PERFORMANCE MEASUREMENTS USING STM32F407 AND STM32F100 MICROCONTROLLERS

The aim of this thesis was to measure the performance of authentication and encryption algorithms used in Teleste's devices and evaluate which of them is the most fitting. The test environment consisted of two microcontrollers containing the same processors as Teleste is using in some of their HFC-network devices.

The final product of the thesis was a table containing the performance values of the most common authentication and encryption algorithms using the STM32F407 and STM32F100 microcontrollers. The measurements performed consist of the execution times of the processors, the usage of flash and RAM as well as the size of stack when the program is under the heaviest load.

KEYWORDS:

algorithm, table, authentication, encryption, decryption, processors, flash, RAM, stack

SISÄLTÖ

KUVAT

TAULUKOT

KÄYTETYT LYHENTEET

AES	Advanced Encryption Stantard, lohkosalausalgoritmi, jota käytetään tietotekniikassa
DES	Data Encryption Standard, lohkosalausalgoritmi
IDE	Integrated development enviroment, eli ohjelmointi ympäristö
TDES	Triple Data Encryption Standad, uudempi versio DES-algoritmista
MD5	Message-digest-algoritmi, kryptograafinen tiivistefunktio
SHA	Secure Hash Algorithm, kryptograafinen tiivistefunktio
XXTEA	Corrected Block TEA, lohkosalaus-algoritmi
ST	STmicroelectronics, Euroopan suurin mikroprosessorien valmistaja.
RAM	Random-access memory, laitteen käyttömuisti
ARM	Advanced RISC Machines, 32-bittinen mikroprosessoriarkkitehtuuri

1 JOHDANTO

Työssä käsitellään autentikointi- ja salausalgoritmien suoritusarvoja. Tavoitteena on löytää suoritusarvoiltaan parhaat algoritmit sekä salaukseen, että autentikointiin, kun rajoittavana tekijänä ovat kohdelaitteen pienet muistiresurssit. Opinnäytetyön toimeksiantaja oli Teleste Oyj, joka tarjosi testiympäristöksi kaksi mikropiiriä, joita se käyttää laitteissaan. Mittauksissa tuli ottaa huomioon Telesten tiedonsiirrossa käyttämät datapakettien kokoluokat sekä prosessorien pienet resurssit flash- ja RAM-muistissa.

Työn toteutus aloitettiin valitsemalla ohjelmointialustaksi AtollicTrueStudio, sekä ohjelmointikieleksi C. Työssä käytettävät algorimit löytyvät ST-microelectronicsin tarjoamasta krypto-kirjastosta sekä opensource-kirjastoista. Työn toteuttaminen vaati kunnollista perehtymistä sulautettujen järjestelmien työympäristöön sekä tutustumista salaus ja autentikointi menetelmiin.

Autentikointia ja salausta koskevia tutkimuksia on tehty aikaisemminkin, mutta tämä tutkimus eroaa niistä kuitenkin siten, että itse algoritmien toiminta ei ole työssä keskeistä, vaan sen sijaan niiden suorittamiseen vaadittavien resurssien kartoittaminen.

Luvussa kaksi käydään läpi mittauksia varten tehtyä pohjatyötä. Autentikointia ja salausta avataan sekä kaikki tutkimuksessa käytetyt työkalut esitellään. Kolmannessa luvussa käydään läpi itse käytännön toteutusta. Metodit, miten suoritusarvot mitattiin, esitellään yksitellen. Myös mittauksissa ilmenneet ongelmat tuodaan esille sekä niihin löytyneet ratkaisut. Neljännessä luvussa esitellään työssä saadut mittaustulokset. Tämän lisäksi pohditaan, mitä niistä voidaan päätellä, ja valitaan parhaat algoritmit mittaustuloksiin vedoten. Viimeisessä kappaleessa mietitään tavoitteiden onnistumista sekä sitä, mihin niitä voidaan tulevaisuudessa käyttää.

2 ALUSTAVA TUTKIMUS

Työ vaati alkaakseen kattavaa tutkimustyötä siitä, miten ohjelmointi ja testaukset tulisi suorittaa. Salaus ja autentikointi ovat tärkeä osa-alue tietotekniikasta, ja ensimmäinen vaihe oli selvittää, miten ne ohjelmointi maailmassa toimivat. Teleste pyysi, että tutkimukseen sisällytetään tiettyjä algoritmeja. Salauksen osalta nämä olivat AES, DES, TDES ja XXTEA. Autentikoinnin puolelta taas MD5 ja SHA1. Tämän lisäksi mittauksiin voitiin lisätä parhaaksi nähtävät algoritmit, jos sellaisia löytyisi.

Työssä käytettävien työkalujen valitseminen vaati tutustumista sulautettujen järjestelmien työympäristöön. Aluksi oli valittava alusta, jolla itse ohjelmointi toteutettaisiin, sekä kirjastot, jotka sisältäisivät itse algoritmit. Myös kohteena oleviin mikropiireihin tutustuminen oli tarpeellista, jotta itse ohjelmointi levyille onnistuisi.

2.1 Autentikointi

Autentikoinnilla tarkoitetaan todennusta. Ohjelmointi ympäristössä se tarkoittaa yleensä käyttäjän aitouden tunnistamista. Tunnistamiseen käytettäviä metodeja on paljon erilaisia, mutta metodit itsessään eivät olleet työn kannalta merkityksellisiä. Oleellista oli tutkia tunnistamisessa käytettäviä algoritmeja.

Perusperiaate autentikoinnissa käytettäville algoritmeille on datan muuttaminen tietyn kokoiseen tiivisteeseen. Täsmälleen sama data tuottaa saman tuloksen, mutta mikä tahansa muutos tuottaa täysin erilaisen lopputuloksen. Datan muuttaminen alkuperäiseen muotoon on myös mahdotonta ja ainoa tapa selvittää alkuperäinen tulos on kokeilemalla, joka vaatii valtavasti laskentatehoa riippuen algoritmista. (Economictimes, 2017.)

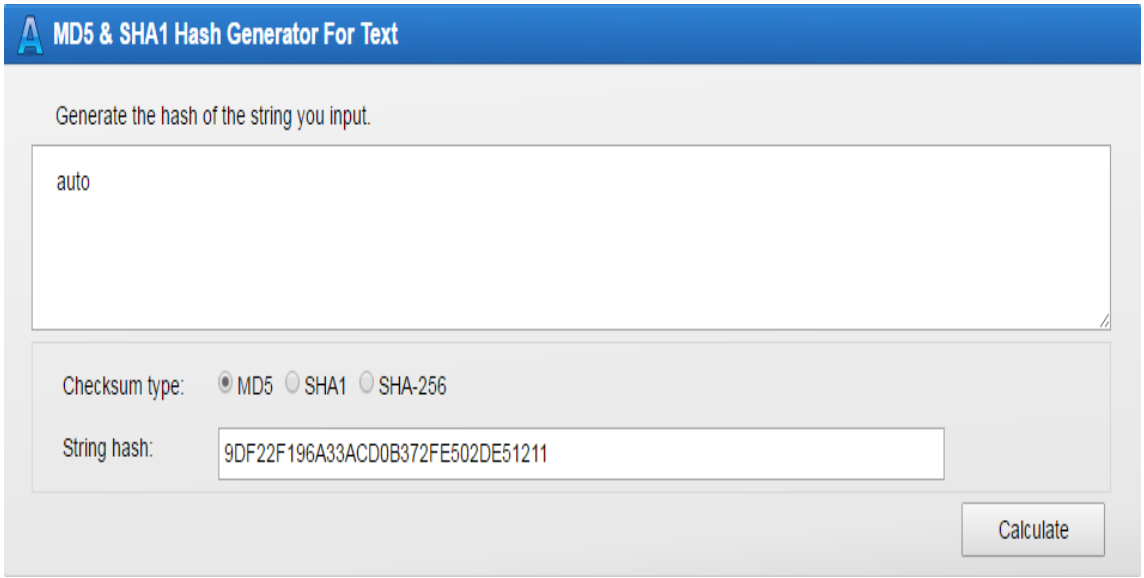
2.1.1 Autentikointialgoritmit

Teleste pyysi sisällyttämään mittauksiin ainakin algoritmit MD5 ja SHA1. Mittauksiin lisättiin myös SHA-2 algoritmin eri versiot SHA256 ja SHA512, sillä niistä sai hyviä vertailukohtia muihin vastaaviin algoritmeihin.

MD5

MD5 on message-digest-algoritmi, jonka alkuperäinen käyttötarkoitus oli kryptografiassa. Nykypäivänä sen katsotaan olevan kuitenkin jokseenkin helposti murrettavissa, joten sitä käytetään enää pääasiassa tiedostojen tarkistus summana (Wikipedia 2017).

MD5-algoritmi toimii samalla periaatteella kuin useimmat muutkin autentikointi algoritmit. Funktio ottaa sisäänsä dataa, jonka se sitten muuntaa 128-bittisen pituiseen tiivisteeseen. Prosessoidusta muodosta on mahdotonta päätellä alkuperäistä sisältöä, eikä tiivistettä voida kääntää vastakkaisprosessilla alkuperäiseen muotoon (Wikipedia 2017). Kuva 1 on esimerkki MD5-algoritmin tiivisteestä sanasta auto.



Kuva 1. Esimerkki MD5-tiivisteestä (onlinemd5, 2017).

SHA

SHA, eli Secure-Hash-Algorithms, koostuu joukosta hash-algoritmeja. Algoritmit jaetaan neljään perheeseen, SHA-0, SHA-1, SHA-2 ja SHA-3. Tässä tutkimuksessa otetaan kuitenkin vain SHA-1 ja SHA-2 perheiden algoritmit mukaan tutkimukseen (Wikipedia 2017).

SHA-algoritmien toiminta on hyvin samankaltainen kuin MD5-algoritmin. Funktio ottaa sisäänsä dataa, jonka algoritmi muuntaa tiivisteeseen, jonka koko riippuu käytettävän

algoritmin versiosta. SHA-1-algoritmin tiiviste on 160 b:n pituinen, kun taas SHA-2-algoritmin tiiviste voi olla 224, 256, 384 tai 512 b:n pituinen (Wikipedia 2017). Kuvassa 2 SHA algoritmien eroja.

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds
MD5 (as reference)		128	128 (4 × 32)	512	Unlimited[3]	64
SHA-0		160	160 (5 × 32)	512	$2^{64} - 1$	80
SHA-1		160	160 (5 × 32)	512	$2^{64} - 1$	80
SHA-2	*SHA-224* *SHA-256*	224 256	256 (8 × 32)	512	$2^{64} - 1$	64
	SHA-384 *SHA-512* *SHA-512/224* *SHA-512/256*	384 512 224 256	512 (8 × 64)	1024	$2^{128} - 1$	80
SHA-3	*SHA3-224* *SHA3-256* *SHA3-384* *SHA3-512*	224 256 384 512	1600 (5 × 5 × 64)	1152 1088 832 576	Unlimited[5]	24[6]
	SHAKE128 *SHAKE256*	*d* (arbitrary) *d* (arbitrary)		1344 1088		

Kuva 2. SHA-perheet (Wikipedia, 2017).

Kaikki SHA-algoritmit käyttävät one-way-compression-funktiota data muunnoksessa. Compression funktio yhdistää iteroimalla kaksi datalohkoa yhdeksi. Muuntokierroksia tehdään 64 tai 80 riippuen käytetystä algoritmin versiosta.

SHA-algoritmeja voidaan käyttää myös muuallakin kuin kryptografiassa. Kuten MD5, myös SHA on hyvä algoritmi tarkistussumman tekemiseen. Tarkistussummaa voidaan käyttää esimerkiksi etsimään korrupoitunutta dataa. Kuten MD5, myös SHA-algoritmeilla on ominaista, että mikä tahansa muutos alkuperäisessä datassa antaa täysin erilaisen syötteen. Jos alkuperäinen data on muuttunut, on myös siitä laskettava tarkistussumma eri. Jos siis alkuperäisen tiedoston tarkistussumma on muu kuin lopullisen, on tiedosto muuttunut jollakin tapaa. Esimerkiksi Git käyttää SHA-1 algoritmia tähän tarkoitukseen (Wikipedia 2017).

2.2 Salaus

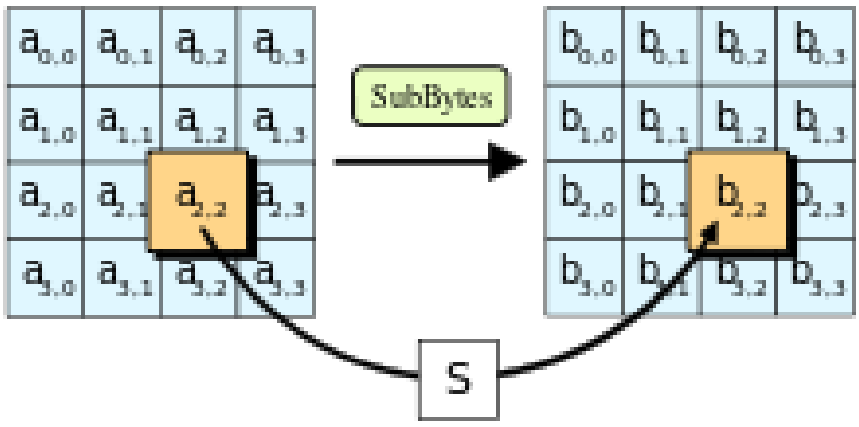
Salauksella tarkoitetaan datan muuttamista tulkitsemattomaan muotoon. Erona autentikointi algoritmeihin on kuitenkin se, että salatun muodon voi kääntää takaisin alkuperäiseen. Tähän käytetään niin sanottua avainta, jonka sekä salaaja, että purkaja tietävät. (bu, 2017.)

2.2.1 Salausalgoritmit

Teleste pyysi, että mittauksiin sisällytetään ainakin AES-, DES-, TDES-, ja XXTEA-algoritmit. Tämän lisäksi mukaan otettiin kaikki AES-algoritmin eri versiot, jotta saataisiin monipuolisempi tutkimus.

AES

AES (Advanced Encryption Standard) on nykymaailman eniten käytetty lohkosalaus algoritmi, jolla voidaan salata elektronista dataa. AES on seuraaja DES algoritmille, sillä se on nopeampi ja vaikeampi murtaa. Se käyttää 16 tavun data lohkoja ja sillä on kolme eri kokoista avainta 128, 192 ja 256-bittiä. Avainta käytetään itse salaus prosessissa ja vain sillä voi salatun datan kääntää takaisin alkuperäiseen muotoonsa. AES muodostaa datalohkoista ja avaimesta matriisit, jonka jälkeen se tekee neljä erilaista muutosprosessia matriiseihin. Alla oleva kuva 2 on yksi operaatio neljästä. Avaimen koko myös määrää montako kierrosta salausta tehdään. 128-bitin avain on 10 kierrosta, 192 on 12 ja 256 14 kierrosta (Searchsecurity, 2017).



Kuva 3. Yksi salaus operaatio neljästä (Wikipedia, 2017).

DES

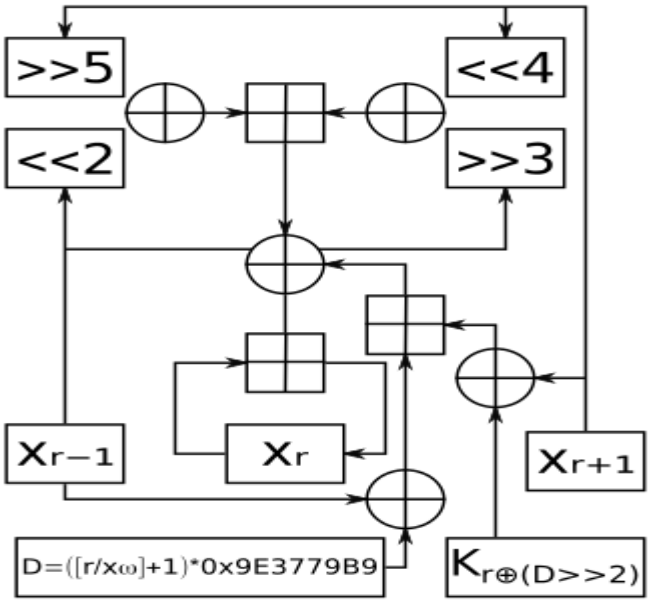
DES (Data encrypt standard) on 1970-luvulla kehitetty lohkosalausalgoritmi. Nykypäivänä sen ajatellaan olevan jokseenkin turvaton, sillä siihen on kehitetty menetelmiä, jolla se voidaan murtaa jo muutamissa tunneissa. DES toimii siten, että se ottaa 64-bitin pituisen lohkon dataa, puolittaa sen ja muuntaa sen avaimen kanssa monimutkaisten operaatioiden avulla toiseen yhtä pitkään lohkoon. Avain on 64 b:n pituinen, ja sen tietävät sekä salaaja, että purkaja. Lohkon purkaminen alkuperäiseen muotoon tapahtuu käänteisellä prosessilla. (Tutorialspoint, 2017.)

TDES

TDES on seuraaja DES algoritmille. Se kehitettiin alun perin siksi, että DESin ajateltiin olevan turvaton nykyajan laitteille. Toimintaperiaatteeltaan se on kuitenkin hyvin samanlainen, mutta avaimen koko on 192-bittiä. Salausoperaatiossa avain on jaettu 3 osaan ja salausoperaatioita suoritettaan 3 peräkkäin jokaisen eri avaimen palan kanssa. TDES on vielä tänä päivänä käytössä joissakin laitteissa, mutta sen hitaan suorituskyvyn takia AES-algoritmi tulee luultavasti korvaamaan sen kokonaan. (Tutorialspoint, 2017.)

XXTEA

XXTEA suunniteltiin alun perin korvaamaan edeltäjänsä BlockTEA-algoritmin heikkoudet. Se käyttää salauksessaan bittioperaatioita, joissa satunnaisten bittien paikkaa vaihdetaan. XXTEA on lohkosalaus algoritmi, jonka lohkot ovat 32 b:n kokoisia. Se vaatii kuitenkin toimiakseen vähintään kaksi lohkoa, eli yhteensä 64 bittiä. syötteen pituus määrää montako salaus kierrosta tehdään. Minimi on 6 kierrosta ja maksimi 32 (Wikipedia, 2017). Tästä johtuen salaus on parempi silloin, kun syötteen pituus on suurempi. Kuva 4 on esimerkki salauksessa käytettävistä bittioperaatioista.



Kuva 4. XXTEA-algoritmin bittioperaatiot (Wikipedia, 2017).

XXTEAN vahvuus on itse algoritmin pieni koko. Se mahdollistaa sen, että RAM muistia ei kulu paljoa prosessoidessa algoritmia. Sulautetussa ympäristössä resurssit ovat usein tärkeä valintakriteeri, ja siksi kyseinen algoritmi otettiinkin tähän tutkimukseen mukaan.

2.3 IDE

Alusta, jonka päällä ohjelmointi toteutettiin, oli valittava monista vaihtoehdoista. Parhaiksi vaihtoehdoiksi valikoitui AtollicTruestudio, Mkeil sekä Xcube, joista lopuksi kokeilun jälkeen atollic osoittautui ominaisuuksiltaan parhaaksi. Alustassa oli kattava tuki eri mikropiireille, sekä hyvät työkalut tutkimaan itse ohjelman suorittamista. Osa työkaluista oli kuitenkin maksullisen version takana, mutta Atollic tarjosi viikon ilmaisen kokeilun maksullisiin ominaisuuksiin, jos oli tarvetta.

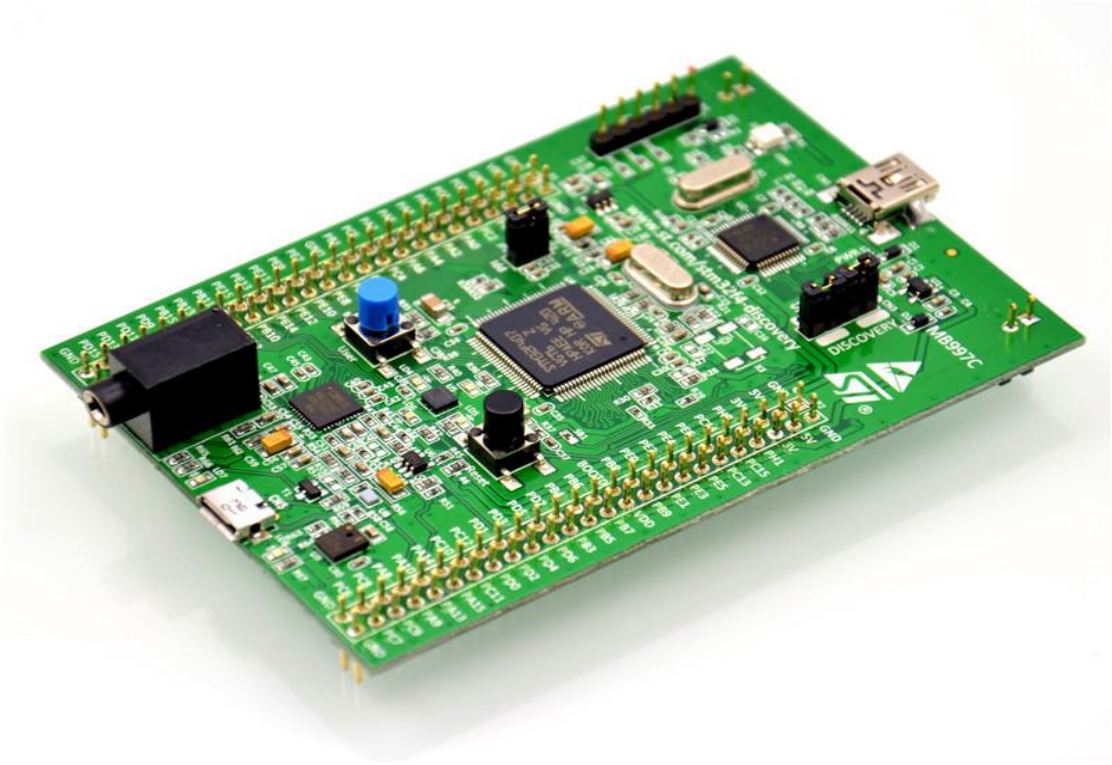
2.4 Kirjastot

Algoritmit sisältävien kirjastojen valinta oli oma prosessinsa. Suurin osa työssä käytetyistä algoritmeista löytyy STmicroelectronicsin tarjoamasta krypto-kirjastosta,

mutta esimerkiksi XXTEA-algoritmi on erillisestä opensource-kirjastosta. Kirjastojen käyttämisen oppiminen vaati myös oman aikansa, jotta itse algoritmeja osasi käyttää oikein. Kirjastojen mukana tuli kuitenkin esimerkkejä, joissa algoritmifunktioita opastettiin käyttämään.

2.5 Prosessorit

Testikäytössä oli kaksi ST-microelectronicsin valmistamaa mikropiiriä STM32F407 ja STM32F100. Piireissä olevia prosessoreja käytetään Telesten laitteissa, joten ne olivat ideaalinen testiympäristö. Molemmat prosessorit ovat suhteellisen pieniä resursseiltaan. STM32F407-levyn prosessorin kellotaajuus on 168 Mhz, ja siinä on1024 kt flash-muistia, sekä RAM-muisti klmessa palassa, 128kt, 64kt ja 16kt. STM32F100-levyn prosessori on vain 24 Mhz. Flash-muistia siinä on 128 kt ja RAM-muistia 16kt. (ST-microelectronics, 2017.) Molempien mikropiirien kohdalla joutui aluksi tekemään pohjatyötä, sillä piireistä tuli valita mitä ominaisuuksia käytetään ja miten. Alapuolella oleva kuva 5, on työssä käytetty STM32F407-levy.



Kuva 5. Kuva työssä käytetystä STM32F407-levystä.

3 KÄYTÄNNON TOTEUTUS

3.1 Työkalujen käyttö

Työkalujen käyttäminen halutulla tavalla tuotti ongelmia. Tämä johtui pääasiassa siitä, että työhön valitut ohjelmat ja kirjastot eivät toimineet täydellisesti. AtollicTrueStudio on vielä verrattain uusi sulautettujen alusta, joten siitä löytyi yllättävän paljon uusia ja jo tiedossa olevia virheitä. Myös ongelmat kirjastojen kanssa tuottivat ylimääräistä vaivaa mutta mitään ylipääsemätöntä ei ilmennyt.

3.1.1 Alusta ja kirjastot

AtollicTrueStudio valikoitui alustaksi lähinnä siksi, että siinä oli kattava tuki työssä käytettäville mikropiireille. Alustalla oli paljon esimerkkejä, miten ohjelmoida ja ajaa koodia prosessoreilla. Mittausten aloittamista hidastivat kuitenkin alustan omat ohjelmointi virheet.

Ensimmäinen ongelma ilmeni jo esivalmisteluissa ennen itse ohjelmointia. Osa kirjastoista, joita käytettiin, toimitettiin binaari-tiedostoina. Ensimmäinen vaihe oli siis kääntää ne. Kirjastojen linkittäminen kääntäjälle automaattisesti tuotti ongelmia, jotka saatiin korjattua linkittämällä kaikki manuaalisesti. Itse kääntäminen ei kuitenkaan aluksi onnistunut, sillä kääntäjä oli eri, kuin se millä binaari-tiedostot oli alun perin käännetty. Ongelma ratkesi, kun kirjastot vaihdettiin saman kääntäjän kääntämiin tiedostoihin.

Suurin ongelma kirjastojen kanssa tuli esiin, kun eri algoritmeja testattiin. Piirin valmistajan toimittamassa krypto-kirjastossa oli virhe, joka aiheutti ongelmia tiettyjen prosessorisarjojen kanssa. Kirjasto oli vielä niin uusi, että ongelmasta ei löytynyt juurikaan tietoa. Lopuksi ongelma ratkesi, kun nettifoorumilta community.st.com löytyi samanlainen tapaus ja siihen ratkaisu. Ongelma saatiin korjattua mutta se viivästytti projektia noin viikon verran.

3.1.2 Virheenkorjaus

Virheenkorjaus, eli yleisemmin tunnettuna debuggaus, muodostui ongelmaksi. Alkuperäisenä suunnitelmana oli käyttää atollicin omia työkaluja, jotka näyttivät tarvittavat asiat helposti monitoroituna. Suurin osa halutuista työkaluista oli kuitenkin vain maksullisessa versiossa, joten koodin tutkimiseen joutui käyttämään alustan omaa debuggeria. Debuggerissa oli kuitenkin ohjelmointivirhe, joka aiheutti ongelmia ohjelman pysäytyspisteiden käytön kanssa. Ohjelman pysäyttäminen ja analysoiminen, oli kuitenkin pakollista tuloksien saamiseksi, joten ongelman kanssa oli pakko tulla toimeen. Tämä kuitenkin aiheutti ylimääräistä työtä, sillä debuggerin ongelma aiheutti projektin uudelleen luomista muutamaan otteeseen.

3.2 Ohjelmointi

Ohjelmoinnin kannalta työ ei ollut kovinkaan hankala. Kun ensimmäisen algoritmin sai toimimaan halutulla tavalla, oli muut helppo toteuttaen samalla tavalla. Perustuntemus C-kielen ohjelmoinnista riitti koodaukseen hyvin. Suurin työ oli kuitenkin toteuttaa kaikki mittaukset, sillä algoritmeja oli monta, sekä jokainen algoritmi piti testata viidellä erikokoisella datapaketilla. Tämän lisäksi kaikki mittaukset tuli tehdä molemmilla prosessoreilla.

Yksi ongelma oli suoritettujen mittausten tulosten saaminen piiriltä. Reaaliaikainen ohjelman tutkiminen ei ollut helppoa Atollicin Ilmaisella versiolla, sillä kaikki siihen tehdyt työkalut olivat maksullisen lisenssin takan. Yksi vaihtoehto olisi ollut oman ohjelman tekeminen, joka olisi lähettänyt USB-kaapelin kautta tarvittavat tiedot. Se olisi ollut kuitenkin turhan haasteellinen toteuttaa. Lopulta päädyttiin toteuttamaan mittaukset kääntäjän oman debuggerin avulla, sekä flash ja RAM-muistin saamiseksi käytettiin Atollicin maksullisten ominaisuuksien viikon mittaista ilmaista kokeilujaksoa.

3.2.1 Testiohjelma

Kaikkien algoritmien mittaamisessa käytettiin hyvin toistensa kaltaista ratkaisua. Ohjelma ottaa sisäänsä annetun kokoisen datapaketin ja prosessoi sen halutulla

algoritmilla. Tämän jälkeen se syöttää algoritmilla prosessoidun datan ulos. Ainoa asia, jota piti vaihtaa, oli sisään menevän datapaketin koko. Itse ohjelman rakennetta ei muuttanut juuri lainkaan se kumpi prosessori oli kyseessä. Kuvassa 6 on toteutus MD5-algoritmin suorittamisesta.

```
180 int MD5(){
181
182 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_CRC, ENABLE);
183 uint8_t inputmessage[1536];
184 int32_t outSize = 0, returnvalue = 0;
185
186 uint8_t digest[16];
187 MD5ctx_stt MD5ctx_st;
188 MD5ctx_st.mFlags = E_HASH_DEFAULT;
189 MD5ctx_st.mTagSize = 16;
190
191 returnvalue = MD5_Init(&MD5ctx_st);
192 if (returnvalue == HASH_SUCCESS)
193 {
194     returnvalue = MD5_Append(&MD5ctx_st, inputmessage,sizeof(inputmessage));
195     if (returnvalue == HASH_SUCCESS)
196     {
197         returnvalue = MD5_Finish(&MD5ctx_st, digest, &outSize );
198         if (returnvalue == HASH_SUCCESS)
199         {
200             printf("hash success");
201         }
202         else
203         {
204             printf("error");
205         }
206     }
207     else
208     {
209          printf("error");
210     }
211 }
212 else
213 {
214     printf("error");
215 }
216 return 0;
217 }
```

Kuva 6. Esimerkki tyypillisestä toteutuksesta.

3.2.2 Suoritusaika

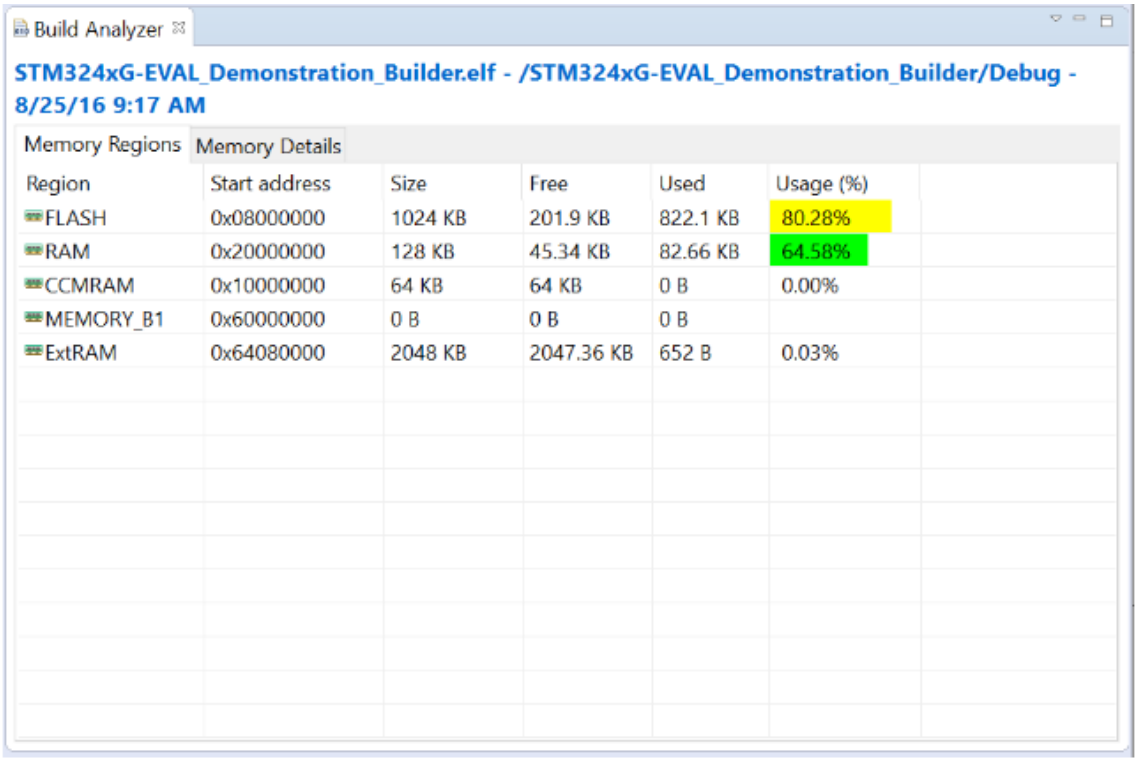
Algoritmien suoritusaikojen mittaamiseen käytettiin ARM-prosessorin omaa sykli-laskuria. Laskuri laitettiin alkamaan juuri ennen funktiota joka prosessoi datan ja lopetettiin välittömästi tämän jälkeen. Tällä tapaa kerättiin kaikkien algoritmien suoritusajat molemmilta prosessoreilta sekä kaiken kokoisilla datapaketeilla mitattuna.

Tuloksista muodostettiin taulukko, johon listattiin sekä prosessorin syklit ja niistä muunnetut millisekunnit. Muunnos oli helppo, sillä ytimen kellotaajuus oli tiedossa molemmista prosessoreista. Mittauksissa tuli myös ottaa huomioon, että ohjelmassa ei ollut päällä mitään mahdollisia keskeytyksiä, jotka voisivat vääristää suoritusaikoja.

Tulokset varmistettiin vielä siten, että suoritusfunktiota kerrottiin tarpeeksi isolla luvulla, jotta suoritusajaksi muodostui sekunteja. Suoritusaika otettiin ylös sekuntikellolla ja verrattiin sitä sitten syklien määrään. Tämä mahdollisti sen, että pystyttiin helposti laskemaan, täsmäsivätkö saadut syklit prosessorin kellotaajuuteen. Tämä oli merkittävää tulosten kannalta, sillä molempien prosessorien tuli käydä maksimiteholla, jotta tulokset vastaisivat Telesten laitteiden prosessoreita.

3.2.3 Flash-muisti

AtollicTrueStudio tarjosi Build Analyzer-nimisen työkalun, jolla sai käytetyn flash-muistin määrän selville. Se kuitenkin vaati maksullisen version, joten Atollicin tarjoama viikon ilmainen kokeilu otettiin käyttöön. Kuvassa 7, on Build Analyzerin näkymä.

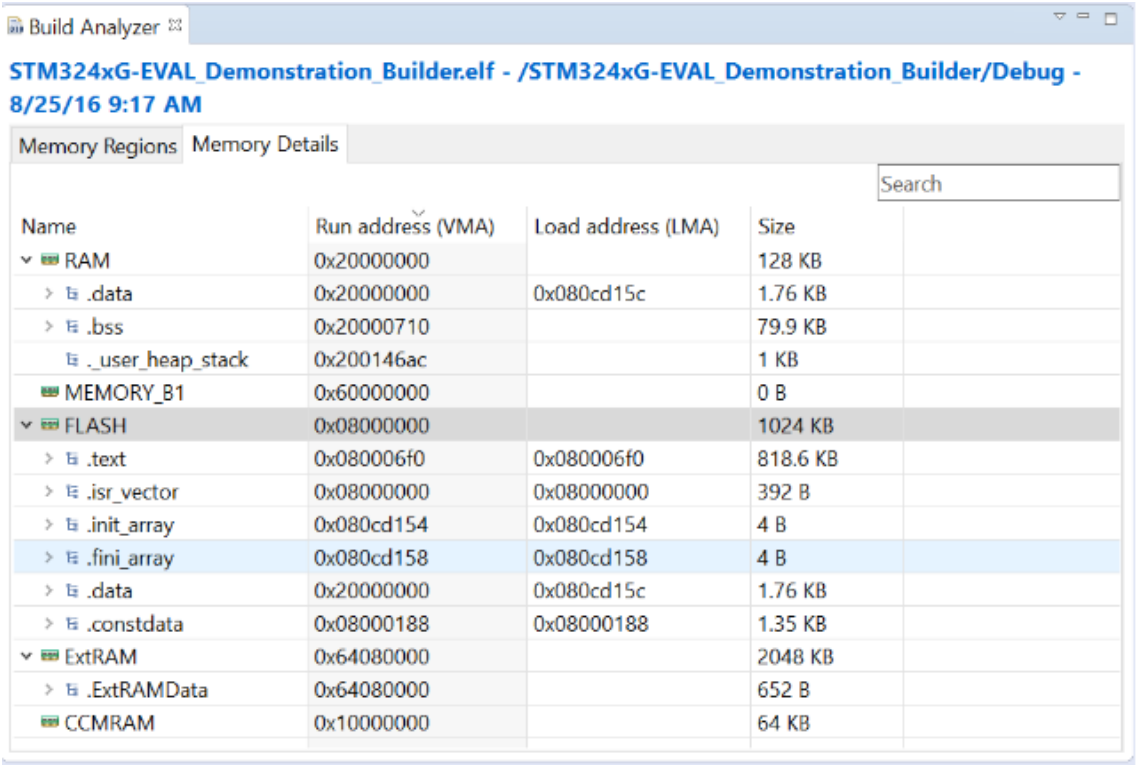


Kuva 7. Esimerkki muistin käytöstä Build Analyzerin käyttöliittymästä.

Käytetty flash-muisti vaihtelee sen mukaan, miten iso itse ohjelma on ja millaista optimointia kääntäjä käyttää. Usein sulautetussa ympäristössä käytetään kääntäjässä optimointia, sillä se poistaa kaiken turhan koodista. Telesten kanssa käytyjen keskustelujen jälkeen tultiin kuitenkin siihen tulokseen, että optimointia ei tarvittu.

3.2.4 RAM-muisti ja pinon käyttö

RAM-muistin käyttö oli vaikeampi saada selville, sillä pino sijaitsee siinä. Pinon kokoon vaikuttaa kuitenkin ohjelman hetkellinen suoritus, joten pysyvä RAM ja pino tuli mitata erikseen. RAM-muistin käytön sai selville samaisella työkalulla kuin flashinkin, Katso kuva 8.



Name	Run address (VMA)	Load address (LMA)	Size
RAM	0x20000000		128 KB
.data	0x20000000	0x080cd15c	1.76 KB
.bss	0x20000710		79.9 KB
._user_heap_stack	0x200146ac		1 KB
MEMORY_B1	0x60000000		0 B
FLASH	0x08000000		1024 KB
.text	0x080006f0	0x080006f0	818.6 KB
.isr_vector	0x08000000	0x08000000	392 B
.init_array	0x080cd154	0x080cd154	4 B
.fini_array	0x080cd158	0x080cd158	4 B
.data	0x20000000	0x080cd15c	1.76 KB
.constdata	0x08000188	0x08000188	1.35 KB
ExtRAM	0x64080000		2048 KB
.ExtRAMData	0x64080000		652 B
CCMRAM	0x10000000		64 KB

Kuva 8. Muistien yksityiskohtainen näkymä Build Analyzerissa.

Työkalulla pystyi mittaamaan myös pinon käytön, mutta pino mitattiin myös perinteisin keinoin. Prosessorin muistirekisteristä selvisi missä pino sijaitsee. Mittaukset

tapahtuivat siten, että pinon arvo otettiin ylös ennen haluttua kohtaa ja jälkeen, jolloin erotuksena saatiin pinon tarkka arvo.

4 TULOKSET

Tutkimuksen tavoitteena oli löytää parhaat algoritmit sekä autentikointiin että salaukseen. Valintaprosessissa tuli ottaa huomioon algoritmien suoritusarvot sekä niiden salauksien vahvuus. Kaikista tutkimuksessa saaduista suoritusarvoista muodostettiin taulukko, johon kerättiin kaikkien algoritmien prosessointiaika sekä prosessorin sykleinä, että millisekunteiksi muunnettuna. Tämän lisäksi ylös otettiin kunkin algoritmin viemä flash ja RAM-muisti sekä pinon koko pahimpina ruuhka-aikoina. Kaikki mittaukset toteutettiin molempien mikropiirien kanssa, sekä kaikki tulokset mitattiin kaikilla viidellä eri datapakettikoolla.

4.1.1 STM32F407

Taulukossa 1, 2 ja 3 on ST32F407-piirillä mitatut prosessointiajat. Jokainen algoritmi on mitattu jokaisella eri pakettikokoluokalla, sekä salauksessa on erikseen mitattu salaus ja purkaminen. Salausta on merkitty kirjaimella E, ja purkua kirjaimella D.

Taulukko 1. Prosessointiaika sykleinä

Packet(Bytes)	8	64	256	512	1536
MD5	3808	5117	9094	14502	35878
SHA-1	4975	7983	15480	25669	66133
SHA_256	19512	23669	34552	49205	107573
SHA_512	29579	29747	82226	134908	345636
AES_128_CBC_E	3856	10108	35680	69473	205473
AES_128_CBC_D	6672	13158	38612	73053	209708
AES_192_CBC_E	4244	11611	41352	80786	238962
AES_192_CBC_D	7426	14909	44738	84383	245308
AES_256_CBC_E	5005	13535	48789	93444	274335
AES_256_CBC_D	8841	17315	51442	96839	278999
AES_128_ECB_E	3766	9675	35011	69454	202567
AES_128_ECB_D	6202	12514	37623	71232	206198
DES_CBC_E	26595	38216	80496	136424	354815
DES_CBC_D	26543	38778	79744	134542	356755
DES_ECB_E	26292	38719	79011	133440	354286
DES_ECB_D	26346	38216	79150	133622	354338
TDES_CBC_E	77728	112037	233107	394123	1018988
TDES_CBC_D	77701	111905	232623	393030	1015374
XXTEA_E	4419	8854	22194	41972	120820
XXTEA_D	4156	8446	21600	40819	117619

Taulukko 2. Prosessointiaika millisekunteina.

Packet(Bytes)	8	64	256	512	1536
MD5	0,022666667	0,030458333	0,054130952	0,086321429	0,213559524
SHA-1	0,029613095	0,047517857	0,092142857	0,152791667	0,39364881
SHA_256	0,116142857	0,140886905	0,205666667	0,292886905	0,640315476
SHA_512	0,176065476	0,177065476	0,489440476	0,80302381	2,057357143
AES_128_CBC_E	0,022952381	0,060166667	0,212380952	0,413529762	1,223053571
AES_128_CBC_D	0,039714286	0,078321429	0,229833333	0,434839286	1,248261905
AES_192_CBC_E	0,025261905	0,069113095	0,246142857	0,480869048	1,422392857
AES_192_CBC_D	0,044202381	0,088744048	0,266297619	0,502279762	1,460166667
AES_256_CBC_E	0,029791667	0,080565476	0,290410714	0,556214286	1,632946429
AES_256_CBC_D	0,052625	0,103065476	0,306202381	0,576422619	1,660708333
AES_128_ECB_E	0,022416667	0,057589286	0,20839881	0,413416667	1,205755952
AES_128_ECB_D	0,036916667	0,074488095	0,223946429	0,424	1,227369048
DES_CBC_E	0,158303571	0,22747619	0,479142857	0,812047619	2,111994048
DES_CBC_D	0,157994048	0,230821429	0,474666667	0,800845238	2,123541667
DES_ECB_E	0,1565	0,230470238	0,470303571	0,794285714	2,108845238
DES_ECB_D	0,156821429	0,22747619	0,471130952	0,795369048	2,109154762
TDES_CBC_E	0,462666667	0,666886905	1,387541667	2,345970238	6,065404762
TDES_CBC_D	0,462505952	0,66610119	1,384660714	2,339464286	6,043892857
XXTEA_E	0,026303571	0,052702381	0,132107143	0,249833333	0,719166667
XXTEA_D	0,024738095	0,05027381	0,128571429	0,242970238	0,700113095

Taulukko 3. STM32F407-piirin flash, RAM ja pinon käyttö.

Size (bytes)			
	Flash	Ram	Stack (datapacket 16-1536 bytes)
MD5	6,21KB	1,17KB	0,168-1,688 KB
SHA_128	5,31KB	1,17KB	0,168-1,688 KB
SHA_256	5,83KB	1,17KB	0,176-1,696 KB
SHA_512	6,76KB	1,17KB	0,312-1,832 KB
AES_128_CBC_E	7,15KB	1,17KB	0,392-4,952 KB
AES_128_CBC_D	8,15KB	1,17KB	0,392-4,952 KB
AES_192_CBC_E	7,18KB	1,17KB	0,400-4,960 KB
AES_192_CBC_D	8,19KB	1,17KB	0,400-4,960 KB
AES_256_CBC_E	7,19KB	1,17KB	0,408-4,968 KB
AES_256_CBC_D	8,19KB	1,17KB	0,408-4,968 KB
AES_128_ECB_E	6,43KB	1,17KB	0,392-4,952 KB
AES_128_ECB_D	7,02KB	1,17KB	0,392-4,952 KB
DES_CBC_E	7,53KB	1,17KB	0,248-4,808 KB
DES_CBC_D	7,64KB	1,17KB	0,248-4,808 KB
DES_ECB_E	7,43KB	1,17KB	0,248-4,808 KB
DES_ECB_D	7,77KB	1,17KB	0,248-4,808 KB
TDES_CBC_E	7,62KB	1,17KB	0,528-5,088 KB
TDES_CBC_D	7,70KB	1,17KB	0,528-5,088 KB
XXTEA_E	6,32KB	2,48KB	2,064-3,584 KB
XXTEA_D	6,57KB	2,48KB	2,064-3,584 KB

4.1.2 STM32F100

Taulukoissa 4, 5 ja 6 ovat STM32F100-piirillä mitatut suoritusarvot.

Taulukko 4. Suoritusajat sykleinä.

STM32100B cycles					
Packet(Bytes)	8	64	256	512	1536
MD5	2671	3444	5845	9038	21822
SHA_128	3962	6448	12762	21175	54839
SHA_256	16678	20305	30112	42992	94768
SHA_512	24313	24445	67497	110701	283517
AES_128_CBC_E	2646	6108	19943	38391	112206
AES_128_CBC_D	3845	7199	20629	38533	110172
AES_192_CBC_E	2878	6950	23010	44482	130393
AES_192_CBC_D	4333	8237	23859	44691	128042
AES_256_CBC_E	3232	7826	26199	50695	148699
AES_256_CBC_D	4940	9396	27214	50974	146038
DES_CBC_E	21883	29331	54868	88916	225128
DES_CBC_D	21867	29231	54482	88146	222823
TDES_CBC_E	64489	85891	159288	257144	648588
TDES_CBC_D	64473	85778	158809	256185	645714
XXTEA_E	3843	8117	21066	40292	117092
XXTEA_D	3705	7946	21013	40182	96714

Taulukko 5. Suoritusajat millisekunteina.

STM32100B cycles	Time(ms)				
Packet(Bytes)	8	64	256	512	1536
MD5	0,111291667	0,1435	0,243541667	0,376583333	0,90925
SHA_128	0,165083333	0,268666667	0,53175	0,882291667	2,284958333
SHA_256	0,694916667	0,846041667	1,254666667	1,791333333	3,948666667
SHA_512	1,013041667	1,018541667	2,812375	4,612541667	11,81320833
AES_128_CBC_E	0,11025	0,2545	0,830958333	1,599625	4,67525
AES_128_CBC_D	0,160208333	0,299958333	0,859541667	1,605541667	4,5905
AES_192_CBC_E	0,119916667	0,289583333	0,95875	1,853416667	5,433041667
AES_192_CBC_D	0,180541667	0,343208333	0,994125	1,862125	5,335083333
AES_256_CBC_E	0,134666667	0,326083333	1,091625	2,112291667	6,195791667
AES_256_CBC_D	0,205833333	0,3915	1,133916667	2,123916667	6,084916667
DES_CBC_E	0,911791667	1,222125	2,286166667	3,704833333	9,380333333
DES_CBC_D	0,911125	1,217958333	2,270083333	3,67275	9,284291667
TDES_CBC_E	2,687041667	3,578791667	6,637	10,71433333	27,0245
TDES_CBC_D	2,686375	3,574083333	6,617041667	10,674375	26,90475
XXTEA_E	0,160125	0,338208333	0,87775	1,678833333	4,878833333
XXTEA_D	0,154375	0,331083333	0,875541667	1,67425	4,02975

Taulukko 6. Muistin käyttö STM32F100-piirillä

STM32100B			
Size (bytes)			
	Flash	Ram	Stack (datapacket 16-1536 bytes)
MD5	5,21KB	0,16KB	0,168-1,688 KB
SHA_128	4,41KB	0,16KB	0,168-1,688 KB
SHA_256	4,88KB	0,16KB	0,176-1,696 KB
SHA_512	5,66KB	0,16KB	0,312-1,832 KB
AES_128_CBC_E	6,34KB	0,16KB	0,392-4,952 KB
AES_128_CBC_D	7,42KB	0,16KB	0,392-4,952 KB
AES_192_CBC_E	6,39KB	0,16KB	0,400-4,960 KB
AES_192_CBC_D	7,43KB	0,16KB	0,400-4,960 KB
AES_256_CBC_E	6,39KB	0,16KB	0,408-4,968 KB
AES_256_CBC_D	7,43KB	0,16KB	0,408-4,968 KB
DES_CBC_E	6,71KB	0,16KB	0,248-4,808 KB
DES_CBC_D	6,73KB	0,16KB	0,248-4,808 KB
TDES_CBC_E	6,81KB	0,16KB	0,528-5,088 KB
TDES_CBC_D	6,80KB	0,16KB	0,528-5,088 KB
XXTEA_E	5,40KB	1,46KB	2,064-3,584 KB
XXTEA_D	5,64KB	1,46KB	2,064-3,584 KB

4.1.3 Johtopäätökset mittaustuloksista

Alusta asti oli selvää, että STM32F407-piirillä mitatut suoritusarvot olisivat nopeammat mitä STM32F100-piirillä saatavat tulokset. F4-levyllä käytetty prosessori oli 168 MHz, eli noin 6 kerta nopeampi kuin F100-piirin prosessorin kellotaajuus 24 MHz. Tämän takia suoritusajat ovat karkeasti 6 kertaa pienemmät.

Taulukoista voidaan lukea, että kaikki suoritusajat pysyvät alle 30-millisekunnin, riippumatta algoritmista tai pakettikoosta. Se onko 30-millisekuntia liian suuri aika, on mahdotonta sanoa tuntematta itse käyttökohdetta. Tutkimusta lähdettiin tekemään sillä olettamuksella, että itse mittauksissa saatujen tuloksien kelvollisuutta ei tarvitse määrittää, vaan Teleste hoitaa sen osuuden.

Tuloksista voidaan kuitenkin eritellä hitaammat ja nopeammat algoritmit. Esimerkiksi autentikoinnin osalta voidaan todeta, että MD5-algoritmi on aavistuksen nopeampi kuin SHA-1. SHA-1 on kuitenkin salaukseltaan vahvempi kuin MD5, joten algoritmin käyttötarkoitus tulee ottaa huomioon tehdessä valintoja.

Mittauksissa on mukana myös SHA-algoritmin versiot 256, ja 512 joiden salaus on huomattavasti vahvempi. Algoritmien prosessointiajat nousevat kuitenkin merkittävästi, joten tämä tulee huomioida, jos käyttökohteena on heikkotehoinen prosessori.

Salausalgoritmien osalta oli jokseenkin selvää, että esimerkiksi DES on muita selvästi hitaampi. Tämä johtuu siitä, että kyseinen algoritmi on kehitetty muita paljon aikaisemmin. DES on myös kohtalaisen helposti murrettavissa nykypäivänä, joten mittauksiin sisällytettiin myös DESin uudempi versio TDES. Mittaustuloksista voidaan kuitenkin todeta, että TDESin prosessointiaika on algoritmin vanhasta teknologiasta johtuen hidas.

Tuloksista voidaan myös päätellä, että AES ja XXTEA ovat selvästi parhaat salaus algoritmit suoritusarvoiltaan. AESin etuna on kuitenkin se, että se tarjoaa useamman eri version sen mukaan, miten vahva salaus tarvitaan. Merkittävää tuloksissa on se, että prosessointiaika ei kasva juurikaan salausta vahvennettaessa.

Flash ja RAM-muistin käyttö kerättiin omaksi taulukoksi. RAM-muistissa sijaitsevan pinon käyttö on ilmoitettu pienimmän ja isoimman paketin raja-arvoilla, sillä pinon koko kasvoi lineaarisesti pakettikokoa kasvatettaessa. Kuten suoritusaikojenkin tulkitseminen, myös flash- ja RAM-muistin sopivan käytön arvioiminen on mahdotonta ilman, että tuntee kohdelaitteen muun ohjelmiston resurssivaatimuksia. Sen verran tuloksista pystyy kuitenkin sanomaan, että mitatut arvot näyttivät vievän vain muutamia prosenttiyksikköjä flash- ja RAM-muistista, joten luultavasti liikutaan hyväksyttävissä rajoissa.

5 LOPUKSI

Tässä opinnäytetyössä tutkittiin autentikointi- ja salausalgoritmien suoritusarvoja. Tavoitteena oli löytää suoritusarvoiltaan sopivat algoritmit Telesten laitteille. Testauksessa käytettiin alustana kahta mikropiiriä STM32F407 ja STM32F100, joiden prosessoreja Teleste käyttää laitteissaan.

Opinnäytetyön tuloksena syntyi taulukko, joka sisältää autentikointi- ja salausalgoritmien suoritusaikoja sekä flash- ja RAM-muistin käytön. Mittaustuloksista voidaan muu muossa todeta, kuinka raskas mikäkin algoritmi on. Parhaaksi algoritmiksi autentikoinnin osalta valikoitui MD5, joka on lähes puolet nopeampi kuin salaukseltaan vastaava SHA-1. Salausalgoritmeista selvästi tehokkaimmaksi osoittautui AES, jonka 128-bittinen versio oli merkittävästi nopeampi kuin salaukseltaan vastaavat XXTEA tai DTES. Taulukosta nähdään myös se, paljonko muistiresursseja vaaditaan. Mittauksia voidaan käyttää apuna valittaessa sopivaa algoritmia salaukseen tai autentikointiin, jos algoritmien suoritusarvot ovat merkityksellisiä prosessoreiden kannalta.

Tutkimuksesta olisi voinut tehdä paljon kattavamman lisäämällä siihen enemmän salauksessa ja autentikoinnissa käytettäviä algoritmeja. Tähän tutkimukseen päätettiin ottaa kuitenkin vain yleisimmät algoritmit mukaan, jotta mittaukset eivät olisi paisuneet liian suuriksi. Jos tulevaisuudessa kuitenkin on tarvetta testata muita algoritmeja, on ne helppo toteuttaa samalla tapaa kuin muutkin mittaukset.

LÄHTEET

Bu 2017. Security-resources. Auth. Viitattu 29.4.2017 https://www.bu.edu/tech/about/security-resources/bestpractice/auth/

Economictimes 2017. Definitation. Authentication. Viitattu 29.4.2017 http://economictimes.indiatimes.com/definition/authentication

ST-microelectronics 2017. Products. Viitattu 29.4.2017 http://www.st.com/content/st_com/en/search.html#q=STM32F4-t=keywords-page=1

searchsecurity 2017. Data Encyption Standards. Viitattu 29.4.2017 http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard

Tutorialspoint 2017. Hash functions. Viitattu 29.4.2017 https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm

Tutorialspoint 2017. Data Encryption Standard. Viitattu 29.4.2017 https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm

Wikipedia 2017. SHA-1. Data integrity. Viitattu 29.4.2017 https://en.wikipedia.org/wiki/SHA-1

Wikipedia 2017. AES. SubBytes. Viitattu 29.4.2017 https://en.wikipedia.org/wiki/Advanced_Encryption_Standard