CHALMERS

ECU-brandvägg

Demonstration av ECU-brandvägg i AUTOSAR-miljö

Examensarbete inom Data- och Informationsteknik

MICHAEL EDEVÅG
ROBERT SWIERCZYNSKI

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

EXAMENSARBETE

ECU-brandvägg

Demonstration av ECU-brandvägg i AUTOSAR-miljö

MICHAEL EDEVÅG
ROBERT SWIERCZYNSKI

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Göteborg 2016

ECU-brandvägg
Demonstration av ECU-brandvägg i AUTOSAR-miljö
MICHAEL EDEVÅG
ROBERT SWIERCZYNSKI

Examinator: Peter Lundin

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
412 96 Göteborg
Telefon: 031-772 1000

Institutionen för Data- och Informationsteknik
Göteborg 2016

FÖRORD

Examensarbetet är utfört på ArcCore AB i Göteborg i samarbete med Institutionen för Data- och Informationsteknik på Chalmers Tekniska Högskola. Examensarbetet omfattar 15hp och utfördes våren 2016 inom dataingenjörsprogrammet 180hp. Vi vill tacka Johan Ekberg för att vi fick möjlighet att utföra detta examensarbete och för den hjälp vi har fått under arbetets gång. Vi vill även tacka Lukas Suter för att alltid varit hjälpsam och tillgänglig för att hjälpa till så fort det uppstått något frågetecken. Vi vill också tacka hela ArcCore för att ha fått oss att känna oss välkomna. Utöver detta vill vi även tacka vår handledare på Chalmers, Sakib Sistek, som varit ett bra stöd under hela arbetet. Slutligen vill vi tacka examinatorn Peter Lundin som har försett oss med information om hur rapporten ska skrivas.

SAMMANFATTNING

I takt med att bilar blir mer och mer uppkopplade mot internet och användare tillåts att styra olika funktioner i bilen via sina telefoner blir säkerhetsaspekten viktig att tänka på. Utvecklingen av självkörande bilar har också bidragit till att mer vikt läggs på att säkerheten ska vara hög. AUTOSAR-standarden används idag av många biltillverkare för att definiera hur mjukvaran i bilens alla ECU-enheter ska vara skriven. AUTOSAR har dock inte någon klart definierad modul för en säkerhets-brandvägg. ArcCore AB arbetar med AUTOSAR-relaterade produkter och har skapat en implementering av AUTOSAR som heter Arctic Core.

På uppdrag av ArcCore undersöker den här rapporten möjligheten att implementera någon form av en brandvägg i Arctic Core. Någon ny modul har inte utvecklats, endast befintliga moduler har använts. Några starka krypterings-algoritmer har ej använts då rapportens fokus endast ligger på att visa att det är möjligt att kryptera. Resultatet av rapporten visar en metod för att implementera en brandvägg i AUTOSAR-miljö och erbjuder också ett enkelt demonstrationsprogram för att visa hur brandväggen både autentiserar användare och krypterar kommunikationen.

Nyckelord: Bil, Säkerhet, AUTOSAR, Kryptering, ECU, Autentisering, Brandvägg

ABSTRACT

Since recent development aim to make modern cars more and more connected to the Internet, the question about how secure the cars computer systems are also gets more important. The development of autonomous cars also contributes to making the security of the computer systems more important. The AUTOSAR standard is today used by many car manufacturers to define how to write and implement the software used in the cars many ECUs, but AUTOSAR lacks a definition of a module for a firewall/security gateway. ArcCore AB is a company working with AUTOSAR related products and it has its own implementation of AUTOSAR called Arctic Core.

On behalf of ArcCore this report examines the possibility to implement some sort of firewall in Arctic Core. A new module was not created, only existing modules were used and combined. Strong encryption formulas are not used since the focus of this report only is to demonstrate that encryption is possible. The result of this report shows that it is possible to implement a firewall in an AUTOSAR environment. The report also provides a simple demonstration program to demonstrate how the firewall both authenticates the users and encrypts the communications.

Keywords: Car, Security, AUTOSAR, Encryption, ECU, Authentication, Automotive firewall

Innehållsförteckning

Definitioner

Arctic EVK-M3 - *IC, beskriven i Kapitel 3, Teknisk bakgrund*
ARText - *Modelleringsspråket som används för att definiera SWC:arna*
ASIL - *Automotive Safety Integrity Level*
AUTOSAR - *AUTomotive Open System ARchitecture*
BSW - *Basic SoftWare*
CAN - *Controller Area Network*
ECU - *Electronic Control Unit*
GIT - *Versionshanteringsprogram*
IC - *Integrated Circuit*
JTAG - *Joint Test Action Group*
Komposition - *En SWC som fungerar som behållare för flera andra SWC's*
MCAL - *MicroController Abstraction Layer*
NHTSA - *National Highway Traffic Safety Administration*
Python - *Ett objektorienterat programmeringsspråk*
Runnable - *"Runnable entities are the smallest code-fragments that are provided by the component and are (at least indirectly) a subject for scheduling by the operating system." [1].*
ST-LINK/V2 - *Debugger för bland annat STM32 från STMicroelectronics*
STM32 - *32-bitars IC-krets baserad på ARM - arkitektur innehållandes processor, RAM, flash-minne, debugg-interface och ett antal ingångar såsom Ethernet och USB*
SOME/IP - *Scalable service-Oriented MiddlewarE on IP*
Spoofing - *En attack där en angripare utger sig för att vara någon annan än den är [2].*
SWC - *SoftWare Component*
WinPcap - *Windows Packet capture*
XOR - *eXclusive OR*

1 Inledning

ArcCore är ett företag som arbetar med AUTOSAR-produkter och finns i Sverige, Tyskland och Indien. Med start 2003 började biltillverkare, underleverantörer och utvecklare av styrenheter arbeta mot att få en gemensam, öppen och standardiserad mjukvarumodell för ECU:n i bilar [3]. Detta resulterade i AUTOSAR[1]. Det dröjde till slutet av AUTOSARs fas 1 år 2006 innan de första riktiga specifikationerna var klara [4]. Fler och fler företag anslöt sig och organisationen beslöt att dela dessa i olika nivåer. Det finns Core partners, Premium partners och Associate partners [4]. ArcCore är för närvarande Associate partner och har tre olika typer av produkter. Dessa är Arctic Studio, Arctic Core och Arctic Bootloader. De olika produkterna används för utveckling av komplett mjukvara för ECU:er.

1.1 Bakgrund

I takt med att bilar blir mer och mer uppkopplade och olika vitala funktioner blir mer och mer beroende av bilens olika datorsystem blir också säkerheten i dessa system en mer och mer viktig fråga. Många av dagens säkerhetstekniker fokuserar mer på att hindra ett intrång i systemet än att minimera skada då ett intrång redan inträffat [5]. Detta är dock också viktigt vilket påvisas av nedanstående citat.

> It's called operational security, and the auto industry -- even the banking industry -- has been slow to adopt it, according to Egil Juliussen, a senior analyst and research director for IHS Automotive. "They assume hackers can't get through their perimeter security, which is not true,"" Juliussen said. "That's a basic principle for security." [5].

Utvecklingen rör sig mer och mer mot ett behov av bättre säkerhet, “Miller said he can imagine a more secure method, such as using cryptography or encrypted messaging within a vehicle's CAN, to make it more difficult to hack.” [5].

Denna rapport ämnar utforska möjligheten att ha två nivåer av skydd mot intrång i bilens interna datorsystem. En första nivå är att autentisera en extern enhet, så som en mobiltelefon, så att en specifik användare endast har tillgång till vissa utvalda resurser på bilens interna nät. Nivå två avser att minimera skadan vid en eventuell attack där angriparen redan har tagit sig förbi första nivån. Detta sker genom att utöver autentiseringen även kryptera den data som skickas internt mellan bilens olika noder. Resultatet är att någon som tagit sig in i systemet även måste komma förbi krypteringen för att kunna avläsa och injicera datatrafik, något som annars är en sårbarhet som finns i många av dagens bilar [6]. Detta scenario, som beskrivs i Firewalls can't protect today's connected cars och här citeras,

> All modern vehicles have a CAN, which acts as a computer superhighway to the vehicle's various electronically controlled components. Once on the CAN, Miller and Valasek discovered which electronic messages controlled various systems, and they were able to send messages to remotely control the brakes, transmission, acceleration and other vital components. [5]

ska alltså ej längre vara en sårbarhet då CAN-trafiken är krypterad [5].

[1] Mer om AUTOSAR kan läsas i kapitel 2.2

1.2 Syfte

ArcCore vill lära sig mer om olika säkerhetstekniker för att kunna skydda privat data i fordonssystem från obehöriga användare på det publika nätet. ArcCore vill därför undersöka möjligheten att implementera en brandvägg som med hjälp av nycklar och certifikat hanterar autentiseringen av användare som begär ut information av systemet. Om möjlighet finns så vill man även kunna kryptera kommunikationen som sänds tillbaka till användaren.

1.3 Mål

Målet är att konstruera ett demonstrationsprogram som består av en programvara som är skriven i språken ARText och C för att visa funktionaliteten av en brandvägg. Programvaran ska kunna autentisera användare från en extern enhet samt kunna kryptera och dekryptera data som skickas mellan ECU och användaren..

1.4 Avgränsningar

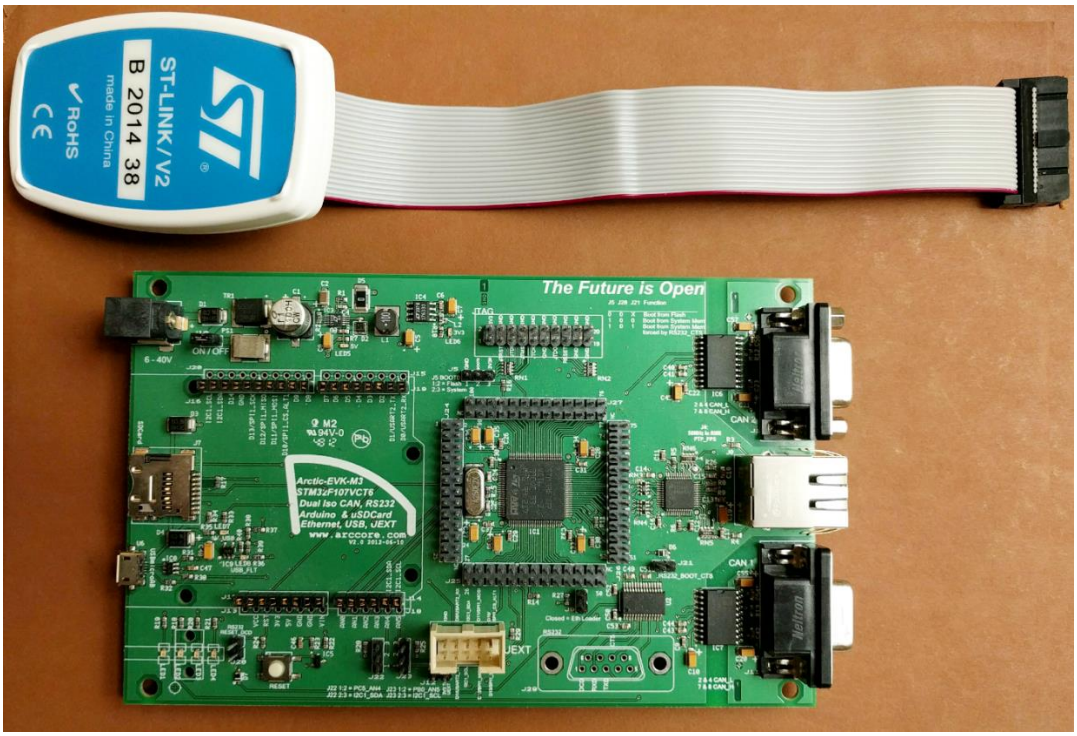
Arbetet ska ej skapa några stora tillägg i ArcCores befintliga produkter utan avser att resultera i en demonstrationsversion av en ECU-brandvägg som sätts ihop av andra system och befintliga komponenter. Krypteringsalgoritmen som används ska inte vara avancerad utan ska vara väldigt enkel. Anledningen till detta är att syftet med demonstrationen endast är att påvisa ett sätt att implementera kryptering. Det ska också vara möjligt att byta krypterings-algoritm i efterhand.

2 Teknisk bakgrund

2.1 Utvärderingskortet Arctic EVK-M3

Ett utvärderingskort utvecklat av ArcCore. Det har en STM32F107 Cortex M3 (32-bitars ARM) processor, dubbla isolerade CAN-kanaler, 10/100 Mb/s Ethernet och stöd för Arduino shields [7].

Det är ett kort som till exempel kan användas som en CAN-gateway i bilar, en flightrecorder eller en CAN till Ethernet brygga [7]. Mjukvaran kan laddas upp på hårdvaran med hjälp av en ST-LINK/V2 JTAG till USB-debugger (se figur 2.1).

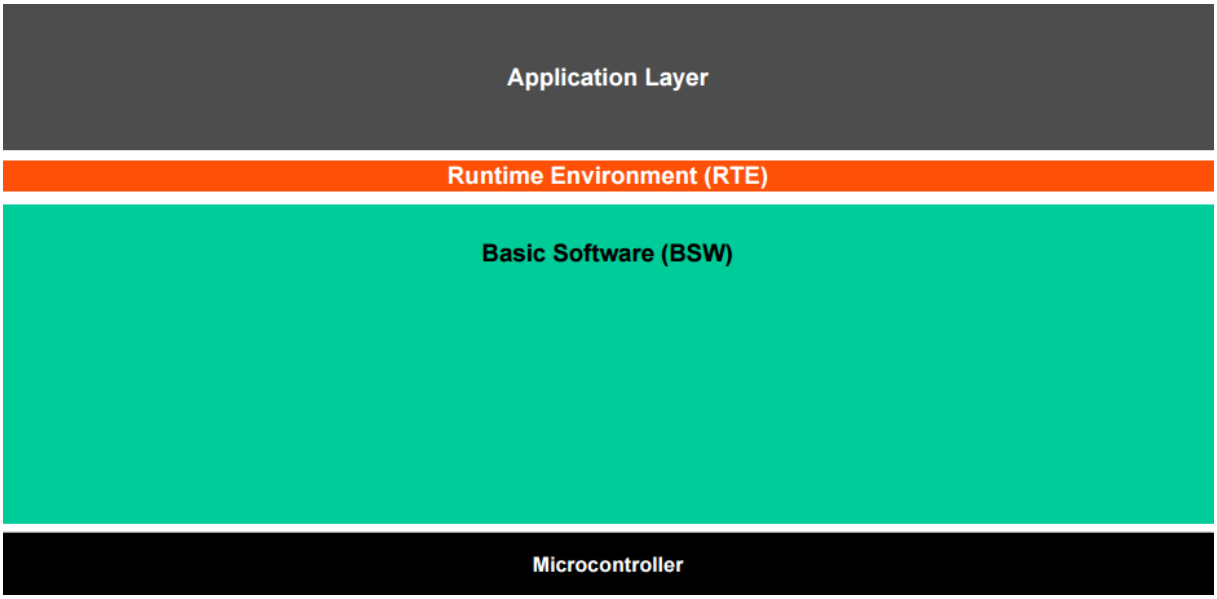


Figur 2.1 Utvärderingskortet Arctic EVK-M3 tillsammans med ST-LINK/V2-debuggern.

2.2 AUTOSAR-Standarden

2.2.1 Introduktion

AUTOSAR är en standard som ämnar förenkla utvecklandet av mjukvara för bilar. Istället för att en viss mjukvara endast går att köra på en viss ECU i en viss bil så ska den mjukvaran som skrivs, i form av SoftWare Components (SWC), går att köra på många olika bilar [8]. Tanken var att en ECU gjord efter AUTOSAR-standard kontrollerar alla system i bilen, från fönsterhissar till centrallås [8]. Dessa olika system är skapade som olika SWC:s som ligger i lagret längst upp i arkitekturen (se figur 2.2) [8]. Komponenterna kombineras i olika kompositioner och kan kommunicera med varandra för att även få information om andra system [8]. "Autosar tillhandahåller själv inga implementationer. Organisationen specificerar bara gränssnitten. Det är upp till olika mjukvarutillverkare att implementera dem. Och upp till biltillverkare att plocka och välja i utbudet för att sätta ihop sin favoritbil." [8].



Figur 2.2 De olika lagren som AUTOSAR består av [9].

2.2.2 BSW - Basic Software

BSW:n innehåller 54 standardmoduler som ger SWC:s tillgång till bilens olika hårdvarusystem [8]. Basmjukvaran är det som motsvarar operativsystemet på en vanlig dator. Eftersom AUTOSAR[2] är en väldigt komplex standard så kommer de moduler som nämns i rapporten enbart att beskrivas kortfattat i Tabell 2.1.

[2] För den intresserade läsaren finns alla specifikationer att läsa på AUTOSARs hemsida. https://www.autosar.org/specifications/release-42/

BswM	- Basic SW Mode Manager, hanterar de andra modulernas lägen.
Cal	- CryptoAbstractionLibrary, krypteringgränssnittet.
Com	- Ett gränssnitt till Rte för kommunikationssignaler.
ComM	- Hanterar användarnas kommunikation.
Det	- Default Error Tracer, användas för diagnostik.
EcuC	- En virtuell modul som innehåller global ECU-information, t.ex. på Pdu-objekt som går genom hela Com-stacken.
EcuM	- Ecu State Manager, hanterar tillståndet som ECU:n befinner sig i.
EthSM	- Ethernet State Manager, tillhandahåller ett abstrakt gränssnitt för ComM, hanterar tillståndet för Ethernetkommunikationen.
Mcu	- Konfigurerar mikrokontrollen.
Os	- Konfigurerar funktionerna i operativsystemet. Baserat på OSEK OS [10].
PduR	- Pdu Router, dirigerar I-PDUs(Interaction Layer Protocol Data Units) mellan transportprotokollmoduler och moduler som hanterar kommunikations-gränssnittet.
Port	- Definierar de fysiska portarna på mikrokontrollern.
Rte	- Runtime Environment, här sker konfigurationen av Rte:n som tillsammans med OS, Com och andra BSW-moduler är en implementering av AUTOSARs VFB, Virtual Function Bus. RTE:n möjliggör SWC:s att kommunicera med varandra oberoende av vilken ECU de befinner sig på.[3]
SecOC	- Secure Onboard Communication. Möjliggör för en BSW-modul att genom verifiering sända data säkert mellan två eller fler mottagare på det inbyggda nätverket.
SoAd	- Socket Adaptor, definierar gränssnittet mellan t.ex. TCP/IP stacken och PduR.
TcpIp	- Definierar TCP/IP stacken i AUTOSAR.

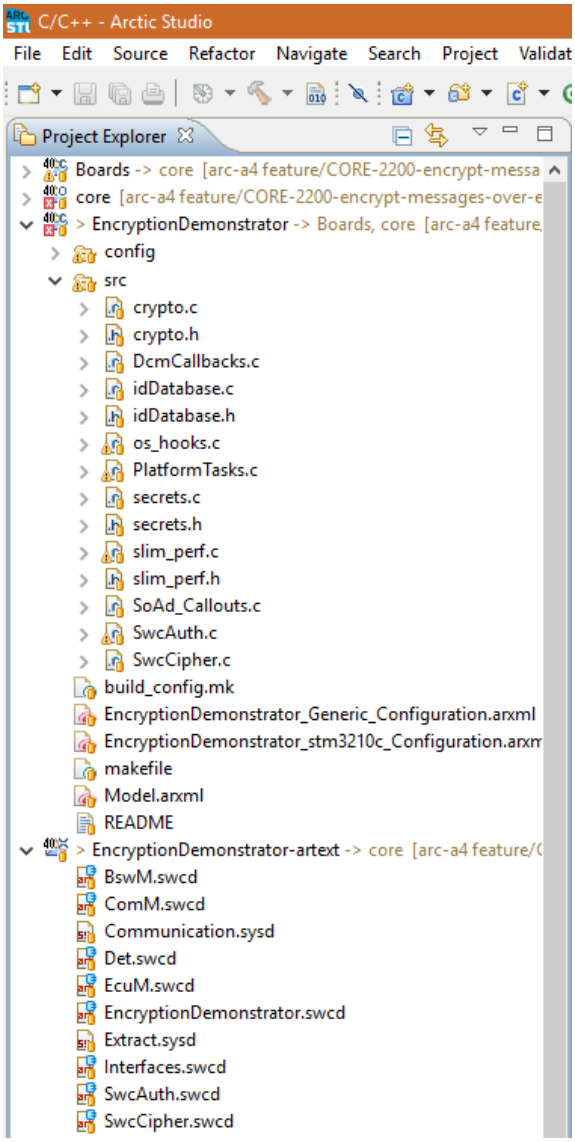
Tabell 2.1 Kort beskrivning av moduler som används i den här rapporten.

2.2.3 SWC - Artext och ECU-projekt

Under utvecklingen av en ECU-programvara behövs två projekt (se figur 2.3). I det ena finns .swcd-filererna, detta kallas för artext-projektet. Efter att .swcd-filerna har definierats exporteras dessa till det andra projektet, ECU-projektet, i form av en AUTOSAR-arxml fil. Detta är ECU-extractet som beskriver hur ECU-mjukvaran ska vara uppbyggd och hur de olika delarna ska fungera, vilka portar de har och vad de olika runnable:arna heter. “ECU extract: is the information from the System Configuration Description needed for a specific ECU” [11].

ECU-projektet innehåller även en source-mapp med en tillhörande .c-fil för varje .swcd-fil (se figur 2.3). Det är här man faktiskt definierar vad som händer i SWC:arnan. Även om de definieras och skapas som olika objekt i Artext-projektet så är det i .c-filerna som själva implementationen skrivs.

[3] RTE och VFB beskrivs mer ingående i det här dokumentethttp://hpi.de/fileadmin/user_upload/fachgebiete/giese/Ausarbeitungen_AUTOSAR0809/NicoNaumann_RTE_VFB.pdf



Figur 2.3 Visar de olika filerna som finns i demonstrationsprogrammets Arctic Studio-projekt, både .swcd filer från artext- projektet och .c filer, ECU-extract och BSWKonfiguration från ECU-projektet.

2.3 Arctic Studio

Arctic Studio är en av ArcCores produkter och även det program som användes för utveckling under detta arbete. Arctic Studio är baserat på Eclipse och är till utseendet väldigt likt. Det skiljer sig dock på vissa viktiga områden så som att det finns en AUTOSAR-vy där det går att se de AUTOSAR-specifika filerna på ett korrekt sätt.

2.4 Arctic Core

Arctic Core är ArcCores implementering av AUTOSARs öppna plattform. Med hjälp av Arctic Core går det att utveckla en fullt fungerande ECU. Arctic Core består av 45 st moduler. MCAL-paketet av Arctic Core tillhandahåller moduler som tar hand om inställningar som är kopplade till en specifik processor. Implementeringen är skriven i C och följer MISRA[4] C,C2 och C3. Källkoden är öppen och tillgänglig för köparen av produkten.

2.5 WinDump

WinDump är ett program för Windows som körs direkt i kommandotolken. Det används för att fånga och analysera nätverkstrafik. Programmet kan, så som i denna rapport, anropas från ett Python-skript. Programmet använder WinPcap-biblioteket [12].

2.6 Wireshark

Wireshark användes för att avlyssna all datatrafik som sändes mellan ECU:n och datorenheten för att på så vis läsa av datapaketen som innehöll de olika meddelanden (se figur 4.5).

2.7 GIT

GIT är ett versionhantergsystem för kod. Det används för att lagra den källkoden som skrivs på ett centralt ställa så att alla delaktiga kan komma åt den. Källkoden laddas ner lokalt från GIT-servern. Efter att ändringar är gjorda är det möjligt att återigen ladda upp källkoden på servern.

[4] MISRA C är en kodstandard framtagen av Motor Industry Software Reliability Association, bilindustrin brukar kräva att standarden följs.

3 Metod

Inledningsvis kommer arbetet till stor del att bestå av informationssökning. För att kunna uppfylla uppsatta mål kommer det att krävas fördjupande kunskaper om AUTOSAR. Eftersom AUTOSAR är en väldigt omfattande standard avsätts en stor del tid till detta. Utöver detta kommer det att krävas inhämtning av information i form av rapportläsning för att få mer kunskap om de olika problem och eventuella lösning som finns angående säkerhet inom bilinstrin.

ArcCores två produkter Arctic Studio och Arctic Core är den mjukvara som kommer att användas och därför kommer viss tid behöva avsättas för att sätta sig in i programvaran genom att göra exempelprojekt och läsa användarmanualer. Mjukvaran kommer att köras på ett av ArcCores framtagna utvärderingskort vid namn Arctic EVK-M3. Detta kort är mer utförligt beskriven i kapitel 2, Teknisk bakgrund.

En befintlig testklass kommer att användas som grund för att testa programvaran, denna kommer att skrivas om som ett skript i språket Python. Skriptet kommer att skicka meddelanden i form av UDP-paket till hårdvaran och även ta emot svar i form av UDP-paket. I detta arbete så användes Ethernet för all kommunikation för att förenkla demonstrationen men i ett verkligt scenario så hade CAN-bussen använts. För att Ethernet-trafiken ska hållas så lik CAN som möjligt så görs valet att använda två olika portar för kommunikation med hårdvaran istället för en. All kod som skrivs läggs upp på ArcCores interna Bitbucket-server med hjälp av Git. I projektet användes SourceTree[5] som Git-klient.

[5] Ett Git-program med ett grafiskt gränssnitt från företaget Atlassian. https://www.sourcetreeapp.com/

4 Genomförande

4.1 Uppstart

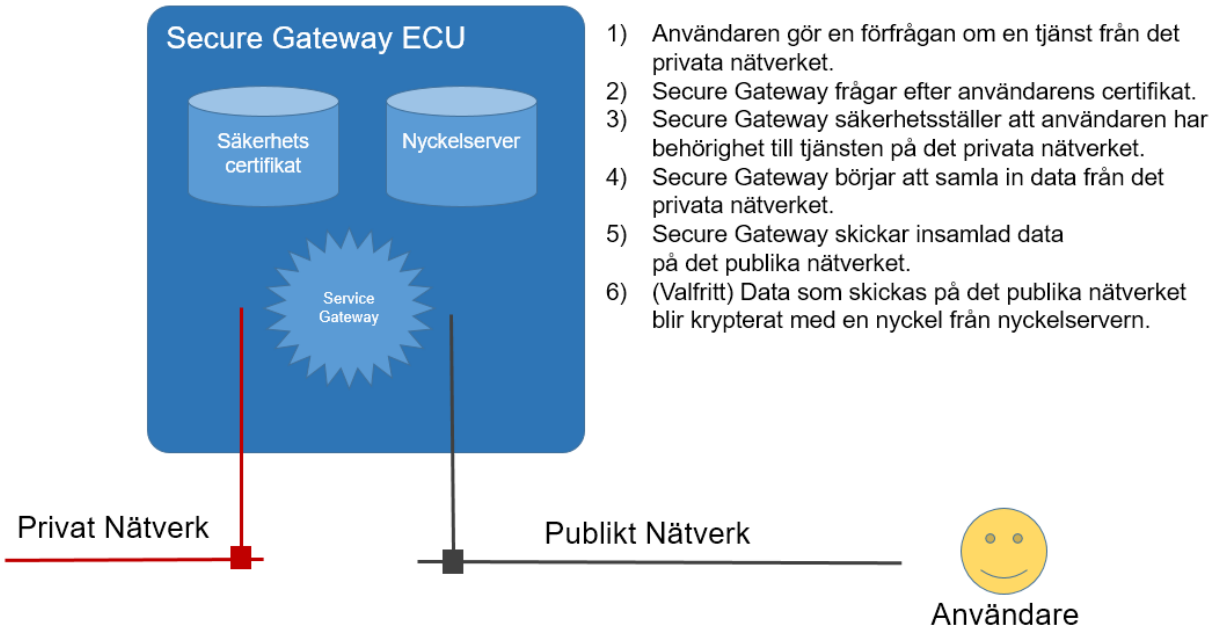
Arbetet startade med att skriva en planeringsrapport och att läsa på om AUTOSAR och ArcCores produkter och hur deras system är uppbyggt. Eftersom det var mycket verktyg, drivrutiner och GIT-filer som behövdes så tog det avsevärd tid att sätta upp utvecklingsmiljöerna. När dessa väl var färdigkonfigurerade tillhandahöll ArcCore två exempel för att ge förståelse för hur Arctic Studio och hårdvaran fungerade. Det första var ett LED-exempel för att med hjälp av olika moduler och lite C-programmering tända de olika LED:arna på chippet. Nästa exempel var för SecOc och gick ut på att prova att skicka meddelanden till SecOc och se svaren, samt ändra koden för att skicka andra meddelanden till SecOc. Under tiden som de två exemplen arbetades igenom så krånglade utvecklingsmiljön flera gånger. Datorbyte, ominstallation av operativsystem, flera avspeglingar från Git och liknande behövdes och gjorde att exemplen och utvecklingsmiljö tog hela vecka två att slutföra.

4.2 Efterforskning

Studier av befintlig information inom området visar att det finns stora brister i säkerheten i bilars datorsystem. I samband med en rapport från 2015 gick FBI, tillsammans med NHTSA, ut och varnade både allmänheten och tillverkare (som på något sätt var involverade med bilsäkerhet) att vara vaksamma mot potentiella brister i säkerheten [13].

Även amerikanska PT&C | LWG, som arbetar med “Forensic Engineering & Recovery Solutions“ har skrivit en artikel om detta problem där de listar olika bilar efter hur enkelt det är att hacka dem och visar att vissa bilar har väldiga brister inom säkerheten [14].

Utvecklingen av en lösning har inom ramen för denna rapport baserats på ett användingsfall (eng: Use Case) som tillhandahölls av ArcCore (se figur 4.1). Användningsfallet antar att en användare som befinner sig på det publika nätverket vill komma åt data som finns på det privata nätverket (1). Ett exempel på detta skulle kunna vara att en användare som via sin mobiltelefon vill avläsa bilens nuvarande hastighet. Secure gateway ber då användaren identifiera sig genom att skicka ett certifikat till brandväggen (2). Brandväggen verifierar om certifikatet är giltigt och om användaren har rättigheter till just den resurs som begärs (3). ECU:n kontrollerar värdet på den resurs som användaren efterfrågar genom att läsa av den interna CAN-bussen.(4). När ECU:n har kontrollerat resursens värde så skickar den ut detta värde på det publika nätet via brandväggen vilket möjliggör för användaren att avläsa värdet (5). Data som skickas ut kan eventuellt krypteras så att enbart användaren som begär data kan läsa av den (6).



Figur 4.1 Illustrerar det användningsfall som tillhandahölls av ArcCore och som användes som utgångspunkt för arbetet.

Det publika nätverket kan bestå av mobiltelefoner, användargränssnitt i infotainmentsystem eller datorer på Internet. Det privata nätverket syftar på den interna kommunikationen mellan ECU:erna i bilen som oftast sker över CAN-bussen men som i framtiden mer och mer kan gå över till SOME/IP.

Enligt en artikel av Lucas Mearian så rekommenderas biltillverkare att både autentisera och kryptera interna meddelanden på CAN-bussen eftersom bussen är väldigt osäker när en angripare väl har fått tillgång till den [3]. Detta är anledningen till att data som skickas, utöver autentisering, även krypteras innan den skickas ut på eller in från det publika nätet.

4.3 Utveckling

4.3.1 BSW-konfigurationen

Arbetet börjar med att studera BSW-konfigurationen i SecOC-demonstrationsprogrammet. SecOC- och Cal-modulen togs bort eftersom brandväggen skulle bli ett fristående program med interna funktioner istället för att använda SecOc-modulen. Insignalen till SecOc togs bort och utsignalen från SecOc mappades om så att den gick förbi SecOc-modulen och direkt till nästa modul istället. För att åstadkomma dessa ändringar krävdes modifiering av PduR-, EcuC-, Com- och SoAd-modulerna. Även ett namnbyte gjordes på signalerna genom hela Com-stacken för att undvika förvirring.

4.3.2 SWC:s och implementeringen

Arbetet fortsatte med modifikationer i applikationslagret där SecOC-demonstrationsprogrammet endast bestod av en SWC. Valet föll på att istället använda två SWC:s med en runnable var. Den ena SWC:n döptes till SwcCipher och hanterar inkommande kommunikation samt kryptering/dekryptering. Den andra heter SwcAuth och hanterar all autentisering. Även här fick portar dirigeras om, denna gång i rotkompositionen när det gällde portarna utanför SWC:arna respektive .swcd-filerna för portarna mellan SWC:arna. Därefter påbörjades den implementering av SWC:arna som görs i .c-filerna. Två SWC:s skapades inuti den gemensamma kompositionen. Testklassen får data inmatad i form av hårdkodade decimala tal med en övre gräns på 255. Detta är på grund av att testklassen konverterar talen till hexadecimala tal och valet gjordes att använda den första siffran som en identifikation av sändaren och den andra siffran som den data som ska skickas. Efter att testklassen har det hexadecimala talet så kommer kryptering att utföras. En bitvis XOR-operation valdes som krypteringsalgoritm. Anledningen till valet är att rapportens syfte enbart är att demonstrera att kryptering är möjlig. Om så krävs kan krypteringen i efterhand bytas ut mot en starkare algoritm. Den mask som har använts är 35 decimalt (00100011 binärt) (se Ekvation 4.1).

$$\begin{array}{rr} & 0000\ 0011 \\ \text{XOR} & 0010\ 0011 \\ \hline & 0010\ 0000 \end{array}$$

Ekvation 4.1 Bitvis XOR-operation av 03 decimalt och 35 decimalt ger resultatet 32 decimalt [15].

Nästa steg är att testklassen tar den indata som hårdkodats och förbereder för sändning. Förberedelserna innebär att testklassen adderar meddelandet med användar-id, krypterar meddelandet och omvandlar det från decimalt till hexadecimalt. Meddelandet skickas via flera olika moduler, bland annat PduR och COM (som är den sista modulen innan meddelandet kommer fram till kompositionen). När meddelandet väl är framme kommer det först att komma till SwcCipher. Denna kommer, med hjälp av en hjälpklass vid namn decrypt.c, åter igen maska meddelandet med en bitvis XOR-mask med samma mask-värde. Då kommer 32, i detta fall, åter att bli 03. Det dekrypterade meddelandet skickas tillbaka till SWC:n som direkt skickar det vidare till SwcAuth då det är SwcAuth som tar beslutet att tillåta eller neka förfrågan. SwcAuth har också en hjälpklass, denna vid namn idDatabase.c som innehåller en lista av tillåtna användare och vilka resurser de har tillgång till (se figur 4.2). Då de dekrypterade meddelandena kommer till SwcAuth kommer denna att maska meddelandena två gånger. Först “0xf0” för att få ut identifikationsbiten av meddelandet och sedan med “0x0f” för att få fram meddelandet. Dessa två värden skickas vidare till idDatabase.c som först kontrollerar att användaren finns med bland godkända användare. Om så inte är fallet returnerar den -1, annars går den vidare och jämför den resurs som har efterfrågats in med vad denna användare har för behörighet. Om dessa stämmer överens returneras 1, annars -1. När SwcAuth har fått svar av idDatabase.c så beslutar den vad som skall returneras till testklassen. Om svaret är 1 från hjälpklassen så skickar SwcAuth “oxaa” till testklassen för att signalera att förfrågan är godkänd. Om svaret från hjälpklassen däremot är “-1” så skickar SwcAuth “0xbb” för att indikera att förfrågan blev nekad.

```
struct UserPriv {
            int id;
            int signal;
};

/*
 * These are the users and the different privileges the users have
 * 20, 50, 60, a0, d0 used by test-case
 */
struct UserPriv userPriv[16] = {
                        {0x00, 0b0000},
                        {0x10, 0b1000},
                        {0x20, 0b0000},
                        {0x30, 0b0000},
                        {0x40, 0b1100},
                        {0x50, 0b0001},
                        {0x60, 0b0010},
                        {0x70, 0b1001},
                        {0x80, 0b0100},
                        {0x90, 0b0000},
                        {0xA0, 0b1111},
                        {0xB0, 0b0010},
                        {0xC0, 0b0101},
                        {0xD0, 0b0011},
                        {0xE0, 0b1011},
};

/*
 * This function first checks the id of the sender to see if
 * it is in the list. If so it checks if it has clearance
 * to access the resource it asked for. If so, return the resource
 * as an ok-message. If not return -1
 */
int checkId(int incomingId, int incomingMsg){
            for(int i = 0; i < 17; i++)
                        if(userPriv[i].id == incomingId)
                                    if(userPriv[i].signal == incomingMsg)
                                                return incomingMsg;
            return -1;
}
```

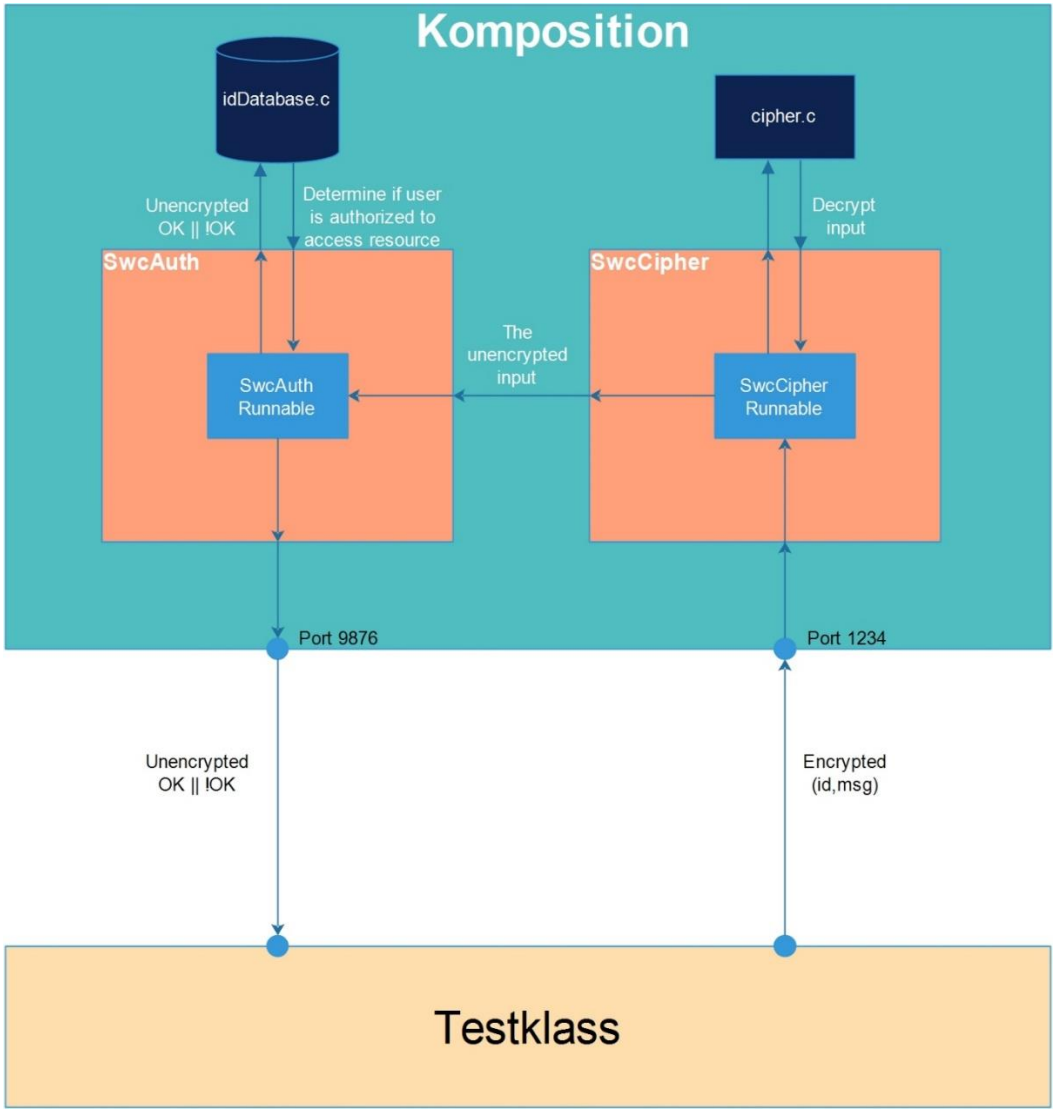
Figur 4.2 Visar koden från idDatabase.c. Här syns alla de 15 användare som finns tillsammans med vilken behörighet de har. Eftersom endast en hexadecimal siffra används för identifikation finns en övre gräns på 15 användare, och en användare är struken för att visa att anrop från denna användare blockeras.

4.3.3 Testklassen-summering

Test-skriptet skrevs i Python. Detta medförde mycket nya kunskaper då Python är ett språk som inte var del av de förkunskaperna som fanns. Trots detta utvecklades ett fungerande skript för att testa den mjukvaran som laddades upp på hårdvaran. Detta skript tar hårdkodad indata och hårdkodad XOR-mask. Det krypterar sedan meddelandena och skickar dem sedan ett och ett som UDP-meddelanden över Ethernet. Skriptet väntar sedan på svar från hårdvaran och när dessa kommer så rensas svaren upp och avkrypteras så att endast den data som är intressant filtreras fram (se figur 4.6 och figur 4.7).

4.3.4 Version 1

Version 1 fungerade enligt beskrivning i kapitel 4.3.2 (se figur 4.3). Informationen kom här in till en SWC och svaret kom från den andra, båda har varsin runnable och svaret var endast ett "OK"- eller "inte OK"-meddelande.



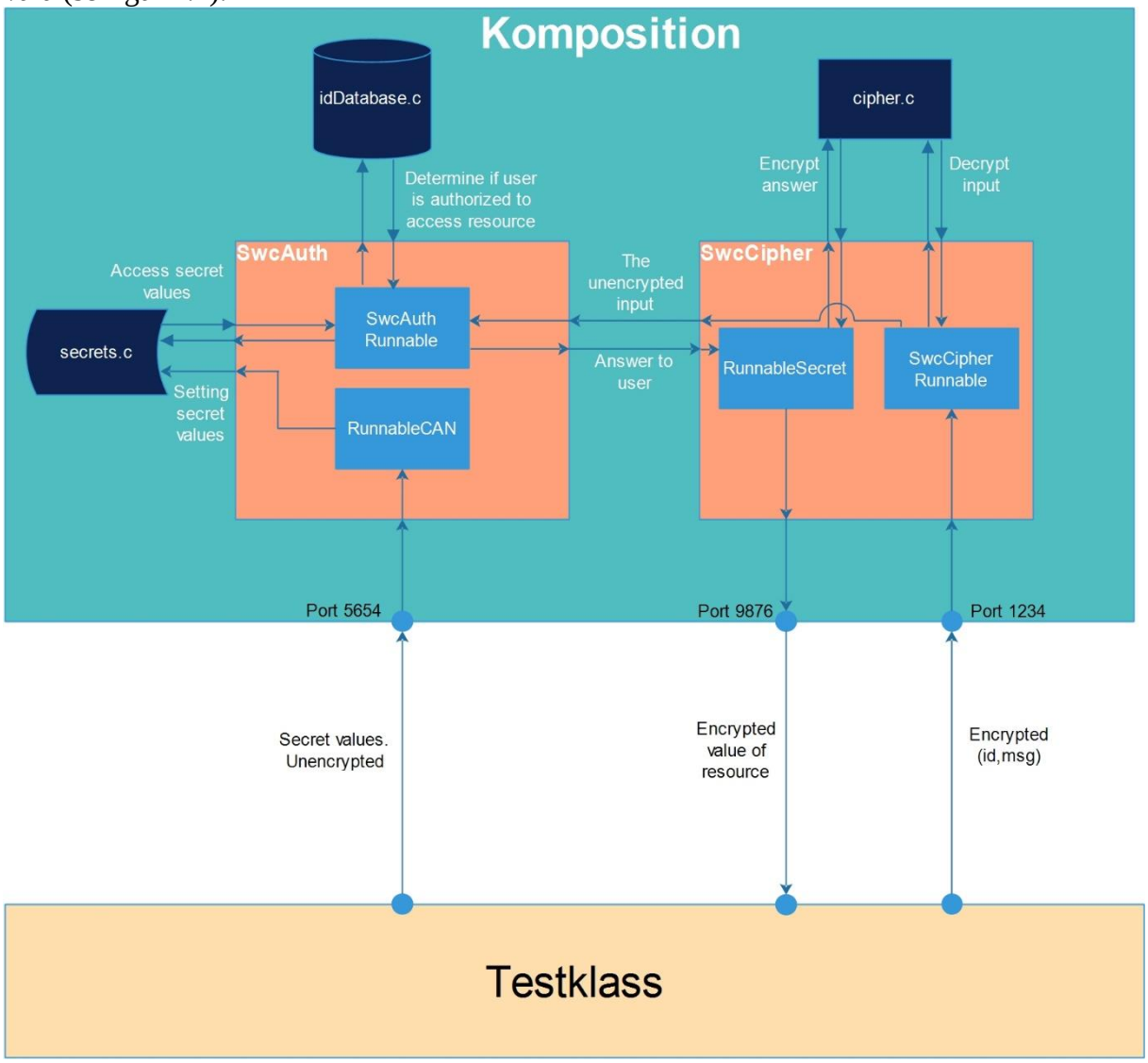
Figur 4.3 Illustration över hur ett meddelande rör sig från testklassen, via olika moduler och SWC:s , samt hur testklassen sedan får tillbaka ett svar. Gäller första versionen av mjukvaran.

4.3.5 Version 2 – slutversionen

Efter att version 1 fungerade som tänkt påbörjades arbetet med version 2. En av de saker som skiljer version 2 från version 1 är att det skapades två nya runnables, en i varje SWC. Detta på grund av att det behövdes en runnable som är ansvarig för indata och en som är ansvarig för utdata, båda SWC:arna kommer i denna version ha både in- och utdata. En ny C-fil skapades också, secrets.c. Tanken är att testklassen börjar testfallet med att kommunicera med RunnableCAN i SwcAuth för att sätta de värden som skall vara CAN-simulerande. Dessa representerar interna värden i bilen och simulerar en CAN-buss. I testfallet är det 4 variabler som anger hastighet, nuvarande växel, gasnivå och antal dörrar bilen har. Dessa sätts till värdena 66 km/h, växel 2, gasnivå 8 av 10 och 5 st dörrar. Testklassen skickar sedan meddelanden på samma sätt som i version 1 till SwcCipherRunnable. Denna runnable dekrypterar med hjälp av crypt.c inkommande meddelande och skickar dem vidare till SwcAuthRunnable som maskar av meddelandet så som i version 1 (se kapitel 4.3.3).

Kontrollen i idDatabase.c fungerar ungefär likadant som i version 1. Om användaren ej har behörighet returnerar idDatabase.c “-1” och SwcAuthRunnable skickar felmeddelandet “0xbb” tillbaka till SwcCiphers RunnableSecret som i sin tur, utan kryptering, och returnerar detta till testklassen. Om användaren däremot är behörig för den signal som efterfrågas returnerar idDatabase.c samma värde som den fick in via meddelande-variabeln. SwcAuthRunnable tar då detta värde och skickar vidare till secrets.c som returnerar värdet för den resurs som efterfrågats. Detta värde returneras av SwcAuthRunnable till SwcCiphers RunnableSecret som krypterar och returnerar meddelandet till testklassen. Testklassen tar mottagen data i en lista, delar listan i separata variabler, dekrypterar indata och presenterar resultat i kombination med en automatiserad kontroll att svaret stämmer med vad det borde

vara (se figur 4.4).



Figur 4.4 Illustration över hur ett meddelande rör sig från testklassen, via olika moduler och SWC:s , samt hur testklassen sedan får tillbaka ett svar. Gäller den färdiga versionen av mjukvaran. Figuren visar även de olika runnables som finns och hur de kommunicerar.

4.3.6 Testklassen

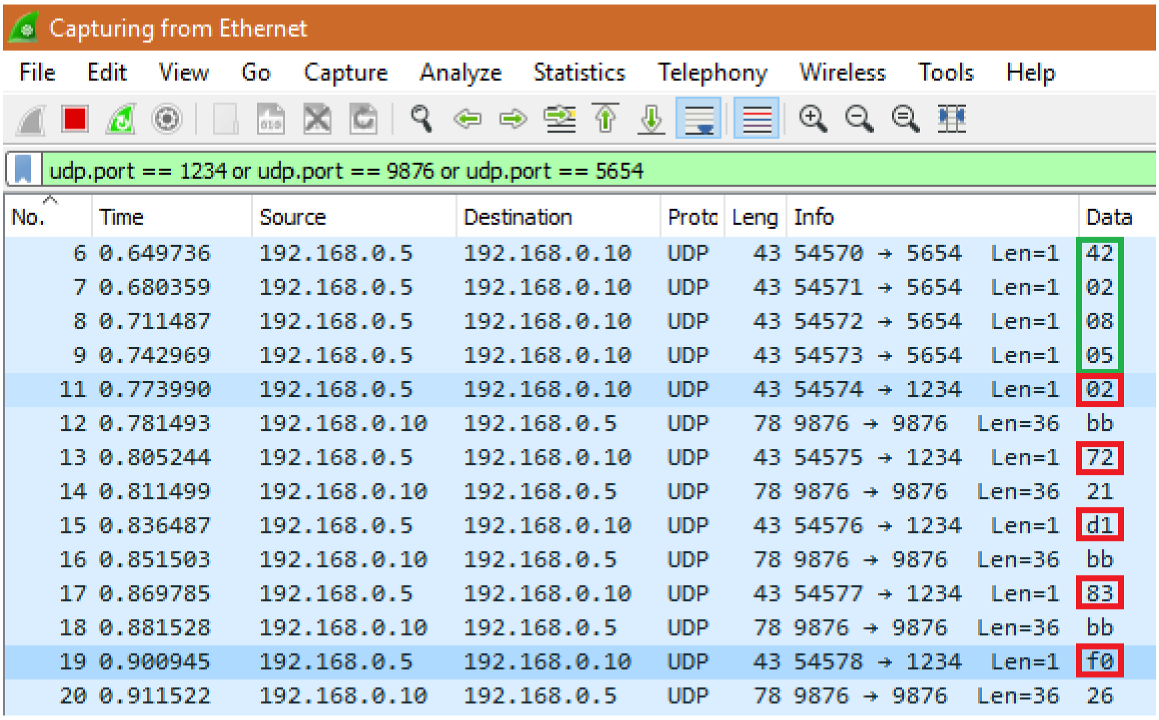
Då testklassen var skriven i Python och behövde modifieras mycket för att passa för detta ändamål istället för SecOC så krävdes mycket tid för att sätta sig in i Python och få syntaxen att fungera korrekt. I testklassen finns en variabel som anger vilket nätverksgränssnitt som WinDump ska läsa av. Denna variabel lyftes ut till en egen fil för att samma testklass skulle gå att köra på olika datorer utan att behöva ändra variabeln efter varje ny avspegling från Git.

Det första testklassen gör är att skicka (sätta) de CAN-simulerande värdena i secrets.c. Detta görs genom RunnableCAN i SwcAuth vilket gör det möjligt att enkelt ändra testfallen. Därefter skickar testklassen de meddelandena menade för kommunikation.

Tesklassen skickar meddelanden av formen (id,msg) med hjälp av hexadecimala siffror vilket betyder att det som syns i hex-kolumnen i Tabell 4.1 är det som kommer att synas i Wireshark då testet körs (se figur 4.5). Det som är i kolumn "status" visar om förfrågan godkänns eller ej och det som är i kolumn "svar" är det svar som kommer från ECU:n i okrypterat format. Kolumn "krypterat svar" visar svaret så som det syns i Wireshark (se figur 4.5).

Anv.	***Anv. dec***	***(id, msg)***	***krypterat (id,msg)***	***hex***	***status***	***svar***	***krypterat svar***
03	*32*	*0010 0001*	*0000 0010*	***0x02***	***-----***	***0xbb** (Nekat)*	***0xbb****
06	*80*	*0101 0001*	*0111 0010*	***0x72***	***OK***	***2** (Växel)*	***0x21***
16	*240*	*1111 0010*	*1101 0001*	***0xd1***	***-----***	***0xbb** (Nekat)*	***0xbb****
11	*160*	*1010 0000*	*1000 0011*	***0x83***	***-----***	***0xbb** (Nekat)*	***0xbb****
14	*208*	*1101 0011*	*1111 0000*	***0xf0***	***OK***	***5** (antal dörrar)*	***0x26***

*Tabell 4.1 De olika användarna som används i testfallet, vad de skickar och vad de borde få som svar. *"bb" krypteras ej då det skickas tillbaka och kan därför ses i klartext i Wireshark i figur 4.5, detta är därför att det inte finns någon anledning att kryptera "bb".*



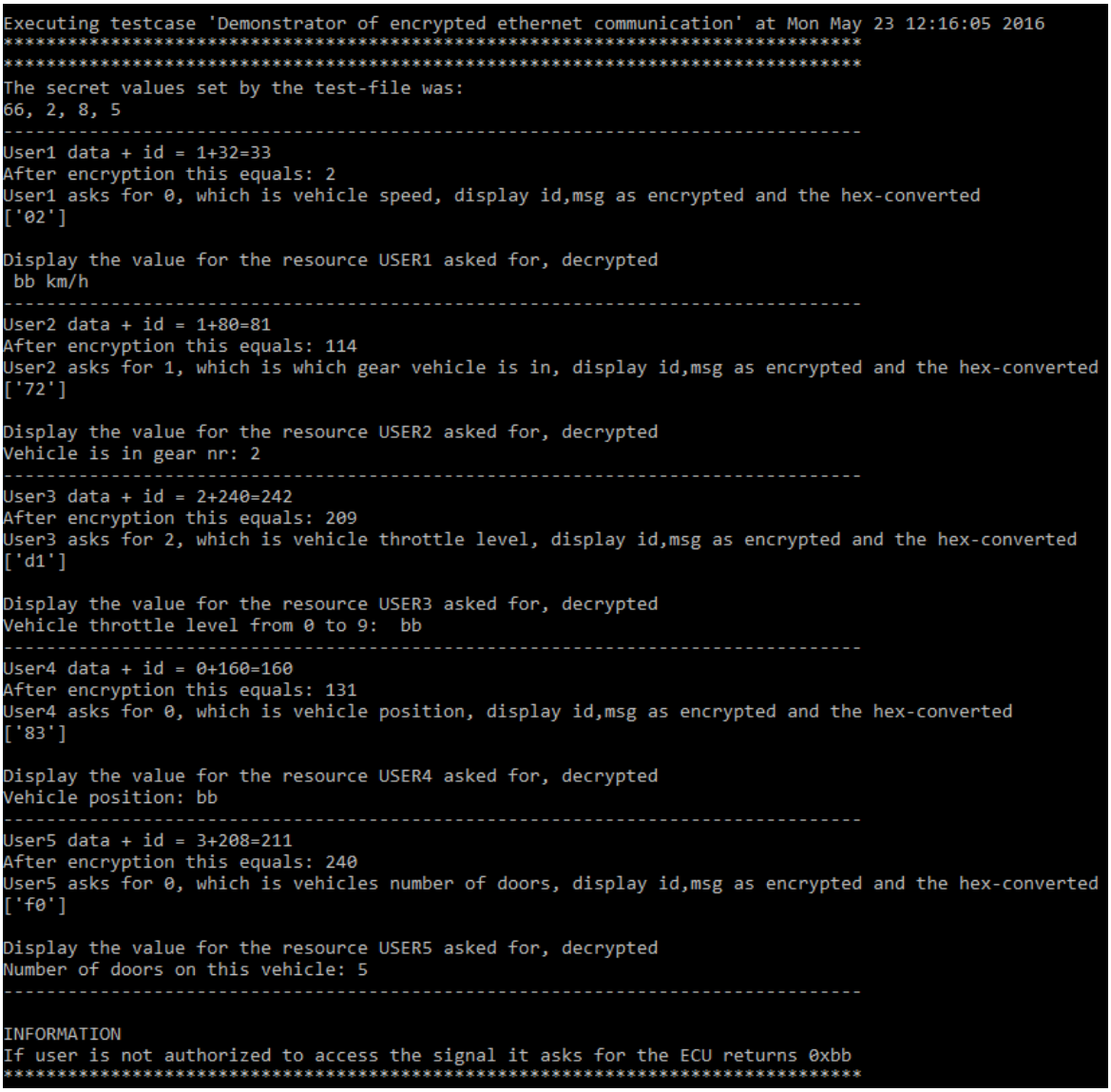
No.	Time	Source	Destination	Protc	Leng	Info	Data
6	0.649736	192.168.0.5	192.168.0.10	UDP	43	54570 → 5654 Len=1	42
7	0.680359	192.168.0.5	192.168.0.10	UDP	43	54571 → 5654 Len=1	02
8	0.711487	192.168.0.5	192.168.0.10	UDP	43	54572 → 5654 Len=1	08
9	0.742969	192.168.0.5	192.168.0.10	UDP	43	54573 → 5654 Len=1	05
11	0.773990	192.168.0.5	192.168.0.10	UDP	43	54574 → 1234 Len=1	02
12	0.781493	192.168.0.10	192.168.0.5	UDP	78	9876 → 9876 Len=36	bb
13	0.805244	192.168.0.5	192.168.0.10	UDP	43	54575 → 1234 Len=1	72
14	0.811499	192.168.0.10	192.168.0.5	UDP	78	9876 → 9876 Len=36	21
15	0.836487	192.168.0.5	192.168.0.10	UDP	43	54576 → 1234 Len=1	d1
16	0.851503	192.168.0.10	192.168.0.5	UDP	78	9876 → 9876 Len=36	bb
17	0.869785	192.168.0.5	192.168.0.10	UDP	43	54577 → 1234 Len=1	83
18	0.881528	192.168.0.10	192.168.0.5	UDP	78	9876 → 9876 Len=36	bb
19	0.900945	192.168.0.5	192.168.0.10	UDP	43	54578 → 1234 Len=1	f0
20	0.911522	192.168.0.10	192.168.0.5	UDP	78	9876 → 9876 Len=36	26

Figur 4.5 Visar paketen i Wireshark. De grönmarkerade visar de CAN-simulerande meddelandena när de sätts av testklassen, de rödmarkerade är förfrågningar från testklassen och övriga är svar från ECU:n.

Testklassen gör även detaljerade utskrifter för att det ska gå att avläsa vad de olika meddelandena betyder (se figur 4.6). Utöver detta gör testklassen en summering för att det ska gå lättare att se värdena, samt ett antal automatiserade testfall för att avgöra om de meddelanden som kommer till testklassen är de som borde komma i kombination med vad testklassen skickar (se figur 4.7).

```
SUMMARY ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Secret values: 66, 2, 8, 5
Secret values hex (sent): 0x42, 0x2, 0x8, 0x5

Values before encryption: 33, 81, 242, 160, 211
Values after encryption: 2, 114, 209, 131, 240

Sent values: ['02'], ['72'], ['d1'], ['83'], ['f0']

Received values: bb, 21, bb, bb, 26
Decrypted received values: bb, 2, bb, bb, 5
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Missing PARSABLE_FILE_REPORT_FILE in GlobalConfig. No parsable file report generated.

Test result:
============
---------- Demonstrator of encrypted ethernet communication ----------

  USER1 asks for which gear the vehicle is in, should be denied             [OK]
  USER2 asks for which gear the vehicle is in, should be granted            [OK]
  USER3 asks for the vehicles throttle level, USER3 does not exist. Denied. [OK]
  USER4 asks for the vehicles speed, should be denied                       [OK]
  USER5 asks for the number of doors on this vehicle, should be granted     [OK]
----------------------------------------------------------------------
```

Figur 4.6 Visar den del av testprogrammets utskrift som sammanfattar alla värden samt presenterar resultaten av och testfallens evaluering.



Figur 4.7 Visar den del av testprogrammets utskrift som visar mest information. Här presenteras alla värden som skickas och tas emot, både krypterat och okrypterat samt vad de betyder.

5 Resultat

För att simulera CAN-kommunikation på ett tillräckligt bra sätt konstruerades testklassen så att den ger ECU:n de värden som skall motsvara bilens interna värden i början av exekveringen av testfallen (se kapitel 4.3.4). Dessa värden motsvarar variabler så som aktuell hastighet, nuvarande växel i växellådan, antal dörrar på denna bil samt gaspedalens nivå på en skala från 1 till 10. Anledningen till detta är att dessa värden lätt ska gå att ändra för att demonstrera att brandväggen fungerar. I ett verkligt scenario hade, förutom antal dörrar, dessa värden kunnat ändras hela tiden.

Demonstrationsprogrammet visar sedan hur fem förfrågningar från fem olika användare skickas från testklassen till ECU:n (se tabell 4.1). Dessa förfrågningar begär värdena för de olika resurserna som finns i bilen och väntar på svar. Efter varje förfrågning tar testklassen emot svaret innan den skickar nästa förfrågning. Dessa förfrågningar och svar visas upp i flera olika versioner. Avkrypterat innan avsändning, krypterat innan sändning, omvandlat under sändning och avkrypterat efter mottagande (se tabell 4.1, figur 4.5, figur 4.6, figur 4.7)

Slutligen visas ett antal automatiserade tester där testklassen avgör om det som mottagits som svar är det som är förväntat svar i kombination med den förfrågan som skickats (se figur 4.7). Testklassen visar också att svaren och förfrågningarna går att avlyssna under färd mellan testklass och ECU men eftersom de är krypterade går det ej att avgöra vad de betyder (se figur 4.5). Trots detta kan både testklass och ECU tolka svar och förfrågningar på ett korrekt sätt.

Det mål som sattes i kapitel 1.3 är härmed uppfyllt och demonstrationen visar att det ej går att avlyssna och tolka meddelandena under färd mellan extern enhet och ECU samt att ECU:n har möjlighet att filtrera vilken data som får komma ut från det privata- till det publika nätet.

6 Slutsats och diskussion

Denna rapport visar att det är möjligt att implementera en kryptering och autentisering av externa enheter som kopplas in i en bil för att filtrera den data, om någon, som enheten bör få tillgång till. Rapporten ger också ett förslag på hur detta exempelvis kan ske. Det finns dock flera aspekter som inte tagits i beaktning vid utvecklingen av demonstrationsprogrammet som också behöver inkluderas i en färdig lösning.

Attacker så som spoofing är fortfarande möjliga: En möjlig attack skulle kunna utnyttja det faktum att sändaren endast identifierar och autentiserar sig med hjälp av ett förbestämt id. Om dessa paket avlyssnas när de skickas från enheten till ECU:n och sedan injiceras igen kan en angripare utföra en spoof-attack. Om dessa meddelanden endast skulle orsaka att ECU:n returnerar ett värde, så som aktuell position av fordonet, skulle det inte utgöra någon säkerhetsrisk då svaret fortfarande är krypterat och attackeraren skulle inte kunna läsa det. Men om de spoofade meddelanden istället är av den typen att de får ECU:n att ändra någonting skulle det kunna utgöra en stor säkerhetsrisk. Om enheten som exempel skulle ge en signal för ECU:n att öka hastigheten och dessa meddelanden blev spoofade skulle det kunna orsaka bilen att öka hastigheten okontrollerbart. För att detta inte ska vara möjligt i slutimplementeringen så måste autentisering via certifikat införas fullt ut och användaren måste signera sina meddelanden.

Starkare krypteringsalgoritm: I denna rapport används endast en bitvis XOR-operation som kryptering. Denna algoritm skulle behöva bytas ut mot en starkare algoritm, eventuellt skulle man kunna använda Cal- och/eller Csm-modulen. Även internt behöver certifikaten och krypteringsnycklarna skyddas med hjälp av kryptering och säkrad minnesåtkomst. Kryptering och avkryptering kan vara väldigt påfrestande för en mikrokontroller, därför rekommenderas en extern hårdvarumodul för avlastning av processorn.

En möjlig lösning för kryptering och autentisering av kommunikationskanalen har tagits fram av TUM CREATE. Den beskriver autentisering för kommunikationen mellan flera ECU:er och en liknande lösning rekommenderas för denna rapportens implementering [16].

6.1 Etik

Om demonstrationsprogrammet ska användas i riktiga sammanhang så finns det också en etisk aspekt att överväga. Mycket testning krävs för att säkerställa att brandväggen fungerar som den är tänkt då eventuella fel kan orsaka svåra olyckor med personskador eller dödlig utgång som påföljd. En olycka skulle även kunna ske om en angripare kan komma in och kontrollera vitala system så som bromsar eller styrning. Detta är en av anledningarna till att XOR-algoritmen skulle behöva bytas ut innan en riktig implementering.

Inom bilindustrin finns det också en standard vid namnet ASIL som beskriver olika säkerhetsnivåer beroende på vad som skall göras och hur vitalt systemet är. Innan en riktig implementation så skulle brandväggs-modulen behöva utvärderas enligt denna standard för att se vilken ASIL-nivå den är klassad för och på så vis avgöra om det finns eventuella brister.

6.2 Hållbar utveckling

En korrekt implementering av brandväggen i alla nya bilar skulle kunna ha både en positiv och negativ påverkan på miljön. Huvudsyftet med brandväggen är att stoppa en angripare från att ta sig in i bilens vitala system. I och med detta så undviker man manipulering av t.ex. motorns bränsleförbrukning vilket skulle ha en negativ påverkan för miljön. Brandväggen skulle även kunna förhindra privatpersoner från att ändra miltalsmätaren innan en försäljning vilket skulle i framtiden kunna få äldre och oftast miljöförstörande bilar av vägen snabbare. Krypteringen i brandväggen kan ha en negativ miljöpåverkan i form av att mer processorkraft och mer el skulle behöva förbrukas.

Referenser

[1] Embedded Systems Course – Lesson 19 AUTOSAR, Real-Time Systems Laboratory, https://retis.sssup.it/sites/default/files/lesson19_autosar.pdf, ???, (Acc 2016-05-27)

[2] Spoofing, http://it-ord.idg.se/ord/spoofing/, ???, (Acc 2016-05-20)

[3] AUTOSAR – A Worldwide Standard is on the Road, http://www.win.tue.nl/~mvdbrand/courses/sse/0910/AUTOSAR.pdf, ???, (Acc 2016-05-15)

[4] AUTOSAR, http://www.autosar.org/, ???, (Acc 2016-05-15)

[5] Firewalls can't protect today's connected cars, http://www.computerworld.com/article/2951878/telematics/firewalls-cant-protect-todays-connected-cars.html, 2015-07-24, (Acc 2016-05-15)

[6] Remote Exploitation of an Unaltered Passenger Vehicle, http://www.ioactive.com/pdfs/IOActive_Remote_Car_Hacking.pdf, 2015, (Acc 2016-05-14)

[7] Introduction to Arctic-EVK-M3, http://download.arccore.com/bootloader/VK-Board%20Getting%20started.pdf, ???, (Acc 2016-05-14)

[8] Autosar – ett Windows för bilelektroniken, Elektronik tidningen, http://www.etn.se/index.php?option=com_content&view=article&id=21003, 2007-04-26, (Acc 2016-05-14)

[9] Layered Software Architecture, https://www.autosar.org/fileadmin/files/releases/4-0/software-architecture/general/auxiliary/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf, 2011-10-06, (Acc 2016-05-24)

[10] Documentation for Arccore Autosar 4 solution version 4.1, http://dev.arccore.com/public/user-doc/UD441x/Os_10190911.html, 2014-04-13, (ACC 2016-05-22)

[11] AUTOSAR, http://www.autosar.org/about/technical-overview/autosar-methodology/, ???, (Acc 2016-05-30)

[12] WinDump - tcpdump for Windows using WinPcap, https://www.winpcap.org/windump/, 2006-10-18, (Acc 2016-15-14)

[13] Motor Vehicles Increasingly Vulnerable to Remote Exploits, Federal Bureau of Investigation, the Department of Transportation and the National Highway Traffic Safety Administration, 2016-03-17, http://www.ic3.gov/media/2016/160317.aspx, (Acc 2016-05-14)

[14] The Most Hackable Cars on the Road, PT&C|LWG| Forensic Engineering Services and Forensic Consultants, 2015-08-19, http://www.ptclwg.com/news/the-most-hackable-cars-on-the-road-1, (Acc 2016-05-14)

[15] XOR bitwise operation, Khan Academy, https://www.khanacademy.org/computing/computer-science/cryptography/ciphers/a/xor-bitwise-operation, (Acc 2016-05-14)

[16] P. Mundhenk, S. Steinhorst, M. Lukasiewycz, S. Fahmy och S Chakraborty: Lightweight authentication for secure automotive networks, Proceedings of DATE, sid. 285–288, 2015

Bilaga A: Körning och testning

Följande steg visar hur demonstrationsprogrammet laddades upp på ECU:n och hur test-skriptet exekverades:

Förberedelser:

1. Ändra IP-adressen på datorns nätverkskort till 192.168.1.5
2. Koppla in nätverkskabeln mellan datorns- och utvecklingskortets nätverkskort.
3. Koppla in USB-kabeln från utvecklingskortet till datorn. (Används till att ladda upp ny mjukvara)

ArcticStudio:

1. Generera alla BSW-modulerna.
2. Kompilera alla filer. Resultatet blir en binär .elf-fil

winIDEA Open[6]:

1. Ladda upp ny mjukvara på utvecklingskortet genom att ladda upp .elf filen till utvecklingskortet.
2. Tryck på "run" så att programmet körs på utvecklingskortet.

Wireshark:

1. Starta Wireshark och välj rätt nätverksadapter att avlyssna.
2. Sätt filtret till "udp.port == 1234 or udp.port == 9876 or udp.port == 5654" (se figur 4.5).
3. Tryck på Enter-tangenten för att filtret ska appliceras.

MSYS:

1. Starta MSYS-kommandotolken som finns i Arctic Studio-mappen
2. Navigera till testklassens rotmapp
3. Kör kommandot "make -f arctictest.mk"
4. Läs av resultatet i kommandotolken. Kan jämföras med de värden som syns i Wireshark för att säkerhetsställa resultatet. (se figur 4.5).

[6] Ett gratisprogram gjort av iSYSTEMS som används för att ladda upp mjukvara och debuggning av Cortex-M och ARM7/ARM9 produkter. Finns på http://www.isystem.com/download/winideaopen