CHALMERS



# Konstruktion av ett prototyplåssystem baserat på NFC-teknik

*Kandidatarbete inom Data- och Informationsteknik*

Daniel Moreau
Fredrik Einarsson
Jonas Hemlin
Kristofer Tapper
Robin Gabre
Sebastian Karlsson

Chalmers tekniska högskola
Göteborgs universitet
Institutionen för Data- och Informationsteknik
Göteborg, Sverige, Juni 2013

Construction of a prototype locking system based on NFC technology

Daniel Moreau
Fredrik Einarsson
Jonas Hemlin
Kristofer Tapper
Robin Gabre
Sebastian Karlsson



Examiner: Arne Linde.

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: NFC-Forum logotype. Licence of usage is explained and can be acquired on their homepage (NFC-Forum, 2013).

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

# Förord

Denna rapport ingår i ett kandidatarbete utfört vid Institutionen för data- och informationsteknik, Chalmers tekniska högskola under våren 2013. Rapporten beskriver utvecklingen av ett NFC-baserat låssystem bestående av en Android-applikation samt mjukvara till en mikrokontroller av Arduino-typ.

Projektgruppen önskar tacka Lars Svensson för den tid han har lagt ner för att leda projektet i rätt riktning.

**Abstract**

Today, the smartphone is a part of everyday life and is under constant development to handle more task for its users. The tasks, which are integrated in the smartphones, are now one less thing a user doesn't need to carry. The purpose of this project is to replace the key ring by making the smartphone capable of unlocking doors. Hence, the report describes how a prototype of a locking system is designed and implemented.

In order to provide the desired functionality, a locking device has been developed around the Arduino platform which has the ability to control an electric bolt. Furthermore, a smartpone application for Android has been developed. The application provides the user with an interface which provides the ability to interact with the locking system. The smartphone application communicates with the locking system through NFC (*Near Field Communication*).

The security of the system is that the user authenticates against the smartphone application with a user-selected PIN code. Furthermore, the communication between the two systems is partly encrypted with RSA encryption algorithm with 512 bit keys.

The market has expressed a need for a product of this kind which likely will be available on the consumer market soon.

**Sammanfattning**

Mobiltelefonen utökas idag ständigt till att klara av att utföra fler och fler uppgifter åt dess användare. De saker, vars funktioner nu erbjuds genom dagens mobiltelefoner, behöver användaren alltså inte längre bära med sig. Det här arbetet syftar till att ersätta nyckelknippan genom att ge mobiltelefonen funktionen att låsa upp en dörr. Därav redogör rapporten för hur en prototyp av ett låssystem designas och implementeras.

För att kunna erbjuda den önskade funktionaliteten har en låsenhet utvecklats kring Arduino-plattformen vilken har möjligheten att styra en elektrisk låskolv. Vidare har en mobilapplikation för Android utvecklats, vilken användaren använder för att kunna låsa eller låsa upp en dörr med en integrerad låsenhet. Mobilapplikationen kommunicerar med låsenheten via NFC (*Near Field Communication*).

Säkerheten i systemet består i att användaren verifierar sig mot mobilapplikationen med förvald PIN-kod och kommunikationen är krypterad med krypteringsalgoritmen RSA med 512 bitars nycklar.

Intresse för en produkt av detta slag finns och kommer troligtvis att dyka upp på den kommersiella marknaden inom en snar framtid.

## Ordlista/Terminologi

**Aktivitet** En Java-klass vilken representerar en enstaka fokuserad uppgift i Android, vanligen en skärm.

**Användare** En person vilken använder mobilapplikationen.

**Android** Ett öppet mobilt operativsystem för främst smarttelefoner och pekplattor.

**Android beam** En teknologi för att sända data mellan två Androidenheter via NFC.

**API** (*Application Programming Interface*) en regeluppsättning för hur en viss programvara kan kommunicera med annan programvara.

**Bluetooth** En standard för trådlös kommunikation mellan enheter.

**Börkrav** Krav som bör ha uppfyllt för en bra produkt.

**Ciphertext** Krypterad text, motsatsen till klartext (*plaintext*).

**DEP** (*Data Exchange Protocol*) ett transportprotokoll på RF-nivå.

**DSAP** (*Destination service access point*) adress vilken utgör mål för NFC kommunikation.

**I2C** (*Inter-intergrated Circuit*) ett master/slave protokoll som dikterar kommunikation över en delad systembuss.

**Key-value-store** Abstrakt datatyp bestående av en samling av par där nyckeln är unik.

**Kommunikationsstack** En implementation av de protokoll vilka används för att åstadkomma kommunikation. Protokollen är strukturerade i en stack.

**Kryptoanalys** Läran om att hitta svagheter i eller knäcka *krypotosystem* utan kännedom om dess hemliga nycklar eller algoritmer.

**Kryptosystem** När termen kryptosystem används syftas algoritmer eller modeller vilka krävs för att implementera kryptering.

**LLC** (*Logical Link Control*) Den process vilken hanterar LLCP.

**LLCP** (*Logical Link Control Protocol*) Det protokoll vilket dikterar hur NFC-kommunikation sköts på länknivå.

**Låsenhet** Den del utav den utvecklade prototypen vilken är integrerad i låset.

**Manifest** En fil i ett Androidproject vilken specificerar globala inställningar för applikationen.

**Material** De fysiska delarna vilken prototypen utgörs av.

**MIU** (*Maximum Information Unit*) indikerar LLC-lagrets datakapacitet.

**MIUX** (*Maximum Information Unit extension*) används då LLC-lagret har större datakapacitet än 128-byte.

**Mobil plånbok** En ansats till att ersätta plånboken med en applikation i en smarttelefon.

**NFC** (*Near Field Communication*) En trådlös kommunikationsteknik över korta avstånd.

**NFC-stack** Specifik kommunikationsstack för NFC.

**NFC Forum** En organisation med målet att sprida och utveckla NFC-teknologin.

**NDEF** (*NFC Data Exchange Format*) Ett dataformat vilket specificerar hur data skall skickas och vad det är för data som skickas.

**NPP** (NDEF push protocol) ett protokoll som beskriver utväxling av data vid användning av NFC. NPP fasas ut till fördel för SNEP.

**OSI-modellen** en konceptuell modell för datorkommunikation i 7 lager.

**PDU** (*Protocol Data Unit*) Ett datapaket utformad i enlighet med de regler som gäller i för ett protokoll inom ett lager.

**RF** (*Radio frequency*) Är en frekvens i området ca 3 kHz till 300 GHz.

**RSA** (*Ron Rivest, Adi Shamir och Leonard Adleman*) Algoritm vilken används vid asymmetrisk kryptering.

**SDP** (*Service Discovery Protocol*) Ett protokoll vilket används när en avsändare inte känner till DSAP värdet för målenheten.

**Server** Ett datorsystem vilken har till uppgift att serva andra system.

**Skallkrav** Krav vilka skall vara uppfyllda i prototypen.

**Smarttelefon** En mobil enhet som kan användas både som avancerad mobiltelefon och handdator.

**Sköld** Från engelskans shield.

**SNEP** (*Simple NDEF exchange protocol*) Ett protokoll beskriver utbytet av data vid användning av NFC.

**SSAP** (*Source service access point*) Den adress vilken brukande applikationen tilldelats vid användandet av NFC.

**SPI** (*Serial Peripheral Interface*) ett master/slave protokoll som dikterar kommunikation över en delad systembuss.

**SQLite** En liten och lättviktig relationsdatabas.

**Target** Målenhet för NFC kommunikation.

**TLV** (*Type-Length-Value*) uppkopplings specifika parametrar som utbytes under initieringsförfarandet.

**Upplåsningsnyckel** Syftar till den sträng vilken ger tillgång till administration eller öppning av låsenheten.

**URI** (*Uniform Resource Identifier*) Sträng vilken används för att identifiera eller namnge en resurs.

**Verktyg** Källkodsbibliotek, utvecklingmiljöer och specifikationer.

**Wifi** Teknik för trådlösa nätverk.

# Innehåll

# 1 Inledning

När mobiltelefonen först lanserades var den stor, otymplig och dyr, med kommunikation som enda syfte. Sedan dess har utvecklingen eskalerat och allt fler funktioner har integrerats, medan själva mobiltelefonen har gjorts mindre, smidigare och mer tillgänglig för allmänheten. Idag äger i princip alla svenskar en mobiltelefon och byter dessutom dessa ofta (MTB, 2013), då det är både relativt billigt att köpa en ny samt att tekniken snabbt framskrider. I och med den stora efterfrågan på mobiltelefoner är konkurrensen inom branschen hårdare än någonsin för producenterna och för att deras produkter ska bli konkurrenskraftiga på den ständigt utvecklande marknaden måste produkterna oupphörligt innehålla nya smarta funktioner.

På senare tid har övergången till så kallade smarta telefoner bidragit ytterligare till att öka funktionaliteten hos mobiltelefonen. Numera är den inte enbart ett redskap för kommunikation, utan en enhet med lika många eller till och med fler funktioner än en persondator. En av anledningarna till denna utveckling är framstegen inom integrerade kretsar och dess tillverkningsteknik. Denna faktor har bidragit till att fler transistorer kan placeras på en mindre yta (Intel Corporation, 2005) och därmed göra elektronikkomponenter både mindre, strömsnålare, snabbare och mer funktionella. Den direkta följden av detta är att mobiltelefonen nu kan inrymma både funktionaliteten av fler kringenheter, till exempel kalkylatorn, samt inneha kraft nog att hantera dem.

Wifi och Bluetooth är exempel på kommunikationstekniker vilka i stort sett alltid finns inbyggda i en mobiltelefon idag. Enligt Deffree (2012) börjar nu även en ny kommunikationsteknik få allt större utbredning, NFC (Near Field Communication). Detta är en teknik för tillförlitlig kommunikation över korta avstånd, vilken bland annat ligger till grund för en senare ansats att försöka ersätta något vi alltid bär med oss, plånboken. Tanken med denna produkt vilken i många fall kallas den mobila plånboken, vilken Gunn (2012) beskriver, är att göra anspråk på att ersätta våra kontanter, rabatt- och kreditkort med programvara i mobiltelefonen.

Idén bakom detta arbete baseras på samma tankegångar vilka ligger bakom utvecklingen av den mobila plånboken; att ersätta något vi alltid bär med oss. Kandidatarbetet åsyftar till att ersätta hemnyckeln genom att utveckla en prototyp av ett låssystem vilket tillåter användaren att på ett snabbt och bekvämt sätt låsa upp dörren till sitt hem med sin mobiltelefon. För att möjliggöra denna funktionalitet används samma kommunikationsteknik vilken ligger till grund för den mobila plånboken, NFC.

Resultatet av kandidatarbetet kan vara en mycket åtråvärd produkt för privatpersoner vilka är trötta på att leta efter sina hemnycklar samt önskar en flexiblare lösning än den klassiska hemnyckeln. Målet är att ytterligare följa trenden mot en allt mer funktionell mobiltelefon genom att också använda den som en hemnyckel.

Visionen för en färdig produkt av detta slag, och alltså inte för resultatet av detta kandidatarbete, är att produkten ska kunna användas i större företag där det finns hundratals dörrar vilka utrustats med den utvecklade låsenheten samt tusentals anställda vilka alla använder sin mobiltelefon för att kontrollera låsenheterna. Många accessnivåer kan finnas där en specifik användare kan ingå i valfritt antal. Produkten kan underhållas av en central enhet vilken snabbt, smidigt och säkert kan konfigurera om systemet efter givna instruktioner.

Det finns också andra visioner för en färdig produkt av detta slag. Låssystemet behöver nödvändigtvis inte vara ett stort, komplext och centraladministrerat system utan kan istället vara litet, simpelt och mobilt system vilket kan användas i cykellås eller hänglås.

## 1.1 Syfte

Syftet med rapporten är att redogöra för hur en prototyp av ett låssystem designas och implementeras. Låssystemet består av en mobilapplikation och en mikrokontroller där mikrokontrollern är integrerad i låset samt exekverar egenutvecklad mjukvara. Skulle det visa sig att produkten inte uppfyller kravspecifikationen (se Appendix A: Kravspecifikation) ska en analys genomföras i vilken en fullständig slutprodukt beskrivs teoretiskt.

## 1.2 Omfattning

Rapporten beskriver endast processen för utvecklingen av en prototyp utav ett låssystem vilket beskrivs i avsnitt 4. Vidare beskriver rapporten de nödvändiga förkunskaperna vilka krävs för utvecklingen av en sådan prototyp. Dessa förkunskaper innehåller i stort hur kommunikation via NFC implementeras samt hur säkerhet appliceras på denna kommunikationsform.

Fokus ligger på att utveckla en prototyp innehållande en låsenhet samt tillhörande mobilapplikation för privatpersoner. Detta görs då ett låssystem bestående av ett större antal låsenheter eller ett större antal användare till mobilapplikationen ökar komplexiteten för hela projektet. Ett sådant låssystem kommer också ställa andra krav på lösningen än de krav vilka ställts då målgruppen för produkten är privatpersoner.

Mobilapplikationen begränsas till Android eftersom det är
lättillgängligt, välkänt och lämpligt för användningsområdet. Denna avgränsning kommer också naturligt av att det bara finns stöd för NFC hos ett fåtal telefoner där merparten av dem är baserade på Android (NFC World, 2013).

Låsenheten baseras på en mikrokontroller från Arduino-plattformen då plattformen är användarvänlig (McRoberts, 2010), har en relativt låg prisbild samt att flertalet källkodsbibliotek finns vilka kan underlätta implementeringen av låsenheten. Vidare har Arduino-plattformen stöd för NFC i form av ett färdigutvecklat påstickskort, en så kallad sköld[1], med tillhörande källkodsbibliotek.

## 1.3 Utmaningar

Under den initiala fasen av låssystemets utveckling identifierades tre huvudsakliga utmaningar vilka beskrivs punktvis nedan.

- Stora delar av kommunikationsstacken för NFC skall implementeras för låsenheten. Implementationen skall utformas efter hur Androids NFC-stack är uppbyggd för att erhålla kommunikation. Att implementera denna kommunikationsstack medför en större utmaning då den är komplex och innehåller ett flertal olika nivåer. Genom att studera Androids källkod framgår det också att dokumentation kring NFC-implementationen är bristfällig, vilket medför ytterligare svårigheter (Android, 2013d).

- Skapa en Android-applikation vilken ska innehålla funktioner för att kommunicera mot mikrokontrollern via NFC. Androidutveckling medför ytterligare svårigheter jämfört med vanlig mjukvaruutveckling då hänsyn måste tas till Androids egenheter, vilka beskrivs närmare i teoriavsnitt 3.3. En förståelse för de källkodsbibliotek vilka hanterar NFC-kommunikationen måste också byggas upp.

[1] från engelskans shield

- Att implementera säkerhet vid NFC-kommunikation mellan två enheter vilka använder olika programmeringsspråk. Då en mikrokontroller generellt inte har mycket minne eller hög klockfrekvens kan dessa egenskaper skapa utmaningar då säkerhet i form av kryptering ofta kräver omfattande beräkningskapacitet, vilket beskrivs närmare i teoriavsnitt 3.2.5. Den utmaning vilken uppkommer då olika programmeringsspråk används är problem med varierande datarepresentationer och implementationer av funktioner, metoder och klasser.

# 2 Metod

I den initiala fasen genomförs en analys från användarens perspektiv där de krav användaren kan ställa på prototypen listas. Dessa krav utformas sedan från låssystemets perspektiv och sammanställs till en kravspecifikation (se Appendix A: Kravspecifikation). Utifrån kravspecifikationen väljs de material, det vill säga de fysiska delarna vilken prototypen utgörs av, vilka är mest lämpade för utformningen av prototypen.

Under designfasen av projektet väljs hur prototypens funktionalitet distribueras ut i låssystemet och ett kommunikationsprotokoll för systemets delar sammanställs. All funktionalitet låssystemet innehåller delas upp på prototypens två enheter. Kommunikationsprotokollet beskriver hur datan som skickas mellan enheterna är uppbyggd, hur kommunikationsförloppet mellan enheterna ser ut samt hur säkerhet appliceras på kommunikationen.

Projektet övergår nu till en iterativ arbetsgång där ytterligare funktionalitet, utifrån kravspecifikationen, försöker implementeras för varje iteration. För att kunna utföra en iteration undersöks vilka eller vilket verktyg såsom källkodsbibliotek, utvecklingsmiljöer och information som krävs för att implementera funktionaliteten. Verktygen utvärderas och de som bedöms kunna bidra till, eller underlätta för, implementation av den sökta funktionaliteten väljs ut. Verktygen används sedan för fortskridandet av iterationen. Den tillagda funktionaliteten utvärderas och om den lever upp till kravspecifikationen påbörjas nästa iteration. Skulle den nyligen tillagda funktionaliteten inte uppfylla de krav som ställs undersöks vad som är möjligt att ta med av den nyligen tillagda funktionaliteten till nästa iteration, eller om resultatet av iterationen skall kasseras.

# 3 Teori

I detta avsnitt beskrivs den teori vilken är nödvändig att förstå vid utvecklingen av prototypen. Teorin bidrar till en ökad förståelse men främjar framförallt till det direkta framskridandet av projektet. De områden läsaren behöver ha kunskaper inom vilka också beskrivs är: NFC, säker kommunikation, Android- och Arduino-plattformen. De följande styckerna motiverar i tur och ordning varför de nämda kunskapsområderna täcks av teoriavsnittet.

Eftersom prototypen använder NFC-teknik som kommunikationsmetod är kunskaper inom NFC relevant. Kunskaper inom NFC, och inte bara inom dess användande, är främst nödvändig för utvecklingen av mjukvaran till mikrokontrollern då i stort sett hela kommunikationsstacken för NFC behöver implementeras (se avsnitt 5.2.2). Detta står i kontrast till utvecklandet av Android-applikationen där kunskaper om hur Androids API för NFC används är nödvändiga medan kunskaper inom dess implementation är mindre viktiga.

Vidare är kunskaper inom säker kommunikation mycket relevant
eftersom ett låssystem ska utvecklas och NFC har inte någon inbyggd kommunikationssäkerhet. En applikation vilken nyttjar NFC bör därför själv implementera säkerhet om behov finns vilket är fallet i detta projekt. Säker kommunikation kan uppnås på många olika sätt och därför är kunskap för att både välja rätt metod och kunskap om den valda metodens implementation av stor vikt.

Utvecklingen av Android-applikationen kräver förutom kunskap i Java-programmering även kunskap kring applikationsutveckling för Android vilket tas upp i detta avsnitt. Bland annat har applikationer för Android en speciell livscykel (Android, 2013e) vilken är nödvändig att känna till. Det är även nödvändigt att känna till de Android-specifika API:er som Android (2013h) tillhandahåller. Särskilt bör vetskap om de Android-specifika API:er för kommunikation via NFC besittas.

För att utveckla en applikation till Arduino-plattformen krävs förutom kunskaper inom C++-programmering även kunskap om Arduino vilket behandlas i detta avsnitt. En Arduino har inte något operativsystem vilket gör att en Arduino-applikation kör direkt på den underliggande hårdvaran. Vidare behövs kännedom om hur kretskorten kommunicerar med varandra redogöras. Slutligen bör kunskaper angående nödvändiga Arduino-specifika källkodsbibliotek samt hur Arduino kommunicerar via NFC beskrivas.

## 3.1 NFC

NFC (*Near Field Communication*) är en trådlös kommunikationsteknik över avstånd upp till ungefär 10 cm vilken standardiserades år 2006 (Timalsina, 2012). Vidare skriver Timalsina att NFC baseras på RFID (*Radio Frequency Identification*), vilket är en väletablerad trådlös kommunikationsteknik som funnits sedan 1983. NFC och bygger vidare på RFID genom att erbjuda tvåvägskommunikation mellan två enheter.

Tekniken bygger på att två mikrochip med stöd för NFC-teknik kommunicerar med varandra genom magnetisk induktion och Timalsina beskriver också att kommunikationen mer specifikt baseras på att ett mikrochip modifierar det andra mikrochipets magnetfält genom att generera radiovågor med frekvensen 13.56 Mhz. Denna modifiering ger upphov till att data kan skickas i hastigheterna 106 Kbps, 212 Kbps eller 424 Kbps. Dock kan endast en enhet i taget modifiera den andra enhetens magnetfält vilket endast ger halv-duplex kommu-

nikation[2] (Timalsina, 2012).

NFC-enheter kan vara antingen passiva eller aktiva. Timalsina (2012) förklarar att aktiva enheter har strömförsörjning och återfinns till exempel i mobiltelefoner medan passiva enheter saknar strömförsörjning och återfinns till exempel som taggar på planscher eller i busskort. Vidare skriver Timalsina att två aktiva enheter kan kommunicera mot varandra och att aktiva enheter även kan kommunicera mot passiva enheter genom att den aktiva enheten ger strömförsörjning till den passiva enheten med hjälp av magnetisk induktion. Två passiva enheter kan inte kommunicera eftersom inget magnetfält uppstår utan tillförd energi.

NFC Forum (2013a) standardiserade NFC-tekniken år 2006 och är den organisation vilken har ansvar för att ta fram de specifikationer vilka gör kommunikation via NFC möjlig mellan olika typer av enheter. De har också ansvar för att sprida och uppmuntra användandet av NFC-teknik samt utbilda företag med målet att företagen ska följa de officiella specifikationerna vilket leder till att sömlös funktionalitet erhålls mellan enheter från olika tillverkare.

### 3.1.1 NFC Stack

I de följande teoriavsnitten beskrivs de protokoll vilka utgör NFC-stacken och möjliggör kommunikation mellan två aktiva enheter. Dessa protokoll är utformade av NFC Forum (2013a). Vidare täcker protokollen hela OSI-modellen förutom applikationslagret med tillhörande säkerhet då detta inte ingår i NFC-stacken. Listan nedan är en kort översikt över de protokoll vilka ingår i NFC-stacken och stackens motsvarande OSI-lager beskrivs uppifrån och ned. Dessa OSI-lager samt dess motsvarande lager i NFC-stacken kan beskådas i figur 1.

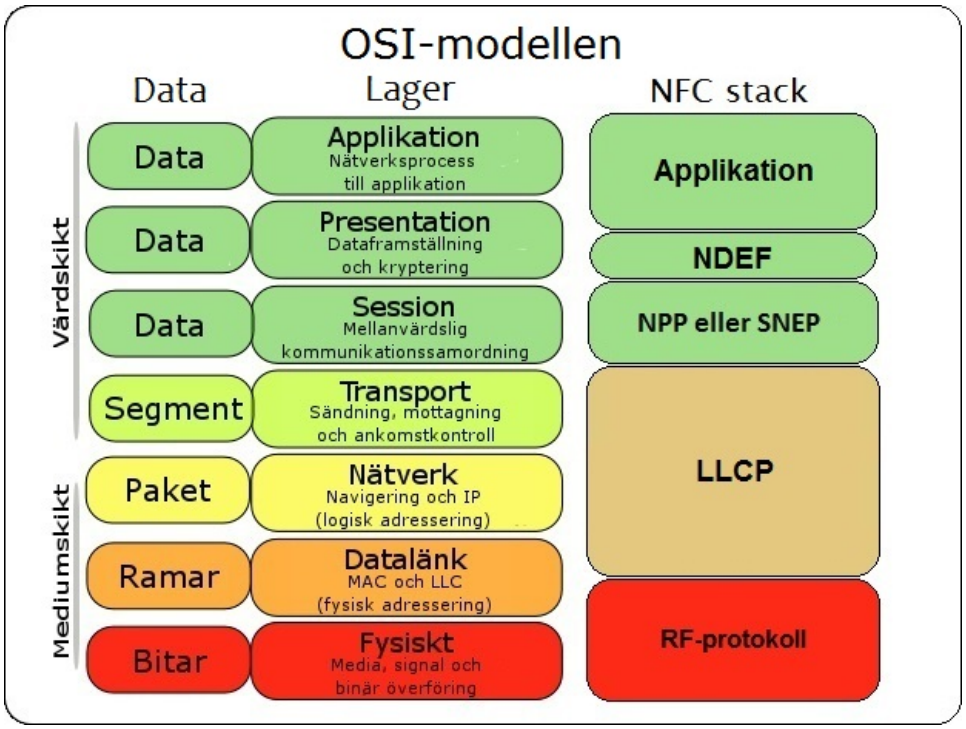
- **NDEF** (*NFC Data Exchange Format*). Data som skall utväxlas mellan två aktiva NFC-enheter skall formateras efter NDEF-protokollet (NFC Forum, 2006). PDU:n som erhålls benämns NDEF-meddelande och utgör standardformatet vid bruk av NFC. Informationen ligger som ett eller flera fält i NDEF-meddelandet som så kallade NDEF-`records`.

- **NPP** (*NDEF Push Protocol*) eller **SNEP** (*Simple NDEF Exchange Protocol*). Vidare används protokollen NPP (Android Open Source Project, 2011) eller SNEP (NFC Forum, 2011b) för att utbyta NDEF-meddelanden mellan två aktiva NFC-enheter. Då NPP har ersatts av SNEP kommer bara SNEP att beskrivas. SNEP har till skillnad från övriga protokoll i stacken två typer av PDU:er. Dessa benämns förfråga samt respons och vilken benämning som används beror på vilken part som sänder.

- **LLCP** (*Logical Link Control Protocol*). För att upprätta och hantera en länk mellan två aktiva NFC-enheter används LLCP (NFC Forum, 2011a). Protokollet hanterar också dataöverföringen mellan enheterna och ser till att den görs utan förluster. LLCP erbjuder dessa tjänster till ovanliggande protokoll. Likt övriga protokoll i stacken har LLCP en tillhörande PDU och denna benämns ram.

- **RF-protokoll**. Den fysiska överföringen av bitarna hanteras av RF-protokollen där bland annat hur och i vilka hastigheter data kan skickas specificeras.

[2] Kommunikation kan endast ske i en riktning i taget



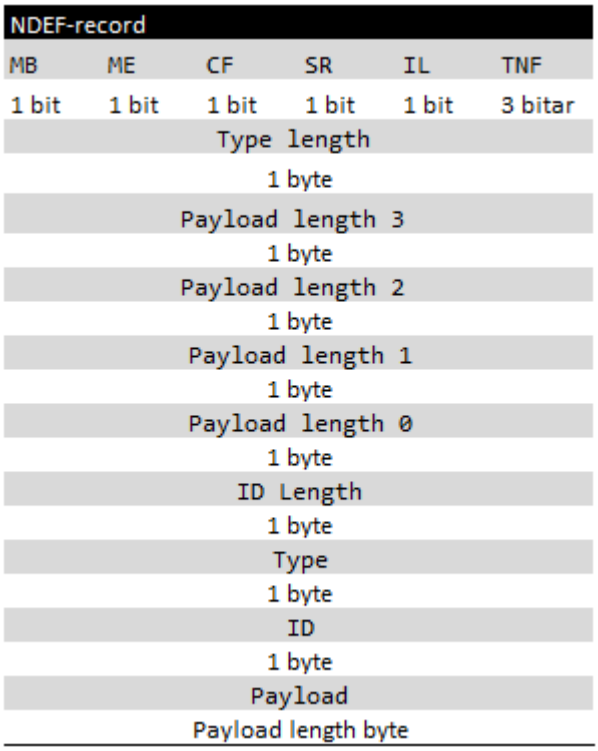

Figur 1: Här visas NFC-stackens protokoll i förhållande till vilket lager i OSI-modellen de utgör. Delvis omarbetad från Wikpedia (2013)

**3.1.1.1 NDEF** NDEF (*NFC Data Exchange Format*) (NFC Forum, 2006) är utformat för att kapsla in applikationsdata på ett effektivt sätt med minimal overhead. NDEF-meddelanden byggs upp av ett eller flera `records` enligt figur 2, där ett `record` innehåller applikationsdata från en applikation. NDEF garanterar inte att leverans av data är tillförlitlig och specificerar således inte vad målenheten skall göra vid mottagandet av ett felaktigt utformat NDEF-meddelande. Det lämnas till brukande applikationer att komplettera med datagarantier och felhantering vilka gemensamt brukar kallas QOS (*Quality Of Service*).

| NDEF-message | | | | | | |
|---|---|---|---|---|---|---|
| Record | R1 | R2 | ... | ... | ... | Rn |
| MB | 1 | 0 | 0 | 0 | 0 | 0 |
| ME | 0 | 0 | 0 | 0 | 0 | 1 |

Figur 2: Ett NDEF-meddelande består av en eller flera `records`.

Varje `record` utformas efter en fördefinierad struktur, vilken är illustrerad i figur 3, och består av ett antal fält. Punktlistan nedan beskriver mer utförligt vad varje fält i en `record` har för syfte.




Figur 3: Utformningen av ett `record`.

- **Header.** En `record header` är en byte stor där varje bit har olika betydelser.
  - **MB** (*message begin*). Denna bit indikerar att detta är den första `record` i NDEF-meddelandet. Denna bit är ettställd i första `record`, nollställd i övriga.
  - **ME** (*message end*). Denna bit indikerar att detta är den sista `record` i NDEF-meddelandet. Denna bit är endast ettställd i sista `record`, nollställd i övriga.
  - **CF** (*chunk flag*). Om CF är ettställd indikerar denna att NDEF-meddelandet är `chunked`. `Chunk`-mekanismen tillåter att ett stort meddelande delas upp över flertalet `records`. Denna mekanism kommer inte att behandlas i denna text då den utvecklade mjukvaran inte nyttjar detta.
  - **SR** (*short record*). Om SR är ettställd indikerar denna att `payload length` utgörs av en byte istället för fyra. Detta används då en datamängd på mindre än 256 byte skall överföras.
  - **IL** (*ID length*). Om IL är nollställd indikerar denna att `ID length`-fältet inte existerar i `record`. Däremot om IL är ettställd existerar `ID length`-fältet.
  - **TNF** (*type name format*). Dessa tre bitar definierar strukturen av `type`-fältet i `record`. Flertalet värden på TNF fältet finns, men det värde vilket är relevant för den utvecklade mjukvaran är `0x02` vilket står för `Media-type`. `Media-type` inrymmer allt från enkel text till bilder och video.
- **Type length.** Definierar antalet byte vilket utgör `Type`-fältet.

- **Payload length 0-3.** Varje delfält i `payload` är en byte stort vilket ger `payload length`-fältet en total storlek på fyra byte. `Payload length` definierar storleken på `payload`-fältet vilket innehåller applikationsdata. I en `record` med SR nollställd kan `payload`-fältet vara $8 * 2^8$ byte stort, det vill säga $4,29$ gigabyte. Med SR ettställd kvarstår enbart ett `payload length`-fält vilket ger `payload`-fältet en maximal storlek på 255 byte.

- **Type.** Detta fält definierar vilken typ av data `payload`-fältet innehåller. Värdet på detta fält måste följa de krav vad gäller struktur och kodning vilka specificerats av TNF-fältet i `header`. `Type`-fältet används för att ange vilken applikation meddelandets data tillhör, men om detta inte behöver specificeras kan typvärdet *text/plain* anges vilket står för att innehållet i `payload` är ASCII-formaterad text.

- **ID.** Värdet på detta fält är unikt och utgör en unik identifierare kallad URI (*Uniform Resource Identifier*). `Records` kan således skiljas åt genom att studera värdet av URI, och NDEF garanterar att detta nummer är unikt genom att tillhandahålla en generator.

- **Payload.** Detta fält innehåller applikationsdata.

#### 3.1.1.2 SNEP

SNEP (*Simple NDEF Exchange Protocol*) (NFC Forum, 2011b) är det protokoll i NFC-stacken vilken hanterar utbytet av NDEF-meddelanden. Det är ett så kallat förfrågan/respons protokoll (*request/response*). En klient skickar en SNEP-förfrågan till en server vilken, beroende på förfrågans innehåll, returnerar ett passande svar om det erfordras. Alla enheter vilka implementerar NFC-stacken har en SNEP-process körandes vilken kan agera både klient och server och erbjuder datautbytestjänster till ovanliggande applikationer. SNEP-lagret implementerar en fragmenteringsmekanism vilken används då underliggande lager i NFC-stacken kräver att en SNEP-respons eller SNEP-förfråga fragmenteras i och med att bara en mindre mängd data kan skickas i taget.

**Förfråga** SNEP specificerar förfrågans utseende enligt figur 4. En SNEP-förfråga är en generell benämning på de PDU:er vilken SNEP-klienten skickar. Begreppet förfrågan behöver således inte syfta på innebörden av PDU, utan snarare på vilken av parterna som agerar klient.

| SNEP-förfråga | | | |
|---|---|---|---|
| Version | Request | Length | Information |
| 1 byte | 1 byte | 4 byte | n byte |

Figur 4: Utformningen av en SNEP-förfråga.

- **Version.** Detta fält definierar vilken SNEP-version förfrågan bygger på. Versionsfältet är indelat i två delfält kallat `Major` och `Minor` om fyra bitar vardera, där `Major` innehåller heltalet av versionsnumret och `Minor` innehåller decimaldelen av versionsnumret. För att genomföra en SNEP-förfråga måste klient och server förstå varandra och det avgörs av SNEP-versionen. Vid mottagande av en SNEP-förfrågan kontrollerar servern versionsnumret. Stämmer numret till fullo överens med den version servern använder kan SNEP-sessionen fullbordas. Stämmer `Major` delen men inte `Minor` är det möjligt att

slutföra sessionen om servern anpassar sig efter klienten, men skiljer sig `Major` delen kan sessionen inte genomföras och servern ger avslag på den mottagna förfrågan.

- **Request.** Detta fält är åtta bitar stort och anger typen av SNEP-förfråga. Olika SNEP-förfrågor erfordrar olika svar från servern.
- **Length.** Detta fält anger hur många byte vilka utgör informationsfältet. `Length`-fältet är fyra byte långt vilket resulterar i att informationsfältet kan innehålla $4,29$ gigabyte data.
- **Information.** Innehållet i detta fält beror på innehållet i `Request`-fältet, det vill säga vilken sorts förfråga det är. Vanligtvis förekommer ett NDEF-meddelande i informationsfältet.

I figur 5 visas alla SNEP-förfrågor och i följande punktlista beskrivs de mer utförligt.

| SNEP-förfrågor | | |
|---|---|---|
| Kod | Namn | Beskrivning |
| 0x00 | Continue | Skicka kvarvarande fragment |
| 0x01 | Get | Returnera ett NDEF-meddelande |
| 0x02 | Put | Ta emot ett NDEF-meddelande |
| 0x03-7E | | Reserverat för framtida bruk |
| 0x7F | Reject | Skicka inte kvarvarande fragment |
| 0x80-0xFF | | Reserverade värden |

Figur 5: En figur över samtliga SNEP-förfrågor.

- **Continue.** `Confinue`-förfrågan skall enbart ges då mottagen respons från servern är en `Get`-förfrågan vilken är fragmenterad. Svaret indikerar att klienten har förmåga och önskar ta emot kvarstående fragment. I en `Continue`-förfråga skall inte något `Information`-fält närvara.
- **Get.** En `Get`-förfråga har kod `0x01` och utformas enligt figur 6. Förfrågan innebär att klienten ber servern att returnera information inkapslat i ett NDEF-meddelande. Vilken information klienten önskar specificeras i NDEF-meddelandet vilket finns i `Get`-förfrågans `Information`-fält. Dock utgör de fyra första byten i `Information`-fältet fältet `Acceptable Length` vilket anger maxstorleken av NDEF-meddelande i serverns respons.

| SNEP Get-förfråga | | | | |
|---|---|---|---|---|
| SNEP-Header | | | Information | |
| Version | Get | Length | Acceptable Length | NDEF-Message |
| 1 byte | 1 byte | 4 byte | 4 byte | n byte |

Figur 6: Utformning av en `Get`-förfråga.

- **Put.** En `Put`-förfråga har kod `0x02` och utformas enligt figur 7. Förfrågan innebär att klienten ber servern att ta emot det inkapslade NDEF meddelandet.

| SNEP Put-förfråga | | | |
|---|---|---|---|
| SNEP-Header | | | Information |
| Version | Put | Length | |
| 1 byte | 1 byte | 4 byte | n byte |

Figur 7: Utformning av en Put-förfråga.

- **Reject.** `Reject`-förfrågan ges då klienten inte har möjlighet att ta emot kvarstående fragment. I en `Reject`-förfråga skall inte något `Information`-fält närvara.

En förfrågan kan skickas i sin helhet eller i fragmenterad form. Sändaren delar upp meddelandet i fragment vars storlek kan väljas helt fritt dock med ett undantag. Det första fragmentet måste innehålla hela förfrågans `header`. Mottagaren kan genom att inspektera `Length`-fältet i fragmentet konstatera att förfrågan vilken togs emot var kortare än det indikerade och kan då skicka en förfråga till sändaren att skicka resterande fragment.

**Respons** SNEP specificerar utformningen av en SNEP-respons enligt figur 8. En SNEP-respons är en generell benämning på den PDU vilken SNEP-servern skickar. Begreppet respons behöver således inte syfta på innebörden av PDU, utan snarare på vilken av parterna som agerar server. Fälten i en respons fyller samma funktion som i en förfråga med undantag för andra byten vilket i en respons kallas `Response` istället för en förfrågas `Request`. `Response` innehåller en av de koder för responser vilka visas i figur 9.

| SNEP-respons | | | |
|---|---|---|---|
| Version | Response | Length | Information |
| 1 byte | 1 byte | 4 byte | n byte |

Figur 8: Utformning av ett SNEP-svar.

I figur 9 visas samtliga SNEP-responser. De vilka anses relevanta för projektet beskrivs utförligare i följande punktlista.

| SNEP-responser | | |
|---|---|---|
| Kod | Namn | Beskrivning |
| 0x00-7F | | Reserverat till förfrågor |
| 0x80 | Continue | Skicka kvarstående fragment |
| 0x02 | Success | Lyckad sändning |
| 0x82-BF | | Reserverat för framtida bruk |
| 0xC0 | Not Found | Resurs otillgänlig |
| 0xC1 | Excess Data | Resurs för stor för att skickas |
| 0xC2 | Bad Request | Felutformad förfrågan |
| 0xC3-DF | | Reserverat för framtida bruk |
| 0xE0 | Not Implemented | Sökt funktionalitet finns inte |
| 0xE1 | Unsupported Version | Protokollversion stödjs ej |
| 0xE2-FE | | Reserverat för framtida bruk |
| 0xFF | Reject | Skicka inte kvarstående fragment |

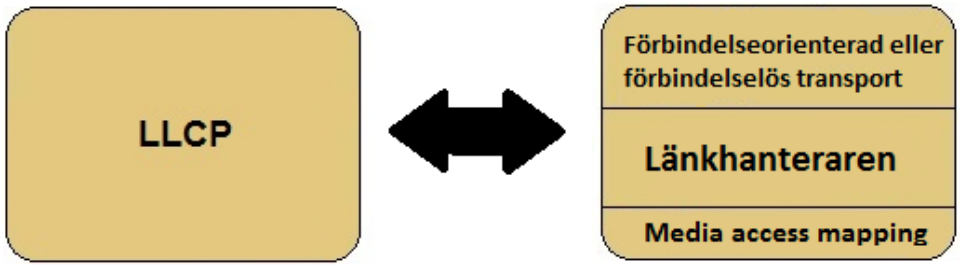
Figur 9: Värden på Response-fältet samt dess betydelse.

- **Success.** Detta svar indikerar att servern klarat av att genomföra klientens `Get` eller `Put` förfrågan. Utformningen på `Success`-svaret skiljer sig beroende på om det ges som svar på en `Get` eller `Put` förfrågan. I fallet `Get` följer svaret den generella utformningen med `Response` satt till `0x81` vilket indikerar `success`. I `Information`-fältet förekommer NDEF-meddelandet vilket ges som svar på `Get`-förfrågan. I fallet `Put` sätts response till `0x81`, men `Information`-fältet närvarar ej.
- **Continue.** Svar på en SNEP-`request` då längden angiven i `Length`-fältet är större än det meddelande vilket tagits emot. Förfrågan antags då vara fragmenterad.
- **Bad request.** Ges om förfrågan inte följer fördefinierat utseende. Svarskoden är `0xC2` och inget `Information`-fält förekommer.
- **Unsupported version.** Ges då klientens SNEP version inte stämmer överens med vad servern kan hantera. `Major`-delen av `version`-fältet skiljer sig åt alternativt klarar servern inte av att överbygga de skillnader vilken olika `Minor`-värden medför. Inget `Information`-fält förekommer.

En respons kan skickas i sin helhet eller fragmenterad. Första fragmentet måste vara tillräckligt stort för att rymma hela `respons-headern`. Vid mottagande av en fragmenterad respons kan mottagande part avgöra om den skall vänta fler fragment genom att studera värdet i responsens `Length`-fält.

#### 3.1.1.3 LLCP

LLCP (*Logical Link Control Protocol*) (NFC Forum, 2011a) är protokollet vilket sköter samtliga datalänkar med andra NFC-enheter. För att öka överskådligheten hos protokollet har det delats upp i fyra centrala delar: `Media access mappning`, länkhanteraren, förbindelselös transport och förbindelseorienterad transport. På varje telefon vilken stödjer NFC körs en LLC-process vilken implementerar LLCP. Processen tillhandahåller ovan nämnda transporttjänster till ovanliggande lager. LLCP-lagret ser med dessa delar ut enligt figur 10.




Figur 10: En mer detaljerad bild utav LLCP-lagret.

**Media access mapping** Knyter samman ovanliggande protokoll med de underliggande RF-protokoll (se avsnitt 3.1.1.4), vilka bland annat specificerar tekniken vilken står för den faktiska överföringen av data mellan två NFC-enheter.

**Länkhanteraren** Detta är den del av LLCP vilken serialiserar all kommunikation till och från NFC-enheter. Länkhanteraren kan hantera flera parallella datalänkar och sköter övervakningen av deras tillstånd genom att genomföra ett så kallat symmetriförfarande. Eftersom NFC är halv-duplex måste länkhanteraren hålla reda på vilken enhet som har rätten att sända. Under ett normalt utbyte åstadkommes en balans i kommunikationen genom att parterna sänder en LLCP-ram var. Om endera part har rätten att sända, men inte har något att sända, genomför länkhanteraren symmetriförfarandet genom att skicka en speciell LLCP-ram (SYMM) för att lämna över rätten att sända. Symmetriförfarandet gör det möjligt att upptäcka länkförluster vilka uppkommer av att till exempel kommunikationsavståndet blivit för stort. Om en part i kommunikationen förs utom räckhåll upptäcks detta genom att all form av svar uteblir.

**LLCP:s transporttjänster** LLCP erbjuder två transporttjänster till det ovanliggande SNEP-lagret (se avsnitt 3.1.1.2), förbindelseorienterad transport och förbindelselös transport. Förbindelseorienterad transport riktar sig till applikationer vilka kräver garanterad leverans av data medan förbindelselös transport riktar sig till applikationer vilka kräver snabb leverans utan garantier.

För att unikt kunna identifiera de tjänster i ovanliggande lager vilka brukar LLCP använder LLC en SAP (*Service Access Point*). SAP kan liknas vid en brevlåda i vilken brukande tjänster lämnar meddelanden (data) vilka de önskar att överföra till en tjänst körande på målenheten. Till varje SAP hör exakt en tjänst.

- **Förbindelselös transport.** Förbindelselös transport skickar onumrerade ramar och ger inga garantier till brukande tjänst att data tas emot. Mindre kontroll av datatrafiken leder till minskad overhead och därigenom till snabbare transport. Denna typ av transport lämpar sig således för applikationer vilka lägger stor vikt på snabbhet och inte kräver några transportgarantier. Om en applikation kräver snabbhet och säkerhet lämnas det till denna att implementera säkerheten själv. En datalänk av denna sort kallas för logisk datalänk och karaktäriseras enbart av SSAP (*Source Service Access Point*), SAP för sändande tjänst, och DSAP (*Destination Service Access Point*) SAP för den mottagande tjänsten.

- **Förbindelseorienterad transport.** Förbindelseorienterad transport skickar numrerade ramar och garanterar brukande tjänster säker leverans av data, men till en kostnad av ökad overhead. Förbindelseorienterad transport lämpar sig för applikationer vilka ställer högre krav på säkerhet än snabbhet. Två för NFC centrala tjänster, SNEP samt NPP, använder denna transportmetod (NFC Forum, 2013b).

  En datalänk av denna typ kallas för uppkoppling. En uppkoppling karaktäriseras av SSAP och DSAP men också ett virtuellt tillstånd vilket bestäms av fyra så kallade tillståndsvariabler vars olika värden definierar uppkopplingens tillstånd. Samtliga tillståndsvariabler kan anta ett värde mellan 0 och 15. Vilket tillstånd som uppkopplingen är i bestämmer hur LLC skall agera vid sändande och mottagande av ramar. Variablernas betydelse beskrivs i punktlistan nedan:

  - **VS** (*send state variable*). VS står för nästa numrerade ram vilken lokal LLC skall skicka över aktuell uppkoppling.
  - **VR** (*recieve state variable*). VR indikerar vilket nummer vilken nästa numrerad ram LLC väntar på att ta emot på aktuell uppkoppling.
  - **VSA** (*send acknowledgement*). VSA anger senast korrekt mottagna numrerade ram vilken lokal LLC mottagit på aktuell uppkoppling.
  - **VRA** (*recieve acknowledgment state variable*). VRA står för senast sända ram vilken LLC skickat över aktuell uppkoppling.

  Utöver dessa variabler håller också LLC reda på lokalt mottagningsfönster RWL (*Local Recieve window size*) och mottagningsfönstret i målenheten RWR (*remove recieve window*) vilka båda står för hur många numrerade ramar vilka får skickas mellan enheterna utan att VSA uppdateras. Anledningen till att VSA inte uppdateras efter varje sändning är att *overheaden* på uppkopplingen minskas.

  Nedan följer en kort redogörelse för hur tillståndsvariablerna dikterar hur LLC skall agera vid sändning och mottagning samt hur detta resulterar i säker överföring:

  - **Sändning av numrerad ram.** När LLC önskar att sända över aktuell uppkoppling kontrolleras att VS är lika med VSA + RWR. Detta villkor måste alltid vara uppfyllt och utgör en så kallad invariant. Invarianten syftar till försäkra att alla föregående ramar tagits emot och att mottagande LLC är redo för försändelsen. Först när detta villkor är uppfyllt tillåts den lokala LLC-processen att skicka en numrerad ram över aktuell uppkoppling. I ramen placeras nuvarande VS och efter sändning ökas VS med ett.
  - **Mottagning av numrerad ram.** När den lokala LLC-processen mottager en numrerad ram vilken innehåller ett VS-värde som är lika med lokalt VR, packas ramens innehåll upp och ges till den brukande tjänsten i ovanliggande lager. Efter detta genomför LLC ett ACK-förfarande. Om dessa värden skiljer sig åt har ett fel uppstått och en felbeskrivande ram (FRMR-ram) skickas till motpartens LLC.
  - **ACK-förfarande.** Detta förfarande kan utföras på två olika sätt. Har den lokala LLC-processen möjligheten att skicka en informationsbärande ram utförs bekräftelsen med hjälp av den. Har den lokala LLC-processen inte möjligheten att

skicka en numrerad ram används en dedikerad ACK-ram (RR-ram). Vid båda dessa alternativ placeras aktuell VR i ramen och VRA sätts till detta VR.

Vid mottagande av en numrerad ram eller en dedikerad ACK-ram, kontrolleras det VR vilket följde med ramen. Eftersom VR indikerar vilken ram LLC på sändande enhet väntar att ta emot, tolkas detta som en implicit bekräftelse på att alla VR-1 ramar tagits emot korrekt. VR värdet vilket följde med ramen blir det nya värdet på VSA.

**TLV-parametrar** Eftersom NFC körs på flertalet olika enheter vilka skiljer sig åt vad gäller mjuk- och hårdvara måste LLCP vara flexibelt nog för att kunna köras oberoende av plattform. Denna flexibilitet erhålles genom att specifika parametrar, kallade TLV-parametrar, för en datakanal kan förhandlas fram mellan de två kommunicerande LLC-processerna. De viktigaste TLV-parametrarna för detta projekt är MIUX, LTO och VERSION.

- **MIUX** (*Maximum Information Unit extension*) Denna parameter indikerar hur stor en ram maximalt får vara vilket skall skickas över uppkopplingen. Detta tillåter att enheter med begränsade databuffertar kan bruka LLCP. Om denna parameter utelämnas i förhandlingen antar MIU (*Maximum Information Unit*) ett standardvärde på 128 byte. Om parametern sätts följer att MIU initieras enligt: MIU = 128 + MIUX.
- **LTO** (*Link Time Out*). Parametern indikerar hur länge en uppkoppling kan vara inaktiv innan den betraktas som förbrukad och stängs ner. Genom att anpassa LTO kan enheter med låg beräkningskapacitet använda LLCP och kommunicera via NFC, vilket är fallet för en ATMEL-baserad mikrokontroller (Atmel Corporation, 2013).
- **VERSION** (*Version Number*). Denna parameter innehåller versionsnummer för den implementerade versionen av LLCP och är uppdelat i två delfält kallade `Major` och `Minor`, där `Major` innehåller heltalet av versionsnumret och `Minor` innehåller decimaldelen av versionsnumret. För att upprätta en länk måste båda parternas `Major`-del överensstämma. Huruvida en länk tillåts att upprättas även om `Minor`-del inte överensstämmer är upp till specifik implementation.

De nämnda parametrarna MIUX och LTO kan antingen utbytas efter att en uppkoppling etablerats eller så utbyts alla tre nämnda parametrar under RF-protokollens initieringsförfarande (se avsnitt 3.1.1.4) mellan två enheter. Det förstnämnda förfarandet kallat PAX (*Parameter Exchange*) används inte av prototypen och kommer inte att behandlas vidare.

**LLCP-meddelanden vid uppkopplingsorienterad transport** Trots att ramarna vid uppkopplingsorienterad transport benämns som numrerade är fallet inte alltid så. I de ramar vilka är numrerade närvarar ett `sequence`-fält och i de ramar vilka inte är numrerade förekommer inte detta fält. Detta gör att den generella utformningen på en LLCP-ram vilken visas i figur 11 anger ett `sequence`-fält på 0 eller 8 bitar. Den följande punktlistan beskriver de fält vilka utgör en LLCP-ram.

| LLCP-ram | | | | |
|---|---|---|---|---|
| | | Header | | Payload |
| DSAP | PTYPE | SSAP | Sequence | Information |
| 6 bitar | 4 bitar | 6 bitar | 0 eller 8 bitar | n byte |

Figur 11: Utformningen av ett ram till förbindelseorienterad transport.

- **DSAP** (*Destination SAP*). Detta fält administreras av mottagarens LLC och utgör adressen vilken sändaren önskar att upprätta en uppkoppling med. DSAP skall användas som SSAP(se nedan) när ett eventuellt svar ges på den mottagna ramen.
- **PTYPE** (*Payload data unit type field*). Detta fält anger vilken sorts ram det är och formen på denna. Alla ramar är inte utformade på exakt samma sätt då de är avsedda för olika ändamål. Genom att undersöka PTYPE-fältet kan mottagaren utläsa vilka fält som skall närvara och vad fältens innehåll betyder.
- **SSAP** (*Source SAP*). Anger vilken SAP ovanliggande tjänst blivit tilldelad och önskar att bruka. Skall användas som DSAP när ett eventuellt svar ges på den mottagna ramen.
- **Sequence.** Detta fält är indelat i två stycken underliggande fält. NS är här samma nummer som VS. NS innehåller således en siffra $0-15$ vilken visar vilket sekvensnummer ramen har i den lokala LLC-processen. NR är samma som VR. NR innehåller alltså en siffra $0-15$ vilken visar vilket nummer den lokala LLC-processen väntar på att ta emot. `Sequence`-fältet finns enbart med i de ramar som är numrerade. Sekvensnumret används i LLC-protokollets ACK-mekanism finns beskriven ovan.
- **Information.** Innehåller data från ovanliggande lager, t.ex. SNEP eller NPP vilka i sin tur kapslar in applikationsdata. Alternativt kan `Information`-fältet innehålla LLC specifika parametrar. Vad informationsfältet innehåller kan utläsas av PTYPE-fältet.

Nedan listas de typer av ramar LLCP erbjuder vilka anses vara relevanta för projektet. Alla de utvalda typerna av ramar illustreras i figur 12 och beskrivs mer utförligt i punktlistan nedan.

| LLCP-ramar | | | | | |
|---|---|---|---|---|---|
| | | SNEP-Header | | | Information |
| Ram | DSAP | PTYPE | SSAP | Sequence | |
| Connect | DSAP | 0100 | SSAP | - | TLV-parametrar, n byte |
| Connection complete | DSAP | 0110 | SSAP | - | TLV-parametrar, n byte |
| Symmetry | 0000 | 0000 | 0000 | - | - |
| Information | DSAP | 1100 | SSAP | NS, NR | Data, n byte |
| RR | DSAP | 1101 | SSAP | - , NR | - |
| DISC | DSAP | 0101 | SSAP | - | - |
| DM | DSAP | 0111 | SSAP | - | Reason, 1 byte |

Figur 12: Visar utformningen av relevanta ramar för projektet.

- **CONNECT** (*Connect*). När en NFC-enhet önskar etablera en uppkoppling mot en annan NFC-enhet används denna ram. DSAP sätts till en det värde som tillhör sökt

tjänst, till exempel `0x04` vilket indikerar att sändaren önskar etablera en uppkoppling mot SNEP-processen vilken kör på målenheten. SSAP sätts till det värde applikationen blivit tilldelad. Observera att `Sequence`-fältet inte existerar i en `Connect`-ram och det beror på att en `Connect`-ram är onumrerad. `Information`-fältet specificeras till att innehålla TLV parametrar, vilka kan skickas med.

- **CC** (*Connection complete*). Denna ram skall ges som svar på en `Connect`-ram om en uppkoppling kan upprättas. DSAP sätts till `Connect`-ramens SSAP värde. SSAP sätts till det fördefinierade värdet sändaren angivit i DSAP om denna är i bruk. Precis som `Connect`-ramen är denna ram onumrerad och dess `Information`-fält innehåller uppkopplingsspecifika parametrar. När denna utväxling är avklarad är uppkopplingen etablerad. De satta värdena på SSAP och DSAP kommer att användas i samtliga ramar vilka skickas på den nyetablerade uppkopplingen.

- **I** (*Information*). När uppkopplingen är etablerad kan utbytet av applikationsdata inledas. Informations-ramen är ämnad för att överföra data mellan två NFC-enheter. För att kunna upptäcka eventuell dataförlust är denna ram numrerad och har därför ett `sequence`-fält. `Information`-fältet innehåller i detta fall data från ovanliggande applikationer.

- **SYMM** (*Symmetry*). Denna ram används i den tidigare beskrivna symmetri-proceduren för att undvika länk-*timeout* under pågående kommunikation. När en NFC-enhet har exklusiv rätt att skicka data men saknar något att skicka och önskar bibehålla uppkoppling genomförs symmetri-proceduren under vilken denna ram spelar en central roll. Denna ram är inte numrerad och har således inte ett `sequence`-fält.

- **RR** (*Ready recieve*). Denna ram används för att indikera att föregående ram togs emot korrekt när ingen nyttig data kan skickas. Detta är således LLCP:s dedikerade ACK-ram. Denna ram är delvis numrerad och innehåller enbart VR.

- **DISC** (*Disconnect*). När en av parterna vill stänga ner uppkopplingen används DISC-ramen. När en LLC-process tar emot en DISC-ram kommer processen inte att skicka fler ramar över denna uppkoppling men uppkopplingen stängs ned först då en DM-ram mottagits.

- **DM** (*Disconnect mode*). Utgör svar på en `Connect`-ram när en uppkoppling inte kan etableras mellan enheterna och på en DISC-ram. Då en DM-ram har skickats stänger den sändande LLC-processen ned uppkopplingen. I `Information`-fältet anges anledningen till att DM-ramen sänts och i figur 13 visas de värden `Information`-fältet kan innehålla.

| LLCP DM Reasons | |
|---|---|
| Kod | Beskrivning |
| 0x00 | Skall indikera att LLC har tagit emot en DISC ram och uppkopplingen betraktas som förbrukad |
| 0x01 | Skall indikera att LLC har tagit emot en uppkopplings-orienterad ram men på en SAP som inte har någon aktiv uppkoppling |
| 0x02 | Skall indikera att LLC har tagit emot en Connect-ram på en SAP som inte är i bruk |
| 0x03 | Skall indikera att LLC har tagit emot en Connect-ram men uppkopplingen vägrades av ovanliggande tjänst |
| 0x10 | Skall indikera att LLC permanent inte kommer att godta uppkoppling på specifik SAP |
| 0x20 | Skall indikera att LLC temporärt kommer att vägra uppkoppling på specifik SAP |
| 0x21 | Skall indikera att LLC tämporärt kommer att vägra samtliga uppkopplingar |
| Övriga | Skall inte används av LLC vid sändandet av en DM ram. Vid mottagade tolkas värdet till 0x00 |

Figur 13: Visar de anledningar till varför en DM-ram kan sändas.

- **FRMR** (*Frame reject*). Ramen är uppbyggd enligt figur 14 och indikerar att en i sammanhanget oväntad ram, alternativt en felutformad ram, har tagits emot av LLC-processen. Fyra flaggor används för att precisera felets natur. W-flaggan påvisar att ramen är felutformad. I-flaggan indikerar att `Information`-fältet i mottagen `Information`-ram är felutformat. R- och S-flaggorna påvisar att en ram mottagits med felaktig VS och eller VR. `Sequence`-fältet innehåller mottagna tillståndsvariabler, det vill säga de felaktiga VR och VS vilka hittas i den mottagna ramen. Fälten VS, VR, VSA och VRA är samtliga tillståndsvariabler tillhörande LLC. Ramen spelar en väsentlig roll i felhanteringen vilken finns beskriven ovan.

| LLCP FRMR-ram | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Header | | | Information | | | | | | | | |
| DSAP | PTYPE | SSAP | W | I | R | S | Sequence | VS | VR | VSA | VRA |
| 6 bitar | 4 bitar | 6 bitar | 1 bit | 1 bit | 1 bit | 1 bit | 1 byte | 4 bitar | 4 bitar | 4 bitar | 4 bitar |

Figur 14: Visar hur en FRMR-ram är uppbyggd.

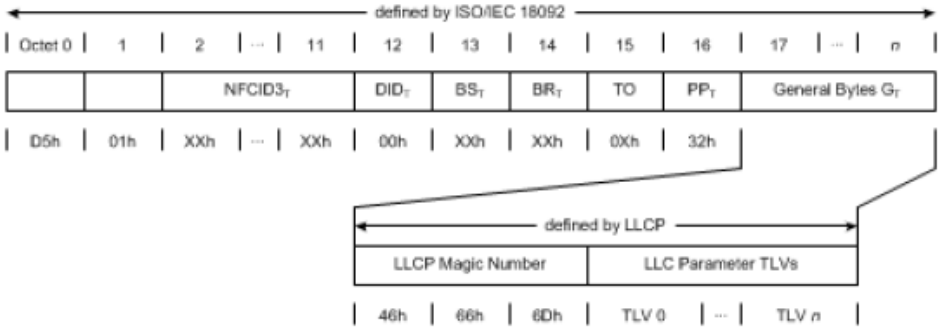
#### 3.1.1.4 RF-protokoll

Längst ner i NFC-stacken finns det tre RF-protokoll. Det första protokollet, vilket också är allra längst ner i OSI-modellen, är *NFC Analog Technical Specification* (NFC Forum, 2012a) vilket specificerar den analoga karaktäristiken för radiofrekvensgränssnittet vilken en NFC-enhet använder. Ovanpå det protokollet ligger det två andra protokoll vilka är ISO/IEC 18092 (International Organization for Standardization, 2004), också kallat NFCIP-1, och ISO/IEC 14443 (International Organization for Standardization, 2008). Dessa två protokoll hanterar kommunikation mot aktiva samt passiva NFC-enheter. Det utvalda NFC-chippet PN532 (NXP Semiconductors, 2013b) från NXP Semiconductors (2013a), vilket används i utvecklingen av prototypen, implementerar alla tre protokollen.

Protokollet NFCIP-1 används för att implementera SNEP (se avsnitt 3.1.1.2) och ISO/IEC 14443 används för att implementera NPP (se avsnitt 3.1.1) vilket medför att det förstnämnda som används i utvecklingen av prototypen. Detta protokoll specificerar bland annat moduleringstekniker, kodningsanvisningar, överföringshastigheter, datakollisionskontroll och ramformat. Vidare definierar NFCIP-1 initieringsförfarandet, datautbytesförfarandet (DEP) och inaktiveringsförfarandet vilket används vid kommunikation mot en aktiv NFC-enhet vid användning av SNEP.

**Initieringsförfarande** I initieringsförfarandet sänds en `ATR_REQ` (*Attribute Request*) från den initierande enheten till målenheten, vilken sänder tillbaka en `ATR_RES` (*Attribute Response*) som svar. En `ATR_REQ` och en `ATR_RES` är snarlikt uppbyggda och beskrivs samtidigt i följande stycke.

De två första byten i en `ATR_REQ` eller `ATR_RES` (se figur 15) är `0xD4` och `0x00`. Därefter följer NFCID3 vilket är ett slumpartat identifikationsnummer vilket identifierar den skickande enheten och ska vara detsamma under en hel kommunikations-session. Följande fem byte i en `ATR_RES` respektive fyra byte i en `ATR_REQ` (TO uteblir), är parametrar vilka sköts av NFCIP-1. Slutligen följer så kallade generella byte vilka inleds med de magiska LLCP numren och efter dem radas de TLV parametrar (se 3.1.1.3) upp som önskar utbytas i initieringsförfarandet.




Figur 15: Utförandet av en `ATR_RES`. NFC Forum (2011a).

**Datautbytesförfarande** DEP (*Data Exchange Protocol*) är det protokoll som följs vid utbytandet av data mellan två aktiva NFC enheter på RF-protokoll-nivå och är ett blockorienterat transportprotokoll med inbyggd felhantering och kedjemekanism. Denna mekanism används då datan vilken ska överföras inte får plats i en ram.

**Inaktiveringsförfarande** Efter att datautbytet ägt rum kan initiatorn påbörja ett inaktiveringsförfarande utav DEP. Detta görs genom att initiatorn antingen skickar en `deselect`- eller `release`-förfråga till målenheten vilken returnerar ett svarsmeddelande som godkänner förfrågan. `Deselect`-proceduren inaktiverar DEP och initiatorn kan sedan reaktivera DEP genom att skicka en förfråga till målenheten. `Release`-proceduren inaktiverar DEP utan möjlighet till reaktivering och enheterna försätts i ett initialt läge där de väntar på ett nytt initieringsförfarande.

#### 3.1.1.5 Ett exempel på ett teoretiskt kommunikationsförlopp

En applikation som körs på en NFC-enhet önskar att skicka data till en applikation som körs på en målenhet.

**Initieringsförfarande** Kommunikationsförloppet inleds med att de kommunicerande enheternas NFC-chip utbyter `ATR_REQ` och `ART_RES` enligt beskrivning i avsnitt 3.1.1.4. För kommunikationen utbyts relevanta TLV-parametrar och kommunikationen övergår i upprättandet av uppkopplingen.

**Uppkopplingsupprättande** För att kunna sända datan måste LLC-processen på den sändande enheten först upprätta en uppkoppling genom att skicka en `connect`-ram. För att indikera att det är en `connect`-ram sätts PTYPE till $0100_2$. `Connect`-ramens SSAP sätts till den adress vilken aktuell applikation blivit tilldelad. På samtliga NFC-enheter körs en SNEP-server. Denna server använder en välkänd SAP (`0x04`) och det är denna SAP som används som DSAP i `connect`-ramen om applikationsdatan skall utväxlas med SNEP vilket ofta är fallet.

När LLC-processen på målenheten ter emot en `connect`-ram, och DSAP och SSAP är giltiga, bearbetas de TLV-parametrar vilka eventuellt medföljde `connect`-ramen. Avsedd applikation, i detta fall SNEP-servern, meddelas om att någon önskar utbyta data. SNEP-servern kontrollerar huruvida datautbyte är möjligt. Är datautbyte inte möjligt meddelas LLC vilken då sänder en DM-ram och uppkopplingen avslutas. Är utbyte av data möjligt meddelas LLC om detta vilken då skickar en CC-ram. I denna ram sätts DSAP till `connect`-ramens SSAP och SSAP till `connect`-ramens DSAP. VS, VR, VSA och VRA sätts samtliga till noll.

Vid mottagandet av CC-ramen sätter den förstnämnda LLC-processen VS, VR, VSA och VRA till noll. Uppkopplingen är nu etablerad och kommunikationen övergår i fasen informationsutbyte.

**Informationsutbyte** Den lokala LLC-processen sänder iväg den första informationsramen vilken kapslar in applikationsdata. Eftersom det är den första informationsramen sätts sekvensnumret NS och NR till 0. Om ytterligare informationsramar skickas ökas successivt sekvensnumret VS och NS. Informationsfältet innehåller SNEP-ramen som i sin tur innehåller datan som skall överföras.

Vid mottagande av informationsramen erhålls den inpackade SNEP-ramen vilken levereras till SNEP-servern. Om applikationen på målenheten önskar svara kapslar LLC på målenheten in svaret i en informationsram med sekvensnumret NS = 0 och NR = 1. Det första indikerar att detta är den första informationsramen vilken målenheten skickar till sändarenheten, det senare indikerar att föregående informationsram tagits emot korrekt. Om inget svar ges skickar LLC på målenheten en RR-ram med sekvensnummer NR = 1 vilken fyller samma funktion som informationsramen men utan medföljande data.

Datautbytet fortlöper med successivt ökande tillståndsvariabler tills antingen ett fel uppstår eller tills utbytet är färdigt. Båda resulterar i att kommunikationen övergår i fasen för uppkopplingsavslutning.

**Uppkopplingsavslutning** Applikationen signalerar den lokala LLC-processen om att utbytet av data är färdigt. Den lokala LLC-processen skickar då en DISC-ram till LLC på målenheten och den lokala LLC-processen väntar på mottagandet av en DM-ram. Då

målenheten mottager DISC-ramen ser målenheten uppkopplingen som avslutad och LLC meddelar SNEP-servern att uppkopplingen stängs ned och skickar en sista DM-ram över uppkopplingen. När den lokala LLC-processen mottager denna DM-ram slutar även målenhetens LLC att ta emot ramar och dess SNEP server meddelas att uppkopplingen är avslutad.

## 3.2 Säkerhet

NFC har likt RFID i sitt grundutförande inga inbyggda säkerhetsfunktioner vilka förhindrar avlyssning, datamodifiering eller attacker såsom en *man-in-the-middle attack*. Det är därmed upp till sändare och mottagare att applicera säkerhet om behovet finns. Följande avsnitt beskriver hur dessa säkerhetsbrister kan åtgärdas och baseras till stor del på artikeln *Strengths and Weaknesses of Near Field Communication (NFC) Technology* skriven av Mohamed Mostafa Abd Allah (2011). Dessutom följer en genomgång av kryptering, då det är det enklaste sättet att skydda sig mot de nämnda hoten, teorin i krypteringsavsnittet bygger på teori från boken *RSA and Public-Key Cryptography* skriven av Mollin (2002).

### 3.2.1 Avlyssning

Eftersom NFC är en trådlös kommunikationsmetod där ett magnetfält är informationsbärande kan icke avsedda enheter läsa av fältet. Tack vare NFC:s korta kommunikationsavstånd på omkring 10 centimeter besitter NFC ett inbyggt hinder mot avlyssning. Avlyssningsutrustningen måste föras in i de kommunicerande enheternas magnetfält vilket innebär att avlyssningsutrustning måste placeras nära enheterna vilket underlättar upptäckten av utrustningen. Det finns dock ingen garanti på att informationen kan fångas upp på betydlig längre avstånd eftersom flertalet parametrar påverkar hur långt radiosignalerna kan färdas. Till exempel har ett experiment där kommunikation försökte avlyssnas på avstånd uppåt tio meter genomförts med framgång (Allah, 2011). De parametrar vilka påverkar avlyssningsavståndet mest beskrivs i varsitt kommande stycke.

**Antenn** Vilken antenn sändare och mottagare använder spelar en avgörande roll vid avlyssning. Dess dimensioner, utformning och omslutande material är några viktiga faktorer som påverkar hur väl en antenn kan sända och ta emot radiovågor (Allah, 2011). I telefoner används en relativt liten spole vilket starkt begränsar signalstyrkan. Vidare är mobiltelefoner i dagsläget ofta inneslutna i robust plast eller aluminium som också minskar signalstyrkan.

**Effekt** Effekten vilken signalen sänds med avgör till stor del hur långt den kan färdas. En signals effekt är starkt sammankopplad med bärvågens amplitud och större amplitud i bärvågen medför att signalen har en större effekt. En signal vilken sänds med hög effekt och följaktligen stor amplitud kan färdas längre än en signal vilken sänds med låg effekt och liten amplitud.

**Moduleringsteknik** Moduleringstekniken, enkelt uttryckt hur informationen mönstras in i bärvågen, är också avgörande i sammanhanget. Olika moduleringstekniker medför olika mönster i bärvågen. Hur kraftig modulering som används uttrycks i procent och symboliserar hur tydligt mönstret vilket utgör informationen blir. NFC bygger på en variant av amplitudmodulering kallad ASK (*Amplitude Shifting Keying*) vilket innebär att informationen

mönstras in via amplitudvariationer av signalen (Allah, 2011). I detta sammanhang betyder kraftig modulering stora amplitudvariationer hos bärvågen som informationen representeras av. Kraftig modulering uppåt 100 procent innebär att bärvågen stundom helt släcks ut och svagare modulering kring tio procent innebär mindre variationer i amplitud hos bärvågen. Det följer att olika moduleringstekniker med olika kraftfull modulering resulterar i signaler med olika egenskaper varav en är hur långt signalen effektivt kan bära information. Vid kraftig modulering kan signalen färdas längre än vid svag modulering. Kraftig modulering är således känsligare för avlyssning.

**Omgivning** Allah (2011) beskriver att omgivning spelar en avgörande roll i hur långt radiosignaler kan färdas. Till exempel finns i en byggnad väggar och andra föremål vilket kraftigt begränsar signalens förmåga att breda ut sig. Bakgrundsbrus, andra radiovågor från mobiltelefoner, radioapparater och övrig elektronik stör radiosignalen och minskar dess räckvidd.

### 3.2.2 Datamodifiering

Enligt Allah (2011) är datamodifiering när en attackerande enhet försöker störa kommunikationen mellan två enheter genom att modifiera den information vilken skickas mellan enheterna. Den enklaste formen av datamodifiering är att störa ut de informationsbärande signalerna. Mer avancerad datamodifiering innebär att modifiera signaler, det vill säga den informationen vilken utbyts, på ett sådant sätt att informationen fortfarande uppfattas som korrekt av mottagaren, men nu främjar angriparens syfte. Hur stor sannolikheten är att genomföra en lyckad datamodifieringsattack bestäms till stor del av den använda kodnings- och moduleringstekniken.

Moduleringstekniken bestämmer hur information mönstras in i bärvågen och kodningstekniken avgör hur varje bit representeras. Om amplitudmodulering används, vilket är fallet vid NFC-kommunikation, bestämmer kodningstekniken om en logisk etta och nolla representeras av en hög respektive låg amplitudnivå. *Manchesterkodning* används för de högre överföringshastigheterna, det vill säga 212 kbit/s och 424 kbit/s. *Modifierad Millerkodning* används för den lägre överföringshastigheten på 106 kbit/s (Allah, 2011).

Allah (2011) hävdar att om kraftig modulering används blir det praktiskt taget omöjligt att lyckas med en datamodifieringsattack. Om svag modulering används, nedåt 10 procent, blir en datamodifieringsattack teoretiskt sett möjlig att genomföra om *Manchesterkodning* används. Vid *Modifierad Millerkodning* är modifiering svårare och begränsas enbart till speciella variationer av bitföljder. Allah fastställer att för att ha högsta möjliga skydd mot en datamodifieringsattacker skall en kraftig modulering kombineras med *Modifierad Millerkodning*. Denna kombination är dock den som är känsligast mot avlyssning då kraftig modulering används.

### 3.2.3 Replay attack

Ett potentiellt hot är *replay attack* vilket beskrivs mer utförligt av Mollin (2002). Krypterade meddelanden kan kopieras av en illasinnad avlyssnare och utan att avlyssnaren kan dekryptera meddelandet skickas det till den angivna mottagaren som då uppfattar det korrekt. För att undvika replay attacker krävs att varje krypterat meddelande är unikt. Detta brukar

lösas vilket Mollin (2002) beskriver med till exempel tidsstämplar eller andra unika taggar i meddelandet för att verifiera att mottagna meddelanden är korrekta.

### 3.2.4 Man-in-the-middle attack

En så kallad *man-in-the-middle* attack går ut på att en illasinnad enhet agerar mellanhand mellan sändare och mottagare (Allah, 2011). Sändare och mottagare handlar i god tro om att de talar direkt till varandra. Mellanhanden får tillgång till känslig information och har dessutom möjligheten att ändra innehållet i datan.

En attack av detta slag är i praktiken omöjligt att genomföra på en NFC-länk oavsett om kommunikationen sker mellan två aktiva enheter eller en aktiv och en passiv enhet (Allah, 2011). För att kunna imitera sändaren måste mellanhanden aktivt störa ut informationen vilken parterna försöker utbyta eller försöka skapa en signal som perfekt överlappar befintlig signal. Minsta avvikelse i utstörningen eller överlappningen resulterar i att datan vilken skickas över länken blir förvrängd och om de två enheterna aktivt lyssnar på länken kan denna förvrängning upptäckas.

### 3.2.5 Kryptering

Kryptering är ett brett område och teorin vilken behandlas i detta avsnitt är främst teori vilken kan nyttjas för att applicera säkerhet vid NFC-kommunikation. Allah (2011) ser kryptering som lösningen på samtliga av de attacker vilka behandlats i tidigare avsnitt. Han betonar etableringen av en säker datakanal mellan de kommunicerande enheterna under vilken asymmetrisk kryptering och symmetrisk kryptering spelar en avgörande roll. Begreppet säker datakanal syftar i projektets fall på NFC-kommunikation där känslig data krypteras för att kunna undvika ovan beskrivna säkerhetshot. Mer specifikt behandlar nedanstående avsnitt asymmetrisk kryptering [3], först genomgång av hur krypteringen går till praktiskt för att senare gå in djupare på teori bakom användbara algoritmer och kryptosystem för projektet. I detta avsnitt tas även symmetrisk kryptering upp för jämförelse mot asymmetrisk kryptering. Hot mot kryptosystem kommer också att presenteras eftersom dessa ses som viktiga att kunna förstå och undvika. Hela avsnittet bygger på teori från boken *RSA and Public-Key Cryptography* av Mollin (2002).

För att säkert kunna skicka meddelanden innehållande känslig information kan asymmetrisk kryptering (Mollin, 2002) användas. Med denna metod används en publik krypteringsnyckel för att kryptera meddelanden och en privat krypteringsnyckel för att dekryptera meddelanden. Den publika krypteringsnyckel är inte hemlig men den kan inte heller användas för att dekryptera meddelanden. Den privata krypteringsnyckel är däremot hemlig och det är viktigt att den inte kommer i en angripares händer. Det går att dra parallellen mellan krypteringsmetoden och en låst brevlåda. Vem som helst kan lägga post i brevlådan (skicka meddelanden till låsenheten) men bara brevbäraren (låsenheten) med tillgång till nyckeln för att låsa upp brevlådan kan läsa breven.

För att kunna nyttja asymmetrisk kryptering i praktiken finns kryptosystemet *RSA* (Mollin, 2002) att tillgå. Det finns även andra kryptosystem för asymmetrisk kryptering till exempel *ElGamal*[4] (Mollin, 2002), dock finns det betydligt fler färdiga källkodsbibliotek imple-

[3] Från engelskans *Public-key encryption*
[4] Döpt efter skaparen Taher ElGamal

menterade med RSA än ElGamal att använda för till exempel C/C++.

Mollin (2002) skriver att för att förstå varför asymmetrisk kryptering anses vara säker, med tillräckligt stora krypteringsnycklar, krävs en förståelse av komplex beräkningsteori och så kallade *One-way functions*. Vidare skriver Mollin att *One-way functions* är funktioner som är lättaätt beräkna åt ena hållet men beräkningsmässigt omöjliga att beräkna inversen av. Så kallade *Trapdoor One-way functions* fungerar på samma sätt som *One-way functions* men med tillgång till en så kallad *Trapdoor* blir inversen billigätt beräkna. En *Trapdoor* är alltså information som krävs för att kunna beräkna inversen, och det är ovanstående teori som asymmetrisk kryptering bygger på. Den privata krypteringsnyckeln fungerar alltså som *Trapdoor* (Mollin, 2002).

Nedan listas vilka fördelar och nackdelar Mollin (2002) anser att det finns med asymmetrisk kryptering. En annan välanvänd krypteringsmetod kommer tas upp för jämförelse; symmetrisk kryptering. Symmetrisk kryptering bygger i korthet på att både sändare och mottagare har tillgång till en likadan hemlig krypteringsnyckel och med denna nyckel går det att både kryptera och dekryptera meddelanden.

Fördelar med asymmetrisk kryptering:

- Säkert, endast den privata krypteringsnyckeln behöver hållas hemlig (Mollin, 2002). Den privata nyckeln skickas aldrig över kanaler och ges inte ut till sändare av krypterad data.
- Hög livslängd på krypteringsnycklar leder till att administrationen blir enkel eftersom den privata krypteringsnyckeln aldrig delas ut. Detta leder till att nycklarna inte behöver bytas lika frekvent (Mollin, 2002).

Nackdelar med asymmetrisk kryptering:

- Långa krypteringsnycklar, 1024 bitar rekommenderas, jämfört med symmetrisk kryptering där det räcker med 128 bitar (Mollin, 2002).
- Beräkningstungt, långsammare än symmetrisk kryptering. RSA är cirka 1000 gånger långsammare än Data Encryption Standard (DES) som är ett vanligt kryptosystem för symmetrisk kryptering (Mollin, 2002).

Mollin (2002) skriver att det finns två stora hot mot själva kryptosystemet. Dels att en angripare kommer över den privata krypteringsnyckeln och dels attacker riktade mot kryptosystemet genom att ta reda på *klartext*[5] utan att känna till den privata nyckeln (kryptoanalys). För att en angripare inte ska komma över den privata kryperingsnyckeln krävs att den hålls hemlig och det ses inte som en svårighet då den inte delas ut till obehöriga. Vidare skriver Mollin att det finns två olika typer av attacker mot ett kryptosystem: passiva attacker och aktiva attacker. En typ av passiv attack går ut på avlyssning av ett krypterat meddelande analyseras med kryptoanalys för att försöka förstå och dekryptera meddelandet (*Ciphertext-only*). En annan typ av passiv attack går ut på att kryptera ett eget meddelande med den publika nyckeln för att försöka knäcka systemet (*Chosen-plaintext*). Aktiva attacker är svårare att genomföra och man-in-the-middle attacken beskriven i ovanstående sektion är ett exempel på attack som kan riktas även mot själva kryptosystemet. Dock är det redan påpekat att detta i praktiken är omöjligt att genomföra vid NFC-kommunikation. Aktiva attacker kommer därför inte att betraktas som ett hot mot kryptosystemet vid NFC-kommunikation.

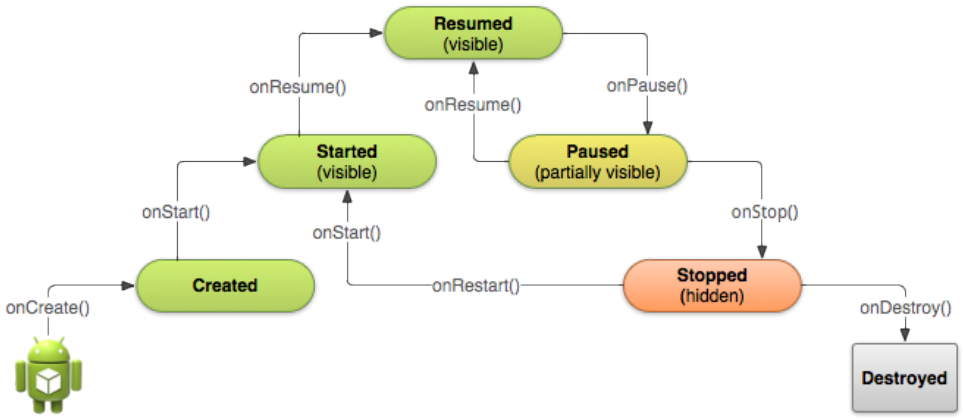
[5] Från engelskans plaintext

## 3.3 Android

Enligt Android (2013b) är Android världens mest använda operativsystem för mobila enheter. Android är baserat på Linuxkärnan och kan med hjälp av denna stödja många olika typer av hårdvara. Detta gör Android kompatibelt med enheter från många olika tillverkare och det är en av anledningarna till att Android har blivit dominerande. En annan anledning till dominansen är Androids öppna källkod och dess användande av världens mest använda programspråk, Java (Android, 2013b).

För att komma igång och skriva applikationer för Android är det några saker en utvecklare behöver känna till. Förutom grundläggande kunskaper inom Java-programmering och mjukvaruutveckling i allmänhet bör en utvecklare av Android-applikationer inneha kunskaper inom i huvudsak tre områden. Dessa tre områden beskrivs i nedanstående stycken.

Först bör en utvecklare inneha kunskaper om den särskilda livscykel (Android, 2013e), se figur 16, en Android-aktivitet har. Livscykeln beskriver de olika tillstånd en aktivitet kan befinna sig i samt de olika automatiska anrop som sker då aktiviteten tar sig mellan dem. Livscykeln ger ytterligare funktionalitet men också vissa svårigheter så som att viktiga data måste sparas och återställas när användaren till exempel vrider mobiltelefonen.




Figur 16: Bilden som är publicerad av Android (2013e) beskriver en Android-aktivitets livscykel. Särskilt beskriver den de olika tillstånd den kan befinna sig i samt de metodanrop som sker när den tar sig mellan dem.

För det andra måste en utvecklare av Android-applikationer ha kunskap om de Android-specifika API vilket Android (2013h) tillhandhåller. Dessa API beskriver de funktioner vilka är unika för Android-enheter. Där beskrivs till exempel hur ett användargränssnitt byggs upp, hur en lokal databas sätts upp och hur applikationen kommunicerar mot andra applikationer och mot omvärlden. Särskilt användbart för projektet är det API vilket specificerar kommunikation över NFC.

Vidare bör en utvecklare också känna till filstrukturen hos ett Android-projekt, de specifika verktyg för Android-utveckling som finns samt vilka filer som definierar vad. Applikationernas grafiska gränssnitt definieras till stor del av XML-dokument och därför krävs kunskap om hur dessa byggs upp. XML-dokument används även vid andra delar av uppbyggnaden av Android-applikationer.

### 3.3.1 NFC för Android

Android innehåller funktionalitet för att kommunicera via NFC (Android, 2013f). Dels finns det funktionalitet för att skriva och läsa från taggar och dels finns det funktionalitet för att kommunicera mot andra enheter vilka har Android som operativsystem. Eftersom detta arbete nyttjar kommunikation mellan två aktiva NFC-enheter är det den senare som är mest relevant. Detta eftersom det är en Androidenhet mikrokontrollern imiterar.

Vidare skriver Android (2013f) att det är möjligt att skicka vilken typ av NFC-meddelande som helst via NFC för Android men att de rekommenderar meddelanden av typen NDEF (NFC Data Exchange Format) vilken är den av NFC Forum definierade standarden. När ett NDEF-meddelande tas emot hanteras det av den process vilken är ansvarig för att den inkommande datan skickas vidare till rätt applikation. En applikation kan deklarera vilka typer av data den kan ta emot i dess manifest.

För den eftersökta kommunikationen mellan två Androidenheter används en teknik vilken kallas *Android Beam* (Android, 2013f). Denna teknik bygger på utbyten av NDEF-meddelanden med hjälp av SNEP. *Android Beam* fungerar genom att två enheter förs mot varandra vilket förbereder skickandet av ett meddelande. Sändningen slutförs genom att användaren trycker på skärmen. Tekniken sker alltså utan att någon form av parning[6] behövs så som är fallet i andra kommunikationsmetoder, exempelvis Bluetooth vars parningsmekanism beskrivs närmare av Chang (2007).

Från och med Android version 4.0 görs *Android Beam* tillgänglig via ett antal API:er (Android, 2013f). Dessa definierar allt ifrån ihopsättande av ett meddelande till hur systemet automatisk förbereder sändande och mottagande. Första steget i användandet av kommunikation via NFC är att få tag på ett objekt som representerar NFC-hårdvaran. Den representeras som ett objekt av typen NFCAdapter. Objektrepresentationen fås via den statiska metoden `getDefaultAdapter`.

Vidare skriver Android (2013f) att en aktivitet kan ställas in för sändning genom att en klass implementerar ett särskilt interface, `CreateNdefMessageCallback`. Implementationen ger klassen förmågan att dynamiskt skapa och skicka meddelanden via *Android Beam*. För att sedan möjliggöra sändningen från en aktivitet behöver denna välja den implementerade klassen genom att kalla på metoden `setNdefPushMessageCallback`. För att ta emot meddelandet behövs, förutom att det står angivit i manifestet, metoden `onNewIntent` skrivas över och i denna behöver meddelandet fångas upp för att sedan tolkas.

### 3.3.2 Design av Android-applikationer

För att skapa grafiska gränssnitt vilka en användare enkelt kan förstå och använda finns generella designprinciper att utgå ifrån vilka kallas Jakob Nielsen's Hueristics (Nielsen, 1995). Jakob Neilsen's Heuristics är 10 stycken riktlinjer vilka har blivit tumregler att följa vid interaktionsdesignsutveckling. Nedan listas vilka reglerna är och hur dessa följs (Nielsen, 1995).

- *Visibility of system status* - Systemet ska informera användaren om vad som händer.
- *Match between system and the real world* - Systemet ska använda ett språk med ord och fraser vilka användaren känner igen och förstår istället för ett systemorienterat språk.

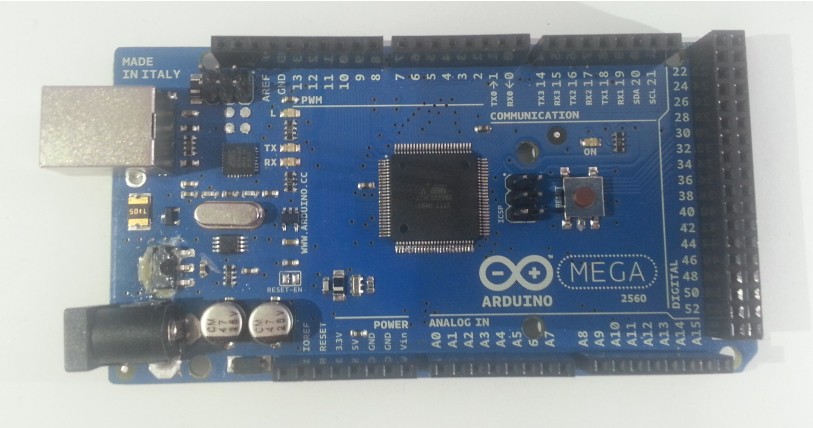
[6] Parning eng pairing

- *User control and freedom* - Användare begår ofta misstag genom att trycka på fel knappar därför behövs "nödutgångar" för att enkelt kunna komma tillbaka från oönskade platser. Stöd för *undo* och *redo* ska också finnas.
- *Consistency and standards* - Följ plattformskonventioner, användaren ska inte behöva fundera på hur olika saker fungerar. Exempel för Android, meny- och back-knappen fungerar och gör vad de skall göra.
- *Error prevention* - Försök undvika att fel uppstår och varna användaren innan fel kan uppstå.
- *Recognition rather than recall* - Användaren skall inte behöva minnas instruktioner utan dessa skall vara tydliga och enkla när dessa behövs.
- *Flexibility and efficiency of use* - genvägar och snabbkommandon kan användas för att snabba upp interaktionen för erfarna användare.
- *Aesthetic and minimalist design* - Framhäv viktig information och ta inte med onödiga detaljer i designen, framförallt i dialoger. Onödig information tar uppmärksamhet från viktig information och minskar dess synlighet.
- *Help users recognize, diagnose, and recover from errors* - Felmeddelanden ska vara enkla och förstå, det vill säga inga felkoder, tala om vad som gick fel och hur det kan lösas.
- *Help and documentation* - Det bästa är om systemet går att använda utan hjälp men om hjälp ska finnas tillgänglig skall den vara lätt att nå, enkel att söka i, fokuserad på användarens uppgift och inte alltför stor.

## 3.4 Arduino

En Arduino (Arduino, 2013d) är en mikrokontroller, det vill säga en mikroprocessor med tillhörande nödvändig elektronik kopplad till ett flertal in- och utgångar, vilken främst är avsedd för hobbymarknaden. Arduino-plattformen har framställts med användarvänlighet i fokus och har en stor gemenskap online (McRoberts, 2010). All den kod vilken bygger upp hela Arduino-plattformen är licenserad under antingen *GNU General Public License* eller *GNU Lesser General Public License* (Arduino, 2013c) vilket bland annat ger möjlighet för ett arbete av detta slaget att nyttja all denna kod.

Flertalet modeller i olika storlekar och prestandaklasser finns men en av de vanligare och den modellen vilken kommer beskrivas mer ingående, illustreras i figur 17 är *Arduino Mega 2560 R3*. Arduino skriver på produktsidan (Arduino, 2013b) för *Arduino Mega 2560 R3* att denna mikrokontroller baseras på mikroprocessorn ATmega2560 från Atmel Corporation med klockhastigheten 16 MHz. Vidare kan läsas på *Arduino Mega 2560 R3* produktsida att denna enhet bland annat innehar 16 analoga ingångar, 54 digitala in/utgångar, USB-kontakt, ström-jack och återställningsknapp. Minnesstorleken på flash-minnet är 256 KB, SRAM (*Static Random-Access Memory*) är 8 KB stort och 4 KB EEPROM (*Electrically Erasable Programmable Read-Only Memory*) finns.




Figur 17: Arduino Mega 2560 R3.

En mikrokontroller av Arduino-typ kan köras med en *bootloader* vilket är ett litet förinstallerat program (Arduino, 2013a) och på *Arduino Mega 2560 R3* utgör den omkring 1 KB av flash-minnet (Arduino, 2013b). En *bootloader* är grovt beskrivet ett litet operativsystem vilken gör exekveringen möjlig av den kod som användaren av mikrokontrollern har laddat in i minnet.

Arduino-plattformens utvecklingsmiljö (Arduino, 2013a) är enkel i sitt utförande och programspråket som används är C++. Utvecklingsmiljön innehåller allt som behövs för att en användare ska kunna skriva sin kod för att sedan kompilera och för att slutligen ladda upp koden mot vald Arduino-modell. Tyvärr erbjuder inte utvecklingsmiljön många funktioner där de mest användbara är autoformatering och möjligheten att inkludera externa Arduino-specifika källkodsbibliotek.

Ett Arduino-program utgörs av en INO-fil vilken är uppbyggd av två delar: `setup` och `loop` (Arduino, 2013e). Den som körs först och endast en gång är `Setup` och där initieras ett programs alla variabler, objekt, etcetera. Efter detta exekveras delen `loop` vilket är själva hjärtat i ett Arduino-program. Denna del innehåller den funktionella delen av ett program och upprepas tills programmet termineras genom stimuli utifrån.

### 3.4.1 Arduino-specifika källkodsbibliotek

Ett flertal källkodsbibliotek finns till användarens hjälp och flertalet av dessa är fokuserade på kommunikation mot kringenheter. Källkodsbiblioteket finns dokumenterade på Arduinos officiella hemsida (Arduino, 2013d). Andra källkodsbibliotek finns men är då skrivna av användare och hittas i Arduino Playground (Arduino, 2013d). De källkodsbibliotek vilka tas upp i detta avsnitt är de bibliotek vilka har använts i implementationen av låsenheten.

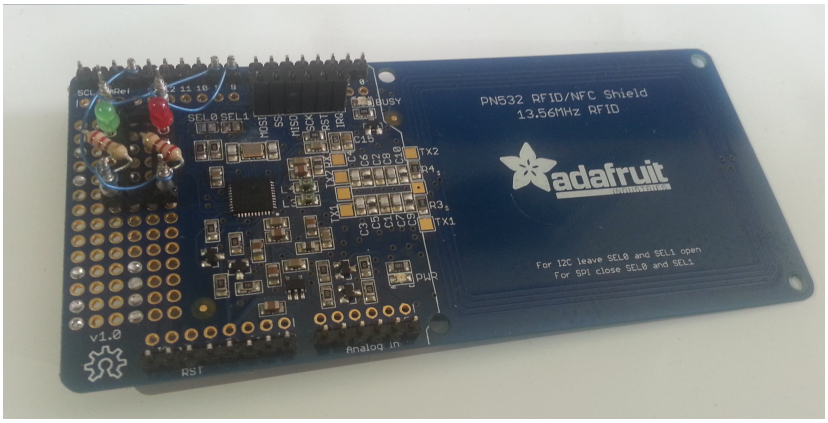
För att kommunicera mot kringenheter via kommunikationsprotokollet I2C (*Inter-Intergrated Circuit*) behövs standardbiblioteket `WIRE`. Om Arduino-modellen *Arduino Mega 2560 R3* används kommer källkodsbiblioteket använda pin nummer 20 och 21 för SDA (*Serial Data Line*) respektive SCL (*Serial Clock Line*). Dessa pinnar är kopplade till motsvarande pinnar på kringenheten. Att förstå och använda källkodsbiblioteket är trivialt och tas därför ej upp i denna text.

Källkodsbiblioteket AVR/SLEEP, vilket finns beskrivet på Arduino Playground (Arduino,

2013d)) används för att mikrokontrollern ska gå ned i strömsparläge. Detta strömsparläge kan varieras mellan ett flertal olika lägen, vilka stänger av olika stora delar av mikrokontrollern. Det läge vilket sparar mest ström är `SLEEP_MODE_PWR_DOWN` och väl inne i detta läge kan mikrokontrollern endast väckas genom en avbrottssignal via pinne två eller tre. Innan kommandot anropas, vilket försätter mikrokontrollern i strömsparläge, måste det specificeras hur den ska väckas. Detta görs genom att ange vilken pinne som ska te emot avbrott och om avbrott ska utlösas vid logisk etta eller nolla. Då mikrokontrollern har väckts genom en avbrottssignal kommer en speciell uppvakningfunktion exekveras där det mottagna avbrottet ska nollställas. Efter att uppvakningsfunktionen har exekverats kommer mikrokontrollern fortsätta exekvera där strömsparläget aktiverades.

### 3.4.2 NFC-skölden PN532 NFC/RFID Shield

För att en Arduino ska kunna nyttja kommunikation via NFC krävs att en så kallad sköld, vilken har NFC-funktionalitet, är ansluten till enheten. En sköld av det beskrivna slaget är Adafruit Industries (2013) *PN532 NFC/RFID Shield*, vilken illustreras i figur 18 och baseras på NFC-chippet *PN532* (NXP Semiconductors, 2013b) från företaget NXP Semiconductors (2013a). Skölden har en RF-antenn vilken specificeras till att ha en räckvidd på 10 cm.




Figur 18: *PN532 NFC/RFID Shield* från företaget Adafruit Industries. På skölden syns också en monterad diodkrets.

Skölden kommunicerar mot en mikrokontroller med tekniken SPI (*Serial Peripheral Interface*) eller I2C. För teknikerna SPI och I2C finns källkodsbibliotek att tillgå, skrivna av Adafruit Industries (2012), vilka kontrollerar kommunikationen mellan skölden och mikrokontrollern. Vidare ger också källkodsbiblioteken ytterligare funktionalitet då manipulation av taggar erbjuds. Dock innehåller källkodsbiblioteken inte stöd för kommunikation mot en aktiv NFC-enhet (se 3.1, NFC), vilket är det detta arbete önskar nyttja.

De protokoll som Adafruit Industries (2012) har implementerat för *PN532 NFC/RFID Shield* är de RF-protokoll vilka är de som finns längst ner i NFC-stacken. Dessa protokoll möjliggör den faktiska överföringen av data och det implementerade protokollet med högst abstraktionsnivå är DEP (*Data Exchange Protocol*) vilket beskrivs i avsnitt 3.1.1.4.

För att kontrollera *PN532 NFC/RFID Shield* finns ett stort antal kommandon vilka beskrivs utförligt i användarmanualen till *PN532* (NXP Semiconductors, 2007) och de beskriv-

ningar vilka görs i detta avsnitt härrör från detta dokument. Ett kommando skickas över vald kommunikationsteknik inkapslad i en ram vilken är specificerad enligt figur 19 (NXP Semiconductors, 2007). Denna ram inkapslar också de svar som fås från *PN532 NFC/RFID Shield*. Punktlistan nedan beskriver ramen mer utförligt.

| PN532 Anrops- och svarsram | | | |
|---|---|---|---|
| Inledande byte | Startkod för paket | Paketlängd | Kontrollsumma för paket |
| 0x00 | 0x00FF | LEN | LCS |
| 1 byte | 2 byte | 1 byte | 1 byte |

| Riktningsidentifierare | Data | Kontrollsumma för data | Avslutande byte |
|---|---|---|---|
| TFI | PD0 - PDn | DSC | 0x00 |
| 1 byte | n+1 byte | 1 byte | 1 byte |

Figur 19: Anrops- och svarsram för kommunikation mot PN532.

- **Inledande byte**[7]. En byte vilken inleder en kommandoram innehåller `0x00`.
- **Startkod för paket.** Två byte innehållande `0x00` respektive `0xFF`.
- **Paketlängd.** En byte vilken indikerar hur många byte datafältet innehåller, det vill säga från TFI till och med PDn.
- **Kontrollsumma för paket.** En byte innehållande en kontrollsumma vilken satisfierar villkoret LEN +LCS = `0x00`.
- **Riktningsidentifierare.** En byte vilken indikerar om *PN532 NFC/RFID Shield* är sändare eller mottagare.
  - `0xD4` anger att *PN532 NFC/RFID Shield* är mottagare av ramen.
  - `0xD5` anger att *PN532 NFC/RFID Shield* är sändare av ramen.
- **Data.** Paketlängd-1 byte av paketdata vilka sträcker sig från PD0 till PDn. Detta fält innehåller alltid ett kommando skickat till skölden eller ett svar skickat från skölden. Maximalt kan 262 byte skickas i en ram till skölden.
- Kontrollsumma för data. En byte innehållande en kontrollsumma vilken satisfierar villkoret TFI + PD0 + PD1 + ... + PDn + DCS = `0x00`.
- Avslutande byte[8]. En byte vilken avslutar en kommandoram innehåller `0x00`.

De kommandon som kan nyttjas grupperas i fyra grupper, generella kommandon, RF-kommunikation, `initiator` och `target`. I gruppen generella kommandon finns kommandon som diagnostiserar NFC-chipet, läser och skriver till register samt konfigurerar skölden i viss mån. Gruppen RF-kommunikation innehåller två kommandon vilka konfigurerar radio-fältet

[7]De byte vilka är placerade efter den avslutande byten eller innan den inledande byten kan vara godtyckligt många och ha godtyckliga värden så länge följden 0x00 0xFF inte förkommer.

[8]De byte vilka är placerade efter den avslutande byten eller innan den inledande byten kan vara godtyckligt många och ha godtyckliga värden så länge följden 0x00 0xFF inte förkommer.

respektive används för radio-fälts reglering. Tredje gruppen, `initiator`, innehåller de kommandon vilka används då *PN532 NFC/RFID Shield* agerar initiator till NFC-kommunikationen, och den sista gruppen omfattar de kommandon vilka används då *PN532 NFC/RFID Shield* är målenheten för NFC-kommunikation.

*Då PN532 NFC/RFID Shield* har mottagit en kommandoram korrekt sänds en ACK-ram tillbaka till sändaren vilken är uppbyggd enligt figur 20.

| PN532 ACK-ram | | | |
|---|---|---|---|
| Inledande byte | Startkod för paket | ACK-kod | Avslutande byte |
| 0x00 | 0x00FF | 0x00FF | 0x00 |
| 1 byte | 2 byte | 2 byte | 1 byte |

Figur 20: Utförandet av en PN532 ACK-ram.

Om en kommandoram vilken mottas och tolkas korrekt av *PN532 NFC/RFID Shield* innehåller felaktiga värden skickas en ERROR-ram till mikrokontrollern vilken är uppbyggt enligt figur 21.

| PN532 ERROR-ram | | | |
|---|---|---|---|
| Inledande byte | Startkod för paket | Paketlängd | Kontrollsumma för paket |
| 0x00 | 0x00FF | 0x01 | 0xFF |
| 1 byte | 2 byte | 1 byte | 1 byte |

| ERROR-kod | Kontrollsumma för paket | Avslutande byte |
|---|---|---|
| 0x7F | 0x81 | 0x00 |
| 1 byte | 1 byte | 1 byte |

Figur 21: Utförandet av en PN532 ERROR-ram.

De kommandon vilka beskrivs i denna text är de vilka varit till nytta för utvecklingen av prototypen och de återstående kommandona tas ej upp, bland annat utelämnas kommandogrupperna RF-kommandon och `initiator`-kommandon helt då de ej nyttjas.

**3.4.2.1 Generella kommandon till PN532** Dessa kommandon utför läsningar och skrivningar till register, diagnostisering utav NFC-chippet, viss konfiguration av skölden samt att skölden kan ställas i olika vilolägen.

**SAMConfiguration** När *PN532 NFC/RFID Shield* startas upp befinner sig NFC-chippet i läget `LowVbat`, genom att anropa kommandot SAM (*Security Access Module*) Configuration vaknar skölden och går in i ett läge där den behandlar också alla andra kommandon[9]. Kommandot `SAMConfiguration` är uppbyggt enligt figur 22, vilken visar fälten `TFI`, `PD0` till `PD3` i en kommandoram.

[9]Om `SAMConfiguration` används för att skölden ska gå ur `LowVbat` läget måste `Mode` sättas till `0x01`.

| PN532 SAMConfiguration - Anrop | | | | |
|---|---|---|---|---|
| TFI | PD0 | PD1 | PD2 | PD3 |
| 0xD4 | 0x14 | Läge | Timeout | [IRQ] |
| 1 byte | 1 byte | 1 byte | 1 byte | 1 byte |

Figur 22: Utförandet av anropet av kommandot SAMConfiguration.

- **0xD4** anger att *PN532 NFC/RFID Shield* är mottagare av ramen.
- **0x14** är den kommandospecifika koden.
- **Läge.** Här väljs hur SAM (*Security Access Module*) ska användas. Då *PN532 NFC/RFID Shield* inte har någon SAM väljs normalt läge då en SAM inte används, genom att ange `0x01`.
- **Timeout.** Om SAM inte används sätts denna parameter till `0x00`.
- **IRQ**, frivilligt fält. Om en `IRQ-lina` ska användas för smidigare kommunikationen mot en Arduino anges `0x01`, annars `0x00`.

Svarsmeddelandet till `SAMConfiguration` är uppbyggt enligt figur 23, vilken visar fälten `TFI` och `PD0` i en svarsram. Det ända som svaret innehåller, förutom `TFI`, är den kommandospecifika svarskoden `0x15`.

| PN532 SAMConfiguration - Svar | |
|---|---|
| TFI | PD0 |
| 0xD4 | 0x15 |
| 1 byte | 1 byte |

Figur 23: Utförandet av svaret till kommandot SAMConfiguration.

#### 3.4.2.2 Kommandon till PN532 då NFC-skölden är målenhet

Då *PN532 NFC/RFID Shield* är målenheten för kommunikationen finns ett antal kommandon som är dedikerade för detta ändamål. Dessa kommandon ger bland annat funktionalitet i form av att konfigurera skölden som en målenhet och att sända samt ta emot data.

**TgInitAsTarget** För att konfigurera *PN532 NFC/RFID Shield* som en målenhet anropas kommandot `TgInitAsTarget` enligt figur 24, vilken visar fälten `TFI`, `PD0` till `PDn` i en kommandoram.

| PN532 TgInitAsTarget - Anrop | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| TFI | PD0 | PD1 | PD2 - PD7 | PD8 - PD25 | PD26 - PD36 | PD37 | PD38 - PDXX | PDXX | PDXX - PDn |
| 0xD4 | 0x8C | Läge | Mifare param. | FeliCa param. | NFCID3t | LEN Gt | Gt | LEN Tk | Tk |
| 1 byte | 1 byte | 1 byte | 6 byte | 18 byte | 10 byte | 1 byte | Len gt byte | 1 byte | Len tk byte |

Figur 24: Utförandet av anropet av kommandot TgInitAsTarget.

- **0xD4** anger att *PN532 NFC/RFID Shield* är mottagare av ramen.

- **0x8C** är den kommandospecifika koden.
- **Läge.** Val av operationsläge.
  - bit 0 satt indikerar att skölden endast kommer påbörja kommunikation mot passiva NFC-enheter.
  - bit 1 satt indikerar att skölden följer protokollet DEP (*Data Exchange Protocol*) och kommer endast påbörja kommunikation om en `ATR_REQ` ram mottas.
  - bit 2 satt indikerar att skölden emulerar en tagg av typen ISO/IEC14443-4 och endast kommer påbörja kommunikation om en RATS ram mottas.
- **Mifare parametrar**, 6 byte. Parametrar vilka behöver definieras då kommunikation mot en passiv NFC-enhet i 106 Kb/s önskas.
- **FeliCa parametrar**, 18 byte. Parametrar vilka behöver definieras då kommunikation mot en passiv NFC-enhet i 212/424 Kb/s önskas.
- **NFCID3t**, 10 byte, ingår i utformningen av `ATR_RES` (se avsnitt 3.1.1.4).
- **LEN Gt** anger hur många byte, max 47, `Gt` består av.
- **Gt**, `LEN Gt byte`, anger de generella byte som används i `ATR_RES` (se avsnitt 3.1.1.4).
- **LEN Tk** anger hur många byte, max 48, `Tk` består av.
- **Tk**, `LEN Tk byte`, anger de historiska byte vilka används i `ATS` när skölden emulerar en tagg av typen ISO/IEC14443-4.

Svarsmeddelandet till `TgInitAsTarget` är uppbyggt enligt figur 25, vilken visar fälten `TFI`, `PD0` till `PDn` i en svarsram. Svaret innehåller `TFI`, den kommandospecifika svarskoden `0x8D`, en byte vilken indikerar i vilket läge skölden blivit konfigurerad samt det första meddelandet från initiatorn.

| PN532 TgInitAsTarget - Svar | | | |
|---|---|---|---|
| TFI | PD0 | PD1 | PD2 - PDn |
| 0xD5 | 0x8D | Läge | Initiatorns meddelande |
| 1 byte | 1 byte | 1 byte | n-1 byte |

Figur 25: Utförandet av svaret till kommandot TgInitAsTarget.

**TgSetData** Detta kommando används då data önskas skickas enligt `DEP` och är uppbyggt enligt figur 26, vilken visar fälten `TFI`, `PD0` till `PDn` i en kommandoram. Förutom den kommandospecifika koden innehåller ramen de byte vilka önskas skickas. Observera att maximalt 262 byte kan skickas åt gången och om fler ska skickas måste också kommandot `SetMetaData` användas.

| PN532 TgSetData - Anrop | | |
|---|---|---|
| TFI | PD0 | PD1 - PDn |
| 0xD4 | 0x8E | Utgående data |
| 1 byte | 1 byte | n byte |

Figur 26: Utförandet av anropet av kommandot TgSetData.

Det svar vilket erhålles från skölden är uppbyggt enligt figur 27, vilken visar fälten `TFI`, `PD0` och `PD1` i en svarsram. Förutom att svaret innehåller den kommandospecifika svarskoden `0x8F` samt `TFI` finns också ett statusfält vilket indikerar om sändningen utfördes korrekt, vilket indikeras med `0x00`, eller inte.

| PN532 TgSetData - Svar | | |
|---|---|---|
| TFI | PD0 | PD1 |
| 0xD5 | 0x8F | Status |
| 1 byte | 1 byte | 1 byte |

Figur 27: Utförandet av svaret till kommandot TgSetData.

**TgGetData** Detta kommando används då data önskas hämtas enligt `DEP` och är uppbyggt enligt figur 28, vilken visar fälten `TFI` och `PD0` i en kommandoram, och innehåller förutom `TFI` endast den kommandospecifika koden.

| PN532 TgGetData - Anrop | |
|---|---|
| TFI | PD0 |
| 0xD4 | 0x86 |
| 1 byte | 1 byte |

Figur 28: Utförandet av anropet av kommandot TgGetData.

Svaret från skölden är uppbyggt enligt figur 29, vilken visar fälten `TFI`, `PD0` till `PDn` i en svarsram, och innehåller den kommandospecifika svarskoden, `TFI`, ett statusfält vilket indikerar om sändningen utfördes korrekt, vilket indikeras med `0x00`, eller inte och slutligen finns också datan vilken önskade hämtas vilket kan vara upp till 262 byte.

| PN532 TgSetData - Svar | | |
|---|---|---|
| TFI | PD0 | PD1 |
| 0xD5 | 0x8F | Status |
| 1 byte | 1 byte | 1 byte |

Figur 29: Utförandet av svaret till kommandot TgGetData

# 4 Kravanalys och design

En kravspecifikation (se Appendix A: Kravspecifikation) utformades för hela produkten och för de ingående delarna. Kraven sammanställdes enligt den beskrivna metoden (se avsnitt 2) och en sammanfattning av kravspecifikationen finns under avsnitt 4.1 Prototypkrav. Den analys vilken gav de krav en användare kan ställa på prototypen baserades på de krav projektgruppen skulle ställt på ett liknande system.

Ett kommunikationsprotokoll sammanställdes på sådant sätt att prototypkraven uppfylldes. Detta protokoll specificerar hur kommunikationen mellan enheterna är uppbyggd, hur kommunikationsförloppet är designat, vilken information som utbyts och hur kommunikationssäkerheten är löst. Hela protokollet beskrivs nedan under avsnitt 4.2.

## 4.1 Prototypkrav

Produktens upplåsningsförfarande, alltså den tid som användaren spenderar väntande vid låset efter att ha valt kommando i mobilapplikationen, ska ta maximalt fem sekunder men det bör ta endast en sekund. Tiderna har valts med användarkomfort i åtanke och skulle upplåsningsförloppet ta lång tid försämras denna. Vidare bör produkten kunna konfigureras med hjälp av mobilapplikationen för ett antal väl valda parametrar vilka antingen ger ökad säkerhet till systemet eller utökad funktionalitet. Då prototypen besitter en högre grad av intelligens erhålles mer flexibilitet än hos vanliga lås. Detta kapitaliseras på genom att ge användaren mer valfrihet vad gäller säkerhet och önskad funktionalitet.

Låsenheten bör vara felsäker, till exempel ska låset förbli låst vid strömavbrott. Vidare bör formfaktorn av låsenheten anpassas för det utrymme den avses installeras i, såsom låshus i dörr eller kassaskåp. Senare bör en fungerande prototyp färdigställas där låsenheten fysiskt har installerats.

Mobilapplikationen bör vara lättförståelig och mycket enkel att hantera då det är användandet av en vanlig metallnyckel som ska konkurreras med. Vidare bör inga kända buggar finnas samt upplåsningsnycklar till låsenheten bör kunna delas med andra mobiltelefoner vilka har mobilapplikationen installerad. På så vis erhålles en flexiblare lösning än vad som erbjuds av metallnycklar, då delning av en digital motsvarighet inte kräver att en låssmed involveras. Mobilapplikationen bör även tillhandhålla någon form av personverifiering, till exempel PIN-kod, vilket gör systemet säkrare.

## 4.2 Kommunikationen mellan enheterna

Kommunikation via NFC kan åstadkommas på olika sätt men i detta arbete kommer metoden *peer-to-peer* användas. Anledningen till detta är att Android, genom *Android Beam*, endast kan använda den metoden för kommunikation med en aktiv NFC-enhet, vilket detta arbete inriktar sig mot. För att mikrokontrollern ska kunna kommunicera mot Android via *peer-to-peer* måste mjukvaran för mikrokontrollern imitera Androids NFC-stack. Detta måste göras då Android endast har stöd för att använda *Android Beam* mot andra Android-enheter. Teori kring detta beskrivs mer omfattande i avsnitt 3.3.1.

Förutom att lösa kommunikationen via NFC har ett kommunikationsprotokoll på applikationsnivå upprättas vilken tar särskild hänsyn till följande punkter:

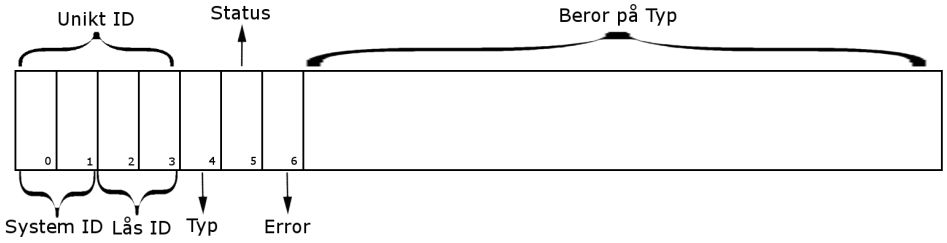
- Låsenheten ska identifiera sig mot mobilapplikationen då en mobilapplikation kan inneha ett flertal upplåsningsnycklar till olika låsenheter och mobilapplikationen ska kunna avgöra vilken nyckel som ska användas.
- Om mobilapplikationen inte har tillgång till den upplåsningsnyckel, vilken erfodras av låsenheten, skall den inte utföra begärda kommandon.
- Det skall inte vara möjligt för en tredje part att få tillgång till en upplåsningsnyckelnyckel.
- Det skall finnas möjlighet att distribuera upplåsningsnycklar mellan användare.
- Android-applikationen skall kunna byta upplåsningsnyckel till låsenheten.

Nedan beskrivs det utarbetade kommunikationsprotokollet.

### 4.2.1 Meddelandetyper

För att göra tolkandet av meddelandena smidig och enkel har ett antal olika meddelandetyper utformats. De huvudsakliga typerna benämns $1 - 5$. Meddelandena representeras på mobilapplikationssidan av en klass kallad NFCPMessage. På mikrokontrollern och vid sändandet representeras meddelandet av ASCII-strängar.

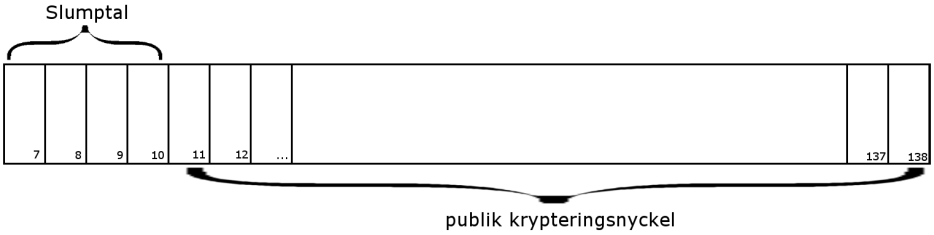
Gemensamt för de fem olika typerna är betydelsen av de första sju tecknen vilka illustreras av figur 30. De fyra första tecknen skapar tillsammans en identifiering av det aktuella låset. Fyra tecken ger $2^{32}$ olika alternativ vilket möjliggör lite över fyra miljarder olika låsenheter. Vidare anger nästkommande tre tecken typen, statusen och eventuell felkod för meddelandet.




Figur 30: Figuren illustrerar det de olika meddelandetyperna har gemensamt.

Efter de sju första tecknen följer ett varierande antal tecken. Hur många dessa är och vad de betyder bestäms av meddelandets typfält. Följande figurer illustrerar meddelandetyperna och då typ 3 inte innehåller någon ytterligare information finns ingen figur. Typerna används enligt förloppen i avsnitt 4.2.2.

Typ 1, vilken illustreras av figur 31, lägger till ett fyrsiffrigt slumptal följt av en 128 byte publik krypteringsnyckel. Meddelandet skickas från låsenhet till smarttelefon som ett första steg i ett upplåsningsförfarande.

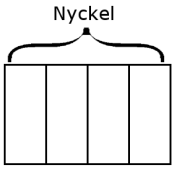



Figur 31: Figuren illustrerar typ 1.

Typ 2, vilken illustreras av figur 32, lägger till ett 128 byte stort fält med krypterad information. Den krypterade datan består av ett slumptal och en upplåsningsnyckel till låsenheten.


Figur 32: Figuren illustrerar typ 2.

Typ 4, vilken illustreras av figur 33, lägger till en 4 byte nyckel.




Figur 33: Figuren illustrerar typ 4.

Typ 5, vilken illustreras av figur 34, vilken visas i figur lägger till ett krypterat fält som är 128 byte stort. Den krypterade datan utgörs av ett slumptal samt nuvarande och ny nyckel till en låsenhet.


Figur 34: Figuren illustrerar typ 5.

### 4.2.2 Kommunikationsförlopp

För låssystemet har tre kommunikationsförlopp utarbetats. Det första beskriver ett typiskt försök till upplåsning och det andra en lösning till distribution av nycklar. Ett tredje förlopp har också utarbetats för att konfigurera om upplåsningsnyckeln till låsenheten.

| Upplåsningsförlopp | |
|---|---|
| Mobilapplikation | Låsenhet |
| 1. Mobilapplikationen startas upp. | Låsenheten är passiv och väntar på att detektera ett magnetfält. |
| 2. Personverifiering genomförs genom att användaren anger den förvalda PIN-koden i mobilapplikationen | |
| 3. Då mobilens knapplås är av, förs mobiltelefonen i närheten av låsenheten. | Låsenheten detekterar mobiltelefonens magnetfält och vaknar upp. |
| 4. | Låsenheten skickar ett meddelande av typ 1 bland annat innehållande en publik krypteringsnyckel, ett slumptal samt låsenheternas ID. |
| 5. Mobiltelefonen tar emot meddelandet och söker med hjälp av låsenhetens ID i mobilens lokala databas efter de specifika nycklar som finns för det särskilda låset. | |
| 6. Finns ingen nyckel kan förloppet inte fortskrida men finns detta fortsätter förloppet. | |
| 7. Telefonen krypterar nyckeln samt det mottagna slumptalet med hjälp av den publika krypteringsnyckeln och skickar detta i form av ett medelande av typ 2. | |
| 8. | Låsenheten dekrypterar med sin privata krypteringsnyckel och jämför resultatet med sitt aktuella nyckel och slumptal. |

Figur 35: Upplåsningsförlopp.

| | Mobilapplikation | Låsenhet |
|---|---|---|
| 9. | | Om nyckel är lika med den dekrypterade nyckeln samt det genererade sumptalet är lika med det mottagna, skickar låsenheten en signal för upplåsning. |
| | Alternativa steg | |
| 10. | Om ett svar, innehållande information om låsenheten, beviljade upplåsningen önskas hålls mobiltelefonen kvar mot låsenheten. | |
| 11. | | Låsenheten skickar ett svar av typ 3 där den anger reslutatet samt eventuellt felkod. |
| 12. | Mobiltelefonen tar emot meddelandet och visar informationen för användaren. | |

Figur 36: Upplåsningsförlopp.

**4.2.2.1 Kommunikationsförlopp vid distribution av nycklar** Eftersom projektets mål är att utveckla en prototyp mot privatpersoner och för att prototypen inte ska bli för komplex har valet att dela upplåsningsnycklar mellan två olika mobiltelefoner gjorts. Den mest användarvänliga lösningen vore att ha en central enhet som är ansvarig för distribution av upplåsningsnycklar. Dock skulle en sådan lösning medföra utmaningar vilka går utöver projektets syfte. En lösning att dela upplåsningsnycklar direkt mellan olika mobiltelefoner medför således mindre svårigheter under utvecklingen. Användarvänligheten borde inte minska utav denna lösning då det vanligen endast är ett fåtal användare i ett hushåll. Ett större system för distribution är under dessa förhållanden onödigt. Om ett hushåll däremot har tillfälliga besökare som behöver ha tillgång till huset utan att först fysiskt möta en av familjemedlemmarna som redan har tillgång till huset skulle en central enhet vara att föredra.

Den ena mobiltelefonen har den aktuella upplåsningsnyckeln till det aktuella låset, och skickar den via NFC till den andra mobiltelefonen vilken lägger in den i sin databas. Nedan beskrivs hur delningen går till. Mobiltelefon A avser den mobiltelefon som har för avsikt att dela en upplåsningsnyckel med mobiltelefon B. Proceduren har konstruerats för att vara så användarvänlig som möjligt och beskrivs nedan.

1. Användaren på mobiltelefon A väljer ut en upplåsningsnyckel för delning.
2. Mobiltelefonerna förs mot varandra och användaren av mobiltelefon A trycker på skärmen för att dela.

3. Ett meddelande av typ 4 skickas från mobiltelefon A till mobiltelefon B.

4. Mobiltelefon B tar emot upplåsningsnyckeln och användaren bekräftar att nyckeln skall läggas till i databasen.

**4.2.2.2 Konfigureringsförlopp** Eftersom det finns ett behov av att ändra upplåsningsnyckeln till låsenheten då en användare inte längre ska ha tillgång till låset har ett förlopp för att ändra låsenhetens upplåsningsnyckel utarbetats. Alla användare som har den nuvarande nyckeln har möjlighet att byta ut den. Följande illustrerar förloppet.

1. Punkt ett och två genomförs enligt upplåsningsförloppet.

2. Användaren av mobilapplikationen väljer och knappar in en ny upplåsningsnyckel.

3. Förloppet som följer är nu enligt upplåsningsförloppet med skillnaden att vid punkt 7 skickas ett meddelande av typ 5 istället för av typ 2.

### 4.2.3 Säker kommunikation

För att uppfylla kravet som ställts på säker kommunikation måste de fyra säkerhetshoten som beskrivits i teoridel 3.2 avvärjas. De beskrivna säkerhetshoten: avlyssning, datamodifiering, *replay attack* och *man-in-the-middle attack* kommer att försvåras avsevärt och förhoppningsvis avhjälpas helt med hjälp av kryptering av känslig information. Med känslig information menas upplåsningsnyckeln till låsenheten samt slumptalet vilka sänds i meddelanden av typ 2 från mobilapplikationen (se avsnitt 4.2.2). Asymmetrisk kryptering (se avsnitt 3.2.5) kommer att användas och mer specifikt kryptosystemet RSA. Nedanstående punkter kommer att behandla hur tidigare beskrivna hot ska motverkas med hjälp av RSA-kryptering och andra åtgärder. Till sist beskrivs det krypteringsförlopp vilket prototypen skall utföra.

- **Avlyssning.** Detta kan inte undvikas, men sårbarheten för denna sortens attack kan minskas. Väljs en av de högre överföringshastigheterna, dvs 212kbit/s eller 424kbits/s, används svagare modulering vilket minskar räckvidden som signalerna effektivt kan bära information. Antennens effekt kan inte styras, men dess räckvidd är begränsad till 10 cm vilket försvårar avlyssning. Vidare kan varken antennens utformning eller omgivningen inte påverkas. Då informationen är krypterad kan avlyssning inte ge någon insikt i vad som skickas.

- **Datamodifiering.** Denna typ av attack kommer att undvikas med hjälp av en krypterad kanal. Kryptering medför att angriparen inte vet vad som skickas och mer avancerad datamodifiering förlorar sitt syfte. Vidare kan överföringshastigheten på 106kbit/s väljas vilket medför att kodningstekniken *Modifierad Miller* används, vilket gör datamodifiering svår att genomföra. Dessvärre ackompanjeras denna bithastighet av kraftig modulering vilket resulterar i försämrat skydd mot avlyssning. Varför kraftig modulering inte är ett säkerhetsproblem framgår av föregående punkt.

- **Man-in-the-middle attack.** Attacker av detta slag är i teorin omöjligt att åstadkomma mot NFC-kommunikation och ses därför inte som ett hot, vilket beskrivs i avsnitt 3.2.4.

- **Replay attack.** För att motverka *replay attack* (se avsnitt 3.2.3) genererar låsenheten ett slumptal vilket skickas tillsammans med den publika krypteringsnyckeln. Detta tal krypteras av mobilapplikationen tillsammans med låsenhetens upplåsningsnyckel för att skapa unika meddelanden vid varje förlopp och på så vis göra *replay*-attacker omöjliga att genomföra. Det sista steget i förloppet är för låsenheten att dekryptera meddelandet och se till att slumptal samt låsenhetens upplåsningsnyckel stämmer med vad som förväntas (asvnitt 4.2.2 beskriver kommunikationsförloppet i sin helhet).

#### 4.2.3.1 Krypteringsförlopp

Krypteringsförloppet finns till för att motverka ovanstående säkerhetshot, och detta stycke beskriver hur förloppet går till för att skapa en kanal där känslig data kan skickas utan att utsättas för säkerhetsrisker.

Det första som sker under förloppet är att låsenheten genererar ett slumptal vilket används för att kunna skapa ett unikt meddelande. Låsenheten skickar sedan sin publika krypteringsnyckel tillsammans med slumptalet till mobilapplikation då den önskar skicka ett krypterat meddelande. Mobilapplikationen använder den publika krypteringsnyckeln för att kryptera ett meddelande innehållandes slumptalet och upplåsningsnyckeln till låsenheten. När det krypterade meddelandet skickats av applikationen och sedan mottagits hos låsenheten dekrypteras det med hjälp av låsenhetens privata krypteringsnyckel. Slutligen jämförs det slumptal vilket finns i det dekrypterade meddelandet med det slumptalet vilket genererades i början av krypteringsförloppet, om dessa överensstämmer anses det mottagna meddelandet vara verifierat och dess innehåll ges till den körande applikationen på mikrokontrollern.

# 5 Material och verktyg

Nedan listas de material och verktyg vilka användes vid framställandet av prototypen. Med material menas de fysiska delarna vilket lösningen består av och med verktyg menas de mer immateriella delarna såsom
källkodsbibliotek, utvecklingsmiljöer eller specifikationer.

## 5.1 Mobilapplikation

Androidplattformen, vilken beskrivs i avsnitt 3.3, valdes för utvecklandet av mobilapplikationen då Android är ett väletablerat mobilt operativsystem med en stor användarbas (Android, 2013b). Android-plattformen valdes för dess stora gemenskap samt dess öppna källkod. Vidare var Android och tillverkare av Android-baserade mobiltelefoner, enligt Barlas (2010), tidiga med att inkludera NFC-tekniken och Android har därför detta ett bra stöd för tekniken i dagsläget.

För att få det bästa möjliga stödet för NFC väljs de nyare Android-versionerna. Android 4.0 och uppåt väljs då det, enligt Google Inc (2013), från och med denna version finns utökade NFC-funktioner i form av *Android Beam* (Android, 2013g).

De mobiltelefoner vilka har funnits tillgängliga för projektets räkning är följande: LG Optimus L9 (E 610), HTC One X och Samsung Galaxy S3 (GT-I9300). Alla tre mobiltelefoner har stöd för NFC, men mobiltelefonen från LG har Android version 4.0 medan mobiltelefonerna från HTC och Samsung har Android version 4.1.2. Dock finns det ingen skillnad på API-nivå för NFC-kommunikation mellan dessa versioner. Vidare är modellerna från HTC och Samsung tillverkarnas respektive flaggskepp medan den från LG var en budgetmodell.

### 5.1.1 Utvecklingsverktyg

För utveckling av Android-applikationer behövs förutom JDK (*Java Development Kit*) (Oracle Corporation, 2013b) också Android-specifika program vilka laddas hem i form av Android SDK (Android, 2013a). Paketet innehåller bland annat programmet Eclipse (Eclipse, 2013) och diverse tillägg som möjliggör utveckling och testning av Android-applikationer.
Användningen av Eclipse för det nämnda ändamålet kan ses som standard för Android-utveckling och eftersom projektgruppen innehar tidigare positiva erfarenhet av Eclipse valdes detta trots att andra alternativ finns.

### 5.1.2 Använda API:er

Java har ett API (Oracle Corporation, 2013a) med många och användbara klasser och metoder. Förutom detta finns det ett särskilt API för Android som bygger på Javas API, men innehar ytterligare funktionalitet vilken är specifik för en Android-enhet. Androids API innehåller många klasser vilka är till hjälp vid Android-utveckling och har därför använts. I anslutning till detta API finns också utförliga guider och användarmanualer. Ett specifikt paket[10] (*package*) som var användbart var *android.nfc* (Android, 2013c) vilket innehåller de klasser och metoder som en Android-utvecklare behöver får att implementera en applikation som kommunicerar via NFC.

[10] från engelskans package

För kryptering användes två paket från Java API för säkerhet,
*java.security* (Oracle Corporation, 2013c) och *javax.crypto* (Oracle Corporation, 2013d). *Java.security* utnyttjades för att skapa RSA-baserade nyckelpar från den mottagna nyckelspecifikationen och *javax.crypto* användes för att kryptera meddelanden med den skapade krypteringsnyckeln. På grund av att det ej gick att göra en av metoderna serialiserbar i de ovannämnda krypteringsbiblioteken användes paketet *Bouncy Castle Crypto API* (Legion of the Bouncy Castle, 2013) för att få fungerande kryptering på Android-plattformen.

## 5.2 Låsenhet

Valet av hårdvara till låsenheten föll på Arduino-plattformen, vilket beskrivs i avsnitt 3.4. Arduino är användarvänligt och har en stor användarbas (McRoberts, 2010) samt att färdigutvecklade källkodsbibliotek för diverse användningsområden, däribland NFC, att tillgå. Vidare är Arduino öppen källkod och dess produkter är relativt billiga. Att välja Arduinoplattformen är framförallt en tidseffektiv lösning då flera NFC-påstickskort finns tillhanda vilka lätt kan anslutas till en Arduino vilket medför att ingen hårdvara behöver utvecklas för att implementera låsenheten.

En Arduino av typen *Arduino Mega R3* vilken beskrivs i avsnitt 3.4 valdes tillsammans med NFC-sköldenen *PN532 NFC/RFID Shield* från tillverkaren Adafruit Industries vilken beskrivs i avsnitt 3.4.2. Det finns ett flertal mikrokontroller under benämningen Arduino och *Aduino Mega R3* valdes ut då den hade det minne och den processorkraft vilket krävdes för att klara av de uppgifter låsenheten hanterar, men *Aduino Mega R3* är fortfarande en relativt simpel mikrokontroller och innehar inte mycket mer funktioner, minne och processorkraft än det som projektet kräver. NFC-sköldenen *PN532 NFC/RFID Shield* valdes då den jämfört med ett flertal andra NFC-sköldar hade större funktionalitet då fler tekniker kan användas för kommunikation mot mikrokontrollern.

### 5.2.1 Utvecklingsverktyg

Den utvecklingsmiljö vilken använts vid utveckling av låsenheten är Arduinos utvecklingsmiljö version 1.0.4. Denna utvecklingsmiljö användes då den är enkel och lätt att använda. Vidare erhålls denna plattform av Arduino-organisationen och flertalet användbara kodbibliotek medföljer.

### 5.2.2 Val av källkodsbibliotek för implementation av NFC-stacken

De källkodsbibliotek (Adafruit Industries, 2012) vilket företaget bakom
*PN532 NFC/RFID Shield* erbjöd visade sig inte ge all den funktionalitet projektet krävde. Källkodsbiblioteket gav endast möjlighet att kommunicera mot passiva NFC-enheter och inte aktiva vilket brukas i detta projekt.

Källkodsbiblioteket *Embedded-PN532* skapat av Michael Wier (2012) är ett källkodsbibliotek under *öppen källkodslicens* för att skapa kommunikation mellan en Arduino-plattform med ansluten NFC-sköld och en Android-baserad mobiltelefon. Det källkodsbibliotek som följde med PN532 *NFC/RFID Shield* ligger som grund i *Embedded-PN532*. Grunden har byggts på av ett antal lager vilka gör kommunikation mot Android möjligt. Upphovsmannen garanterar att kommunikationen fungerar då Android 2.3.3 används (Wier, 2012), vilket är en tidigare version än den versionen projektet arbetar mot.

Dokumentationen för *Embedded-PN532* är bristfällig vilket gör det svårt att tolka och förstå. Vidare är endast delar ur de behövda protokollen implementerade och faktum framkommer att detta källkodsbibliotek är producerat i hobby-syfte.

Källkodsbiblioteket *Open NFC* är en utmanare till *Embedded-PN532*, vilken tillhandahålls av företaget Inside Secure (2013) och är det källkodsbibliotek Android använder för att kommunicera över NFC (Wilson, 2013). Detta bibliotek erbjuder en fullständig implementation av NFC-stacken och således besitter biblioteket stora fördelar gentemot *Embedded-PN532*. Att Open NFC inte användes kan härledas till att det hittades en tid in i projektet då en ökad förståelse för *Embedded-PN532* uppkommit samt att ett underliggande operativsystem i stort sett krävs för att källkodsbiblioteket ska fungera. Detta bedömdes vara något som varken utrymmesmässigt finns eller planerats ge plats åt på mikrokontrollern. Vidare behövdes en NFC HAL(*Hardware Abstraction Layer*) modul skrivas vilken länkar samman *Open NFC* med det aktuella NFC-chippet, vilket framstod som ett omfattande arbete. Dessa anledningar gjorde att *Embedded-PN532* valdes trots dess brister och låsenheten kommer alltså att använda *Embedded-PN532* för att åstadkomma kommunikation via NFC.

De protokoll *Embedded-PN532* implementerar är LLCP, NPP och NDEF. Viktigt att notera är att endast delar eller bråkdelar av dessa protokoll är implementerade och det är omöjligt att använda detta källkodsbiblioteket till applikationer vilka behöver utbyta stora mängder data. Den mängd data vilken kan skickas åt gången är begränsad då *chaining*-mekanismen i LLCP, fragmenterings-mekanismen i SNEP och *chunk*-mekanismen i NDEF inte är implementerade. Dock kommer inte det låssystem vilket arbetas fram i projektet behöva skicka meddelanden vilka kräver dessa mekanismer (se avsnitt 4.2.1). Detta gör att användandet av *Embedded-PN532* är lämplig trots dess brister.

### 5.2.3 Val av källkodbibliotek för implementation av kommunikationssäkerhet

Det finns flera källkodsbibliotek för att implementera kryptering i C/C++ där det mest använda källkodsbiblioteket är *OpenSSL*(The OpenSSL Project, 2009). Dock visade det sig att *OpenSSL* kräver mycket minne och är således inte optimerat för ett litet inbyggt system som Arduino. Ett försök till att minska det minnesutrymme *OpenSSL* kräver, var att plocka ur specifik funktionalitet ur källkodsbiblioteket. Denna lösning fungerade dock inte ty *OpenSSL* bygger inte på lös bindning[11] vilket betyder att en klass är mer eller mindre oberoende av övriga klasser.

Det som eftersöktes var ett litet krypteringsbibliotek gjort speciellt för inbyggda system med begränsad prestanda och krypteringsbiblioteket *PolarSSL*(Offspark B.V, 2013)visade sig vara detta. *PolarSSL* är lämplig för inbyggda system eftersom det bygger på lös bindning alltså perfekt för mikrokontrollern där endast ett fåtal funktioner behövdes och på så sätt sparades minnesutrymme. I detta källkodsbiblioteket var det enkelt att plocka ut de delar som behövdes, vilket i stort sett bara var dekryptering av krypterade meddelanden.

### 5.2.4 Protokoll, specifikationer och källkodsbibliotek

De protokoll och källkodsbibliotek vilka har blivit implementerade eller modifierade är följande:

- SNEP (Simple NDEF Exchange Protocol)(NFC Forum, 2011b)

[11] Från engelskans loose coupling

- LLCP (Logic Link Control Protocol)(NFC Forum, 2011a)
- Källkodsbiblioteket *Embedded-PN532* (Wier, 2012)
- Källkodsbiblioteket PolarSSL (Offspark B.V, 2013)

De protokoll, specifikationer, manualer och källkodsbibliotek vilka har bidragit till ökad förståelse inom arbetet är följande:

- NDEF (NFC Data Exchange Format) (NFC Forum, 2006)
- NPP (NDEF Push Protocol) (Android Open Source Project, 2011)
- Användarmanual till PN532 (NXP, 2007)
- Applikationsguide till PN532 (NXP, 2010)
- Källkodsbiblioteket ismb-snep-java (Lotito, 2013a)
- ISO/IEC 18092 - NFCIP (*Near Field communication, Interface and Protocol*), del 1. (International Organization for Standardization, 2004)
- Källkodsbiblioteket *OpenSSL* (The OpenSSL Project, 2009)

# 6 Genomförande

Utvecklingen av låsenheten och mobilapplikationen genomfördes i linje med kravspecifikationen och kommunikationsprotokollet vilka beskrivs i avsnitt 4 Kravanalys och Design. Arbetet med de två enheterna förflöt parallellt tills de uppnådde tillräcklig funktionalitet vilket gjorde att det var möjligt att sammanfoga dessa till en prototyp.

## 6.1 Utveckling av mobilapplikationen

Utvecklingen av applikationen för Android med stöd för NFC har bedrivits i iterativ form. En funktion har lagts till i taget för att sedan testas. Vidare har utvecklingen av kommunikation till en början skett med en Android-enhet på var sida av länken. Detta eftersom applikationen behövde ha grundläggande funktionalitet när mikrokontrollern var redo att kommunicera men även med anledning av att det finns bättre verktyg för testning till Android än för mikrokontrollern. När mikrokontrollerns kommunikationsstack var redo övergick arbetet till kommunikation mellan smarttelefon och mikrokontroller.

Först utvecklades en applikation vars enda funktion var att skicka och ta emot information över NFC. För att implementera denna funktion användes *Android Beam* då det är den enda funktion moderna Android-enheter använder för att skicka och ta emot NDEF-meddelanden med en aktiv motpart. Funktionaliteten utarbetades för att följa kommunikationsprotokollet beskrivet i avsnitt 4.2.

När kommunikationen var på plats gick utvecklingen vidare för att lägga till personverifiering. Något sätt att verifiera att rätt person hade tillgång till mobiltelefonen, och därigenom upplåsningsnycklar till olika låsenheter, behövde läggas till. Med anledning av detta implementerades PIN-kod med en bakomliggande timer för att få tillgång till applikationens alla funktioner. Utöver det behövdes även funktionalitet vilken automatiskt loggar ut användaren när tiden har gått ut samt funktionalitet för att ändra PIN-kod implementeras.

När ovanstående personverifiering var på plats, var applikationen redo för att okrypterat låsa upp ett lås vars låsid och upplåsningsnyckel behövde hårdkodas in i källkoden. Då detta uppenbart inte är en skalbar lösning utvecklades en bakomliggande databas med ansvar för att hantera nyckelpar. En lokal relationsdatabas användes, då stöd för detta redan finns färdigt i Android. Implementationen kan göras effektivare med så kallad `Key-value store`, speciellt anpassad för ändamålet, men detta finns inte lättillgängligt i Android och används således inte. Applikationens relationsdatabas består av en tabell med låsenhetens id som primärnyckel och låsenhetens upplåsningsnyckel som andra kolumn. Databasen är av typen SQLite (SQLite, 2013).

Databasens funktioner integrerades i kommunikationsförloppet och en aktivitet med särskilt ansvar för att administrera databasen utvecklades. Aktiviteten kan lista, lägga till samt ta bort nyckelpar i databasen. Ett behov av en funktion för delning av nyckelpar, med andra Android-enheter, fanns också. Delningsfunktionen sker till stor del automatiskt, användaren väljer först vad som skall delas och därefter sker kommunikationen via NFC mellan mobiltelefonerna. Delningen slutförs genom bekräftelse av tillägg i databasen från mottagarparten.

När de ovanstående stegen slutfördes utvecklades en funktionellt fungerande applikation med kommunikationssäkerhet. Med hjälp av Javas API:s (Oracle Corporation, 2013b) säkerhetsbibliotek, vilka beskrivs i avsnitt 5.1.2, upprättades en RSA krypterad länk mellan två Android-enheter vilka exekverar den utvecklade applikationen. Detta gjordes då det var

enklare att upprätta denna länk mellan två applikationer skrivna i samma högnivåspråk., då Mikrokontrollern krävde ett annat källkodsbibliotek skrivet i C för att implementera kryptering. För att samma nyckelpar ska behållas genom hela kommunikationsförloppet behövde Android-aktivitetens livscykel iakttas noga, då olika händelser triggar återskapandet av aktiviteter och därigenom datan aktiviteten innehar.

## 6.2 Utveckling av låsenheten

Utvecklingen av låsenheten beskrivs lager för lager, nerifrån och upp genom hela den NFC-stack vilken har implementerats. NFC-stacken följer OSI-modellen, dock följs inte OSI-modellen rakt av då NFC-stacken har distribuerat ut funktionaliteten olika mellan OSI-lagren och vissa av dessa lager har slagits samman. De översta fyra lagren har implementerats i mjukvara vilka körs på mikrokontrollern då skölden endast har RF-protokollen implementerade. Då en mikrokontroller önskar nyttja RF-protokollen sker kommunikation mot skölden och hur denna funktionalitet implementerats ses i avsnitt 6.2.1.1.

Felhanteringen vilken implementerats i vardera lager är rigorös med felmeddelanden vilka klart och tydligt visar var felen uppstår och preciserar felets natur. Valet togs att låta upplåsning och konfigurationsförlopp starta om vid fel, då merparten av felen är svåra att parera på ett naturligt sätt. Beslutet togs också på grund av att om ett fel gjorde att ett förlopp avbröts skulle nästkommande kommunikationsförsök utgöra en säkerhetsrisk om det tilläts att återuppta det nuvarande förloppet.

### 6.2.1 RF-lagret

De protokoll, utvecklade av Adafruit Industries (2012), vilka specificerar den fysiska överföringen av NFC-meddelanden är implementerade i *PN532 NFC/RFID Shield* och utvecklades således inte. Dock krävs styrning av de funktioner vilka RF-protokollen erbjuder, såsom att skicka eller att ta emot meddelanden. Detta sker genom kommunikation mellan mikrokontrollern och skölden och vilka kommandon som skickas beskrivs i nedanstående avsnitt.

#### 6.2.1.1 Kontroll av PN532 NFC/RFID Shield

Skölden har en stor mängd kommandon vilka kan nyttjas och de som används av projektet beskrivs i avsnitt 3.4.2 (NFC-skölden *PN532 NFC/RFID Shield*). Källkods-
biblioteket *Embedded-PN532* implementerar de kommandon vilka ger den kontroll av skölden detta arbete söker, men de parametrarna vilka skickas med ett kommando har i viss grad modifierats. Detta gjordes i samband med att NPP byttes ut till förmån mot SNEP (se avsnitt 6.2.4).

De kommandon vilka anropas från att skölden startas upp tills dess att data utbytes med en aktiv NFC-enhet, beskrivs i figur 37 och förklaras i listan nedan. Observera att alla värden i figur 37 är hexadecimala.

| Handling | Kommando | Förklaring av kommando | Respons | Förklaring av respons |
|---|---|---|---|---|
| Få skölden att vakna upp samt göra den redo att ta emot andra meddelanden. | 14<br>01<br>00<br>01 | Kommando SAMConfiguration<br>SAM ska ej användas<br>Parameter används ej<br>IRQ ska användas | 15 | Responskod till kommandot. |
| Kommandot SAMConfiguration skickas endast vid uppstart av *PN532 NFC/RFID Shield*. | | | | |
| Skölden konfigureras som målenhet för kommunikation mot en aktiv NFC-enhet. | 8C<br>02<br>00 00 00 00<br>00 40<br>01 FE 00 00<br>00 00 00 00<br>00 00 00 00<br>00 00<br>AA 99 88 77<br>66 55 44 33<br>22 11<br>0D<br>46 66 6D<br>01 01 11<br>03 02 00 7A<br>04 01 10<br>00 | Kommando TgInitAsTarget<br>Använder endast DEP<br>Mifare parametrar<br><br>FeliCa parametrar<br><br><br><br>NFCID3t<br><br><br>Längd av generella byte<br>Magiska LLCP byte<br>TLV parameter VERSION<br>TLV parameter MIUX<br>TLV parameter LTO<br>Längd av historiska byte | 8D<br><br>25<br>1E D4 00<br>1F 26 3F AF<br>B2 A0 2D 83<br>1E 35<br>00 00 00 32<br>46 66 6D<br>01 01 11<br>03 02 00 13<br>04 01 96 | Responskod till kommandot.<br><br>Läge.<br><br>NFCID3i<br><br><br>RF-parametrar<br>Magiska LLCP byte<br>TLV parameter VERSION<br>TLV parameter MIUX<br>TLV parameter LTO |
| Skölden väntar nu på ett ATR_REQ kommando från en initiator, när detta fås skickas automatiskt meddelandet ATR_RES av skölden och enheterna är nu redo att utbyta data. | | | | |
| Skölden skickar följande data till initiatorn. | 8E<br>... | Kommando TgSetData<br>Utgående data | 8F<br><br>00 | Responskod till kommandot.<br><br>Status. 00 indikerar att inga error uppkom. |
| Skölden väntar på att ta emot data från initiatorn. | 86 | Kommando TgGetData | 87<br><br>00<br><br>... | Responskod till kommandot.<br><br>Status. 00 indikerar att inga error uppkom.<br><br>Ingående data |

Figur 37: Tabellen är en omarbetning från originalet skrivet av NXP Semiconductors (2010)

- När skölden först startas upp skickas kommandot `SAMConfiguration`, vilket gör skölden mottaglig för andra kommandon. Observera att kommandot `SAMConfiguration` endast anropas vid uppstart av skölden och anropas inte igen såvida inte skölden startas om.

- För att en ny uppkoppling ska ske mellan skölden och en aktiv NFC-enhet måste kommandot `TgInitAsTarget` anropas med de i figur 37 angivna parametrarna. Efter anropet av kommandot väntar skölden på ett meddelande av typen `ATR_REQ`[12] från en aktiv NFC-enhet, kallad initiatorn. När `ATR_REQ` inkommit sänds automatiskt ett `ATR_RES`[13] som svar till den initierande enheten. Sedan returneras det inkomna `ATR_REQ` till mikrokontrollern i svaret på `TgInitAsTarget`-kommandot. Ett exempel på ett `ATR_REQ` meddelande kan utläsas i figur 37 efter de inledande byten `0x8D` och `0x25`.

- Nu är skölden redo att utbyta data med den initierande enheten med hjälp av två

[12] Meddelandena `ATR_RES` och `ATR_REQ` beskrivs i avsnitt 3.1.1.4
[13] Samma som fotnot ovan.

kommandon, `TgSetData` och `TgGetData`, vilka sänder respektive tar emot data.

### 6.2.2 Kommunikationen mellan Arduino och NFC-sköld

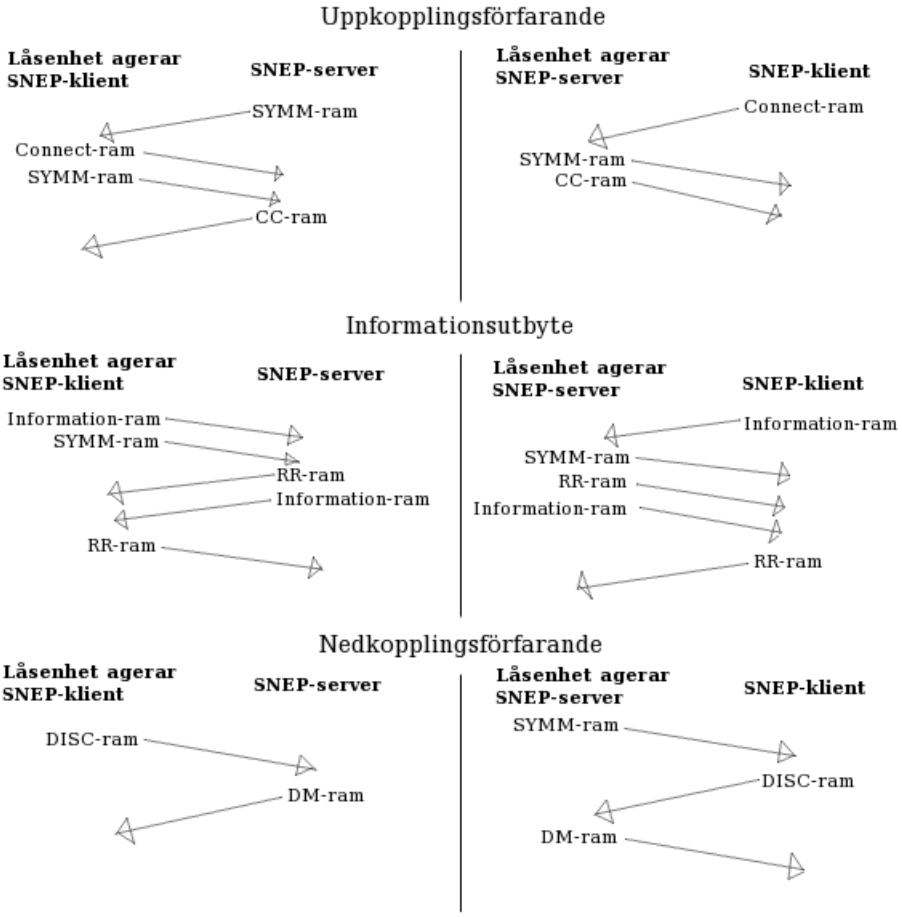
Källkodsbiblioteket *Embedded-PN532* implementerar kommunikation mot skölden via SPI, men valet att istället använda kommunikationstekniken I2C gjordes. Detta beslut togs för att skölden i grundutförande använder I2C och för att I2C är enklare att förstå då den har en högre abstraktionsnivå än SPI. I2C använder buffertar till ut- och ingående meddelanden vilka manipuleras med hjälp av olika funktioner och får på så sätt en högre abstraktionsnivå än SPI, som skickar och tar emot meddelanden direkt på kabeln, en byte i taget. Nackdelen med att använda I2C är att buffertarna, vilka är fem till antalet, kan ta relativt stor del av Arduino-plattformens begränsade SDRAM-utrymme. Standardstorleken på buffertarna är 64 byte vilket ger en storlek på 320 byte.

Att byta ut SPI mot I2C gjordes relativt enkelt då det källkodsbibliotek vilket medföljde *PN532 NFC/RFID Shield* för kommunikation mot en Arduino fanns i två versioner, där den ena bygger på SPI och den andra bygger på I2C. Vidare innehåller källkodsbiblioteken samma uppsättning funktioner, vilket gör att ett byte mellan versionerna enkelt då en funktion kan bytas mot en likvärdig funktion ur den andra versionen av källkodsbiblioteket. Dessvärre behöver I2C:s buffertar vara relativt stora då skölden inte klarar att ta emot segmenterade meddelanden. Då meddelanden av storleken 200 byte ska skickas, varav cirka 150 byte applikationsdata, valdes en buffertstorlek på 256 byte, vilket resulterar i en total SDRAM användning på 1280 byte. Storleken på *Arduino Mega 2560 R3* RAM-minne är 8 KB varvar 1 kbyte består av mikrokontrollerns minneshanterare och 2 KB består av de resterande delarna av programmet, togs beslutet att det finns utrymme för att använda I2C.

### 6.2.3 LLCP-lagret

Android version 4.0 implementerar LLCP (*Logic Link Control Protocol*), vilket beskrivs i avsnitt 3.1.1.3, något annorlunda än Android version 2.3, vilket *Embedded-PN532* är skrivet för. Detta medförde att befintlig kod för detta lager behövde modifieras. Information direkt från Google Inc. kring hur protokollet var implementerat i Android gick tyvärr inte att finna, men ett projekt kallat ismb-snep-java skrivet av Lotito (2013b) hittades för kommunikation mot Android 4.0 vilket nyttjar ett NFC-chip likt det som används i detta arbete.

Lotitos projekt ismb-snep-java är skrivet i programspråket Java och visar klart och tydligt hur LLCP skall användas för att både skapa en uppkoppling samt hantera denna mot en mobiltelefon med Android 4.0. Detta projekt användes därför som en mall vid modifieringen av LLCP-lagret i källkodsbiblioteket *Embedded-PN532*. Figur 38 visar hur enheterna nyttjar LLCP i Lotito (2013b) projektet då Android 4.0 används, och detta är också det förfarandet som *Embedded-PN532* skrivs om till att utföra. Observera att LLCP-ramarna utnyttjas olika beroende på om data ska skickas eller tas emot, det vill säga om låsenheten agerar SNEP-klient respektive SNEP-server.




Figur 38: Figuren beskriver hur LLCP ska användas i kommunikation mot en Android-enhet, då antingen en SNEP-`Put`-förfråga ska skickas eller tas emot.

De TLV-parametrar vilka utbyts med hjälp av `ATR_REQ` och `ATR_RES`, beskrivna i avsnittet 3.1.1.4, är VERSION, MIUX och LTO. Dessa beskrivs i avsnitt 3.1.1.3 och utformas enligt figur 37 i kolumn Kommando på de rader markerade med TLV-parametrar. Följande punktlista anger parametrarnas värden och en förklaring till dessa.

- **VERSION** fick värdet `0x11` då det är delar ur version 1.1 av LLCP vilket implementeras.
- **MIUX** sattes till `0x7A` då detta ger en MIU på 250 byte, vilket är tillräckligt stort för de meddelanden vilken utbyts.
- **LTO** sattes till `0x10` för att detta visade sig vara det minsta värde vilket kommunikationsförloppet kan fullföljas med. Ett lägre värde är önskvärt då det resulterar i ett snabbare kommunikationsförlopp.

### 6.2.4 NPP- eller SNEP-lagret

NPP (*NDEF Push Protocol*) är implementerat i *Embedded-PN532* men då NPP är ett protokoll vilket nu har blivit utbytt, samt att det inte är speciellt väldokumenterat vilket inses av Android Open Source Project (2011), gjordes valet att byta ut detta till förmån för SNEP (*Simple NDEF Exchange Protocol*). Det nya protokollet SNEP, vilket beskrivs i avsnitt 3.1.1.2, är i förhållande till NPP rättfram och enkelt att förstå samt väldokumenterat. Detta val gjordes även då det i kommunikation mot Android fortfarande går att använda NPP då Android-enheten faller tillbaka på det gamla protokollet om den känner av att det nya protokollet SNEP inte kan användas (Android, 2013f).

Eftersom data utbyts i båda riktningarna i kommunikationen gjordes valet att ge låsenheten funktionalitet att agera både som SNEP-klient och SNEP-server. Detta betyder att låsenheten både kan skicka samt ta emot SNEP-förfrågor. Den SNEP-förfråga för vilket stöd implementerades för är `Put`, vilken beskrivs i avsnitt 3.1.1.2 där också SNEP förklaras.

Då låsenheten agerar som SNEP-klient skickas en Put-förfråga enligt figur 39 där fältet Version har värdet `0x10` vilket anger att SNEP version 1.0 används. Vidare innehåller fältet `Request` värdet `0x02` vilket är den specifika koden som anger att detta är en `Put`-förfråga och `Length` innehåller storleken av det NDEF-meddelande vilket förekommer i fältet `Information`. Om överföringen lyckades kommer SNEP-responsen `Success` skickas av servern vilket indikerar att `Put`-förfrågan genomfördes korrekt, annars sänds en SNEP-respons vilket indikerar vilket fel som uppkom. Valet gjordes att betrakta alla SNEP-responser utom `Success` som felaktiga vilket medför att förloppet börjar om då dessa mottages.

| SNEP Put-förfråga | | | |
|---|---|---|---|
| SNEP-Header | | | Information |
| Version | Put | Length | |
| 1 byte | 1 byte | 4 byte | n byte |

Figur 39: Figuren visar de värden vilka används i en `Put`-förfrågan.

När låsenheten agerar server väntar låsenheten på att ta emot en `Put`-förfråga och då denna mottagits undersöks om förfrågans SNEP-version stödjs. Om versionen stödjs tas det mottagna NDEF-meddelandet om hand, vilket förekommer i fältet `Information`, och SNEP-responsen `Success` skickas till klienten. Om versionen inte stödjs eller något annat fel uppkommer avbryts kommunikationen och förloppet börja om.

### 6.2.5 NDEF-lagret

NDEF (*NFC Data Exchange Format*), vilket beskrivs i avsnitt 3.1.1.1, är implementerat i *Embedded-PN532* och källkodsbiblioteket innehåller de funktioner vilket arbetet eftersöker. Detta medför att NDEF-lagret inte utvecklades vidare. I teoriavsnitt 3.1.1.1 beskrivs NDEF och nedan beskrivs hur NDEF användes för projektets ändamål.

Eftersom en mindre mängd applikationsdata utbyts består de NDEF-meddelanden vilka skickas mellan låsenhet och applikation av endast en `record` utformad enligt figur 40. För att visa att endast ett `record` är inkapslat i NDEF-meddelandet är flaggorna `MB` (*Message begin*) och `ME` (*Message end*) satta till `0x1`. Då mindre än 255 byte behöver skickas i detta `record` sätts `SR` (*Short record*) till `0x1` vilket indikerar att maximalt 255 byte kan rymmas i `Payload`

fältet. Fältet `TNF` (*Type name format*) sätts till `0x02` vilket står för media type. I `Type`-fältet skrivs strängen text/plaindå ASCII formaterad text placeras i `Payload`-fältet.

| NDEF-record | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| MB | ME | CF | SR | IL | TNF | Type length | Payload length | Type id | Payload |
| 0x1 | 0x1 | 0x0 | 0x1 | 0x0 | 0x02 | 0x0A | n | text/plain | applikationsdata |
| 1 bit | 1 bit | 1 bit | 1 bit | 1 bit | 3 bitar | 1 byte | 1 byte | 0x0A byte | n byte |

Figur 40: Figuren visar de värden vilka används i ett NDEF-record.

Då låsenheten agerar SNEP-server behandlas det mottagna NDEF-meddelanden av en kontroll då det mottagna meddelandet skall vara utformat precis som visas i figur 40 och om avvikelser förekommer släpps meddelandet. Om låsenheten agerar SNEP-klient byggs ett NDEF-meddelande upp innehållandes ett `record` beskrivet i figur 40 där applikationsdatan vilken ska skickas placeras i det avsedda `Payload`-fältet, och sedan skickas meddelandet inkapslat i en SNEP-förfråga till en SNEP-server.

### 6.2.6 Mjukvara med inbyggd säkerhet

Mjukvaran är ämnad att efterlikna en mobilapplikation vilken brukar *Android Beam* och dess uppgifter är att implementera de utarbetade kommunikations-förloppen (se avsnitt 4.2.2) samt applicera kommunikationssäkerhet (se avsnitt 4.2.3). I avsnitt 6.2.1 till 6.2.5 beskrivs den NFC-stack vilken implementerats och i detta avsnitt beskrivs hur den implementerade NFC-stackens funktionalitet används för att implementera kommunikationsförloppen.
Säkerheten applicerades genom att följa det krypteringsförlopp vilket är beskrivet i avsnitt 4.2.3.

Upplåsningsyckeln till låsenheten placeras på EEPROM, ett icke flyktigt minne. När låsenheten först startas upp har den en fördefinierad nyckel vilken kan konfigureras med hjälp av mobilapplikationen. Då en konfiguration av nyckeln sker byts den gamla nyckeln ut. Genom att bruka EEPROM påverkas inte den ändrade nyckeln.

Kommunikationsförloppen vilka mjukvaran ska implementera är *Upplåsningsförlopp* och *Konfigureringsförlopp*. Dessa två förlopp är uppbyggda av sändningar och mottagningar av ett antal SNEP-förfrågor och kan således byggas upp av två mindre delförfaranden bestående av att antingen sända en SNEP-förfråga eller att ta emot en SNEP-förfråga. Hur de två delförfaranden implementeras beskrivs i avsnitt 6.2.6.1 och hur kommunikationsförloppen implementeras beskrivs i avsnitt 6.2.6.2.

#### 6.2.6.1 Sändnings- och mottagningsförfarande av en SNEP-förfråga

I sändningsförfarandet agerar låsenheten SNEP-klient och bygger först en `Put`-förfråga enligt avsnitt 6.2.4 innehållande ett redan byggt NDEF-meddelande. Om så erfordras går nu låsenheten ner i strömsparläge[14]. När en mobiltelefon, vilken kör den utvecklade mobilapplikationen, är inom kommunikationsavstånd går låsenheten ur strömsparläge och en uppkoppling, enligt avsnitt 6.2.3, etableras mellan enheterna på vilken förfrågan skickas. Låsenheten väntar sedan på att SNEP-responsen `Success` skickas från

[14] Att använda ett strömsparläge kan vara fördelaktigt då det kan ta en längre tid innan en mobiltelefon önskar påbörja ett kommunikationsförlopp.

mobilapplikationen vilket indikerar att sändningen var lyckad. Då en SNEP-respons har tagits emot stängs uppkopplingen ner. Om den mottagna SNEP-responsen inte är av typen `Success` returneras ett felmeddelande som, om så erfordrats, skrivs ut på serial-porten.

Mottagningsförfarandet implementerades genom att låsenheten agerar SNEP-server. Först väntar låsenheten på att en mobiltelefon, vilken kör den utvecklade mobilapplikationen, är inom kommunikationsavstånd. Här går dock låsenheten inte ned i strömsparläge eftersom mottagningsförfarandet är beläget mitt i de pågående förloppen (se 4.2.2 Kommunikationsförlopp). När en mobiltelefon är inom kommunikationsavstånd upprättas en uppkoppling enligt avsnitt 6.2.3 och en `Put`-förfråga skickas till låsenheten. NDEF-meddelandet i förfrågan genomgår en kontroll enligt avsnitt 6.2.5 och om meddelandet godkänns placeras den i ett fält för att kunna användas senare, samt att låsenheten skickar SNEP-responsen `Success` till klienten. Sedan stängs uppkopplingen ner enligt avsnitt 6.2.3 och meddelandet returneras till den brukande applikationen på mikrokontrollern, men om meddelandet inte godkändes i kontrollen returneras istället ett felmeddelande till mikrokontrollerns brukande applikation.

#### 6.2.6.2 Upplåsnings- och konfigureringsförlopp

Till en början bygger låsenheten ett NDEF-meddelande enligt avsnitt 6.2.5 innehållande ett meddelande av typ 1 (se avsnitt 4.2.1). Sedan utförs ett sändningsförfarande med det byggda NDEF-meddelandet. I detta sändningsförfarande specificeras mikrokontrollern att gå ner i strömsparläge.

Sedan utförs ett mottagningsförfarande. Efter detta dekrypteras den känsliga informationen och meddelandet verifieras enligt krypteringsförloppet (se avsnitt 4.2.3). Om meddelandet är korrekt tänds en grön diod vilket indikerar att användaren har rätten att använda låsenheten, men annars tänds en röd diod vilket indikerar att användaren inte har rätten att använda låset. Det mottagna meddelandet är antingen av typ 2 eller 5 (se avsnitt 4.2.1) och om meddelandet är av typ 2 ges en signal på en pin som går till ett eventuellt lås. Är meddelandet däremot av typ 5 ändras låsnyckeln till det som specificeras i meddelandet.

Nästa steg är att låsenheten ger feedback till mobilapplikationen. Detta görs villkorligt beroende på om smarttelefonen befinner sig inom kommunikationsavstånd. Om smarttelefonen inte befinner sig inom kommunikationsavstånd utförs inte sändningsförfarandet och pågående förlopp avbryts. Befinner sig smarttelefonen inom kommunikationsavstånd utförs sändningsförfarandet i vilken ett meddelande av typ 3 sänds. I detta sändningsförfarande går låsenheten inte ner i strömsparläge. Mjukvaran implementerar slutligen en fördröjning vilken gör att dioden förblir tänd i tio sekunder samt att utsignalen till eventuellt lås kvarstår under 10 sekunder.

### 6.2.7 Fysisk implementering

Den fysiska delen av arbetet, det vill säga byggandet av prototypen utgjorde en mindre del av projektet. Detta föreföll naturligt då en färdigbyggd NFC-sköld och mikrokontroller från Arduino användes. Skölden krävde en mindre mängd förarbete vilket bestod av att kontaktpinnar löddes fast. Med pinnarna på plats monterades skölden på mikrokontrollern.

Då låsenheten skall styra någon form av elektroniskt lås konstruerades en diodkrets vilken är ämnad att ge feedback till användaren. Diodkretsen består av en grön och röd diod med tillhörande resistanser som är ämnade att begränsa strömmen. Om Android-applikationen sänder ett korrekt utformat meddelande av typ 2 eller 5 (se avsnitt 4.2.3) och följaktligen har en tillgång till låset visas detta genom att den gröna dioden tänds. Om mobilapplikationen

sänder ett felaktig meddelande, och följaktligen inte har tillgång till låset tänds den röda dioden.

# 7 Resultat

Detta avsnitt ämnar beskriva resultaten av kandidatarbetet samt relatera dessa till den från början uppsatta kravspecifikationen (se Appendix A: Kravspecifikation). Såväl skallkrav som börkrav från kravspecifikationen behandlas.

## 7.1 Produkten som helhet

Projektet har resulterat i ett låssystem bestående av en mikrokontroller vilken exekverar mjukvara skriven i C++ och en Android-applikation skriven i Java. Android-applikationen kommunicerar mot låsenheten med hjälp av Androids egna API för NFC-kommunikation, kallat *Android Beam*. Låsenheten kommunicerar mot applikationen genom att imitera en Android-enhet vilket görs med hjälp av den implementerade kommunikationsstacken. Kommunikationen sker enligt det uppsatta kommunikationsprotkollet (se avsnitt 4.2) och kommunikationen är krypterad med 512 bitars RSA.

De krav (se Appendix A: Kravspecifikation) vilka uppfylls av prototypen som helhet listas nedan[15].

- (1) Endast användare med den rätta nyckeln till låsenheten kan agera mot låsenheten.
- (2) Kommunikation mellan mobilapplikation och låsenheten sker via NFC.
- (3) Känslig information överförs krypterat.
- (6) Låsenhetens nyckel kan ändras med hjälp av mobilapplikationen.
- (7) Flera användare kan låsa upp samma lås då den specifika nyckeln kan delas via *Kommunikationsförlopp vid distribution av nycklar* (se avsnitt ref. 4.2.2).

De krav vilka inte uppfyllts inom detta område är:

- (4) (5) Upplåsning tar sex sekunder och inte under fem sekunder vilket var det lägsta kravet.

De mobiltelefoner vilka låssystemet har testats och garanterat fungerar med är HTC One X, Samsung Galaxy S3 (GT-I9300) och LG Optimus L5 (E 610). Utöver dessa borde alla mobiltelefoner med Android version 4.0 eller högre fungera att använda i låssystemet då de innehar samma NFC-funktionalitet (Google Inc, 2013).

## 7.2 Mobilapplikation

Arbetet med Android-applikationen (Tapper m. fl., 2013b) har resulterat i en funktionell och stabil. Mobilapplikationen uppfyller alla applikationsspecifika skallkrav samt majoriteten av de börkrav vilka togs fram under kravanalysen. Nedan följer en lista av de krav[16] applikationen möter och inte möter utifrån kravspecifikationen (se Appendix A: Kravspecifikation). Listan följs av en mer ingående beskrivning av applikationen.

[15] En siffra inom parantes korresponderar direkt mot det kravet i kravspecifikationen vilket har samma nummer.

[16] se fotnot 15.

- (6) Applikationen har möjlighet att konfigurera låsenheten.
- (8) Applikationen följer Jacob Nielsens heuristik (Nielsen, 1995).
- (9) Applikationen är noga testad och har inga kända buggar.
- (10) Applikationen fungerar inte på alla Android-enheter med NFC. Detta eftersom från och med Android 4.0 ändrades NFC:s kommunikationsstack radikalt. Utvecklingen gick från en av Android egenkomponerad kommunikationsstack till en av NFC Forum (2011b) definierad standard. Därför har det valts att endast utveckla stöd för den senare och därför stödjs endast Android version 4.0 eller senare.
- (11) Nycklar till låsenheter kan delas med andra applikationer över NFC.
- (12) Applikationen kräver personverifiering med hjälp av PIN-kod.

Under utvecklingsfasen av applikationen har det även gjorts observationer angående NFC och dess kommunikationssnabbhet. Ett NDEF-meddelande från en mobiltelefon till en annan går att skicka nästintill ögonblickligen. Om den mottagande mobiltelefonen önskar skicka ett meddelande tillbaka krävs dock att mobiltelefonerna separeras så att ingendera telefon detekterar något RF-fält. Först då kan telefonerna föras inom kommunikationsavstånd och svarsmeddelande skickas.

Applikationen (Tapper m. fl., 2013b) består av över 2000 rader Java-kod och till det över 500 rader XML-definierade gränssnitt. Vidare består den av 16 klasser uppdelat på fem paket. Av klasserna är åtta aktiviteter vilka beskrivs närmare i avsnitt 7.2.1.

### 7.2.1 Design

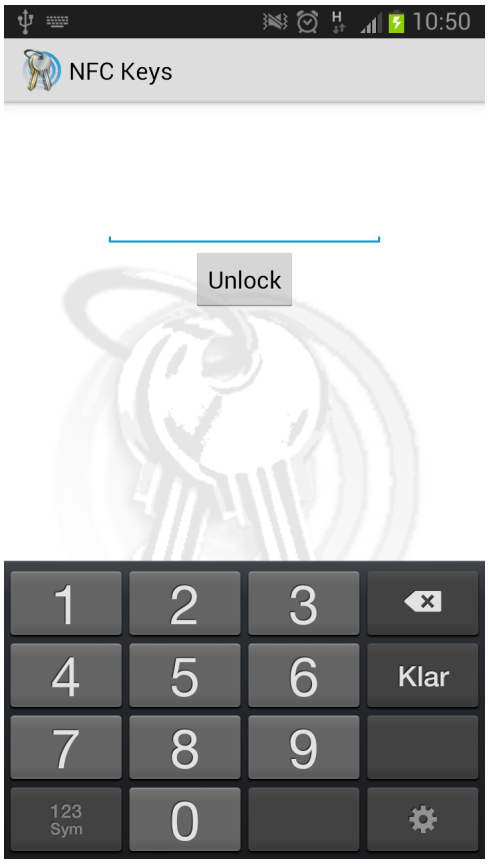
Applikationen är skapad för att enkelt kunna tillhandahålla de funktioner användaren behöver. Med en enkel design uppstår inga onödiga problem för användaren och med återkommande typsnitt, färger och plattformskonventioner uppstår en trygghet. För att anknyta designen till tidigare ställda krav har utvecklingen följt Jacob Nielsens heuristik (se avsnitt 3.3.2). Dessa tumregler har till stor del följts och nedan beskrivs hur de har applicerats för att öka användarvänligheten:

- Användaren har en överblick över systemets status; figur 3 visar ett exempel på den feedback användaren får från låssystemet. Enkla bilder beskriver vad som ska göras eller vad som har hänt.
- Standarder och givna konventioner har följts; användaren känner igen knappar och behöver inte oroa sig för vad olika funktioner gör, då detta är allmänt känt.
- Felmeddelanden har försökts att undvikas när dessa inte varit nödvändiga, detta då det är bättre att få användaren att undvika misstag än att ha bra felmeddelanden.
- Direkta instruktioner ges till användaren på enkel engelska; när instruktioner visas på skärmen är dessa alltid aktuella och användaren behöver inte minnas tidigare givna instruktioner.

- Enkel design utan onödig information; användaren ska enkelt förstå och kunna använda applikationen. Förhoppningsvis tilltalar även den enkla designen användaren rent estetiskt.

Menyer visas som *action bars* för att öka synligheten. Figurerna nedan visar i tur och ordning aktiviteterna `Login`, `Main`, `Success`, `Settings`, `Password` och `Keys`. Följande avsnitt har för avsikt att beskriva ovannämnda aktiviteter och dess funktion.

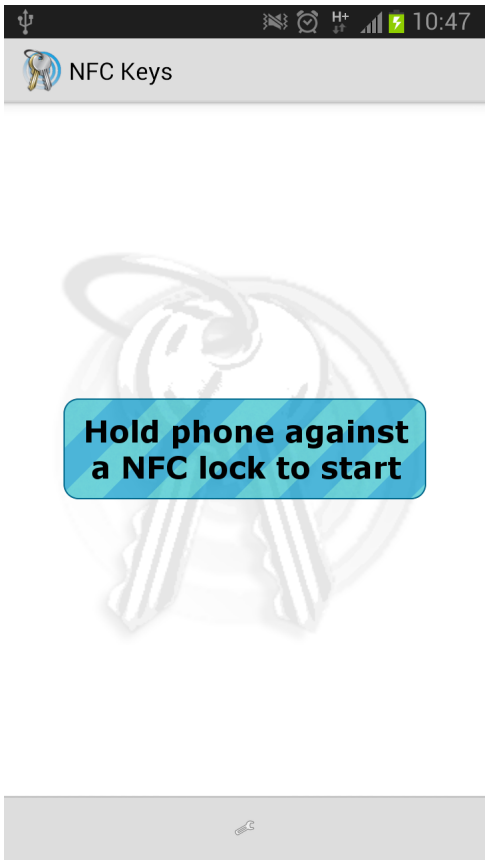
**7.2.1.1 Login** Login (se figur 41), är den första aktiviteten användare av Android-applikationen möts av. Den huvudsakliga funktionen för denna aktivitet är att förhindra otillåten användning av applikationen. Aktiveten implementerar ett system för personverifiering med hjälp av PIN-kod vilken fungerar som en port till övriga delar av applikationen. Ges inte rätt PIN-kod går inte övriga delar av applikationen att nå. När korrekt PIN-kod anges loggas användaren in och en timer startas, timern loggar ut användaren automatiskt efter en bestämd tid.




Figur 41: Figuren illustrerar Login.

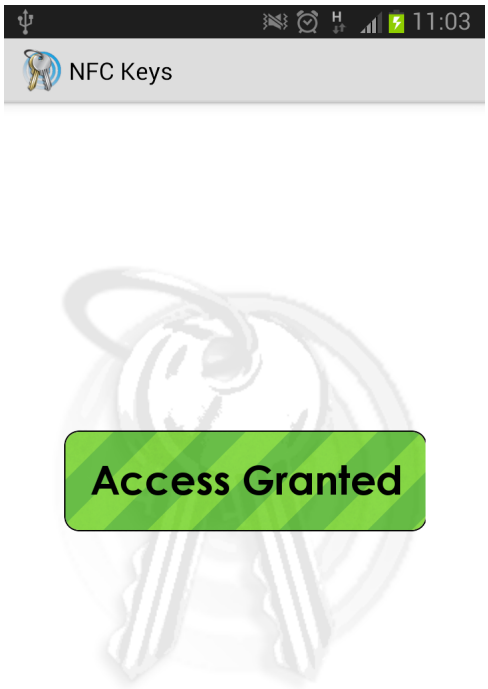
**7.2.1.2 Main** När rätt PIN-kod angivits i Login-aktiviteten möts användaren av nästa aktivitet, Main, vilken illustreras av figur 42. Hela applikationen bygger runt denna aktivitet och det är således även applikationens största aktivitet. Den är ansvarig för NFC-kommunikationen och att automatiskt skicka rätt NDEF-meddelande baserat på det senast

mottagna NDEF-meddelandet. Detta sker enligt ovanstående kommunikationsprotokoll (se avsnitt 4.2). Då rätt nyckel, till låsenheten, används vid ett fullbordat kommunikationsförlopp skickas användaren till Success-aktiviteten. På motsvarande sätt, då felaktig nyckel används, visas Denied-aktiviteten istället. Då det inkommande meddelandet är delning av nyckel krävs en bekräftelse från användaren innan nyckeln adderas till databasen för nyckelpar.



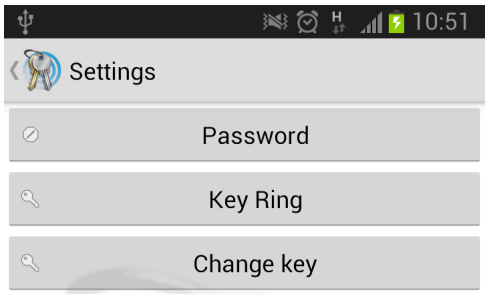

Figur 42: Figuren illustrerar Main.

**7.2.1.3 Success och Denied** `Success`- och `Denied`-aktiviteterna har till uppgift att visa huruvida ett kommunikationsförlopp fullbordats eller ej. `Success`-aktiviteten enligt 43, visas för användaren då upplåsning eller konfiguration lyckats. Vid ett anrop visas aktiviteten bara en kortare stund för att sedan avslutas och applikationen går tillbaka till `Main`-aktiviteten. Utöver `Success` finns den motsvarande aktiviteten `Denied`. `Denied`-aktiviteten har till uppgift att visa att ett upplåsning eller konfigurationsförfarande misslyckats



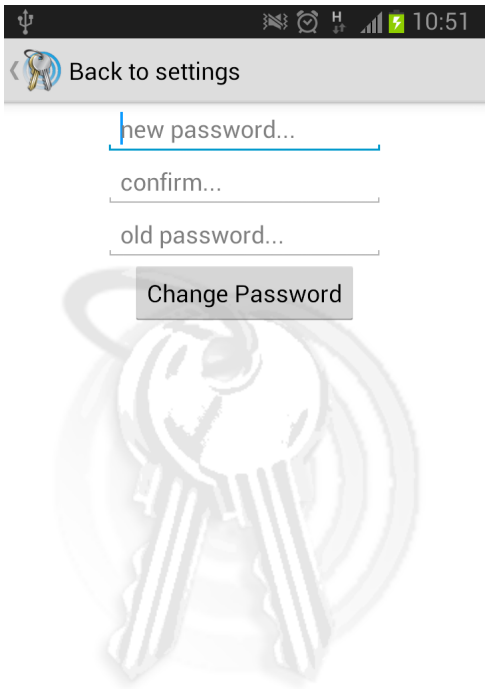

Figur 43: Figuren illustrerar Success.

**7.2.1.4 Settings** `Settings`-aktiviteten nås via knappen längst ner i Main-aktiviteten, som visas i figur 44. Aktiviteten har till uppgift att rada knappar för de inställningar vilka kan ändras. Prototypen innehåller tre inställningar men aktiviteten kan expanderas för att rymma ytterligare knappar. Då användaren trycker på en knapp öppnas motsvarande aktivitet: `Password` eller `Keys`. Den tredje knappen öppnar en dialogruta, enligt figur 44, vilken låter användaren administrera nyckeln hos låsenheten genom att ändra meddelandetypen.




Figur 44: Figuren illustrerar Settings.

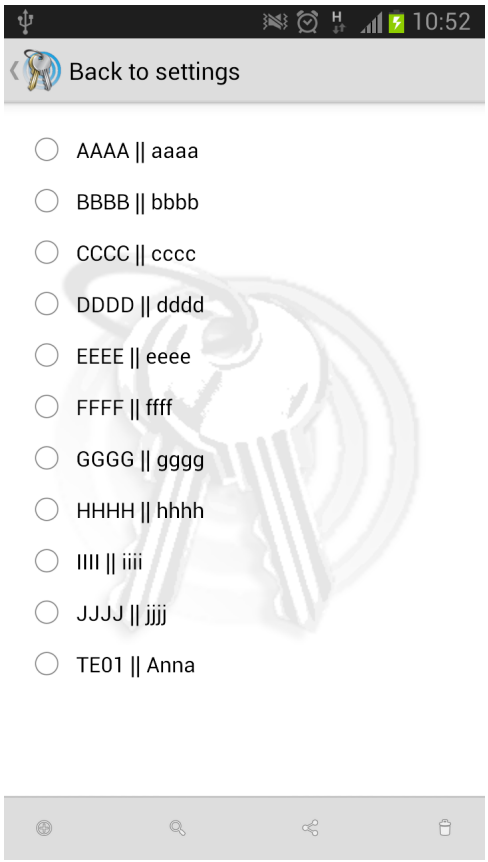
**7.2.1.5 Password** Denna aktivitet, vilken visas i figur 45, har den enkla uppgiften att ändra PIN-kod. För att ändra PIN-kod måste den nuvarande koden anges innan en ny PIN-kod sparas. En korrekt utformad PIN-kod måste vara minimalt fyra siffror lång.




Figur 45: Figuren illustrerar Password.

**7.2.1.6 Keys** Aktiviteten Keys, vilken illustreras av figur 46, har till uppgift att hantera nyckelpar. Bakom aktiviteten ligger en lokal relationsdatabas av SQLite-typ (SQLite, 2013) vilken är ansvarig för att lagra nyckelpar. Aktiviteten ger ett ovanliggande gränssnitt till följande databasfunktioner: lista alla nyckelpar, lägga till eller ta bort ett nyckelpar, söka efter nyckelpar och dela nycklar med andra smarta telefoner vilka har applikationen installerad.

Aktiviteten visar de nyckelpar, vilka finns i databasen, i en lista. Härifrån används även funktionerna `create`, `search`, `share` och `delete` som visas i en *Actionbar* längst ner i fönstret. Dessa funktioner är till för att hantera nyckelpar i databasen. När `create`, `search` och `delete` används visas dialogrutor där ytterligare information visas. `Delete` och `share` kräver att ett specifikt nyckelpar markeras i listan innan de går att använda. Skilt från de andra funktionerna skickar share vidare användaren till en ny aktivitet vilken delar nyckelparet med en annan telefon, vilken har applikationen installerad, via NFC-kommunikation.




Figur 46: Figuren illustrerar Keys.

## 7.3 Låsenhet

Den utvecklade låsenheten implementerar nödvändiga delar utav NFC:s kommunikationsstack vilka möjliggör kommunikation via *Android Beam* och detta behandlas mer i detalj i avsnitt 7.3.1. Kommunikationsförloppet har tidtagits och resultatet presenteras i avsnitt 7.3.2. Mer generella resultat och om kraven vilket kravspecifikationen specifikt ställer på låsenheten har uppfyllts behandlas nedan.

Utfallet av de krav vilka kravspecifikationen specifikt ställde [17] mot låsenheten presenteras i följande punktlista.

- (13) Låssystemet är nästintill felsäkert. Den ända kända buggen är att låsenheten ibland fastnar i strömsparläge då ett RF-fält kommer i kontakt med låsenheten precis då låsenheten går ner i strömsparläge. Vidare förblir låset låst då strömavbrott inträffar.
- (14) Formfaktorn har inte anpassats men låsenheten är liten nog för att kunna installeras i en testmiljö, dock är den inte liten nog för att installeras i till exempel en dörrs låshus.
- (15) Låsenheten uppfyller inte kravet då den förbrukar nominellt $0,55$ watt under normal användning, vilket anses vara en för hög effekt. I den normala användningen har

[17] En siffra inom parantes korresponderar direkt mot det kravet i kravspecifikationen vilket har samma nummer.

kommunikationsförloppen försummats då dessa endast utförs ett fåtal gånger per dygn. Under ett kommunikationsförlopp förbrukas $1,8$ watt nominellt.

Vidare följer en lista i vilken övriga resultat presenteras.

- Låsenheten stödjer dekryptering med hjälp av kryptosystemet RSA och krypteringsnycklarna utgörs av 512 bitar
- En diodkrets har utformats, vilken beskrivs i avsnitt 6.2.7, med syfte att ge feedback till användaren angående hur förloppen fortskridits.
- Låsenheten stödjer kommunikation mot skölden *PN532 NFC/RFID Shield* via I2C samt implementerar de kommandon vilka behövs för att skölden ska stödja aktiv NFC-kommunikation. De kommandon vilka implementeras är en delmängd av sköldens alla kommandon och alla dessa omfattas av avsnitt 3.4.2.
- All den kod vilket omfattar mikrokontrollerns drivrutiner (Tapper m. fl., 2013a)har resulterat i 2500 rader C++ kod. Kompilerat och uppladdat på mikrokontrollern tar detta 28880 byte flash av de totalt 256 kB flash-minne vilket finns tillgängligt. Användandet av SDRAM har kontrollerats under exekvering och har uppgått till maximalt 4209 byte av 8 kB.

### 7.3.1 Den implementerade NFC-stacken

Låsenheten erbjuder stöd för aktiv NFC-kommunikation. Stödet utgörs av en NFC-stack bestående av minimalistiska implementationer av NDEF, SNEP och LLCP. De protokoll vilka ingår i NFC-stacken men inte är implementerade i stacken är RF-protokollen, då dessa hanteras av `PN532 NFC/RFID Shield`. De avkall vilka har gjorts i implementationen har resulterat i ett antal begränsningar av stackens funktionalitet, vilka tas upp i avsnitt 8.3.3). Dock hanterar den implementerade NFC-stacken all den funktionalitet vilket låssystemet kräver.

Punktlistan nedan beskriver den funktionalitet vilket varje lager i den implementerade NFC-stacken erbjuder. Denna funktionalitet har erhållits dels genom modifiering av *Embedded-PN532* (Wier, 2012) men också av större tillskott egenproducerad kod.

- **NDEF.** Detta lager hanterar formatering av NDEF-meddelanden, vilka endast kan innehålla en `record` och det `record` är av karaktären short `record`.
- **NPP eller SNEP.** Protokollet NPP har ersatts med SNEP, vilket är implementerat efter delar ur NFC-Forums specifikation (NFC Forum, 2011b). Implementationen stödjer funktionalitet som SNEP-server, SNEP-klient samt sändning och mottagning av *Put*-förfråga.
- **LLCP** LLCP-relaterad kod har modifierats till den grad att endast vissa metodnamn kan tillskrivas Wier (2012). Kod vilken lagts till hanterar konstruktion av specifika ramar, mer korrekt uppkopplingshantering (vilket inkluderar hantering av SAP och sekvensnummer) samt metoder vilka sköter kommunikationsförloppen mer autonomt. Lagret stödjer uppkopplingsetablering, informationsutbyte och nedkoppling. TLV-parametern MIUX är satt till `0x7A` vilket ger att 250 bytes skickas samt tas emot varav 226 byte

applikationsdata. TLV-parametern LTO har satts till `0x10` vilket ger en Link Time Out på 160ms, vilket är tillräckligt för att även en av de långsammare Android-baserade mobiltelefonerna (LG Optimus L5) ska kunna genomföra kommunikationsförloppen.

- **RF-protokollen** Den funktionalitet vilket kan nyttjas utav RF-protokollen är konfiguration av NFC-chippet till att använda aktiv kommunikation samt sändning och mottagning av meddelanden vilka skickas via aktiv kommunikation.

### 7.3.2 Tidtagning av kommunikationsförlopp

I detta avsnitt presenteras resultatet av tidtagningar på kommunikationsförloppen beskriven från låsenhetens perspektiv, vilka visas i figur 47. Observera att bara ett av de fyra nedan markerade alternativa förlopp med en asterix kan inträffa. Samt att förloppet är tidtaget då användaren från början varit inloggad på mobilapplikationen och att den under hela förloppet finns inom kommunikationsavstånd. Nedan följer en kort redogörelse för varje handling vilka utgör denna tabell.

| Tidtagning av kommunikationsförlopp | |
|---|---|
| Handling | Tid (ms) |
| Formatering av NDEF-meddelande | 14 |
| Sändningsförfarande | 1318 |
| Mottaningsförfarande | 1688 |
| Deformatering av NDEF-meddelande | 1 |
| Dekryptering | 3486 |
| Lyckad upplåsning* | 0 |
| Misslyckad upplåsning* | 0 |
| Lyckad konfigurering* | 14 |
| Misslyckad konfigurering* | 0 |
| Sändningsförfarande | 1728 |
| Totalt | 8264 - 8278 |

Figur 47: I tabellen visas tidtagningen i kommunikationsförloppet.

- Formatering av NDEF-meddelandet inkluderar läsande av aktuell (det vill säga antingen förinställd eller användarspecificerad) nyckel, skapande av ett meddelande typ 1 enligt avsnitt 4.2.1.
- Sändnings- och mottagningsförfarandena utförs enligt avsnitt 6.2.6.1.
- Avdemontering NDEF-meddelandet utförs enligt avsnitt 6.2.5.
- Dekryptering utförs enligt det förfarande vilket finns beskrivet i avsnitt 4.2.3.
- De alternativa förloppen inkluderar kontroll av innehåll i demonterat NDEF-meddelande och efterföljande jakning eller nekning på förloppet. Det är enbart lyckad konfigurering vilket tillför ytterligare tid till förloppet. Detta kan härledas till upprepade accesser av EEPROM när den nya nyckeln läggs in.

Från figuren framgår det att det tar $8,2$ sekunder att genomföra ett helt kommunikationsförlopp. Detta är längre än de 5 sekunder vilket specificeras i kravspecifikationen (se Appendix A: Kravspecifikation). Dock kan denna tid minskas om användaren avstår från den frivilliga feedbacken, vilket då resulterar i att ett kommunikationsförlopp tar 6.5 sekunder. Reflektioner kring hur sändnings- och motagningsförfarandet kan förbättras presenteras i avsnitt 8.1.1, medan reflektioner relaterade till hur krypteringen kan förbättras presenteras i avsnitt 8.1.2.

# 8 Diskussion

Kandidatarbetets syfte var att redogöra för hur ett låssystem baserat på NFC kan designas och implementeras. I början av arbetet identifierades tre huvudutmaningar (se avsnitt 1.3) vilka tillsammans ger säker kommunikation via NFC mellan en Android-enhet och en Arduionobaserad mikrokontroller. Tillsammans utgör enheterna ett låssystem. För att övervinna dessa utmaningar har en omfattade studie av relevanta tekniker gjorts för att sedan användas i ett konstruktionsinriktat projekt.

Arbetet angreps på ett iterativt sätt och flöt på enligt förväntan när det gällde Androidapplikationen. Vid utvecklande av mjukvaran för mikrokontrollen uppenbarade sig mer problem än vad som förutsetts vilket resulterade i att slutprodukten inte har all den funktionalitet och inte uppfyller alla de krav vilka sattes upp i början av projektet. Men syftet med projektet är trots allt att konstruera en prototyp.

Under både den initiala fasen och designfasen av kandidatarbetet antogs naivt att det källkodsbibliotek (Adafruit Industries, 2012) vilket medföljde NFC-skölden *PN532 NFC/RFID Shield* implementerade den NFC-stack vilket arbetet sökte. En tid senare när källkodsbiblioteket *Embedded PN532* (Wier, 2012) upptäcktes begicks samma misstag igen, då vi trodde att detta bibliotek hade en funktionell implementation av NFC-stacken. Dessvärre visade det sig att inte heller detta var sant och det insågs att projektets fokus skulle komma att ligga på att utveckla en fungerande NFC-stack, istället för att utveckla den funktionellt berikade prototyp vilket var tanken från början. På grund av att en fungerande kommunikationsstack var essentiellt för projektet kunde fokus inte läggas på börmål 13, 14 eller 15 (se Appendix A: Kravspecifikation) och dessa har således inte uppfyllts.

## 8.1 Prototypen som helhet

I det här avsnittet diskuteras prototypens funktionalitet och hur förbättringar skulle kunna åstadkommas. Prototypen har konstruerats utefter den funktionalitet vi önskat att se i ett system av denna karaktär. Kommunikationsprotokollet har utvecklats så att kommunikationsförloppet mellan mobilapplikation och låsenhet är användarvänliga. Asymmetrisk kryptering är även implementerad för att garantera säker kommunikation. I efterhand hade ett annat kryptosystem varit att föredra vilket diskuteras ingående i avsnitt 8.1.2. I sin helhet uppfyller prototypen den funktionalitet vi önskat, men alla krav som ställts är inte helt uppfyllda och vissa har utelämnats.

På grund av skallkravet angående upplåsningstiden (se Appendix A: Kravspecifikation) inte uppfylldes beskrivs det i avsnitten 8.1.1, 8.1.2 och 8.1.3 teoretiska lösningar för att nå kravet. Enligt figur 47 framgår det att de handlingar i vilka tid kan sparas berör krypteringen eller informationsutbytet och nämnda avsnitt behandlar dessa var för sig. Om dessa tidsbesparande åtgärder realiseras tror vi att en lyckad upplåsning kan göras på 1.5 sekunder, eller 3 sekunder om feedback till mobilapplikationen önskas.

En del av syftet med projektet var att skapa ett låssystem som var mer flexibelt än ett vanligt lås. Detta anser vi till viss del är uppfyllt då viss konfigurerbarhet erhålles då applikations PIN-kod kan ställas in likaså kan låsenhetens upplåsningsnyckel väljas av användaren. Dock så önskas att användaren skall besitta större möjligheter att konfigurera systemet. Exempel på önskvärda konfigureringsmöjligheter är alternativa personverifieringsmetoder såsom ansiktsigenkänning eller mönsterlösenord som alternativ till PIN-kod.

### 8.1.1 Tidsbesparing relaterat till informationsutbyten

Under utformandet av kommunikationsprotokollet (se avsnitt 4.2) har användarvänlighet prioriterats. Detta har lett till ett halvautomatiskt och intuitivt förlopp för upplåsning med minimalt ingrepp från användaren. Dock har det visat sig att denna fokus på användarvänlighet har lett till en långsammare lösning. Med nuvarande lösning skickas två meddelanden, en i vardera riktning, för att åstadkomma en upplåsning eller konfigurering. Det har visat sig att denna lösning är långsam då *Android Beam* tycks vara konstruerat för i huvudsak envägskommunikation, vilket observeras i avsnitt 7.2.

**Teoretisk lösning med fokus på förbättrat kommunikationsförlopp** Med anledning av ovanstående resonemang skulle ett kommunikationsförlopp där endast ett meddelande behövde skickas för en lyckad upplåsning ge ett markant snabbare förlopp. En sådan lösning skulle dock kräva ingrepp från användaren vid varje upplåsning, då användaren skulle behöva välja ett lås av potentiellt många vilka användaren har tillgång till. Behovet av att välja lås skulle därmed göra lösningen mindre skalbar. Det skulle också innebära att en inledande parning mellan låsenhet och mobilapplikation skulle behöva genomföras i syfte att utbyta krypteringsnycklar.

Sammanfattningsvis har den lösning vilken projektet har resulterat i lett fram till den mest användarvänliga lösningen. Detta trots att det har visat sig att lösningen har blivit något långsammare på grund av fokusen på användarvänlighet.

**Teoretisk förbättring av nuvarande lösning med fokus på sänd och mottagningsförfaranden** Ytterligare ett tillvägagångssätt till att minska den tid kommunikationsförloppen tar hade varit att använda `Get`-förfrågor istället för `Put`-förfrågor i förloppen. Om `Get`-förfrågan hade implementerats i SNEP-lagret och använts hade antalet nödvändiga sändningsförfaranden från låsenheten minskats till ett. I detta sändningsförfarande skickar låsenheten en `Get`-förfråga innehållande ett meddelande av typ 1 och mobilapplikationen svarar med ett `Success`-respons innehållande ett meddelande av typ 3 eller 5. Sista sändningen, det vill säga den frivilliga feedbacken till mobiltelefonen, kan skickas med en SNEP-`put` men är enligt tidigare utläggning frivillig.

Då detta tillvägagångssätt används kan det krypterade meddelandet nå låsenheten efter endast ett sändningsförfarande vilket är omkring $1,5$ sekunder (se figur 47). Detta är en förbättring från nuvarande tre sekunder. De $1,5$ sekunderna baseras på att ett sändningsförfarande vilket använder en `Get`-förfråga tar lika lång tid att utföra som ett sändningsförfarande vilket använder en `Put`-förfråga, samt att ett annat kryptosystem används vilket är mer tidseffektivt än det nuvarande.

Anledningen till att `Get`-förfrågan inte implementerades baserades på observationer av de SNEP-PDU:er mobilapplikationen skickade, vilka uteslutande utgjordes av `Put`-förfrågor. Istället för att chansa på att *Android Beam* hade stöd för `Get`-förfrågan togs det säkra före det osäkra och `Put`-förfrågan användes.

### 8.1.2 Tidsbesparing relaterat till kryptosystemet

Med ett mer välanpassat och utarbetat kryptosystem hade krypteringens genomslag på tidsförloppet kunnat minskas avsevärt. Den nuvarande lösningen fungerar bra på applikations-

sidan då smarttelefoner har processorkraften vilken behövs för att hantera kryptosystemet. Dock fungerar inte lösningen lika bra på låsenhetssidan då den har en mindre kraftig processor än en smarttelefon och tiden det tar för låsenheten att dekryptera meddelanden är beaktningsvärd. Ett mindre beräkningstungt kryptosystem hade varit att föredra. Att krypteringen är dåligt anpassad beror på flera faktorer. En faktor är bristfällig kunskap när det gäller kryptering, då ingen i gruppen hade förkunskaper i ämnet. Ytterligare en faktor är tiden då mycket tid och arbete har gått åt att implementera fungerande NFC-kommunikation. Med mer resurser än förväntat lagda på kommunikationen har andra utvecklingsområden blivit lidande och i projektets fall anses det att krypteringslösningen blev något av en nödlösning. Med mer tid hade ett bättre och snabbare kryptosystem kunna konstrueras.

Till en början var symmetrisk kryptering (se avsnitt 3.2.5), vilken är en vanlig och välbeprövad krypteringsmetod, aktuell. Detta stycke beskriver tankegångarna om varför symmetrisk kryptering valdes bort till fördel för asymmetrisk kryptering. Vid symmetrisk kryptering finns det enbart en privat krypteringsnyckel. En symmetrisk nyckel är hemlig och används för både kryptering och dekryptering. Det stora problemet med symmetrisk kryptering är att distributionen av krypteringsnycklar blir svår att hantera. Om en angripare får tag på den hemliga nyckeln räcker det inte med att låsenheten tilldelas en ny nyckel, dessutom behöver alla applikationer med access till given låsenhet även få denna nya nyckel distribuerad till sig. Det insågs ganska fort att detta skulle bli ett problem då en central databas, vilken hanterar och distribuerar nya nycklar, hade behövts. Detta sågs som ett bemödande som sträcktes utanför ramarna för grundidén med ett prototyplåssystem baserat på NFC-teknik. I efterhand borde en lösning med symmetrisk kryptering undersökts bättre då Mollin (2002) menar att symmetrisk kryptering är 1000 gånger snabbare beräkningsmässigt än asymmetrisk och betydligt kortare krypteringsnycklar kan användas (128 jämfört med 1024).

Flera alternativa lösningar borde också setts över, som alternativa krypteringslösningar för att kunna hålla nere beräkningskostnader för en bättre krypteringslösning och således ett snabbare prototyplåssystem:

- **Hybrid-kryptosystem.** Flera kryptosystem utnyttjas tillsammans för en elegantare lösning. Allah (2011) rekommenderar en hybrid-krypto lösning för NFC-kommunikation. Det Allah anser vara det bästa tillvägagångssättet för att skydda NFC-kommunikation är att använda asymmetrisk kryptering till exempel RSA för att dela en hemlighet. Hemligheten i det här fallet är privata krypteringsnycklar vilka används för symmetrisk kryptering. Den asymmetriska krypteringen vilken är långsam sker bara en gång vid så kallad `parning`, den symmetriska krypteringen vilken är snabb sker efter att enheter har fått tillgång till privata krypteringsnycklar.

- **Challenge-response authentication.** Applikationen löser utmaningar, som verifieras av låsenheten för autentisering (Internet Security Glossary, Version 2). Tassos Dimitriou beskriver ett *challenge-response* protokoll närmare i A Lightweight RFID Protocol to protect against Traceability and Cloning attacks. Detta protokoll hade troligtvis kunnat implementeras istället för nuvarande kryptosystem för att få en effektiv lösning.

I teoridelen avsnitt 3.2.5 är rekommendationen vid användning av RSA 1024 bitars krypteringsnycklar, dock blev detta ett problem. I huvudsak på grund av begränsad beräkningskapacitet hos mikrokontrollern och därför kunde enbart 512 bitars krypteringsnycklar användas. Eftersom både det använda biblioteket på låsenheten (PolarSSL, se 5.2.3) och Javas API för

säkerhet kommer med stöd för 512 bitars RSA har detta valts att användas. Detta ses inte som ett stort bakslag eftersom det nu finns fungerande kryptering och med en mer beräkningsanpassad mikroprocessor i hårdvaran skulle 1024 bitar kunna användas då all funktionalitet redan finns på plats.

**Teoretisk förbättring av krypteringsförfarande** Om en hybridlösning likt den Allah (2011) föreslår hade implementerats skulle tidsförloppet kunnat förkortas ned markant. RSA hade i sin befintliga implementation brukats enbart för överenskommelse av den symmetriska krypteringsnyckeln. Denna överenskommelse hade bara behövt göras första gången en mobilapplikation kommer i kontakt med låsenheten. Efter denna överenskommelse brukas den symmetriska krypteringsnyckeln för att åstadkomma symmetrisk kryptering med tillexempel AES. Mollin (2002) säger att symmetrisk kryptering är mindre beräkningstung och följaktligen snabbare än asymmetrisk kryptering och förbättringar upp till 1000 gånger kan uppnås. Detta betyder att alla krypteringsförfaranden, annat än det första, istället för $3,5$ sekunder skulle ta nedåt $3,6$ ms.

Att denna lösning inte implementerades beror till fullo på tidsbrist och det faktum att RSA-krypteringen inte implementerades helt problemfritt. Det finns färdigimplementerade AES-bibliotek att tillgå för Arduino-plattformen (Brian, 2008). Huruvida ett kryptosystem bestående av RSA och AES hade inrymts på mikrokontrollern är dock oklart då detta inte undersöktes.

### 8.1.3 Uppgradering av hårdvara

Tidsvinster kan också erhållas genom att uppgradera låsenhetens hårdvara. Försök att göra detta gjordes sent i projektet men utan framgång. Försöket bestod av att byt ut *Arduino Mega* mot en *Arduino Due*. Den sistnämnda erbjuder stor prestandaökning då den är av ARM-arkitektur. Med 32-bitars instruktionsuppsättning och en drygt fyrfaldig klockfrekvensökning till 84 Mhz skulle stora tidsvinster vara möjliga. Tyvärr befinner sig stödet för denna Arduinomodell i utvecklingsstadiet och de kodbibliotek som projektet bygger på fungerar inte med *Due*. Således kunde inte en uppgradering fullföljas.

Enligt observation i avsnitt 7.2 är det enbart krypteringsförloppet där tidsbesparingar är möjliga. Detta gör att vi ser ett förbättrat krypteringssystem som en lämpligare lösning än en kostsam hårdvaruuppgradering. Vidare är antagligen tidsvinsterna med ett förbättrat kryptosystem, där både asymmetrisk och symmetrisk kryptering används, större än den som erhålls av bättre hårdvara.

## 8.2 Mobilapplikation

Utvecklingen av Android-applikationen flöt på smidigt då de inblandade hade goda Javaprogrammeringskunskaper sedan innan. De Android-specifika kunskaper som behövdes inhämtades från utvecklingssidan för Android (2013h). När denna sida inte gav tillräckliga svar användes det i många fall svar tillgängliga på hemsidan Stack Overflow av Stack Exchange Network (2013)).

Ett av de största problem vilket stötts på under utvecklingen av applikationen är att Androids aktiviteter skapas och återskapas enligt det mönster vilket anges i dess livscykel (se avsnitt 3.3). Detta medför att variabler förstörs. För att behålla dessa variabler finns det

diverse standardfunktioner. Dessa funktioner fungerade inte problemfritt vid skapandet av säker kommunikation via NFC.

Problemet med standardfunktionerna var att de krävde att informationen vilken skulle sparas behövde vara serialiserbar, något som upptäcktes att klasser som krävdes för krypteringen inte var. Detta trots att dokumentation tydligt angav att serialisering skulle varit möjlig. Det här gav stora svårigheter då kommunikationsförloppet krävde ett antal återskapningar av aktiviteter samtidigt som dessa skulle behålla information (till exempel variabler). Problemet löstes slutligen med hjälp av ett byte till Bouncy Castle Crypto API (Legion of the Bouncy Castle, 2013) som ingår i Android API:s (Android, 2013d).

Mobilapplikationen hade även kunnat göras mer användarvänlig genom att på något sätt skriva över standardsättet att skicka meddelanden. Standardsättet att skicka meddelande tar hela skärmen i anspråk utan att ge vidare information om vad som skickas. Det skulle även vara önskvärt att kunna genomföra upplåsningsförloppet utan ingrepp från användaren. Det finns det dock ingen möjlighet att ersätta standardsättet med nuvarande API. Det får helt enkelt bli en önskning till framtida *Android Beam* API då det skulle ge möjlighet till flexiblare lösningar. Istället för att det står Touch to Beam"hade man kunnat skriva "Hold near to unlock"eller något liknande för att skapa ökad användarvänlighet. NFC känns utvecklat för specifika ändamål och inte särskilt flexibelt och kreativt från ett utvecklarperspektiv och detta är något som har märkts av under hela utvecklingsförloppet.

NFC-tekniken är omogen vilket kan ses när det gäller olika versioner av Android. Det var t.ex. inte förrän Android version 4.0 som Android fick en av NFC Forum definierad kommunikationsstack för kommunikation mellan två aktiva enheter (Google Inc, 2013). Innan version 4.0 hade Android en egentillverkad kommunikationsstack för detta ändamål, NPP (Android Open Source Project, 2011). Sedan Android 4.0 implementeras dock SNEP (NFC Forum, 2011b) vilket förhoppningsvis får fortsätta mogna med plattformen.

Det skall påpekas att det inte har lagts stort fokus på säkerheten i mobilapplikationen, då upplåsningsnycklar till exempel visas öppet i applikationen. Detta gjordes för att det skulle blir enkelt att felsöka under utvecklingsfasen för kommunikationsprotokollet. Om man skulle vidareutveckla prototypen skulle mer fokus behövts läggas på att utveckla applikationen för att motverka att en angripare kan tjuvtitta och få reda på upplåsningsnyckel eller annan känslig data. Under projektets gång har personverifiering genom PIN-kod ansetts som tillräcklig säkerhet för mobilapplikationen något som alltså inte räcker vid vidareutveckling med sikte på en färdig produkt.

Ett av målen med låssystemet var att skapa ett mer flexibelt låssystem än de befintliga som finns idag. Delar av dessa flexibilitetsförbättringar kan implementeras enbart i mobilapplikationen utan att låsenheten påverkas. Utöver de funktioner som blev implementerade skulle temporära nycklar med antigen tidsbegränsning eller användningsbegränsning läggas in. Mer flexibilitet hade kunnat uppnås med olika nivåer av behörighet hos olika nycklar. Det ger systemet möjlighet att implementera nyckelhierarkier vilket i sin tur leder till möjligheter att använda huvudnycklar och rättigheter att ändra nyckeln i låset eller rättigheter att dela nycklar vidare.

## 8.3 Låsenheten

Utvecklingen av låsenheten har varit intressant då denna del av projektet har utforskat obeträdd mark. Då NFC är en relativt ny teknik visade det sig vara svårt att hitta färdigutvecklade

hjälpmedel till Arduino-plattformen för att realisera låssystemet. I avsnitt 8.3.1 tas tankar upp kring valet av Arduino som plattformen till låsenheten. Vidare beskrivs vår syn på Arduinios utvecklingsmiljö i avsnitt 8.3.2 och vilken utvecklingsmiljö vilken kan användas istället för denna. Slutligen presenteras tankar och reflektioner kring den implementerade NFC-stacken i avsnitt 8.3.3.

### 8.3.1 Reflektioner kring Arduino-plattformen

Efter att ha arbetat med Arduino har det insetts att plattformen inte är färdigutvecklad. Detta är inte förvånande då plattformen inriktar sig mot utveckling av hobbyprojekt. Vidare har upphovsmännen bakom plattformen gjort den till öppen källkod för att uppmuntra användarna att själva utveckla plattformen. Tyvärr samordnar inte upphovsmännen denna utveckling nämnvärt och den har därmed skett mer eller mindre sporadiskt. Detta har resulterat i att dokumentationen av ett källkodsbibliotek är generellt undermålig. Ett konkret exempel på detta är Arduinos implementation av I2C-kommunikation där bristande information kring buffertstorlekar ledde till svårfunna fel.

Trots dessa tillkortakommanden hade vi troligtvis valt att basera projektet på Arduino om vi gjort om det idag. Varför vi skulle valt Arduino-plattformen och inte någon annan plattform beror mer på att vi inte känner till några andra alternativ. Det enda vilket vi anser motiverar valet för Arduino är för att den är användarvänlig och att det är enkelt att komma igång med den. De sköldar vilka finns att tillgå till Arduino kan med små ingrepp anpassas till i stort sett vilken plattform som helst och således utgör inte NFC-skölden någon begränsning vid val av plattform.

### 8.3.2 Reflektioner kring utvecklingsmiljöer till Arduino

Utvecklingsmiljön Arduino tillhandahåller är minst sagt simpel. Den ger ingen möjlighet till insyn i hur kod kompileras och inte heller några verktyg med vilka kompilering kan styras. Ett problem, vilket fick oss att inse behovet av insyn och kontroll vid kompileringen, uppstod när projektet växt sig stort och komplicerat. Kompilatorn valde då en hoppinstruktion vilken inte täckte hela adressrymden projektet tog i anspråk, vilket fick till följd att koden inte alls fungerade och mycket tid gick åt till att hitta felkällan.

Stöd för felsökning finns i form av utskrifter via konsol och mer utvecklade verktyg skulle behövts då alla buggar inte kan hittas via utskrifter, till exempel pekarfel vilka kan vara svåra att upptäcka. Är man inte försiktig kan den uppladdade koden förstöras vid exekvering och felutskrifterna upphöra. Detta sätt att felsöka är således mycket ineffektivt och processen tar längre tid än den skulle gjort om ett bättre verktyg fanns att tillgå. Dessutom plågas utvecklingsmiljön av en del buggar varav den mest påstridiga är att kontakten med mikrokontrollern tappas gång efter annan. Allt detta har lett till att det totala intrycket av utvecklingsmiljön inte är till dess förtjänst.

Vi har i efterhand hittat utvecklingsmiljön Atmel Studio 6 från Atmel Corporation (2012), vilken är betydligt mer funktionellt berikad, men också mer komplicerad, än den utvecklingsmiljö Arduino erbjuder. Atmel Studio erbjuder mer utvecklad felsökning och både insyn och kontroll av kompileringen. Ett försök till en övergång till Atmel Studio gjordes men det visade sig vara svårt att från Atmel Studio få tillgång till de Arduino-specifika källkodsbiblioteken, vilket gjorde att en större satsning på att byta utvecklingsmiljö inte genomfördes. Om vi i

början av arbetet fått kännedom av Atmel Studio och de svagheter Arduinos utvecklingsmiljö har, är det mycket sannolikt att vi genomgående skulle använt Atmel Studio.

### 8.3.3 Reflektioner kring den implementerade NFC-stacken

I detta avsnitt presenterar vi våra reflektioner kring de problem och insikter vi tillgodogjort oss vid implementationen av NFC-stacken. I avsnitt 8.3.3.1 diskuteras de begränsningar vår implementation har, varför dessa ansågs nödvändiga och hur dessa kan åtgärdas. Styckena nedan beskriver de största problem vilka har stötts på i samband med implementationen.

Problemen vilket tagit överlägset längst tid under projektets utvecklingsfas relaterar alla till implementationen av NFC-stacken. Det visade sig mycket svårt att gå från stackens specifikationer till en fungerande implementation. Dessa svårigheter uppkom då specifikationerna är svåra att tolka och sätta samman till ett helt kommunikationsförlopp. Hade det inte varit för Lotito (2013a) projekt vilken implementerade förloppen på ett fungerande och överskådligt sätt, hade det antagligen tagit avsevärt mycket längre tid innan kommunikationen mellan enheterna fungerade.

Mobiltillverkarna verkar också ha haft problem att tolka de nämnda specifikationerna då NFC-stackens implementation skiljer sig tillverkarna emellan. Detta kan styrkas då implementationen på HTC One X skiljer sig åt från Samsung Galaxy S3 och detta tycks inte bero på hårdvarumässiga skillnader då de båda, enligt AnandTech (2012) och Inc (2012), använder NFC-chippet PN544 från NXP Semiconductors (2013c). Dessa skillnader i tolkning har visat sig genom observationer av vilka ramar vilka sänds på LLC-nivå. Till exempel sänder Samsung Galaxy S3 fler SYMM-ramar (se avsnitt 6.2.3) i kommunikationen än HTC One X och följaktligen slutfördes ett kommunikationsförlopp något snabbare på HTC One X än Samsung Galaxy S3. Vidare var mobiltelefonerna olika vad avser de LTO-värden (se avsnitt 3.1.1.3) vilka kunde användas för ett fungerande kommunikationsförlopp. Mindre värden fungerade för HTC One X än Samsung Galaxy S3, vilket antagligen härrör i att HTC One X sänder mindre antal LLCP-ramar. Slutligen spelar även mobiltelefonernas övriga hårdvara in och detta har påvisats med LG Optimus L9, vilken är den prestandamässigt svagaste telefonen vilket används i projektet. Den ter sig vara för långsam för att kunna hantera NFC-kommunikation i de hastigheter projektet kräver.

Alla skillnader vilka förekommer implementationerna emellan kan antagligen härledas till att NFC är en relativt ung teknik. Alla standarder och tillhörande implementationer har till synes inte rotat sig lika djupt som de har gjort hos äldre tekniker. Om några år när NFC har uppnått en högre grad av mognad och standarder och implementationer rotat sig, tror vi att NFC skulle kunna bli en utbredd och välanvänd teknik.

**8.3.3.1 Begränsningar** En del avkall har gjorts för implementationen av NFC-stacken i detta projekt. Dessa avkall samt deras anledningar beskrivs i de kommande styckena. I och med att dessa avkall har gjorts medför det att låsenheten, om låsenheten ämnat att släppas för kommersiellt bruk, inte kan bli NFC-certifierad (NFC Forum, 2012b). De delar utav NFC-stacken vilka inte implementerats är de vilka inte ansågs vara nödvändiga för att utveckla en låsenhet med den sökta funktionaliteten (se 4.1 Prototypkrav). Avkallen gjordes också för att det bedömdes orimligt att implementera hela NFC-stacken på ett inbyggt system, i alla fall de baserat på Arduino-plattformen. Detta framförallt då arbetet var tidsbegränsat men också på grund av bristande plats och beräkningskapacitet förknippat med Arduino.

På NDEF-lagret stödjs enbart sändandet av `Short`-records och enbart ett enskilt record. Vidare är inte `chunking`-mekanismen implementerad. Förklaringen till detta är att låssystemet enbart skickar meddelanden vilka maximalt är 256-byte långa och enbart består av ASCII formaterad text. Således fanns det inget behov att implementera stöd för mer flexibel hantering av NDEF-meddelanden.

Det implementerade SNEP-lagret erbjuder dataöverföring mellan enheterna och detta utförs genom att skicka Put-förfrågor. Implementation av Get-förfrågor, vilket kanske skulle ge ett snabbare kommunikationsförlopp (se avsnitt 8.1.1), släpptes då dessa gjorde hantering av SNEP mer komplex utan att tillföra någon garanterad funktionalitet. Inte heller har feedback, i form av SNEP-responser annat än `Success` implementerats, då detta skulle mångdubblat detta lagers komplexitet.

På LLCP-lagret är det länkhanteraren vilken inte till fullo har den funktionalitet vilket specificeras av NFC Forum (2011a). Implementationen av en uppkoppling är nästintill korrekt. För att erhålla en helt korrekt hantering av en uppkoppling skall LLC hålla reda på dels det lokala mottagningsfönstret (se avsnitt 3.1.1.3) men även det hos mottagarens LLC. I aktuell implementation görs ingendera. Korrekt hantering av mottagningsfönster hade relativ enkelt kunnat implementeras, men eftersom dessa inte utgjorde någon begränsning under sändningsförloppet hade dessa inte fyllt någon nödvändig funktionalitet. Vidare så har inte LLC möjlighet att hantera flera aktiva länkar vilket också specificeras av NFC-forum. Att implementera stöd för flera parallella uppkopplingar bedömdes vara alltför komplext och tidskrävande för att rymmas inom projektets ramar. Dessutom ställer vi oss tveksamma till om det alls hade varit möjligt att implementera på en Arduino-plattform med begränsad beräkning- och minneskapacitet.

Något vilket gäller vår implementation av samtliga delar i stacken är att de är oförlåtande. Uppstår fel i något lager i stacken avbryts pågående förfarande ovillkorligen och hela kommunikationsförloppet startas om. Detta är inte ett försummande från vår sida eftersom det flesta av dessa fel är av en sådan natur att återhämtning är svårt. Felen är oftast relaterade till diverse *time-outs* i kommunikationen, antingen mellan sköld och mikrokontroller eller mellan låsenhet och mobiltelefon. Vidare ses det som fel att tillåta sändning eller mottagningsförlopp återupptas vid fel eftersom detta utgör inskränkningar på säkerheten. Skulle till exempel fel uppstå i det andra sändningsförfarandet skall detta inte återupptas utan hela förloppet skall startas om.

# 9 Slutsats

Projektet startades med syftet att skapa ytterligare funktionalitet för mobiltelefoner med hjälp av NFC-tekniken. Målet var att utveckla en prototyp bestående av en mobilapplikation och en låsenhet vilket också slutfördes. Denna prototyp innehar framförallt en unik egenhet i form av en av oss framställd NFC-stack placerad i låsenheten, vilken möjliggör NFC-kommunikation mot Android-baserade mobiltelefoner. En sådan stack fanns inte att tillgå vid projektets start.

Tidigare kommersiella lösningar, vilka utför samma uppgift som det nu utvecklade låssystemet och använder sig av liknande teknik baseras nästintill uteslutande av passiva taggar. Då vår lösning baseras på två beräkningskapabla enheter kan säkerhet appliceras på en helt annan nivå än för de lösningar vilka finns idag där en enhet i kommunikationen är en passiv tagg.

En lösning vilken släpps sommaren 2013 är Lockitron (2013) vilken utför samma uppgift som den utvecklade prototypen. Denna lösning använder dock Bluetooth för kommunikation mellan låsenhet och mobilapplikation, vilket jämfört med NFC inte har lika stor inneboende säkerhet. Lockitron har redan innan produktsläpp dragit in över 2 miljoner dollar tack vare förbokade exemplar och detta visar tydligt på att det finns ett stort intresse för en produkt likt projektets prototyp.

De två visioner vilka presenterades i inledningen anser vi vara fullt realiserbara om prototypen utvecklas vidare. Visionen angående att använda låssystemet i ett hänglås eller cykellås kan realiseras redan i dagsläget då låssystemet innehar all nödvändig funktionalitet. Den andra visionen om att använda låssystemet till ett stort företags alla dörrar kan visserligen inte realiseras i dagsläget men det vi har utvecklat är grunden för att konceptet ska kunna genomföras. Vi är övertygade om att lösningar likt våra visioner kommer att finnas tillgängliga på marknaden inom en snar framtid, vilket leder till att den traditionella nyckelknippan successivt ersätts av modernare lösningar.

# 10 Referenslista

## Referenser


Adafruit Industries (2012). *Adafruit Industries Repositories*. GitHub. URL: https://github.com/adafruit (hämtad 2013-02-11).

— (2013). *Adafruit PN532 NFC/RFID Controller Shield for Arduino*. Adafruit Industries. URL: http://www.adafruit.com/products/789 (hämtad 2013-05-19).

Allah, M.M.A. (2011). "Strengths and Weaknesses of Near Field Communication (NFC) Technology". I: *Global Journal of Computer Science and Technology* 11.3, s. 51–56.

AnandTech (2012). *The HTC One X for AT and T Review*. AnandTech. URL: http://www.anandtech.com/show/5779/htc-one-x-for-att-review/7 (hämtad 2013-05-18).

Android (2013a). *Android SDK*. Android Developers. URL: http://developer.android.com/sdk/index.html (hämtad 2013-05-19).

— (2013b). *Android, the world's most popular mobile platform*. Android Developers. URL: http://developer.android.com/about/index.html (hämtad 2013-05-17).

— (2013c). *android.nfc*. Android Developers. URL: http://developer.android.com/reference/android/nfc/package-summary.html (hämtad 2013-05-19).

— (2013d). *Downloading the source*. Android Open Source Project. URL: http://source.android.com/source/downloading.html (hämtad 2013-05-17).

— (2013e). *Managing the Activity Lifecycle*. Android Developers. URL: https://developer.android.com/training/basics/activity-lifecycle/index.html (hämtad 2013-05-17).

— (2013f). *Near Field Communication*. Android Developers. URL: http://developer.android.com/guide/topics/connectivity/nfc/nfc.html#p2p (hämtad 2013-05-17).

— (2013g). *Near Field Communication*. Android Developers. URL: http://developer.android.com/guide/topics/connectivity/nfc/index.html (hämtad 2013-05-17).

— (2013h). *Package Index*. Android Developers. URL: http://developer.android.com/reference/packages.html (hämtad 2013-05-17).

Android Open Source Project (2011). *Android NDEF Push Protocol Specification*. Google User Content. URL: http://static.googleusercontent.com/external_content/untrusted_dlcp/source.android.com/sv//compatibility/ndef-push-protocol.pdf (hämtad 2013-05-14).

Arduino (2013a). *Arduino Development Enviroment*. Arduino. URL: http://arduino.cc/en/Guide/Environment (hämtad 2013-05-17).

— (2013b). *Arduino Mega 2560*. Arduino. URL: http://arduino.cc/en/Main/ArduinoBoardMega2560 (hämtad 2013-05-17).

— (2013c). *Arduino Repositories*. GitHub. URL: https://github.com/arduino/Arduino (hämtad 2013-05-04).

— (2013d). *Arduinos officiella hemsida*. Arduino. URL: http://arduino.cc/ (hämtad 2013-04-20).

— (2013e). *Bare Minimum*. Arduino. URL: http://arduino.cc/en/Tutorial/BareMinimum (hämtad 2013-05-17).

Atmel Corporation (2012). *Atmel Studio 6*. Atmel. URL: http://www.atmel.com/microsite/atmel_studio6/ (hämtad 2013-05-19).

— (2013). *Atmel Corporation*. Atmel. URL: http://www.atmel.com/ (hämtad 2013-05-20).

Barlas, P (2010). *Google's New Android Helps Mobile Payment Nexus S first with NFC chips that can facilitate sales via mobile phones*. URL: http://proxy.lib.chalmers.se/login?url=http://search.proquest.com/docview/1002103939?accountid=10041.

Brian, G (2008). *AES implementation for byte-oriented processors*. URL: http://forum.arduino.cc/index.php?topic=88890.0 (hämtad 2013-06-06).

Chang R. Shmatikov, V. (2007). *Formal Analysis of Authentication in Bluetooth Device Pairing*. The University of Texas at Austin. URL: http://www.cs.utexas.edu/~shmat/shmat_fcs07.pdf (hämtad 2013-05-10).

Deffree, S. (2012). *Near-field communications to go far in 2013*. EDN Networks. URL: http://www.edn.com/design/communications-networking/4402986/Near-field-communications-to-go-far-in-2013 (hämtad 2013-05-11).

Eclipse (2013). *Eclipse*. Eclipse. URL: http://www.eclipse.org (hämtad 2013-05-19).

Google Inc (2013). *Introducing Android 4.0*. Android. URL: http://www.android.com/about/ice-cream-sandwich/ (hämtad 2013-05-17).

Gunn, M (2012). "Google Reinvents the Wallet". I: *Bank Systems and Technology* 48.4, s. 8.

Inc, Chipworks (2012). *Inside the Samsung Galaxy SIII*. Chipworks. URL: http://www.chipworks.com/blog/recentteardowns/2012/06/01/inside-the-samsung-galaxy-siii/ (hämtad 2013-05-18).

Inside Secure (2013). *Core Edition*. Open NFC. URL: http://open-nfc.org/wp/editions/core/ (hämtad 2013-05-12).

Intel Corporation (2005). *Moore's Law: Raising the Bar*. Intel. URL: http://download.intel.com/museum/Moores_Law/Printed_Materials/Moores_Law_Backgrounder.pdf (hämtad 2013-05-11).

International Organization for Standardization (2004). *ISO/IEC 18092*.

— (2008). *ISO/IEC 14443. Identification cards – Contactless integrated circuit cards – Proximity cards*.

Legion of the Bouncy Castle (2013). *Bouncy Castle Crypto API*. The Legion of the Bouncy Castle. URL: http://www.bouncycastle.org (hämtad 2013-05-17).

Lockitron (2013). *Lockitron*. Lockitron. URL: http://www.lockitron.com (hämtad 2013-05-20).

Lotito, A (2013a). *ismb-snep-java*. Google. URL: http://code.google.com/p/ismb-snep-java/ (hämtad 2013-05-20).

— (2013b). *ismb-snep-java*. Google. URL: http://code.google.com/p/ismb-snep-java/ (hämtad 2013-05-14).

McRoberts, M. (2010). *Beginning Arduino*. USA, NY: Aspress.

Mollin, A. Richard. (2002). *RSA and Public Key-Cryptography*. USA: Chapman och Hall/CRC.

MTB (2013). *Mobiltelefonförsäljning i Sverige*. MTB - MobilTeleBranchen. URL: http://www.mtb.se/index.php?sid_id=1281&id=1281&ftg=41&ftg_id=41 (hämtad 2013-05-17).

NFC Forum (2006). *NFC Data Exchange Format*. Version 1.0.

— (2011a). *Logical Link Control Protocol*. Version 1.1.

— (2011b). *Simple NDEF Exchange Protocol*. Version 1.0.

— (2012a). *NFC Analog Specification*. Version 1.0.

— (2012b). *NFC Forum Device Requirements*. Version 1.2.

— (2013a). *About us*. NFC Forum. URL: http://www.nfc-forum.org/aboutus/ (hämtad 2013-05-02).

— (2013b). *NFC Forum Technical Specifications*. NFC Forum. URL: http://www.nfc-forum.org/specs/spec_list/ (hämtad 2013-05-24).

NFC World (2013). *NFC phones: The definitive list*. NFC World. URL: http://www.nfcworld.com/nfc-phones-list/#available (hämtad 2013-05-18).

Nielsen, J (1995). *10 Usability Heuristics*. Neilsen Norman Group. URL: http://www.nngroup.com (hämtad 2013-05-16).

NXP (2007). *PN532 User Manual*. NXP. URL: http://www.nxp.com/documents/user_manual/141520.pdf (hämtad 2013-05-19).

— (2010). *PN532 C106 application note*. Adafruit Industries. URL: http://www.adafruit.com/datasheets/PN532C106_Application%20Note_v1.2.pdf (hämtad 2013-05-19).

NXP Semiconductors (2007). *PN532 User Manual*. NXP Semiconductors. URL: http://www.nxp.com/documents/user_manual/141520.pdf (hämtad 2013-05-19).

— (2010). *PN532 C106 application note*. Adafruit Industries. URL: http://www.adafruit.com/datasheets/PN532C106_Application%20Note_v1.2.pdf (hämtad 2013-05-19).

— (2013a). *Home*. NXP Semiconductors. URL: http://www.nxp.com/ (hämtad 2013-05-19).

— (2013b). *PN532*. NXP Semiconductors. URL: http://www.nxp.com/products/interface_and_connectivity/nfc_devices/series/PN532.html (hämtad 2013-05-19).

— (2013c). *PN544*. NXP Semiconductors. URL: http://www.nxp.com/documents/leaflet/75016890.pdf (hämtad 2013-05-20).

Offspark B.V (2013). *PolarSSL - Light-weight, modular cryptographic and SSL/TLS library in C*. PolarSSL. URL: https://polarssl.org/ (hämtad 2013-05-19).

Oracle Corporation (2013a). *Java$^{TM}$ Platform, Standard Edition 7 API Specification*. Oracle. URL: http://docs.oracle.com/javase/7/docs/api/ (hämtad 2013-05-19).

— (2013b). *JDK*. Oracle. URL: http://www.oracle.com/technetwork/java/index.html (hämtad 2013-05-19).

— (2013c). *Package java.security*. Oracle. URL: http://docs.oracle.com/javase/7/docs/api/java/security/package-summary.html (hämtad 2013-05-19).

— (2013d). *Package javax.crypto*. Oracle. URL: http://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html (hämtad 2013-05-19).

SQLite (2013). *SQLite Home Page*. SQLite. URL: http://www.sqlite.org/index.html (hämtad 2013-05-15).

Stack Exchange Network (2013). *Stack Overflow*. Stack Overflow. URL: http://stackoverflow.com/ (hämtad 2013-05-14).

Tapper, K m. fl. (2013a). *iLoveBugs / Arduino*. Github. URL: https://github.com/iLovebugs/Arduino.

— (2013b). *kaety/NFC-Keys*. Github. URL: https://github.com/kaety/NFC-Keys.

The OpenSSL Project (2009). *OpenSSL: The Open Source toolkit for SSL/TLS*. OpenSSL. URL: http://www.openssl.org/ (hämtad 2013-05-19).

Timalsina S.K. Moh, S (2012). "A Review on NFC and NFC-Based Mobile Payment Solution". I: *Journal of Next Generation Information Technology* 3.4, s. 35–44. DOI: 10.4156/jnit.vol3.issue4.4.

Wier, M (2012). *Embedded-PN532*. Github. URL: https://github.com/mweir/Embedded-PN532 (hämtad 2013-05-12).

Wikpedia (2013). *OSI-modellen*. Wikipedia. URL: http://sv.wikipedia.org/wiki/Osi-modellen (hämtad 2013-05-18).

Wilson, R (17 jan. 2013). *Android Ice Cream Sandwich gets NFC stack*.

# 11 Bilagor

## Appendix A: Kravspecifikation

Skallkrav är de krav som måste uppfyllas av vår konstruktion medan börkrav är de krav som ses som sekundära. Börkraven är fördelaktiga att uppfylla och leder till en mer fullständig produkt.

### Skallkrav för produkten som helhet

1. Endast användare med kännedom om låsenhetens identitet skall kunna agera mot låsenheten.
2. Kommunikation mellan mobilapplikation och låsenheten ska ske via NFC.
3. Känslig information skall överföras krypterat.
4. Låset skall kunna låsas upp inom fem sekunder.

### Börkrav för produkten som helhet

5. Upplåsningsprocessen bör vara snabb nog för att låset skall vara upplåst/låst inom 1 sekund efter att kommandot skickats.
6. Låsenheten bör kunna konfigureras av mobilapplikation.
7. Det bör vara möjligt för flera användare att operera mot samma lås alternativt begränsa antalet telefoner som har tillåtelse att låsa upp låset.

### Specifika Börkrav för mobilapplikation

8. Vid förbättring av designen bör utvecklingen följa Nielsen (1995).
9. Noga testad och har inga kända buggar.
10. Mobilapplikationen bör fungera på samtliga Androidtelefoner utrustade med NFC.
11. Nycklar bör kunna delas mellan olika Androidtelefoner över en NFC-länk.
12. Implementera personverifiering med PIN-kod i mobilapplikation.

### Specifika Börkrav för låsenhet

13. Felsäker, t.ex. bör inga kända buggar finnas och vid strömavbrott bör låset förbli låst.
14. Formfaktorn bör vara anpassad för ändamålet.
15. Lösningen bör inte dra för mycket effekt vid normal användning.

# Appendix B: Figurlista

## Figurer