HOVEDPROSJEKT:

Java SmartSign

FORFATTERE:
MORTEN TVENGE, MATS ERIK SMESTAD, BJARNE MANGNES,
OLE MARTIN KROGSET

Dato: 23.05.02

SAMMENDRAG AV HOVEDPROSJEKT

Tittel:	Java SmartSign	Nr. : 9
		Dato: 23.05.02
Deltaker(e):	Morten Tvenge	
	Mats Erik Smestad	
	Bjarne Mangnes	
	Ole Martin Krogset	
Veileder(e):	Erik Hjelmås	
	HiG	
Oppdragsgiver:	ErgoSolutions as	
	Hans Mustadgate 31, 2821 Gjøvik	
Kontaktperson:	Morten Johansen	
Stikkord (4 stk)	Smartkort, Elektronisk ID, Javacard, Biometri	
	Smartcard, Electronic ID, Javacard, Biometic interface	
Antall sider: 102	Antall bilag: A-J	Tilgjengelighet (åpen/konfidensiell): Åpen

Kort beskrivelse av hovedprosjektet:

Denne oppgaven går ut på å lage en prototype for Elektronisk Identifisering på et smartkort. Ved benyttelse av kortet skal brukeren identifisere seg ved bruk av fingeravtrykk og/eller PIN-kode. Løsningen skal brukes til å signere et dokument som skal sendes elektronisk f. eks over Internett. Dermed kan man sjekke at dokumentet er skrevet av oppgitt avsender, og at ikke innholdet i dokumentet er endret underveis.

For å få til dette har vi utviklet to applikasjoner. Den ene befinner seg på smartkortet og foretar selve signeringen. Den andre er en PC-applikasjon som står for kommunikasjon med kortet, og viser at løsningen fungerer.

Forord

Denne oppgaven er utarbeidet som et hovedprosjekt ved Høgskolen i Gjøvik. Prosjektet er et avsluttende ledd i utdanningen for dataingeniører ved HiG, og tilsvarer 6 av til sammen 60 vekttall.

Prosjektet rundt smartkort som ErgoSolutions fremla vakte tidlig interesse blant gruppens medlemmer, da dette er en ny teknologi vi gjerne ville få anledning til å lære mer om. Arbeidet startet rett før jul 2001, og har gjennom de siste månedene gitt oss nyttig erfaring når det gjelder samarbeid og jobbing på et prosjekt av denne størrelsen.

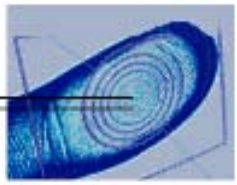
Vi vil rette en spesiell takk til Erik Hjelmås som har vært gruppens veileder. Han har vært til stor hjelp under prosjektets gang, ikke minst som ressursperson på biometridelen. Han har også vært fleksibel og behjelpelig med alle våre spørsmål.

Vi vil også rette en stor takk til Morten Johansen og ErgoSolutions som har stilt opp og svart raskt på våre spørsmål og problemer. Han har også bidratt med sin betydelige kunnskap rundt smartkortteknologi, og vært en viktig støttespiller i utviklingen av dette prosjektet.

Gjøvik 21.05.2002

Morten Tvenge	Mats Erik Smestad
Bjarne Mangnes	Ole Martin Krogset

INNLEDNING



Java SmartSign 2002

1 Innledning

1.1 Definisjoner og terminologi

ADC	Application Delete Certificate. Sertifikat som trengs for å slette applikasjoner fra smartkortet i MultOS.
Administrator	Person som har ansvaret for driften av systemet.
ALC	Application Load Certificate. Sertifikat som trengs for å laste applikasjoner til smartkortet i MultOS.
ALU	Application Load Unit. Programmet som lastes/slettes fra smartkortet i MultOS.
APDU	Application Protocol Data Unit. Dataenheten for smartkortkommunikasjon.
API	Gitt grensesnitt mot et system eller annen kode. Man kommuniserer mot et system ved å bruke fast definerte funksjoner og formater.
Applet	Program skrevet i Java. Eksekveres av et Java Virtual Machine (se JVM).
Applikasjon	Et program som er skrevet for bruk på smartkort eller PC.
ASN.1	Abstract Syntax Notation, Formelt språk for å beskrive data som blir utvekslet mellom forskjellige datasystem.
Autentisering	Prosessen med å bekrefte en oppgitt identitet.
Biometri	Studie av menneskelige, statistiske fenomen. For eksempel fingeravtrykk, stemme, og lignede.
Byte	Målenhet for størrelse av filer på datamaskin 1byte = 8 bit.
C	X509 certificate. Standard for sertifikat.
CA	Certification authority. Utsteder av digitale sertifikater. Som regel et uavhengig organ mange fester tillit til.
Cipher/ decipher	Kryptere / dekryptere.
CLA	APDU Class byte. Byte for å velge en applikasjon på et Javacard
CPU	Central Processing Unit. PC'ens hovedprosessor.
Debug	Feilsøking og retting av datakode.
DC	Decryption. Brukes som forkortelse om dekryptering.
Dekryptering	En datastrøm som er kryptert gjøres lesbar igjen. (Se kryptering).
EEPROM	Electrically Erasable Programmable Read Only Memory. En lagringsbrikke som kan endres ved hjelp av strøm.
EID	Elektronisk Identifikasjon. Smartkort applikasjon for identifisering av bruker.
ErgoGroup (EG)	Tidligere Posten SDS, Oppdragsgiver.
Exception	Feil eller unntak som oppstår ved kjøring av program.
Fingerscanner / Fingeravtrykksleser	Enhet for avlesing av fingeravtrykk, for vår del finnes den på kortleseren.

Garbage collector	System for å frigjøre minne og ressurser som ikke lenger er i bruk.
Gemplus GemXpresso RAD III	Utviklingsverktøy og simulator for Javacard fra firmaet Gemplus.
GUI	Graphic User Interface. Det grafiske brukergrensesnittet.
Hash	Resultatet av en matematisk funksjon, der man ut fra gitte data får en sjekksum på dokumentet. Ved selv små endringer i input-data vil sjekksummen forandre seg. Tilnærmet umulig å regne ”bakover” til original tekst.
HIG	Høgskolen i Gjøvik.
I/O	Input/Output. Kommunikasjon med eksterne enheter.
Identifisere	Påberope seg en identitet
INS	APDU Instruction code. Byte som velger hvilken instruksjon som kjøres på et smartkort.
ISO	International Standard Organization. Jobber med internasjonale standardiseringsmodeller.
ISO 7816-4	Standard for smartkort. Denne delen tar for seg ID-kort, integrerte kretser og kontakter samt standardiserte kommandoer.
ISO 7816-8	Standard for smartkort. Denne delen tar for seg sikkerhetsrelaterte kommandoer.
ISO 7816-9	Standard for smartkort. Denne delen tar for seg tilleggskommandoer for sikkerhetsaspekter ved smartkort.
Javacard	Smartkort med Javacard som operativsystem på kortet. Bygger på, og kodes mot, med programmeringsspråket Java.
JCE	Java Cryptographic Extensions, Suns kryptografiextentions for java. Inneholder en del klasser for, og noen få implementeringer av krypteringsalgoritmer.
JCSI	Java Crypto and Security Implementation, implementasjon fra en Security Provider for å tilby kryptografi til Java JDK.
JDK	Java Development Kit, Javasofts utviklermiljø for Java-applikasjoner.
JRE	Java Runtime Environment. Inneholder JVM og maskinspesifikke klasser for å skape et Java-miljø på et bestemt system.
JVM	Java Virtual Machine. Plattform som alle Java program startes fra og kjøres i.
kb	Kilobyte. 1024 byte.
Klient	Delen av systemet brukeren vil jobbe mot.
Kortleser	Enhet for avlesing/skriving til smartkort.
Lc	APDU length of data field. Byte som forteller hvor mye data som sendes med en enkelt APDU.
Le	APDU length of expected response data. Byte som forteller hvor mye data som forventes i retur fra smartkortet.
mb	Megabyte, 1024 kb.
MultOS	Et operativsystem for smartkort.
OCF	Open Card Framework. Ett rammeverk for Java. Gir mulighet for å sende serier av kommando og svar sekvenser.

P1	APDU parameter 1. Applikasjonsspesifikk parameter for smartkort.
P2	APDU parameter 2. Applikasjonsspesifikk parameter for smartkort.
Padding	Spesifisert måte å fylle opp et visst antall byte/bit for å oppnå en gitt total størrelse på data.
PIN	Personal Identification Number, kode som brukes på blant annet smartkort.
PKCS#1	Standard for sikkerhet som omhandler RSA kryptografi.
PKCS#15	Public Key Cryptography Standard no. 15, RSALabs. Standard som omhandler sikkerhet, og hvordan sertifikater, padding og algoritmer bør benyttes.
Posten SDS	Tidligere Statens Data Sentral. Nå en del av Posten under navnet ErgoGroup.
Precise Biometrics	Firma som jobber med biometri og produkter rundt dette.
PUK	PIN Unblock Code. Kode for å åpne et kort igjen hvis PIN har vært tastet feil 3 ganger.
RAM	Random Access Memory. Arbeidsminne. Innholdet i RAM mistes ved strømtap.
ROM	Read Only Memory. Informasjon blir brent inn i en brikke, og kan ikke slettes.
RSA	Krypteringsalgoritme utviklet av R. Rivets, A. Shamir og L. Adleman.
Security Provider	Leverandør av kryptografi-API.
SHA/SHA-1	Secure Hash Algorithm, algoritme for hashing av data.
Smartkort	Plastkort i kredittkortstørrelse med egen mikroprosessor, minne og lagringsenhet.
Template	En signatur generert ut i fra et fingeravtrykk, for å lagre egenskaper om fingeravtrykket digitalt. Template'en har en størrelse på ca 1,9kb.
Validere	Stadfeste. F. eks identitet, eier av kort etc.
Verifisere	Bekrefte identitet.
X.509	Standard for sertifikater. Beskriver hvilke data som skal være med i et sertifikat, og hvordan dataformatet skal være.

1.1.1 Ord og uttrykk brukt i teksten

For å variere språket i rapporten er det en del ord og uttrykk som brukes om hverandre. Dette er i første rekke for å variere språket, men også for å presisere hva som menes i enkelte tilfeller. Ordene dette gjelder følger nedenfor.

Terminalapplikasjon **Demoapplikasjon** **Demonstrasjonsapplikasjon** **PC-applikasjon**	Brukes om den delen av systemet som befinner seg på PC, og ikke på smartkort. Denne skal kommunisere med smartkortet og EID-applet'en på dette, samt demonstrere at EID-applikasjonen fungerer.
EID **EID-applikasjon** **EID-applet** **Smartkort-applikasjon**	Brukes om den Elektroniske ID-applikasjonen som befinner seg på smartkortet. Denne tar seg av selve signeringen, og inneholder brukerens nøkler og sertifikat.
Kortholder **Bruker** **Kortbruker** **Korteier**	Eieren av smartkortet som vil bruke dette i hverdagen.
Kortutsteder **Issuer** **Admin**	Person eller organisasjon som har ansvaret for å legge inn data i kortene og utstede disse til sluttbruker.
Modul **Klasse**	Selvstendig del av applikasjon eller kode.

1.2 Oppgavedefinisjon

I dagens elektroniske samfunn blir de tjenestene som f.eks Internett tilbyr stadig viktigere. E-post og hjemmesider er blitt dagligdags, og en stadig større del av korrespondansen mellom mennesker skjer elektronisk. Blant annet har også det offentlige innsett at det kan være en betydelig effektiviseringsgevinst å hente i dette. Dette medfører imidlertid at man får et stadig større behov for sikkerhet, fordi det er betydelig enklere å snappe opp og endre informasjon underveis enn før. Mye av problemet ligger i å sikkert kunne fastslå hvem avsender er, og om dokumentet er endret under sending eller ikke. Uten et slikt system for sikker identifisering er det vanskelig og bruke teknologien effektivt, da man i mindre grad kan stole på informasjonen man mottar. Et eksempel på dette kan være å levere tippekupongen over Internett. Vinner man, kan det dreie seg om større pengesummer, og da er man nødt til å ha et system som helt sikkert kan identifisere hvem som sendte kupongen, og at tipperekken ikke er endret under sending. Andre eksempler kan være dokumentforsendelser innad i en bedrift. Det finnes løsninger for elektronisk signering i dag, men disse tar i liten grad for seg gyldigheten av signaturen, altså en uavhengig instans som går god for signaturen som blir brukt.

Formålet med dette prosjektet er å implementere en smartkortløsning som gjør brukeren i stand til å løse problemene med identifisering nevnt ovenfor. Dette vil være en løsning som kan brukes til en rekke formål som involverer identifisering mot et datasystem, alt fra innloggingskort på datamaskiner og nettverk til dokumentforsendelser og handel over Internett. For at løsningen skal bli enklest og sikrest mulig i bruk, tar vi også med skanning av fingeravtrykk. Ved å benytte nøkkelkryptering og digitale sertifikater fra en offentlig sertifiseringsorgan vil vi kunne oppnå tilstrekkelig grad av sikkerhet.

Det ferdige systemet fungerer ved at brukeren installerer en smartkortleser, og tilhørende PC-applikasjon. Dermed vil brukeren, gjennom PC-applikasjonen ha mulighet for å velge et dokument som skal signeres. Dette programmet fungerer ved at brukeren autentiserer seg mot kortet ved hjelp av PIN-kode og fingeravtrykk. Hvis dette stemmer overens, vil kortet generere en digital signatur ut fra valgt dokument, og returnere denne. Brukeren kan da legge denne signaturen sammen med dokumentet og overføre disse samlet. Mottakeren av dokumentet, vil foreta motsatt prosess, ved at man tar signaturen og verifiserer denne ved hjelp av sertifikater fra en offentlig sertifiseringsenhet, for så å finne ut om dokumentet er endret, og om avsenderen er den han gir seg ut for.

Denne oppgaven er imidlertid i stor grad begrenset til selve kortapplikasjonen og ikke det totale systemet, for å kunne holde tidsfristene. Dette medfører at PC-applikasjonen ikke er fullt utviklet, men har en funksjonalitet som demonstrerer at kortapplikasjonen fungerer, samt noen enkle administrative funksjoner. PC-applikasjonen er skrevet i Java, noe som også gjelder kortapplikasjonen, men dette er imidlertid et sterkt redusert subsett av vanlig Java.

1.3 Målgruppe

Målgruppen for denne rapporten er i all hovedsak ErgoSolutions. Det vil være de som vil kunne benytte denne rapporten og våre erfaringer innen Javacard til videre arbeid og utvikling.

Målgruppen for produktet generelt vil til syvende og sist bli vanlige brukere av Internett og e-post. Men siden dette prosjektet baserer seg på å lage en prototype på Javacard for oppdragsgiver, vil målgruppen også her bli ErgoSolutions.

1.4 Faglig bakgrunn og kunnskap

Gruppens fire medlemmer tar driftslinjen på dataingeniørstudiet ved HIG. Gjennom denne utdannelsen har vi tilegnet oss kunnskap innen programmering med hovedvekt på C++. I tillegg har de fleste fra gruppen tilegnet seg en del kunnskap om Java gjennom faget ”Programmering mot www”. Gruppemedlemmene har også hatt systemering og prosjektstyring som et ledd i utdannelsen, og tar kurset ”Sikkerhet i datasystemer” parallelt med prosjektet.

For å kunne fullføre prosjektet var det imidlertid nødvendig og sette seg inn i både smartkortsteknologien generelt, og mer spesifikt, Javacard, som er den plattformen som benyttes. Dette er teknologier som er ganske forskjellige fra det vi hadde kjennskap til fra før, og ble derfor en utfordring. Vi måtte i tillegg sette oss inn i biometri og fingeravtrykksavlesning, og hvordan dette fungerer, noe som også var helt ukjent for gruppen.

1.5 Arbeidsformer

Arbeidsformene i løpet prosjektet har variert en del etter hva vi fant hensiktsmessig. I begynnelsen følte vi at det var viktig at alle medlemmene fikk satt seg skikkelig inn i oppgaven og være med på planlegging av prosjektprosessen. Derfor ble denne delen av prosjektet gjort i felleskap. Dette gjaldt også den første kodefasen, gjennomført hos ErgoSolutions, der man skulle sette seg inn i selve smartkort- og Javacard- teknologien for at alle skulle ha en basis av kunnskap rundt dette.

Etter dette ble det imidlertid en mer variert og oppdelt arbeidsform. For å få tiden til å strekke til ble de forskjellige kodeoppgavene delt opp, og man arbeidet enten en og en eller i team på to, etter hva som var nødvendig. De avsluttende fasene av prosjektet ble naturlig nok mer preget av samarbeid, der man jobbet tett sammen i team med hyppig kontakt. At gruppemedlemmene har bodd på samme sted, har også ført til at man har kunnet holde en nær kontakt seg imellom. Vi har også vurdert arbeidsmengden fortløpende, og fordelt arbeidsoppgavene mellom gruppemedlemmene etter dette.

Hver mandag har gruppen hatt et fast gruppemøte der man oppsummerte hvordan man lå an i forhold til prosjektplanen, og hva man skulle gjøre kommende uke ble klarlagt. I tillegg til dette faste møtet samlet vi oss om det skulle være behov for å ta opp noe i fellesskap, dette kunne være ting som problemer angående løsningsmetoder, uklarheter om oppgaven og annet. Det ble også holdt fire statusmøter med veileder som ble spredt utover semesteret

1.6 Rapportens organisering

Kapittel 1 – Innledning:
Inneholder forord og en kort beskrivelse av hva oppgaven går ut på.

Kapittel 2 – Kravspesifikasjon
Inneholder en detaljert beskrivelse av hva systemet skal bestå av.

Kapittel 3 – Design
Her blir det i detalj beskrevet hvordan vi har laget systemet, og hvilke løsninger som er valgt.

Kapittel 4 – Implementasjon
Inneholder redegjørelse for hvilke verktøy og utviklingsmiljøer som er brukt, og de prinsipper og standarder man følger for dette prosjektet. Her finner man også eksempler på kode, for å vise hvordan dette er etterlevd i praksis.

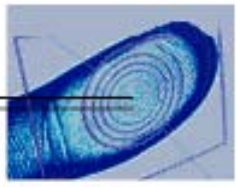
Kapittel 5 – Testing
Inneholder beskriver hvordan testing av applikasjonen har foregått. Dette gjelder både planlegging, valg av teststrategier og gjennomføringen av disse. Kapitelet beskriver også feil som er funnet, og hvordan disse er håndtert.

Kapittel 6 – Diskusjon
Her diskuterer man resultatene fra prosjektet, og eventuelle avvik som har oppstått i forhold til kravspesifikasjonen. Avvik drøftes, og forslag til forbedringer presenteres. Inneholder også gruppens vurdering av arbeidet, samt de erfaringer vi sitter igjen med.

I tillegg følger litteraturliste og vedlegg.

KRAVSPESIFIKASJON

Java SmartSign 2002



2 Kravspesifikasjon

2.1 Introduksjon

Dette dokumentet beskriver den Elektroniske ID-applikasjonen prosjektet skal produsere, som igjen er basert på blant annet kravene fra EID-spesifikasjon gitt av ErgoSolutions. Denne er videre i henhold til kravene som ligger i PKCS#15 standarden fra RSA Inc, samt ISO 7816.

2.1.1 Bakgrunn

2.1.1.1 Om ErgoSolutions

ErgoSolutions er en del av ErgoGroup, tidligere kalt Posten SDS. SDS (Statens DataSentral) ble dannet gjennom et Stortingsvedtak i 1972, og privatisert som aksjeselskap i 1986. Posten kjøpte opp dette i 1995 og dannet ErgoGroup 1.1 2001. ErgoSolutions består i dag av 18 selskaper og leverer et bredt spekter av IT-tjenester. Selskapet har ca. 1800 ansatte og en årsomsetning på over 1670 millioner (2000).

2.1.1.2 Om teknologien

Smartkort ble tatt i bruk så tidlig som i 1970. Disse 1. generasjonskortene var relativt enkle, med kun en minnebrikke. Etter hvert kom 2. generasjons kort som også inneholdt prosessor, og dermed kunne foreta egne utregninger. I senere tid er det også blitt vanlig at kortene har en dedikert kryptoprosessor. 3. generasjonskort er kontaktløse kort og trenger derfor ingen kortleser for å benyttes. I de siste årene har også kortenes grensesnitt blitt mer standardisert med egne rammeverk, samt at minnekapasitet og kraft øker proporsjonalt. I dag er kort med 32 kb minne helt vanlig, mens det for få år siden kun var kort med 16 kb lagringskapasitet som var tilgjengelige på markedet. Javacard-teknologien som skal benyttes er en relativt ny teknologi, og er av en mer ”åpen” smartkort type. Det betyr i all hovedsak at det ikke finnes noen sentral kortautoritet som godkjenner og registrerer nye kort. Man står fritt til å bruke og lage nye kort og applikasjoner.

2.1.1.3 Om prosjektgruppen

Prosjektgruppen består av fire dataingeniørstudenter fra Høgskolen i Gjøvik, med drift av datasystemer som hovedlinje. Gruppesammensetningen er valgt ut fra at gruppens medlemmer har samarbeidet en del i tidligere prosjekter og oppgaver. Derfor kjenner vi hverandre godt, og vet hvilke kvalifikasjoner hver og en har. I tillegg bor man på samme boenhet, noe som letter samarbeidet, spesielt med tanke på prosjektmøter og koordinering av arbeid.

Kriteriet gruppen satte for valg av oppgave, var at den skulle bestå av noe alle interesserte oss for. I tillegg var det ønskelig å få en oppgave som er fremtidsrettet, og introdusere nye og spennende områder innen teknologi man ikke har vært borti før. ”EID på Javacard” var den oppgaven gruppen syntes passet best, og vi var heldige nok til å kunne jobbe videre med denne.

Ingen av gruppemedlemmene har spesiell erfaring eller kunnskap innen smartkortteknologi eller biometri fra før. Derfor blir dette nytt for alle i gruppa, og innebærer at alle har en del å lære før man kan sette oss ned for å starte programmeringen.

2.2 Målsetting

2.2.1 Prosjektets mål

- Sette seg inn i Javacard operativsystemet som den Elektroniske ID’en (EID) skal implementeres på. Dette omfatter også å sette seg inn i utviklingsverktøy, standarder og grensesnitt.

- Sette seg inn i kryptering og dekryptering i forbindelse med datalagring og flyt.

- Utvikle og implementere en applikasjon for sikker Elektronisk Identifisering av en person eller gruppe på et smartkort. Dette skal gjøres i Java (subsett) ut fra spesifikasjon gitt av ErgoSolutions. Se **[Vedlegg D]**. Funksjonaliteten på applikasjonen skal tilsvare eksisterende løsning på MultOS.

- Sette oss inn i biometrisk teknologi, nærmere bestemt fingeravtrykk. Samt å implementere fingeravtrykk-avlesing som en del av autentiseringsprosessen i EID-applikasjonen. Denne applikasjonen skal støtte bruk av fingeravtrykksleser for å identifisere kortholder, eventuelt med PIN-kode i tillegg.

- Utvikle en demonstrasjonsapplikasjon for demonstrering av smartkortløsningen. Dette skal være en PC-applikasjon skrevet i Java, og skal fungere på en PC med Windows-plattform. Denne skal ha en test del for kortholder, og en del for kortutsteder.

- Sammenligne og vurdere sikkerheten i Javacard i forhold til MultOS.

2.2.2 Oppgavebeskrivelse

Elektronisk ID brukes for å kunne autentisere en person for eksempel via Internett, eller nøkkelsystemer slik at en er sikker på at en person er den man gir seg ut for.
ErgoSolutions' tidligere implementasjoner av en slik Elektronisk ID har vært skrevet for MultOS, mens dette prosjektet går ut på å utvikle tilsvarende funksjonalitet på Javacard. Dette er delvis grunnet i at ErgoSolutions er interessert i å se på mulighetene ved bruk av Javacard som operativsystem for smartkort. Da MultOS er sikrere enn Javacard, er det også ønskelig å sammenligne sikkerheten mellom disse, for å funne ut om Javacard kan tilby et tilstrekkelig sikkerhetsnivå til dette formålet.

Oppgaven består i å skrive en applet i Java, som implementerer en elektronisk signatur, såkalt Elektronisk ID, som legges på et smartkort. I tillegg skal det som nevnt utvikles en demoapplikasjonen, som skal skrive persondata til kortet og teste dette ved hjelp av filsystem som skal lages. Skriving til kortet skjer i registreringsfasen, når en person knyttes til smartkortet. Lesing skjer når personen bruker kortet til identifisering. I forbindelse med en del operasjoner, må også data krypteres/dekrypteres av sikkerhetsmessige årsaker. Det blir også vår oppgave å få dette til å fungere ved å bruke funksjonene som finnes i Javacard.

Når det gjelder identifisering av kortholder, skjer dette ved fingerscanning, det vil si avlesing av fingeravtrykk. I tillegg kan man også bruke PIN-kode. Det blir opp til oss å avgjøre om man vil bruke PIN-kode i tillegg til fingerscanning. Denne skjer ved hjelp av en enhet som kobles til PC'en og som skaffes av arbeidsgiver. All programmering mot fingerscanneren skjer ved hjelp av et ferdigdefinert API som vi skal bruke, og som finnes både i C og i Java. Sistnevnte språk er det som vil bli brukt i denne oppgaven.
Ved å bruke API'ets funksjoner for å behandle data fra scanneren, skal man kunne lagre en persons fingeravtrykk på smartkortet, og denne blir da følgelig en del av demoapplikasjonen. Denne skal vise at personidentifiseringen fungerer, både med tanke på verifisering, og avlesning av fingeravtrykk, og den skal kunne kjøres på en vanlig PC.

Mye av utfordringen her blir å få til en fungerende fingeravlesning, og finne en måte å lagre fingeravtrykket på smartkortet. Den komplette EID-applikasjonen må også være liten nok til å få plass på selve smartkortet, noe som nok kan bli en betydelig utfordring, spesielt når fingeravtrykket regnes med.

2.2.3 Systemets omgivelser

En fullt utviklet EID applikasjon vil være rettet mot et bredt spekter av brukere.
Dette faller naturlig da sikker identifisering har mange bruksområder og kundegrupper.
Prosjektets ramme er imidlertid satt til selve kortapplikasjonen som en prototype, og et komplett system er derfor utenfor dette prosjektets rekkevidde.

EID-applikasjonen vil kommunisere med kortleser og omverden via et fastlagt kommandosett definert i kravspesifiseringen av EID som er gitt av arbeidsgiver, se **[Vedlegg D]**. Det vil derfor ikke være noe direkte grensesnitt mot bruker fra selve kort-applikasjonen, men det forholdes til det gitte kommandogrensesnittet. I tillegg skal det utvikles en demoapplikasjon, men denne er beregnet på et mer teknisk sett av brukere. Denne skal utvikles på en PC med Windows-plattform, fortrinnsmessig i Java. Kortapplikasjonen skal ligge på et smartkort som følger ISO-7816 standarden. Et typisk anvendelsesområde vil være at kortet kobles til en PC via en USB-kortleser, og brukes til identifisering ved for eksempel transaksjoner over Internett. Men bruksområdet kan også for eksempel være nøkkelkort og automatisert studentbevis.

2.2.4 Avgrensninger

Selve teknologien bak fingerskanningen blir det ikke vår oppgave å programmere. Det forutsettes bruk av API der vi bruker en matrise/vektor for å representere fingeravtrykket. Prosessering av fingeravtrykk, det vil si sammenligning av matriser ved verifiseringen, skjer i en tilhørende PC. Derfor blir det ikke vår oppgave å programmere selve sammenligningsprosessen, men gruppen må finne ut hvordan fingeravtrykket skal lagres på kortet. Muligheten for at også sammenligningsprosessen kan foregå på kortet skal også undersøkes, med tilhørende implementasjon hvis det lar seg gjennomføre
All programmering i forbindelse med applet'en, det vil si kryptering, signering, PINsjekk og så videre, er også bundet til det ferdigdefinerte kommandoer i Javacard API'et.
Behovet for implementasjon av PIN-kode ved autentisering vil også vurderes.

2.3 Brukerbeskrivelse

2.3.1 Innledning

I dag flyttes mye av papirer og dokumenter over til elektronisk form. Viktige sertifikater og dokumenter utføres mer og mer på PC, til fordel for trykk og papir. I takt med dette øker også kravet for identifisering/autentisering, altså en entydig og ikke minst sikker måte å slå fast hvem du er. Kravet som stilles er mye høyere enn ved vanlig dokumenter, hovedsakelig fordi elektronikken i dag gjør forfalsking betydelig enklere. Økt handel og forretninger over Internett fører til større krav til sikkerheten.
I dette prosjektet skal det utvikles en smartkortapplikasjon for Elektronisk Identifisering. Det skal ta seg av signeringen av elektroniske dokumenter og sertifikater. Kort sagt skal systemet kunne verifisere, på lik linje med signatur, at en gitt person er den han gir seg ut for å være. På denne måten kan for eksempel to forretningspersoner signere et dokument over Internett, samtidig som sikkerheten er ivaretatt.

2.3.2 Brukere

Brukere for dette prosjektet er ErgoSolutions. Et demonstrasjons program skal lages for å presentere EID og biometri på smartkort. EID på smartkort er ikke noe nytt, men ErgoSolutions vil vite mer hvordan smartkort med Javacard OS fungerer, og hvilke muligheter biometrien gir. Selve prosjektet er derfor klassifisert som en forskning på ErgoSolutions vegne.

Verken applet'en på smartkortet eller demoapplikasjonen for PC skal distribueres eller blir brukt i det daglige. Til dette kreves det mye mer utvikling, men det er de første stegene. ErgoSolutions synes Javacard har flere lokkende fordeler og det skal forsøkes på vegne av ErgoSolutions å finne ut om Javacard egner seg som Elektronisk Identifisering.

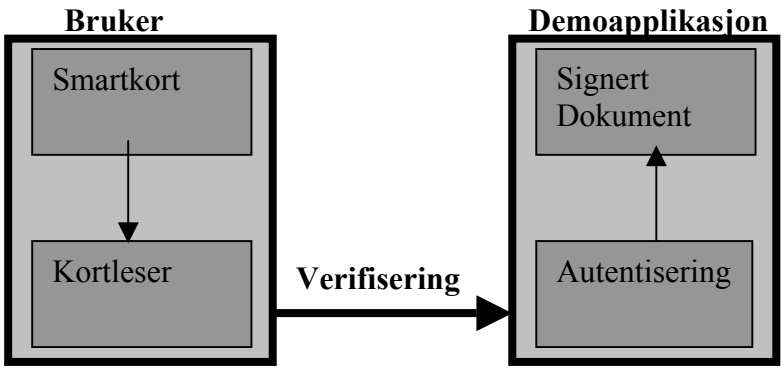
2.3.3 Funksjon

Vi skal i dette prosjektet konsentrere oss om smartkortet som en identifikator. Det vil si at smartkortet først og fremst skal inneholde opplysninger kortholder (sertifikater), og RSA nøkler. Den sistnevnte skal brukes til signering av elektroniske dokumenter. Se også figur 2.1.

På kortet skal det settes opp et dynamisk filsystem som gir økt fleksibilitet og effektiv utnyttelse av plassen. Filsystemet forøvrig vil være hierarkisk oppbygd med adgangskontrollerte filer.

Smartkortet skal beskyttes med en fingeravtrykksleser. Det vil si at i stedet for en vanlig 4-talls PIN-kode som verifisering, brukes personens finger. Dette er enklere enn vanlig PIN-kode som er mest utbredt i dag. Dessuten har folk lett for å glemme eller forveksle PIN-koder, mens fingeravtrykk sørger for at man slipper og huske en hemmelig kode. Muligheten er også åpen for å kombinere PIN og fingerskanning for ytterligere økt sikkerhet.

Kortet skal viser sin funksjonalitet gjennom nevnte demoapplikasjon gruppen utvikler. Opplysninger hentes fra kortet og autentisering skjer ved hjelp av fingerscanneren og ferdige kommandoer som kjøres fra demoapplikasjonen.



Figur 2.1: Grunnleggende funksjonalitet for systemet

2.3.4 Omgivelser

Siden dette prosjektet baserer seg på mye uprøvd teknologi fra oppdragsgiverens side, så blir oppgaven primært bestående av å se om det fungerer tilfredstillende å bruke Javacard i stedet for MultOS. Forskning er derfor et sentralt stikkord. Det skal ikke utvikles et større system, men heller et enkelt system til demonstrasjon og testing. Kodingen er delt i moduler slik at eventuelle videre prosjekter kan konsentrere som om en bit av dette prosjektet og utvide den biten.

Siden dette er en Elektronisk ID er sikkerheten viktig, og det er ønskelig å oppnå størst mulig sikkerhet, tatt i betraktningen at ingen av prosjektgruppens medlemmer ikke har erfaring på dette området.

2.3.5 Ytelse

Ytelsen til EID-applet'en som skal lastes ned på smartkortet, begrenses av hastigheten på chip'en på selve kortet. De funksjonene som applet'en tilbyr, det vil si signering støttet av kryptering, verifiseringsfunksjoner (fingeravtrykk og PIN), filsystemoperasjoner og lignende skal ikke ta unødvendig lang tid for sluttbrukeren. Med unødvendig lang tid, menes at det ikke bør ta mer enn 15 sekunder på en enkelt funksjon. Funksjoner i forbindelse med kryptering, foregår i en co-prosessor på smartkortchipen. Derfor blir tidsforbruket her bedret i forhold til ren bruk av chip-prosessor. Det som kan bli kritisk for ytelsen på dette prosjektet, er behandlingen av fingeravtrykk. Målet er å få plass nok til to fingeravtrykksignaturer (template'er) på kortet. I tillegg vil det, av sikkerhetsmessige årsaker være best å sammenligne disse to på kortet, under verifiseringsdelen av kortholder. Det vil si at prosessorkraften til CPU'en på kortet kan være avgjørende. Den er en 8 bits prosessor som kjører på 7,5 MHz. Det blir en utfordring for denne prosessoren til klare alle de beregningene som hører med når et fingeravtrykk skal verifiseres mot et annet, med et rimelig tidsforbruk, og om det er mulig innenfor de rammene prosjektet har. Spørsmålet som dukker opp er hvor lang tid vil dette ta, dersom det er mulig å gjennomføre sammenligning på kortet. Tidsforbruket i forbindelse med dette må vurderes i utviklingsfasen. Dersom det tar altfor lang tid å sammenligne fingeravtrykkene, må sammenligningsfunksjonene flyttes fra smartkortet og over en PC, der regnekraften er atskillig større.

Erfaringer selv, samt fra andre brukere, viser at funksjoner som tar "unødvendig" lang tid, lett skaper frustrasjon og kan føre til at brukerne velger å ikke bruke produktet. Derfor må det satses på å oppnå tilfredsstillende ytelse under normal bruk. Det er klart at det vil gå med noe tid når kortet initialiseres med sluttbrukerens sertifikater (nøkler) og fingeravtrykk. Fingeravtrykkene avleses til kvaliteten er tilfredstillene og legges på kortet. Kvaliteten dømmes ut fra lysstyrke, kontrast og plassering av finger på fingerscanneren. I/O operasjoner mot kortet tar sin tid på grunn av begrensningene i størrelsen på datapakkene som sendes, samt hastigheten disse blir sendt på (9600 b/s).

Demonstrasjonsapplikasjonen skal også kodes i Java. Dette innebærer at applikasjonen kjøres på en såkalt Java VM (Virtual Machine). For å få akseptabel responstid fra applikasjonen sin side, bør maskinen være av nyere dato.

2.3.6 Begrensninger

Begrensningene for dette prosjektet ligger i smartkortet, med hensyn på den hardware arkitekturen kortet har. Regnekraften til CPU'en er selvsagt svært begrenset, med tanke på den fysiske størrelsen på chip'en (i underkant av en fingernegl). Da man ikke helt vet hvilke beregninger den klarer, og hva slags prosessering som blir for tungt, blir det vår oppgave å finne ut om den klarer håndteringen av fingeravtrykk.

Når et fingeravtrykk skal verifiseres, skjer det en rekke beregninger. Det er mye data som skal sammenlignes før det endelige resultat fremkommer. Det kommer til
å bli forsket en del på kapasiteten til smartkortet i forbindelse med prosessering av fingeravtrykk. Ikke bare med tanke på hastigheten, men også størrelsen på fingeravtrykkene slik de lagres digitalt. Fingerscannerens API genererer en ”template”, eller en matrise med oversikt over egenskapene til fingeravtrykket ut fra bildet fra fingerscanneren. Hvordan denne er bygd opp i detalj, blir litt utenfor vårt arbeidsområde, dersom dette ikke er åpen standard. Størrelsen på denne template’en er avgjørende for om det blir nok plass på kortet. Kortet som skal brukes, har en lagringskapasitet på 24 kb, og det vil bli store utfordringer i å få plass til både EID-applet’en og (helst) to fingeravtrykktemplate’er.

Personinformasjon i sertifikater, samt private RSA nøkler, må også få plass på kortet i forbindelse med EID-applet’en. Ved konstruksjon av filsystemet står man overfor to valg. Man kan enten velge et statisk filsystem, eller et dynamisk. Begge har sine begrensinger. Et statisk filsystem gjør at filsystemet har fast avsatte plasser for lagring av data. Dette setter da grenser (som programmereren definerer) på lengder av navn og lignende. Det passer likevel bra til lagring av nøkler som vanligvis har faste størrelser. Dette kan redusere bruk av lagringskapasitet. Det dynamiske filsystemet åpner for variable lengder, men da må det også lages funksjoner for å rydde opp i dataene som ligger på kortet etter hvert (”garbage-collection”), da data lagres der det er plass. Disse rutinene vil selvsagt gjøre at applet’en blir større. Det blir opp til oss å finne ut hva som er mest hensiktsmessig i forbindelse med plassbesparelse.

Tidligere EID på smartkort, har blitt utviklet for MultOS på smartkort med 8 og 16 kb lagringskapasitet (EEPROM). Der har man brukt Assembler språket som utviklingsverktøy. Dette språket er meget effektivt med tanke på kodeeffektivitet og størrelse. Java er valgt som utviklingsplattform. Java er kjent for ikke å være like plassbesparende og effektiv som Assembler grunnet at det ikke kompileres ned til maskinkode, men det skal dog i teorien noe enklere å programmere. Dette betyr at vår applet vil oppta mer plass på kortet, samtidig som man skal få plass til en fingeravtrykktemplate på ca 1,9 kb. I tillegg bør man regne inn tilsvarende plass for midlertidig lagring av template når et fingeravtrykk skal verifiseres. Det er opplagt at dette blir vanskelig å oppnå, og alternative løsninger må kanskje vurdere underveis. For eksempel å legge sammenligningen av fingeravtrykkene på en ekstern PC. Gruppen vil også se på om det er mulig å gjøre template’en mindre, dersom dette skulle være nødvendig. Dette krever at det ikke gåes for detaljert til verks, grunnet prosjektets tidsramme og omfang.

Man må, så mye som mulig, holde seg til de API’ene som eksisterer for programmeringen mot smartkort og biometri (fingerscanneren). Hvis man går utenom de faste API funksjonene, vil omfanget av prosjektet bli altfor stort til at det vil kunne gjennomføres.

Demoapplikasjonen som skal vise hvordan smartkortet fungerer i praksis, blir et punkt for vurdering om hvor mye tid som skal brukes på denne. Mye står på EID-applet’en og tilhørende fingeravtrykks funksjoner, og hvordan arbeidet med dette går. Om tiden blir for knapp, må arbeidet med demoapplikasjonen nedprioriteres. Det er imidlertid viktig å understreke at denne applikasjonen skal lages. Spørsmålet blir hvor mye som skal gjøres ut av denne.

Blir det tid til overs, kan det selvsagt gjøres mye ut av denne delen av oppgaven ved å legge inn flere funksjoner som mer grundig demonstrerer aspekter ved funksjonaliteten til applet'en.

Den siste fasen av prosjektet, som omhandler drøfting av funksjonalitet og sikkerhet på Javacard i forhold til MultOS, skal også være med. Her kan det dras nytte av egne erfaringer i tillegg til at det må skaffes kjennskap til MultOS systemet. Dersom tiden blir knapp, blir også her arbeidsmengden vurdert, men en rapport skal likevel utformes.

2.3.7 Antagelser

Dette prosjektet innebærer en del forskning, prøving og feiling i forbindelse med fingeravtrykksbehandlingen. Derfor er det noe usikkert hvordan vi kommer oss vel i land. Det må derfor antas at vi kommer til å bruke de API'ene som følger med kortleser/fingerscanner. Smartkortverktøyet som skal brukes, GemXpresso RAD III, er en tilleggsmodul som legges til JBuilder 4. Dette sørger for API mot kortleseren. Precise SDK sørger for API mot biometridelen. Ellers følges rammeverket rundt OCF (OpenCard Framework) og RSA's standarder for nøkler og sertifikater (PKCS#1 og PKCS#15), samt ISO 7816-4 for kommandoer mot selve smartkortet. Man går ut i fra at alt som skal programmeres og utvikles, kan løses ut fra disse gitte API'ene og standardene.

Man må også anta at det ikke oppstår problemer med maskinvare og programvare, slik at dette skaper unødvendige forsinkelser, med tanke på reparasjoner og lignende. Dersom noe slikt skulle skje håper man på rask hjelp fra ErgoSolutions, slik at ikke tid sløses bort.

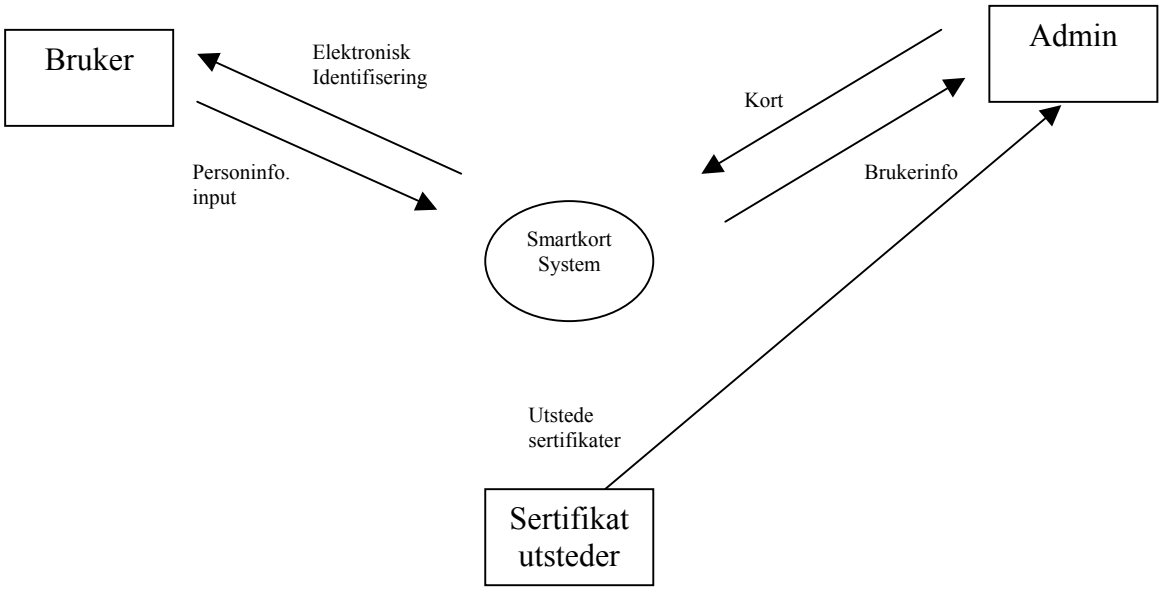
Dersom gruppen møter på programmeringsmessige problemer ikke kan løses internt, er man avhengig av assistanse fra arbeidsgiver og eventuelt veileder. Det må også forventes at gruppemedlemmer gir så mye som mulig, og hjelper hverandre når det oppstår problemer (Jfr. Grupperegelene, **[Vedlegg H]**).

Lagringskapasiteten til smartkortet (24 kb) må antas å være tilstrekkelig for å få plass til EID-applet'en. Det er det mest grunnleggende. Det stilles likevel krav til at koden gjøres så kompakt og effektiv som mulig. Det må i tillegg antas at det blir plass til minst ett fingeravtrykkstemplate. Ellers vil det være umulig å implementere fingeravtrykksidentifikasjon i forbindelse med EID.
Når det gjelder demoapplikasjonen, er denne avhengig av å kunne kommunisere med applet'en på smartkortet, det vil si bruke de funksjonene blir laget i applet'en. Man må gå ut fra at det ikke forekommer kommunikasjonsproblemer som skyldes ekstern programvare, som drivere, spesielle programinnstillinger og lignende.
Til drøftingen av sikkerhetsnivået på Javacard kontra MultOS', antas det å kunne brukes mange av våre erfaringer med Javacard. Tilgjengelig dokumentasjon må kunne skaffes angående MultOS for å i det hele tatt kunne foreta en sammenligning. På dette punktet er det fordelaktig om arbeidsgiver er behjelpelig med dokumentasjon og litteratur.

2.4 Funksjonell spesifikasjon

2.4.1 Funksjonell struktur med dataspesifikasjon og beskrivelser

2.4.1.1 Generell oversikt over systemet



Figur 2.2: Generell kontekst diagram for systemet.

Figur 2.2 gir en grov oversikt over hele systemet sett i ett. Den gir et overblikk over hvem som er i kontakt med systemet, og hvilken kommunikasjon de har med systemet. Det er tre entiteter som blir berørt av systemet:

- **ADMIN** – Administrerende enhet. Denne delen har med selve initieringen av kortet. Det vil si at når en bruker skal få utlevert et smartkort, må persondata og sertifikater lastes ned sammen med EID applet'en. Fra systemet mottar entiteten brukerinfo. Med brukerinfo menes sertifikater, nøkler, fingeravtrykk med mer. Dataflyt til systemet fra ADMIN, vil være kommandokall til smartkortet og dets EID-applet.

- **Sertifikatutsteder:** Sertifikatutsteder vil ordne de nødvendige sertifikater som skal brukes med EID-applet'en. Sertifikatet er klargjort på forhånd ut fra brukerens personinfo, slik at administrerende enhet bare trenger å legge sertifikat med privat krypteringsnøkkel på kortet. Sertifikatutsteder får ingen dataflyter tilbake fra systemet.

- **Bruker**: Bruker er den som får utstedt et smartkort med sin egen Elektronisk ID. Bruker vil gi sin personlige input til systemet. Med det menes personlig informasjon (navn, adresse, fingeravtrykk/PIN) og data som skal signeres. Tilbake fra systemet får bruker en elektronisk signering.

- **Smartkort System:** Dette er hovedprosessen i systemet. I denne prosessen skjer all behandling av smartkortkommunikasjon (autentisering : verifisering og signering) samt behandling av fingeravtrykk. Systemet ordner også grensesnittet mellom bruker/smartkort og administrasjon/smartkort.

På grunn av systemets kompleksitet og omfang, er det nødvendig å dele det opp i mindre selvstendige deler (moduler). Dette gir en bedre oversikt på et mer detaljert nivå. Hver modul regnes som et inkrement som kodes hver for seg. Disse settes sammen til et totalt system etter endt utvikling og testing.

Det har blitt valgt modul ut ifra hvilke deler av systemet som har flest fellestrekk som gjør det praktisk å programmere i moduler:

- **Inkrement 1**: EID–applet. Denne modulen omhandler applet'en som skal ligge på selve smartkortet. Utvikles ut ifra spesifikasjonen for EID som vi har fått av arbeidsgiver, se **[Vedlegg D]**. Dette er selve grunnsteinen i systemet som vi er avhengig av at fungerer før vi kan starte utvikling av de andre modulene. Den er også viktig med tanke på plassbruk på kortet. Resterende plass vil kunne brukes til fingeravtrykk.

- **Inkrement 2**: Biometri; behandling av fingeravtrykk. Denne har 2 mulige alternativer:

  1. **oncard** – Behandling av fingeravtrykk skjer på kortet. Det er plass til begge template'er på smartkortet ,og sammenligningen av fingeravtrykk under identifisering av kortholderen, skjer på selve kortet.

  2. **offcard** – Det er ikke nok plass på kortet til å lagre begge fingeravtrykk-template'ene, og/eller CPU'en på smartkortet har ikke kapasitet til å foreta de nødvendige beregninger under sammenligningsprosessen. Template'ene må da sammenlignes eksternt, på en PC.

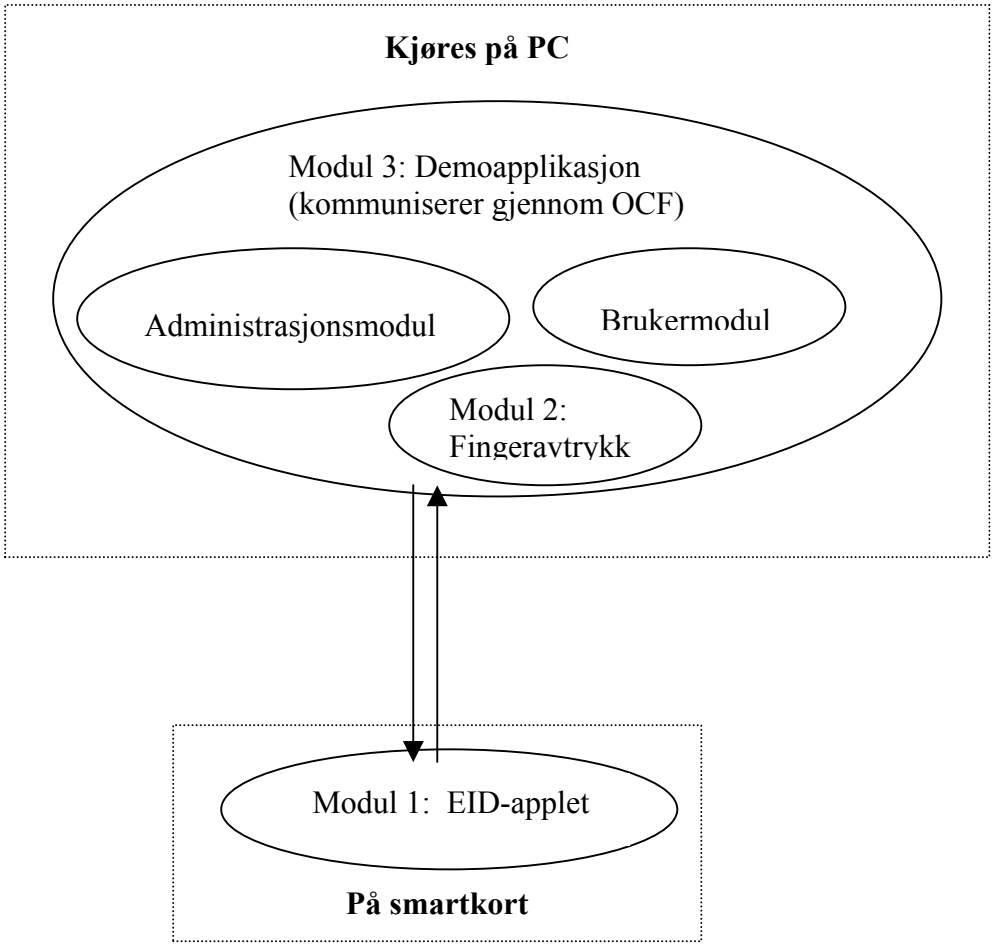
  Hvilken av disse to løsningene som blir valgt vil være et resultat av kapasitetstestingen på kortet (regnekraft og lagring), vurderinger av alternative løsninger, og ikke minst størrelsen til EID-applet'en.

- **Inkrement 3**: Demonstrasjonsapplikasjon. Denne applikasjonene skal teste EID-applet'en og biometrimodulen. Selve inkrementet er naturlig å dele inn i 2 deler, som gjerne kan utvikles samtidig:

  1. **Administrasjonsmodul** (3.1): Denne lages med tanke på initialisering av smartkortet med EID-applet, sertifikatsbehandling, nøkler og registrering av brukers fingeravtrykk til smartkortet.

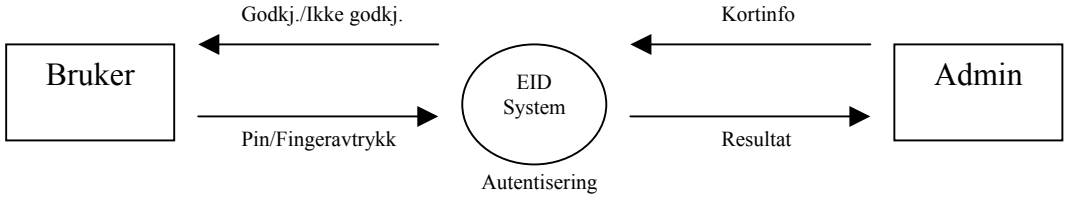
  2. **Brukermodul** (3.2): Denne modulen brukes for testing av EID-applet'ens autentiseringsegenskaper. Det vil si at brukeren kan signere et dokument han har skrevet ut ifra sertifikat som ligger på kortet, og sjekke at denne er riktig. I tillegg vil brukerens verifisering av fingeravtrykk demonstreres.

Da de 3 inkrementene kodes som moduler, vil de i prinsippet kunne kjøres uavhengige av hverandre. I dette prosjektet er det imidlertid et poeng at disse tre kommuniserer med hverandre, og hvert inkrement er en viktig del av systemet.
Av praktiske årsaker og som del prosjektets mål, legges Administrasjonsmodulen og Brukermodulen inn i en applikasjon med felles GUI, men de vil skilles med for eksempel et skilleark eller lignende. Det er viktig å poengtere at disse to modulene gjerne kan kjøres i applikasjoner hver for seg. Demoapplikasjonen bruker biometrimodulen i forbindelse med fingeravtrykkbehandling. Resultater fra biometrimodulen (template'er) kan sendes både til demoapplikasjonen og til smartkortet. Modulene i demoapplikasjonen kommuniserer også med EID-applet'en i form av standardiserte kall og får svar (response). Et kall kan for eksempel være å be om å signere en hashkode fra et tekst, og får ut ifra sertifikatet (den private nøkkelen) en signert hashkode tilbake. Se figur 2.3 for oversikt over det totale systemet.



Figur 2.3: Figuren viser hvordan modulene kommuniserer

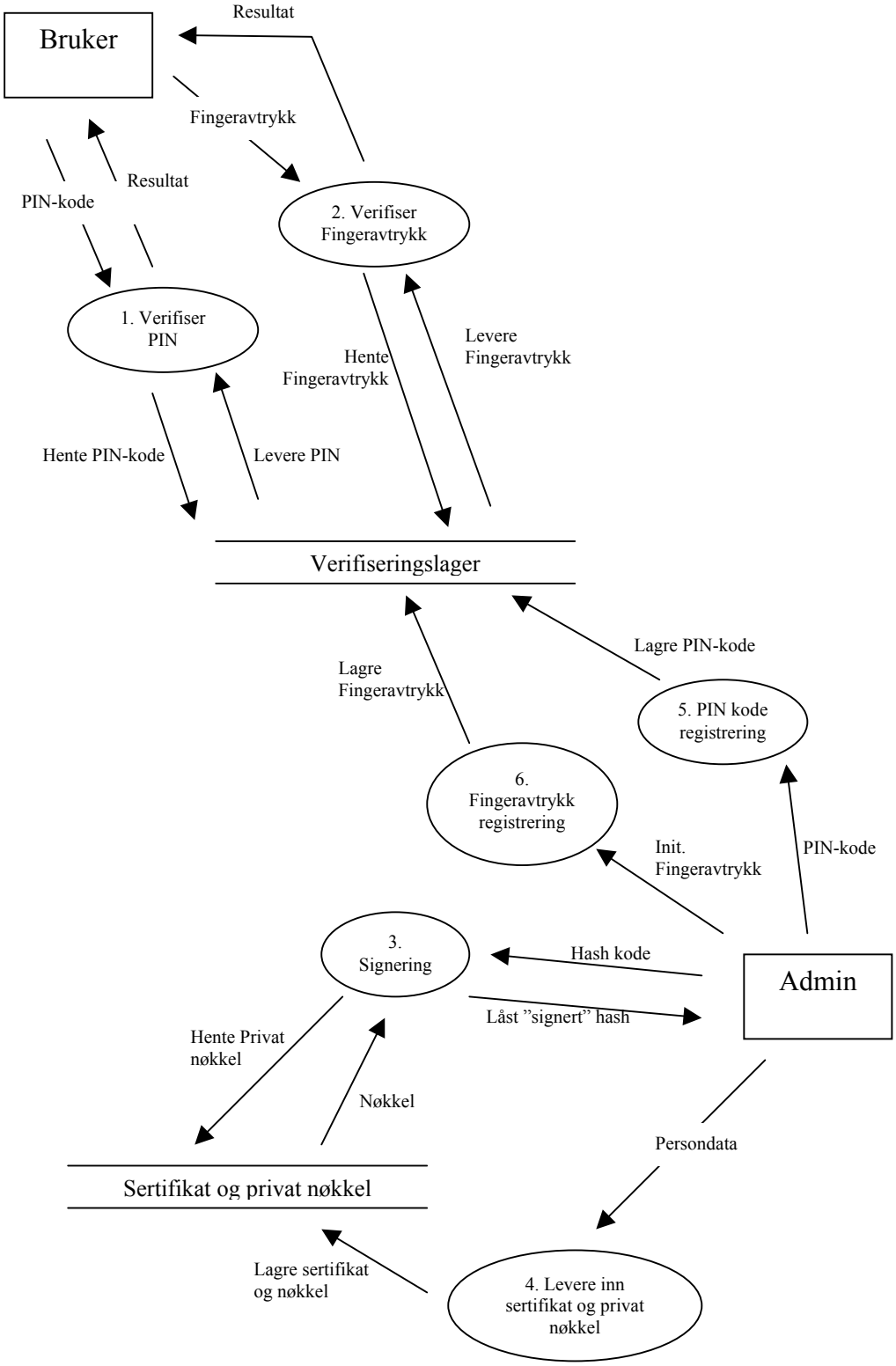
2.4.1.2 Inkrement 1: EID-applet



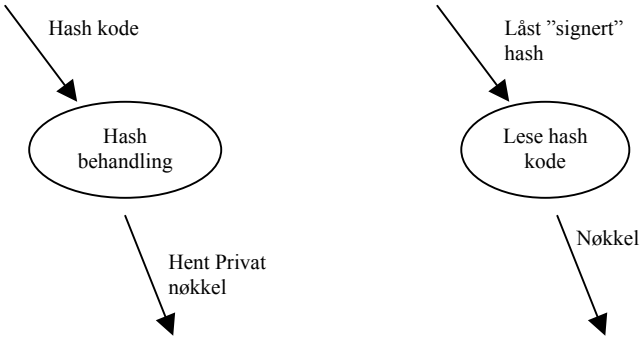
Figur 2.4: Kontekstdiagram EID-modul

Med EID (Elektronisk Identifisering) menes hva som skal til for å identifisere en bruker i ulike sammenhenger. Dette kan dreie seg om adgang til deler av et datasystem, dører i et bygg eller å kunne bekrefte at det dokumentet du har skrevet og sendt elektronisk over til en annen virkelig er blitt skrevet av deg, og at dette skjer på en tilfredsstillende måte. Figur 2.4 viser grov oversikt over EID inkrementet. Videre viser figur 2.5 og 2.6 dataflytene i økende detaljgrad.

I dette inkrementet blir det sett litt nærmere på hva som kreves for at en slik identifisering skal skje tilfredsstillende. Dette innebærer hva slags informasjon som må bli lagret på smartkortet til brukeren og hva han/hun må gjøre for å gjennomføre verifiseringen. I tillegg vil det bli sett på hva som det kreves av terminalen som brukeren benytter.



Figur 2.5: DFD-0 EID-modul



Figur 2.6: DFD-3.1 Signering i EID

Prosessene:

1. Verifiser PIN
En av måtene for brukeren å identifisere seg på er å taste inn en personlig PIN- kode. Dette er en fire sifferet kode som tastes inn enten rett på terminalen eller en ekstern enhet som har forbindelse til terminalen. Når brukeren har tastet inn PIN- koden blir denne sendt til terminalen som i sin tur vil koble seg opp mot et ’verifiseringslager’ hvor alle kodene og informasjonen om brukerne til systemet ligger lagret. Når terminalen har fått kjørt en sjekk opp mot dette lageret vil terminalen sende en melding tilbake til brukeren (resultatet). Denne meldingen kan være en av tre meldinger:

- PIN-koden som ble inntastet var korrekt
- PIN-koden var feil og antall gjenværende forsøk blir returnert, brukeren må da taste inn koden sin på nytt
- Kortet har blitt stengt som følge av at brukeren har tastet inn feil PIN-kode for mange ganger (det vanlige antallet forsøk er som regel 3, men dette er opptil programmererne av kortet. Etter kortet har blitt utgitt vil PIN og PUK bli blokkert for å unngå overskiving eller endringer).

2. Verifiser Fingeravtrykk
Dette er en annen måte som brukeren kan identifisere seg på. og er den verifiseringsmetoden som det vil bli lagt vekt på i dette prosjektet sammen med PIN-kode. Prosessen ovenfor, Verifiser PIN, er meningen skal kunne gi en større sikkerhet om ønskelig, selv om fingeravtrykk alene er regnet som sikkert. Det som skjer idet en bruker legger fingeren på fingeravtrykksleseren er at denne leseren tar bilde av fingeren og lager et template. Det er dette template’et som blir sendt til ’verifiseringslageret’ og sjekket om det har nok likhetstrekk med den originale template’en som allerede er blitt lagret idet brukeren fikk utstedet sitt kort. Et fingeravtrykk består av mange ulike linjer og krysspunkter mellom disse, det er disse som blir sjekket ved en autentisering. Grunnen til at det ikke bare vil være å godkjenne helt like templates, er fordi to bilder ikke vil bli 100% like, dette på grunn av hvor man plasserer fingeren på leseren, hvor hardt man trykker og hvor fuktig/tørr man er på fingeren. Resultatet av autentiseringen kommer ved bruk av meldinger. Disse meldingene vil være en bekreftelse på at fingeravtrykket var godkjent eller en melding som ber brukeren prøve igjen.

3. Signering
Ved signering er hensikten å kunne bevise for en annen bruker at man er den man utgir seg for å være (autentisering), og at det arbeidet som er blitt sendt med ditt navn virkelig er blitt gjort av deg. Så det en bruker gjør når noe han/hun har laget skal elektronisk signeres er å kjøre en hash algoritme, som begge har blitt enige om, på det som er blitt laget. Nå vil senderen overføre hash-verdien/koden over på smartkortet ved bruk av terminalen for å ”signere” hash-koden. Denne signeringen skjer ved bruk av en asymmetrisk algoritme som tar hash-koden den har fått inn i tillegg til brukerens private nøkkel som ligger lagret internt på kortet. Dette vil til sammen være med på å generere en ny kode/verdi som blir hva vi kaller en signert hash. Denne sendes tilbake til terminalen som tar seg av videre behandling av den nye signerte hash-koden. Hva som skjer videre vil bli forklart i inkrement 3.1 som tar for seg demoapplikasjonen som er programmet som står å kjører på terminalen.

4. Levere inn sertifikat og privat nøkkel
Et sertifikat inneholder personopplysninger som sertifikatutsteder går god for. Det inneholder også en nøkkel, dette er den offentlige nøkkelen og som en mottaker bruker for å verifisere signaturen fra senderen. Det finnes også en privat nøkkel som aldri skal forlate kortet. Det er denne nøkkelen som gjør at brukeren kan bevise at han/hun er den man utgir seg for å være. Sertifikat og privat nøkkel er det utstederen av kortet som generer for en ny bruker, men dette gjøres først når de kan gå god for at brukerens identitet er korrekt. Administratoren for bedriften mottar den private nøkkelen og lagrer den på kortets ’sertifikat og privat nøkkel’ lager.

5. PIN-kode registrering
Dette skjer når en ny bruker skal få utdelt et smartkort. Dette vil som regel være tilfelle f.eks når det blir ansatt en ny person ved bedriften. Registreringen er det som tidligere nevnt typisk administratoren ved bedriften som skal gjøre siden det er han/hun som ofte har ansvaret for at den daglige driften går som den skal. Det inkluderer også at de som skal ha rettigheter på forskjellige deler i bedriften får dette. Registreringen skjer ved at den nye brukeren velger et fire sifferet nummer som skal være han/hun sitt personlige adgangskode. Dette nummeret bør av sikkerhetsmessige årsaker ikke være ett nummer som lett kan identifiseres med brukeren, for å unngå at andre kan misbruke kortet. PIN-koden blir lagret sammen med brukerens informasjon på ’verifiseringslageret’.

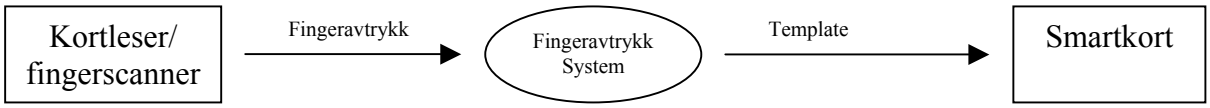
6. Fingeravtrykk registrering
Mye likt som ovenfor bare nå registreres det fingeravtrykk. Registreringen skjer ved at den nye brukeren registrerer et fingeravtrykk som er av god kvalitet. Administratoren registrerer sammen med dette brukerens informasjon på verifiseringslageret.

2.4.1.3 Inkrement 2: Biometri – behandling av fingeravtrykk

Biometri modulen vil ta seg av behandling av fingeravtrykk. Det vil si at modulen får fingeravtrykkbildet fra leseren, og vil ved hjelp av API kall generere et fingeravtrykktemplate. Demoapplikasjon vil gjøre kall mot biometrimodulen slik at den returnerer fingeravtrykktemplate'en. Modulen vil, naturlig nok, motta et kall fra demoapplikasjonen som starter hele prosessen, men vi har valgt å utelukke denne dataflyten for å vise selvstendigheten i modulen.

Det finnes, som tidligere nevnt, to løsningsalternativer. Disse er å dele inkrementet i to deler, der vi kommer til å lage en modell for hvert løsningsalternativ:

1) Oncard
Denne modellen er utgangspunktet dersom det viser seg at det ikke blir nok plass på smartkortet til å lagre to template'er, det vil si to fingeravtrykksignaturer. I sin originale størrelse, har template'en en størrelse på ca 1,9 kb. Da det skal være plass til to av dem, må kortet ha ledig lagringskapasitet på underkant av 4 kb. I tillegg må smartkortet ha en kraftig nok CPU til å gjennomføre beregningene som kreves ved en sammenligning av disse to template'ene. Denne sammenligningsfunksjonen legges inn i EID-applet'en, slik at man kan kalle denne prosedyren fra demoapplikasjonen. Også når biometrimodulen skal sende en template til smartkortet, må det også eksistere en funksjon i EID-applet'en som tar imot og fysisk lagrer denne template'en på smartkortets filsystem. Denne modellen er den beste med tanke på sikkerhet, da det originale fingeravtrykket til smartkortets bruker, aldri forlater selve smartkortet i noen form.



Figur 2.7: Kontekstdiagram for oncard-modul

Av figur 2.7 ser man at oncard-modulen har to entiteter som fingeravtrykksystemet får dataflyter fra. Fra Kortleser/fingerscanner kommer det fysiske bildet (rådata) fra fingerscanneren. I selve systemet blir dette fingeravtrykket konvertert til en fingeravtrykktemplate. Dette er en ganske komplisert funksjon som er implementert i API'et vi skal bruke. Den ferdige template'en sendes så til smartkortet der det lagres ved hjelp av EID-applet'en.

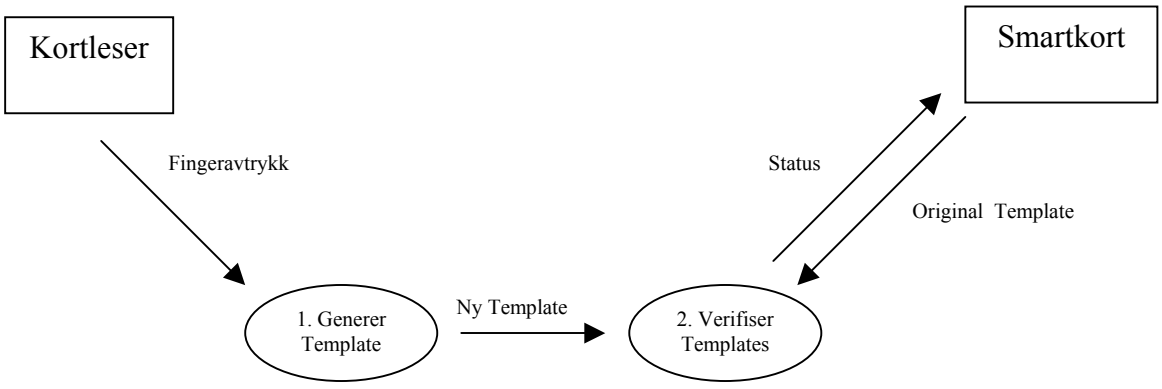
Hvis ikke template'en vil kunne få plass må det forskes på om denne kan gjøres mindre. Dette innebærer å se på hvilke standarder denne template'en følger, og om hvilke risikoer det innebærer å endre på denne. En egen funksjon som forandrer template'en må i så fall lages. Det er meget mulig at det ikke blir tid til dette, da det krever mye tid til å sette seg inn i template'ens oppbygging og virkemåte.

2) Offcard

Denne modellen vil bli bruk dersom det viser seg at oncard-modellen ikke lar seg gjennomføre. Prinsippet denne modellen bygger på, er at biometrimodulen skal ta seg av det meste innen fingeravtrykksbehandlingen. Dersom denne modellen velges, må originalt fingeravtrykk på smartkortet hentes og legges i minnet på den tilhørende PC'en. Dette krever at det eksisterer en funksjon i EID-applet'en som kan returnere fingeravtrykket som er lagret på kortet. Ved autentisering av brukeren, legges også dens fingeravtrykk i minnet. Deretter sammenlignes disse. Resultatet av dette sendes til EID-applet'en.

Figur 2.8: Kontekst diagram for offcard-modellen

I kontekstdiagrammet for "offcard" modellen, figur 2.8, er det 2 entiteter som fingeravtrykkssystemet jobber mot; Kortleser med fingerscanner og EID-applet'en på smartkortet. Systemet mottar et fingeravtrykk (rådata) fra fingerscanneren på kortleseren. EID-applet'en sender, ved hjelp av funksjonskall fra biometrimodulen, template'en av det opprinnelige fingeravtrykket som ligger på kortet. Systemet foretar en sammenligning, og sender resultatet av dette til EID-applet'en.

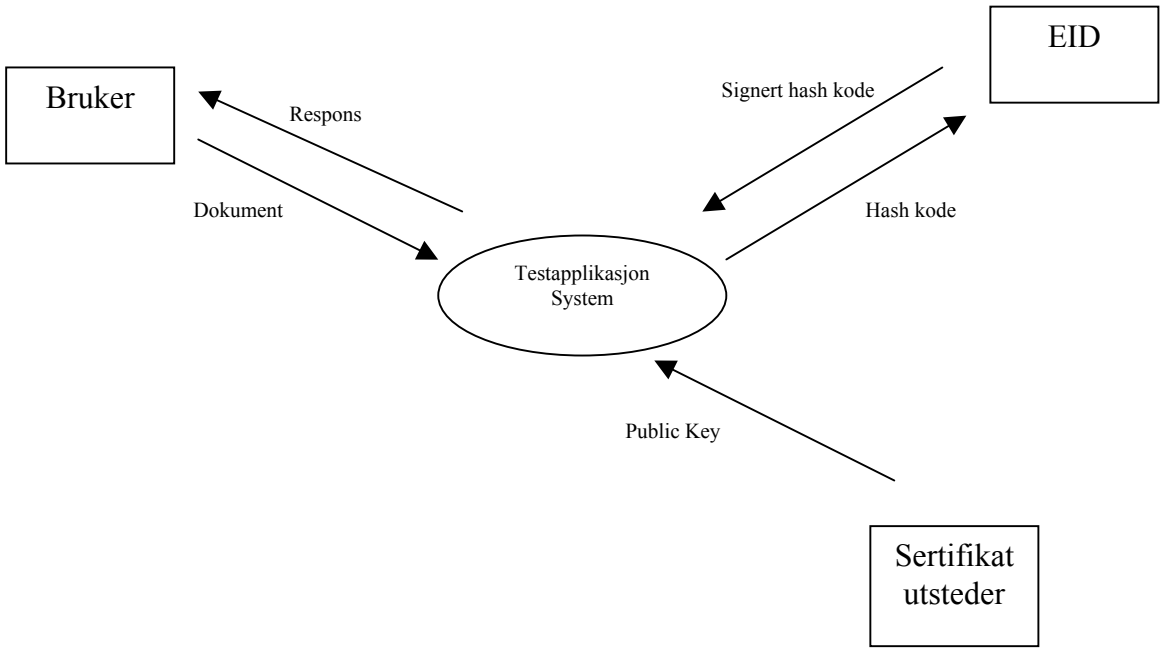


Figur 2.9: DFD-0 Diagram for offcard-modellen

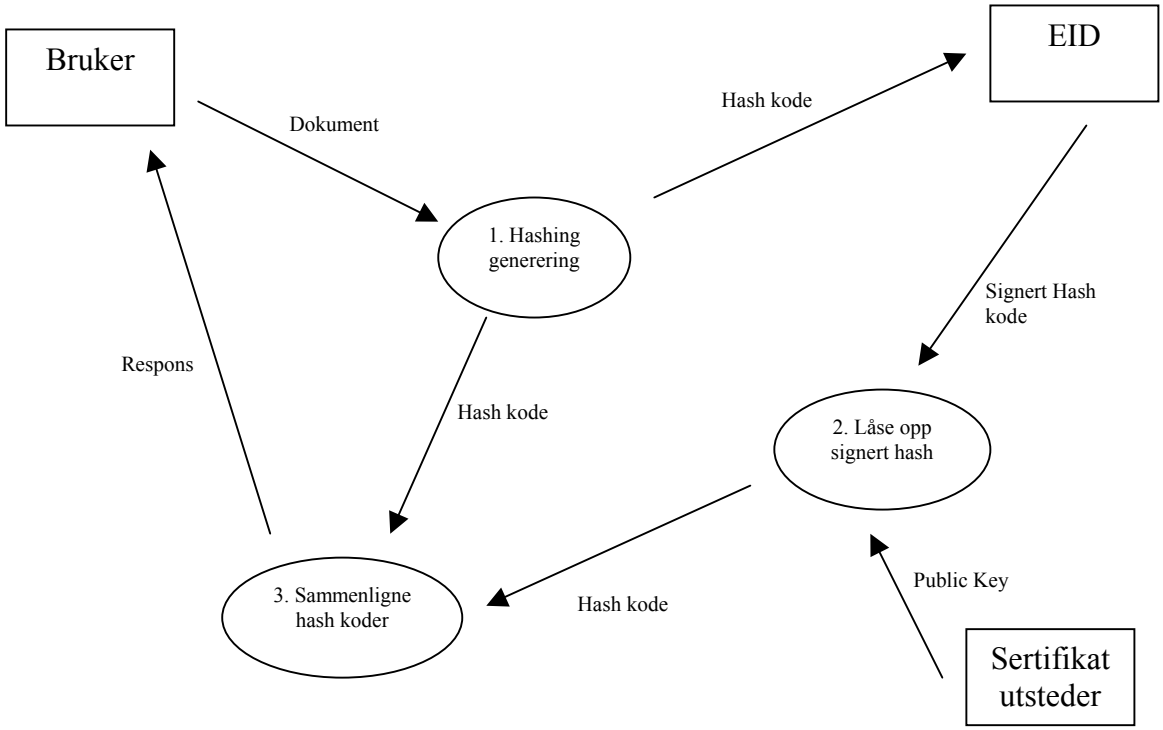
Systemet kan splittes opp i et DFD-0 diagram, se figur 2.9, for å få frem flere funksjoner. Fra fingerscanneren kommer fingeravtrykket som rådata, det vil si et bilde av fingertuppen. Dette må forandres til en template som API'et bruker. Denne genereringen av template'en skjer ved hjelp av API kall. Den ferdige template'en sendes til en prosess som sammenligner template'en med det som ligger på smartkortet. Smartkortets template må hentes via EID-appeleten, som sender template'en inn til funksjonen som sammenligner de to template'ene. Resultatet av dette sendes så til EID-applet'en for videre behandling. Ut fra resultatet, blir da opp til applet'en å avgjøre om brukeren får tilgang til kortets funksjoner.

2.4.1.4 Inkrement 3: Demonstrasjonsapplikasjon

Demonstrasjonsapplikasjonen vil bli delt i to moduler. En for brukere, og en for administratorene. Det er viktig for sikkerheten å få skilt de to modulene. Det vil samtidig gjøre en eventuell senere utvidelse eller forandring noe lettere. I dette prosjektet vil de to modulene smelte sammen i form av en samlet GUI. I et system der sikkerheten er viktigere, bør de enhetene skilles. Bruker- og administrator- modulen vil opptre i form av to forskjellige klasser. Figur 2.10 viser brukermodulens overordnede design.



Figur 2.10: Kontekstdiagram Demoapplikasjon, Inkrement 3.1 (Personmodul → EID)



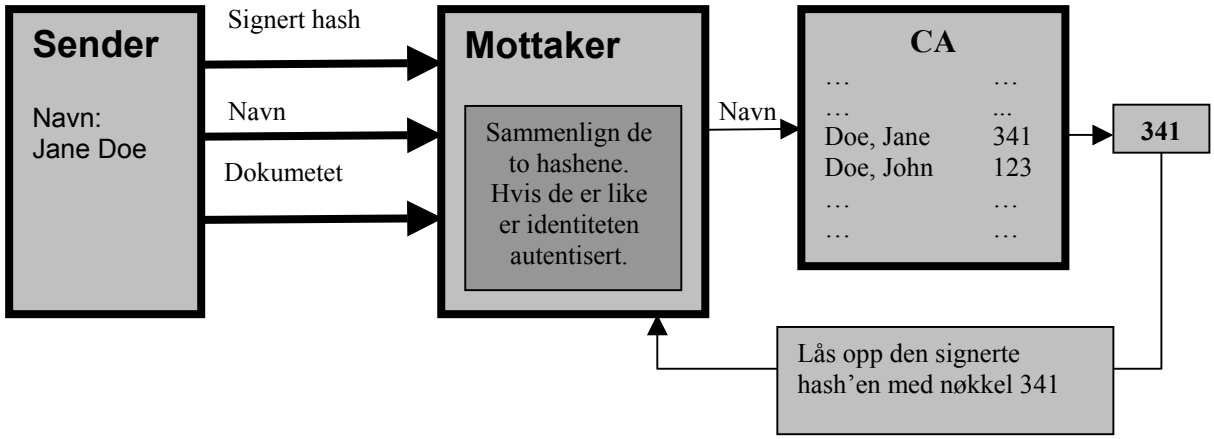
Figur 2.11: Demonstrasjonsapplikasjon DFD-0 (Inkrement 3.1)

Denne modulen, eller klassen er den som brukeren benytter seg av. Modellen i dette prosjektet er meget enkel og helt overordnet. Kortet skal kun brukes til signering av et dokument. Men det er rom for utvidelser på dette området. Figur 2.11 viser flytdiagram for brukerprosessen.

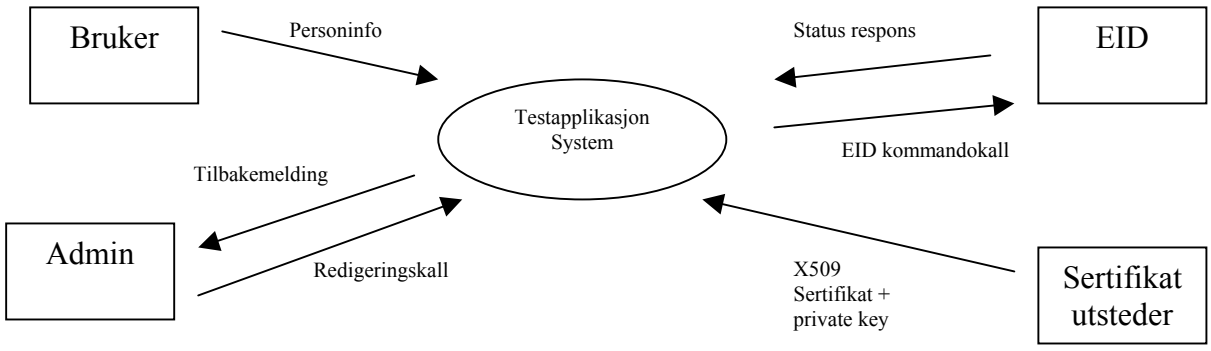
Først vil brukeren autentiseres med fingerscanning og eventuell PIN-kode. Det neste steget da er å signere dokumentet, som er den eneste funksjonaliteten av denne modulen. Det velges et dokument som skal signeres og brukeren trykker på en 'signer' knapp på GUI'en. Applikasjonen vil ikke stå å poll'e hele tiden, men brukeren skal selv hente frem dokumentet og bestemme når signeringen skal skje. Dokumentet blir sendt til en Hash-prosess på maskinen hvor dokumentet gjennomgår en totalforandring. Outputen blir en fastlengde streng generert ut ifra en algoritme som moser igjennom dokumentet. Hash-koden blir som regel mye kortere en dokumentet og det blir nesten et unikt sett med bit for hvert enkelt dokument. Sjansen for at to dokumenter får samme hash er forsvinnende liten. Hash'en blir så sendt til kortet og signert med brukerens private nøkkel. Signaturen blir deretter sendt til en prosess for å låses opp med senderens offentlige nøkkel, som blir hentet fra organisasjonens database. Ideen er nå å sjekke om personen er den han/hun gir seg ut for å være ved å sammenligne hash'en som ble generert av dokumentet (i prosess 1.) mot senderens opplåste signatur (prosess 2.). Denne sammenligningen gjøres i 'sammenligningsprosessen'. Hvis de to hash-kodene er like så er autentifiseringen positiv.

I dette prosjektet har dette i praksis ikke så mye for seg. Med fingercanning og PIN-kode verifisering vet man med stor sikkerhet hvem man har med å gjøre. Men selve nytte effekten kommer når man skal sende autorisasjoner over nettverk/Internett. Da holder det ikke med PIN og/eller fingerscanning.

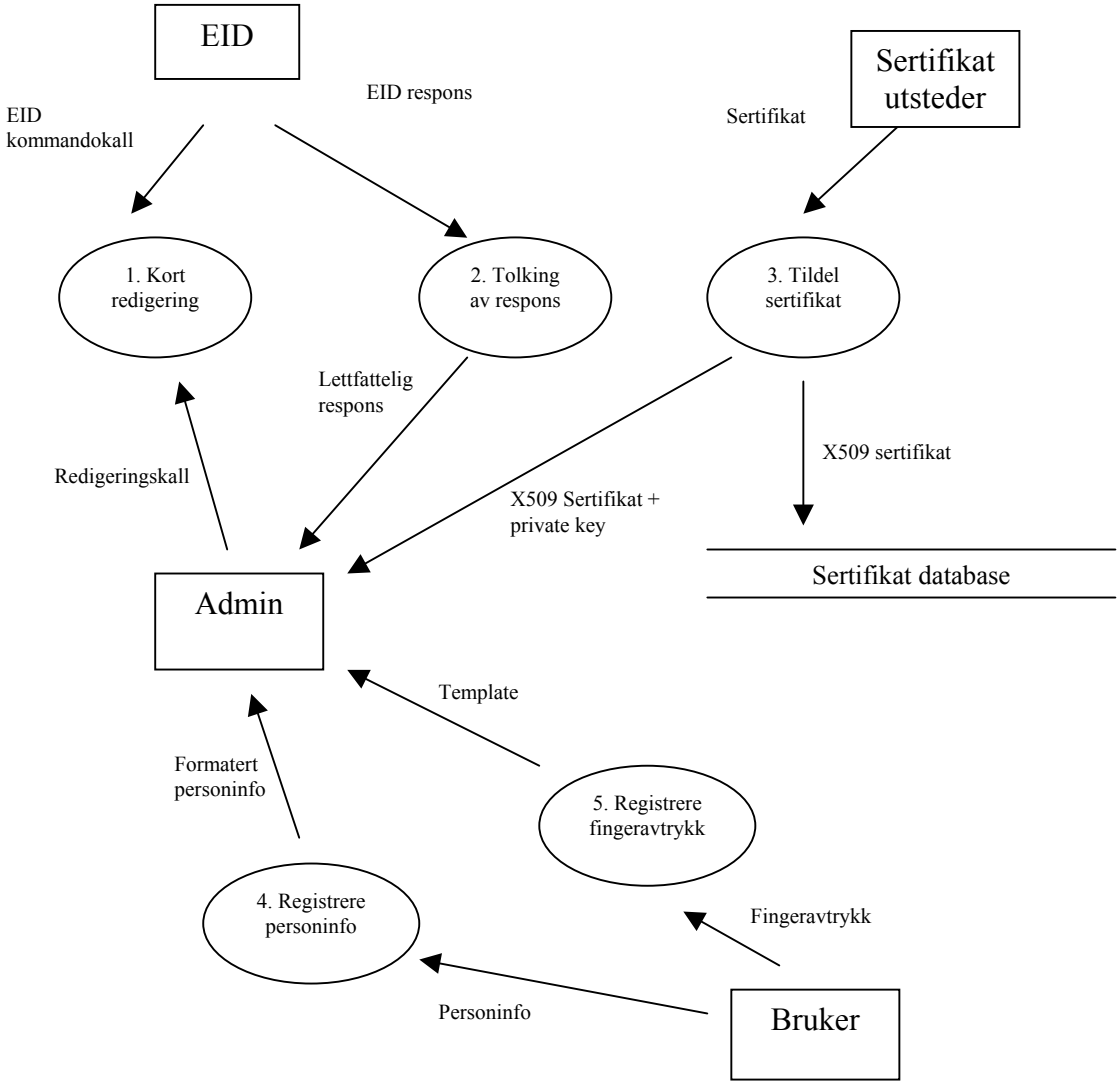
I praktisk bruk i et system over nett, vil 'signaturen' bli sendt til mottaker sammen med dokumentet. Dokumentet blir sendt i klartekst så mottakeren må vite hvilken hash-algoritme som ble benyttet, dette for å kunne generere sin egen hash-kode. Sammenligningen blir gjennomført av mottaker på samme måte som beskrevet ovenfor. Er disse like betyr det at dokumentet virkelig kom fra senderen og at det ikke har skjedd endringer under selv sendingen. Figur 2.12 forklarer selve signerings systemet i bruk. Der har man en CA hvor en mottaker kan hente senders offentlige sertifikat, som inneholder nøkkelen.



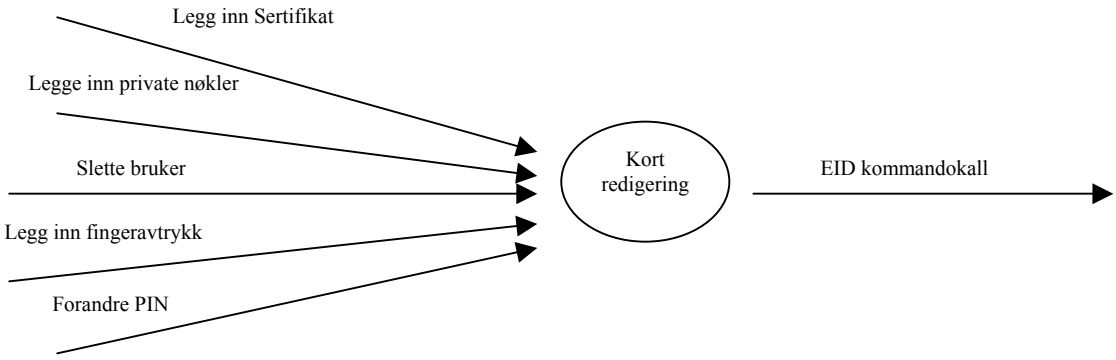
Figur 2.12: Prinsipper omkring signering og hash og PKI



Figur 2.13: Kontekstdiagram Demoapplikasjon, Inkrement 3.2 (Administrasjonsmodul → EID)



Figur 2.14: Demoapplikasjon DFD-0, Inkrement 3.2



Figur 2.15: DFD-1.1 Demoapplikasjon, Kortredigering

Figur 2.13 viser en grov oversikt over prinsippene rundt kommunikasjon og hvilke aktører som er involvert i demoapplikasjonen. Videre viser figur 2.13, 2,14 og 2.15 utvalgte detaljering av dataflyter. Det sentrale punktet i figur 2.14 er Admin; altså dette er representert i form av administreringsdelen i demoapplikasjonen og en administrerende person. Hvis systemet skal brukes i en situasjon som krever sikkerhet, bør administrasjonsdelen legges atskilt ifra brukerdelen.

Administrator mottar data om en person, enten ved en skriftlig henvendelse eller fra en database. Blant de data som er relevante kan Navn, Etternavn, Adresse, Fødselsnummer trekkes frem. Det blir også nødvendig å få et fingeravtrykk fra personen. Dette gjøres ved at fingeren blir scannet til et bilde med tilstrekkelig kvalitet oppnås, og ut ifra dette blir en template laget. Dette blir så den template'en som legges inn på kortet og senere brukes til verifisering av fingeravtrykk. Innhenting av personinfo og fingeravtrykk er delt opp i to prosesser.
Personinfo blir hentet først fra de aktuelle personene, dataene blir behandlet til ønskelig format i et sertifikat, og så sendt til kortutsteder. Prosessen med å sende personinfo fra administrator og kortutstederer ikke tatt med siden det ikke er relevant for prosjektets problemstilling.

Dette fører til neste steg i prosessen. På bakgrunn av informasjon sendt fra administrator, lager kortutsteder en Public key og en Private key som sendes tilbake til administratoren. Med prosessen 'Tildel sertifikat" blir kortets offentlige nøkkel (Public Key) lagt på en database som den tenkte organisasjonen har. Dette er en database som bør være tilgjengelig til enhver tid. Minstekravet er at databasen er tilgjengelig samtidig som autentisering foretas. Oppfylles ikke dette kravet vil ikke autentiseringen fungere. Samtidig som Privat key blir lagt i databasen, så behandles kortet av administrator. Moduler blir lagt til sammen med fingeravtrykket og en eventuell PIN-kode. Kortene er nå klar til bruk og leveres til respektive brukere.

Administrasjonsdelen skal kunne legge til og forandre persondata på et kort. Kortet skal kunne sperres og forandringer av kortdata i form av PIN-kode og fingeravtrykk bør være mulig.

Alle kommandoer fra administrator blir omgjort til smartkort-kommandoer i en prosess kalt "Kort redigering". Dette blir en omfattende prosess. Blant annet må lange data brytes ned til flere sendinger, siden man bare kan sende 255 bytes om gangen, inkludert 5 bytes header og 1 byte trailer. Denne prosessen jobber direkte mot EID-applikasjonen. Funksjonens status består av en 16 bit kode, 2 heksadesimale tall, som indikerer om operasjonen var vellykket eller hva som gikk galt. Denne responsen blir tolket i en annen prosess "Tolking av respons", som tar imot bit'ene og ut ifra disse lager en ny tilbakemelding som er forståelig.

2.4.2 Operasjonelle systemkrav

2.4.2.1 Normal operasjon

Under følger en beskrivelse av hvordan systemet vil fungere under normal bruk.

2.4.2.1.1 Modus og kontroll

Systemet må kjøres på en PC med Windows 98/NT/ME/2000 siden det bare finnes drivere for fingeravtrykkleseren/kortleseren til disse operativsystemene.
Systemet har 2 moduser:

- Registrering av bruker
- Autentisering av bruker ved hjelp av digitale signaturer

I modus 1, Registrering av bruker, vil informasjonen om brukeren bli registrert på kortet. Dette innebærer sertifikater med informasjon som brukerens navn, adresse samt offentlig RSA nøkkel, privat nøkkel, fingeravtrykk og PIN.

I modus 2, Autentisering av bruker, vil foregå ved bruk av fingeravtrykk, det kan også være aktuelt å implementere identifisering med en PIN-kode, som nå er den vanlige identifiseringsmåten på de fleste betalingskort. Bruker vil kunne ha mulighet til å ”signere” dokumentene som han/hun skriver med sin elektroniske signatur. Dette vil være en like god bekreftelse på hvem som skrevet det som en vanlig underskrift/signatur skrevet for hånd.

2.4.2.1.2 Innebygde tester

Det vil være tester på om kommandoene som forsøkes kjørt på kortet, er riktige i følge EID spesifikasjonene, se EID spesifikasjonen **[Vedlegg D]**. Den viktigste testen vil være om fingeravtrykket matcher brukerens fingeravtrykk som allerede er registrert på kortet slik at ingen kan utgi seg for å være en man virkelig er. Om det skulle bli aktuelt med PIN-kode vil denne også bli sjekket. Demoapplikasjonen skal også sjekke svar den får fra EID på smartkortet, samt å teste at den digitale signaturen fra kortet er korrekt.

2.4.2.2 Operasjoner i feilsituasjoner

2.4.2.2.1 Feilrapportering

Om det skulle oppstå en feil skal brukeren få melding om hva som har gått galt. Denne meldingen skal være lett forståelig slik at en bruker/administrator , dersom nødvendig, lett kan rette opp feilen som ble gjort.

2.4.2.2.2 Hva skal gjøres når feil oppstår

Ved alvorlige programmeringsfeil, blir det opp til ErgoSolutions å håndtere dette. Det er arbeidsgiver som kommer til å videreutvikle dette systemet. Moduler som blir brukt i senere systemer, må rettes for eventuelle programmeringsfeil. Disse kan oppdages ved senere utvikling og bruk.

2.4.2.2.3 Sikkerhet

Sertifikater og private nøkler må behandles konfidensielt. Systemet må designes slik at data blir sendt i sikre kanaler (for eksempel kryptert) slik at det blir vanskelig å snappe opp data og bruke disse. Systemet må gjennomgå grundig testing for å unngå sikkerhetshull og eventuelle programmeringsfeil.
Fingeravtrykk og eventuelle PIN-koder er også personlig informasjon som ikke må komme på avveie. Sikkerhetsmessig sett, bør fingeravtrykk bli implementert på smartkortet (jfr. Oncard modellen). Filsystemet på kortet må også ha adgangskontroll på sensitive data, slik at disse ikke kan leses og kopieres.

2.4.3 Begrensninger: Software og Hardware

Å se på begrensningene som software og hardware setter for utviklingen av produktet er en del av prosjektets oppgave. Hvilke begrensninger setter bruken av Javacard? Sikkerheten på grunn av liten lagringsplass på kortet? Prosessering som burde foregått på kortet må kanskje flyttes over til PC'en.

Hva vi vet og må forholde oss til:

Software:

- Inprise (Borland) JBuilder 4.0
- Gemplus GemXpresso RAD III (Utviklingsmiljø for Javacard applets)
- Java JRE, v 1.2.2
- Presice Biometrics API for Java

Hardware:

- PC (Pentium II @ 266 Mhz med 192 mb RAM)
- En kombinert fingeravtrykksleser og smartkortleser
- Smartkort med Javacard OS fra Gemplus, modell GXP211PK. Den har innebygd co-prosessor for krypteringsoperasjoner, og 24 kb lagringskapasitet (EEPROM).

Vi vet at det smartkortet (GXP 211PK) vi skal benytte oss av er på 24k, og at dette vil skape problemer med å få den programvaren og informasjonen som er nødvendig liten nok slik at den får plass på kortet. Prosessoren på kortet er heller ikke stor, 7,5 MHz, noe som fører til at tiden det tar å prosessere en oppgave kan bli så stor at det ikke vil være hensiktsmessig å la dette skje på kortet. Dette er beslutninger som vi må foreta etter hvert som problemene dukker opp. I tillegg til en liten prosessor er det ikke mye informasjon som kan sendes under kommunikasjonen mot smartkortet. Data sendes ved bruk av APDU'er (Application Protocol Data Unit) hvor det ikke er mulig å sende mer enn 255 bytes med data i tillegg til 6 bytes header og trailer informasjon. Vi har også et API som vi må følge, noe som gjør jobben vår noe lettere, men det setter også begrensninger for hva det vil være mulig for oss å gjøre, med tanke på egne spesialiserte kommandoer.

Vi har EID-relaterte standarder som vi må følge:

- ISO 7816-4 Kommandogrensesnitt mot APDU'er
- ISO 7816-8 Sikkerhets relaterte kommandoer
- ISO 7816-9 Ekstra kommandosett
- PKCS#1 Nøkkelformater og padding-mekanismer
- PKCS#15 Filorganisering og kommandogrensesnitt for EID

Vi har også mottatt en teknisk spesifikasjon fra arbeidsgiveren som bygger på PKCS#15 standarden, se **[Vedlegg D]**.
Fingeravtrykksleseren er en Precise 100, som er integrert i kortleseren, Gemplus Touch 430. Denne enheten blir bare levert med drivere for Windows 98/NT/ME/2000 så det må benyttes en maskin med en av disse operativsystemene, dersom drivere til andre systemer ikke kan skaffes.

2.4.4 Aspekter omkring livssyklus

Dette prosjektet innebærer ikke at vi skal lage et ferdig system som skal tas i bruk for en spesifikk oppgave. Det går mer på forskning og utprøving om EID lar seg implementere på Javacard sammen med fingeravtrykksidentifisering, da dette ikke har vært gjort før hos ErgoSolutions. Dersom vi lykkes med dette, danner dette prosjektet grunnlag for arbeidsgiver å vurdere å ta denne teknologien i bruk, dersom det er forsvarlig sikkerhetsmessig. Arbeidet vi gjør kan danne grunnlag for senere prosjekter der EID og fingeravtrykk kan brukes i for eksempel e-handel, eLommebok og lignende, samt områder hvor sikkerheten i dag er for dårlig. Fingeravtrykksverifisering leverer større grad av sikkerhet enn PIN-kode.

Prosjektet kan også danne grunnlag for senere utvikling og tilleggsfunksjoner, når kapasiteten på kortene etter hvert blir større. Med økt lagringskapasitet, åpner det for å legge mange applikasjoner på samme kort, samt raskere prosessering av data på kortet. Det også mulig at rutinene for fingeravtrykk blir enda bedre, sikrere og ikke minst raskere i fremtiden.

All kode som vi lager kan lett overføres til annen maskinvare som støtter API'ene vi bruker. Demoapplikasjonen kodes i Java og kan derfor kjøres plattformuavhengig dersom det finnes drivere til korteleseren/fingerscanneren for den spesifikke plattformen. Det er også et krav at det brukes samme eller tilsvarende smartkort med støtte for samme krypteringsstandarder som brukt i dette prosjektet.

2.4.5 Aspekter omkring installasjon

Det som trengs for at systemet skal kunne tas i bruk, vil være en PC med en fingeravtrykksleser og en smartkortleser tilkoblet.
Kravet til PC'en er at den kan kjøre Java VM, samt ha drivere til kortleser. I tillegg, må Gemplus GemXpresso RAD III være installert for å få tilgang til klasser som brukes til demonstrasjonsapplikasjonen.

Programmet for kommunikasjonen mot smartkortet vil for brukeren være rimelig enkelt å bruke, slik at det ikke vil trengs noen spesiell opplæring i bruken av det.
Det vil være utenfor våre rammer å lage installeringsrutiner for demoapplikasjonen. Installasjon av EID, vil skje i fra demoapplikasjonen. En slik installasjonsrutine ville vært aktuelt å lage for systemer der GemXpresso RAD III ikke er installert. I installasjonsrutinene ville det da vært naturlig å legge inn alle de nødvendige klassene og konfigurasjonsfilene som terminalapplikasjonen trenger.

2.4.6 Akseptansekrav

Med dette mener vi minstekravet som MÅ oppfylles før vi kan si oss ferdig med et inkrement. Vi har delt opp prosjektet i tre inkrementer: EID'en, fingeravtrykkscanningen og demoapplikasjonen.

1) **EID:**
   Dette er det viktigske inkrementet, og denne må kunne fungere før vi sier oss ferdig med den. Vi kan oppleve at størrelsen på modulen vil overskride størrelsen vi har til rådighet på kortet. Hvis størrelsen på modulen er dobbelt så stor som kortets lagringskapasitet vil vi ikke implementere EID'en på kortet, men heller levere koden. Det finnes allerede kort med større lagringskapasitet. Hvis koden er litt større enn lagringskapasiteten på kortet vil vi prøve å trimme koden til den passer. I dette tilfelle vil vi levere både den trimme koden og den utrimmede.

2) **Biometri:**
   Denne delen er den mest eksperimentelle i prosjektet. Det er derfor ikke noe krav at den skal implementeres. Vi ønsker å få implementert den på kortet, jfr. Oncard, men det kan vise seg å bli vanskelig. Neste skritt blir å legge det Offcard, men vi prioriterer EID'en foran fingerscanningen. Hvis EID'en tar for stor plass, vil det gå på bekostning av fingerscanningen. Det kan vise seg at vi må utelukke denne funksjonen for nå. Men koden for en eventuell senere implementasjon vil blir lagt med.

3) **Demoapplikasjon:**
   Her er kravet at vi må lage en applikasjon som må kunne vise at kortet fungerer som det skal; altså kunne signere et dokument. Det er ønskelig å legge til en administreringsdel, der man kan redigere kortet, men på grunn av tidspress er det ikke sikkert det blir mulighet til å utvikle denne.

2.4.7 Prosjektstyring og kvalitetssikringskrav

2.4.7.1 Prosjektets hoveddeler

- Forprosjekt
- Kravspesifikasjon
- Design
- Programmering
- Evaluering/testing
- Klargjøring av rapport

Dette gjenspeiler seg også i gannt diagrammet som brukes for disponering av tid og resurser. Se **[Vedlegg F]**.

2.4.7.2 Krav til statusmøter og beslutningspunkter

Det ble avtalt fire statusmøter med veileder, henholdsvis 30.01, 13.02, 06.03, 03.04 og 24.04.

Beslutningspunkter/milepæler:

- Ferdig forprosjekt
- Ferdig kravspesifikasjon
- Ferdig Design
- Ferdig lesing av grunnleggende bakgrunnsstoff
- Ferdig Koding
- Ferdig Testing
- Ferdig Skriftlig analyse av sikkerhet
- Ferdig Rapport
- Fremføring

Siden vi valgte inkrementell systemutvikling, tilsier dette at skillet mellom de forskjellige delene av prosjektet ikke er klart avskilte. Særlig under kode, design og testingsfasene vil nok en del av disse punktene foregå parallelt. Dette gir oss imidlertid en mer fleksibel arbeidsmetode enn mer tydelige faseinndelte systemutviklingsmodeller. Det henvises også her til **[Vedlegg F]** for nærmere informasjon angående milepæler i gannt diagrammet.

2.4.7.3 Organisering av kvalitetssikring

2.4.7.3.1 Dokumentasjon

- Standard for syntaks på koding, interkommunikasjon, kommentarer, dokumentasjon o.l. Se punkt 4.2.2 og 4.2.3 for detaljene rundt disse.

- Møtelogg/Aktivitetslogger.
  Gir et håndfast dokumentasjon på avgjørelser og ansvarsfordeling, samt hendelsesforløp. Se **[VEDLEGG G]**

- Layout på skrevet materiale. Gir et uniformt bilde av skriftlige arbeider for å oppnå god struktur og lesbarhet. Se vedlagt cd-rom **[\Møtereferater\referat_28.01.pdf]**.

2.4.7.3.2 Tiltak ved avvik/problemer

- I all hovedsak løses problemene via prosjektleder, hvis problemet er at en slik at det kreves diskusjon, innkalles det til møte.

- Dersom prosjektleder i samråd med resten av gruppen ikke finner noen løsning innen ”rimelig tid”, vil veileder/arbeidsgiver kontaktes.

- Ved sykdom av mer langvarig art, omfordeles arbeidsoppgavene.

- Fravær skal meldes til prosjektleder så fort som mulig, senest 1 dag i forveien. Vesentlige grunner bør foreligge. Vedkommende har selv ansvaret for å oppdatere seg etter fraværet, gjerne i samråd med prosjektgruppen.

- Hvis gruppedeltaker ikke greier fristen skal prosjektleder kontaktes før fristen for å finne alternativ løsning. Hvis ikke dette er mulig må enten arbeidsoppgavene omfordeles, eller vedkommende må ta inn igjen det tapte. Muntlig eller skriftlig avviksrapport til prosjektleder kreves.

2.4.7.3.3 Verktøy

- MS Word 2000 som base for rapporten.

- MS Project 98 som base for selve prosjektstyringen.
  Gir på en enkel måte oversikt over milepæler, status på prosjektet, arbeidsfordeling etc.

- Timelogger en enkel oversikt over hver av gruppedeltakernes innsats og arbeidskapasitet, noe som kan være nyttig ved fordeling av arbeidsmengde, samt for hver deltaker individuelt.

- Inprise (Borland) JBuilder 4,Gemplus GemXpresso RAD III
  Brukes for selve kodingen.

2.4.7.3.4 Backup-rutiner

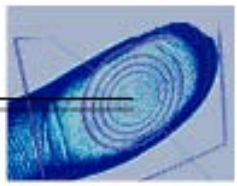
Essensielt å ikke miste viktige data derfor:

- Hver gruppedeltaker har oppdatert virusbeskyttelse.
- Hver bruker tar backup av eget arbeid hver dag.

I tillegg lastes arbeidet en gang i uken til prosjektleders nettområde, samt felles lagring på egen server, Zuzu-5. Zuzu-5 synkroniserer innhold med prosjektleder hver fredag og dette brennes på CD. I tillegg er websiden dublert, både på skolens webserver, samt Zuzu-5.

DESIGN

Java SmartSign 2002



3 Design

3.1 Innledning

Dette kapittelet bygger på kravspesifikasjonen, dataflytdiagrammene og diskusjonen rundt disse, i forhold til hvilke valg vi har tatt med hensyn på løsingen av oppgaven .Leseren skal få et detaljert, men oversiktlig innblikk i hvordan systemet er bygd opp, og hvordan det fungerer. Det vektlegges også å forklare hvordan og hvorfor de valgte og nødvendige løsninger brukes prinsipielt, uten å gå detaljert inn i selve koden. Å gå inn på selve koden, vil være alt for komplekst og uoversiktlig for dette dokumentet. Derfor henvises det, for spesielt interesserte, til dokumentasjonen av kildekoden på vedlagt cd-rom, **[\Dokumentasjon\]**, der kildekoden og spesielle løsninger forklares i detalj. Dokumentasjonen er spesielt skrevet for ErgoSolutions for senere videreutvikling av systemet

Siden systemet har blitt utviklet etter Inkrementell systemutviklingsmetode, er det naturlig at resten av dokumentet, tar for seg inkrement etter angitt prioritet. Se kravspesifikasjonen for nærmere detaljer angående inndeling og organisering av inkrementer.

3.2 Inkrement 1: EID

3.2.1 Innledning

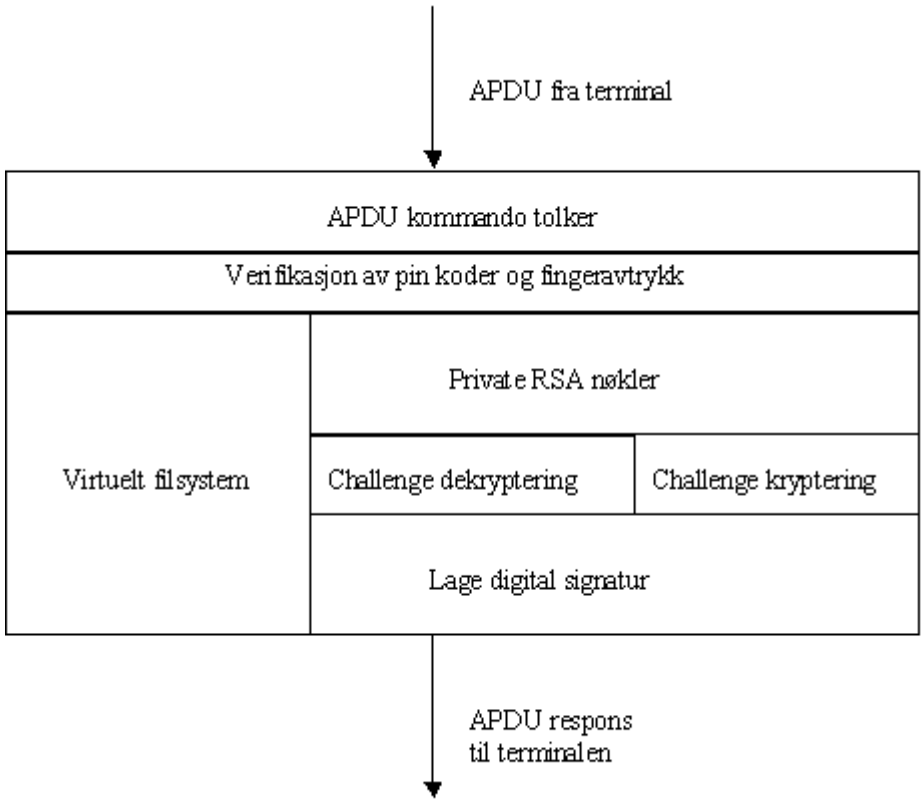
Dette programmet, som vanligvis kalles en ”applet”, er den delen av systemet som skal lastes ned på smartkortet. Denne vil utføre hovedjobben i systemet, nemlig å lage en digital signatur. For å få til dette, trengs en del støttefunksjonalitet:

- Virtuelt filsystem for lagring av filer, i følge ISO7816-4
- Identifisering av kortholder: PIN koder og fingeravtrykk
- Krypteringsalgoritme: RSA
- Krypteringsnøkler: 1024 bits privat (private) RSA nøkler
- Hashalgoritme: SHA
- Signeringsalgoritme: RSA
- X509 sertifikater med 1024 bits offentlig (public) RSA nøkkel

EID designes ut fra ErgoSolutions spesifikasjon, **[Vedlegg D]**. Denne er spesielt laget for smartkort med MultOS som operativsystem, og ISO 7816-4 kompatibelt filsystem. Javacard, som brukes her, er objektorientert, og har derfor ikke støtte for et slikt filsystem. Dette innebærer at et virtuelt filsystem må utvikles for å håndtere filene som EID bruker. Utviklingen av EID applet’en har vært sterkt preget av å følge EID spesifikasjonen, bruk av kryptografiske funksjoner API’et tilbyr, samt tilpassninger av Javacard til ISO7816-4

filsystemet. Dette innebærer at vi har blitt bundet til standarder, og alternative løsninger har derfor blitt noe begrenset. Der vi har vurdert løsninger, er i forbindelse med det virtuelle filsystemet, samt behandlingen av template for verifisering av fingeravtrykk. Delene angående APDU, kryptografi og PIN koder har stort sett blitt utviklet etter API'ets begrensinger.

For å lettest illustrere designprinsippene ved EID applet'en, vises det til figur 3.1, som gir en oversikt over hvordan EID fungerer. EID mottar kommandopakker, APDU'er fra en terminalapplikasjon. Denne blir tolket, og sendt videre til kommandoens respektive funksjon. For at en viss funksjon skal kunne brukes, må PIN koder og/eller fingeravtrykk godkjennes. Deretter kan det enten jobbes mot EID's filsystem, ellers så kan de kryptografiske funksjonene benyttes. Et svar på resultatet av kommandoen returneres til terminalapplikasjonen



Figur 3.1: Oversikt over hvordan EID applet'en er bygd opp. Hver del Diskuteres i detalj på de neste sidene

3.2.2 APDU fra terminal

Dette er datapakken som smartkort bruker for å kommunisere med terminalapplikasjonen. Denne inneholder kommandoinstruksjon samt eventuelle data som skal sendes til kortet. Se tabell 3.1 for hvordan APDU er bygd opp. Kommandoene som brukes mellom EID applet'en og terminalapplikasjonen er definert i ErgoSolutions EID spesifikasjon **[Vedlegg D]**.

Felter	CLA	INS	P1	P2	Lc	Data	Le
Byte nr.	1	2	3	4	5	0-255	Siste
Beskrivelse	Type	Instruksjon	Parameter 1	Parameter 2	Data lengde	Nytte – data	Forventet svarlengde

Tabell 3.1: Oversikt over APDU oppbygning

3.2.3 APDU kommando tolker

Denne kommandotolkeren sjekker APDU header CLA, og spesielt INS byte for å finne ut hvilken funksjon kortet skal kalle videre. Av Javacard API'et, er dette en standardmetode som må implementeres: `public void process(APDU apdu)`. Denne forkaster alle APDU pakker der INS eller CLA byte ikke er støttet. For komplett oversikt over alle kommandoer støttet og dens APDU, se Eros Solutions EID spesifikasjon **[Vedlegg D]**.

3.2.3 Verifikasjon av PIN koder og fingeravtrykk

EID er tilgangsbeskyttet på to måter:

- Kortholder:
  - PIN kode for identifisering og bruk av RSA nøkkel: ID
  - PIN kode for bruk av RSA nøkkel for kryptering/dekryptering: DC
  - PIN kode for bruk av RSA nøkkel for digital signatur: DS
  - Fingeravtrykk lagret på kortet som fil 4701, sjekkes av terminalen ved bruk av EID applet for identifisering av kortholder.

- Kortutsteder:
  - PIN kode for skriving av filer til det virtuelle filsystemet, som gjøres ved initialisering av EID når smartkortet utstedes: IS

Det ble valgt å bruke 3 PIN koder som kortholder må ha validert, før en digital signatur kan lages. De tre private RSA nøklene må beskyttes slik at de ikke skal brukes av andre enn kortholder. Alternativet er å kun bruke fingeravtrykk og ingen PIN koder i det hele tatt, bortsett fra IS PIN koden for administrering. Men dette avhenger av at fingeravtrykket blir verifisert og godkjent i selve EID applet'en, i stedet for terminalapplikasjonen. En slik "oncard" løsning (Se Kravspesifikasjonen), viste seg å være umulig å implementere grunnet hovedprosjektets omfang. Implementasjonen av en slik løsning krever store kunnskaper om biometrialgoritmer for generering og sammenligning av fingeravtrykk-template'er. Arbeidsmengden på dette er et hovedprosjekt i seg selv. Derfor må sammenligning av

fingeravtrykktemplate'er skje på terminalapplikasjonen. Dette foregår ved at kortholderens fingeravtykktemplate hentes fra fil 4701 i filsystemet. Deretter sammenlignes så dette mot kortholderens fingeravtrykk fra fingerscanneren, som avleser fingeravtrykket under verifikasjon og identifikasjon av kortets eier. Dersom disse stemmer overens, tillates bruk av kortet, ved sending av en egen APDU til EID som forteller at fingeravtrykket er godkjent. Se tabell 3.2 for hvordan denne APDU'en ser ut.

CLA	INS	P1	P2	Lc	Data	Le
00	20	00	84	02	90 00	00

Tabell 3.2: APDU for å sette at fingeravtrykket er godkjent.
NB: Denne er ikke definert i EID spesifikasjonen.

Denne måten å fortelle EID at fingeravtrykket er godkjent på, er en sikkerhetsrisiko i seg selv, da hvem som helst med litt smartkort kunnskaper kan, med godkjent ID PIN, sende denne APDU'en til kortet uten å ha vært i nærheten av en fingerscanner engang. Derfor bør fremtidige versjoner av EID implementere "oncard" løsningen, dersom fingeravtrykk skal brukes til å identifisere kortholderen.
For mer informasjon angående dette emnet se **[Vedlegg B]**.

For å styrke den totale sikkerheten til RSA nøklene til et akseptabelt nivå, valgte vi å beskytte disse med egne PIN koder. Alternativt, for brukervennlighet, kan disse kodene settes til samme kode fra terminalapplikasjonen, dog med noe høyere sikkerhetsrisiko.

3.2.4 Virtuelt filsystem

Javacard smartkort bygger på objektorienteringsprinsippet. Derfor har den ikke innebygd støtte for filsystemer. Siden EID spesifikasjonen, [se vedlegg ditt og datt] er filsystemorientert, måtte vi utvikle vårt eget virtuelle filsystem som bruker egendefinerte objekter til å representere filer. I den sammenheng representeres filer av den selvlagde klassen File, se**[Dokumentasjon\Dokumentasjon – eid – Fil representasjon på EID.pdf]** på cd-rom.
Filsystemet er delvis dynamisk og delvis statisk. Dette vil si at noen typer filer genereres automatisk ved installasjon av EID applet, mens andre opprettes etter behov. Se tabell 3.3 for oversikt over filsystemet.

FID	Beskrivelse	Dynamisk	Brukertilgang (ID PIN)
4301	Initialiseringsinfo. Private RSA DC key	Nei	Ingen
4302	Initialiseringsinfo. Private RSA DS key	Nei	Ingen
4303	Initialiseringsinfo. Private RSA ID key	Nei	Ingen
4401	X509 sertifikat til DC key	Ja	Lese
4402	X509 sertifikat til DS key	Ja	Lese
4403	X509 sertifikat til ID key	Ja	Lese
4701	Fingeravtrykktemplate (original)	Ja	Lese (Ingen ved ”oncard” løsning”)
4702	Fingeravtrykktemplate (test hvis ”oncard”)	Ja	Full
5031	PKCS#15 filer, ikke nødvendig for Javacard versjon av EID, men for kompabilitet. Se EID spesifikasjon for nærmere informasjon.	Nei	Lese
5032		Nei	Lese
5110		Nei	Lese
5120		Nei	Lese
5150		Nei	Lese
5160		Nei	Lese

Tabell 3.3: Oversikt over det virtuelle filsystemet i EID applet

Å la noen filer være dynamiske, har den hensikt å spare sårt trengt plass på kortet dersom ikke alle filer brukes. For å begrense størrelsen på filsystemet og koden som omhandler dette, har filsystemet flat struktur og hver fil ligger i en egen File variabel. Ikke alle filene definert i EID spesifikasjonen, er støttet her, da bl.a. PIN og PUK administrasjon er objekter i stedet for filer. Javacard har disse representert som objekter, og det falt mer naturlig å bruke de. Det brukes FID (File IDentifier) for å plukke en bestemt fil. Hver fil har tilgangsbeskyttelse etter hvilken PIN som er validert. IS PIN har full tilgang til alle filer, mens ID PIN kun har begrenset lese tilgang, og ingen skrive tilgang (med unntak av FID=4702). Det er viktig å nevne, at File objektene i EID er gjenbrukbare. Både FID og brukerrettigheter, kan endres, men filens lagringskapasitet kan ikke endres. Gjenbruk er viktig, siden smartkortet vi bruker, ikke har noen form for garbage collector.
Ellers skal filsystemet være mest mulig kompatibelt med ISO 7816-4 standarden. For nærmere informasjon angående filene og APDU kommandoene i forbindelse med filsystemet, se EID spesifikasjonen **[Vedlegg D]**. Filsystemet brukes primært for skriving av data til kortet under utsteding av kortet. Filene som skrives, er de private nøklene, X509 sertifikatene, kortholderens originale fingeravtrykktemplate samt PKCS#15 filene. Dessuten vil sertifikatene leses fra kortet og returneres til terminalapplikasjonene når en digital signatur blir laget, og da brukes filsystemets lesefunksjonalitet. (Read Binary).

3.2.5 Private RSA nøkler

EID bruker tre1024 bits RSA nøkler i forbindelse med generering av den digitale signaturen:

- DC: For dekryptering av Challenge (test av nøkkelpar).
- DS: For selve den digitale signaturen.
- ID: For kryptering av Challenge (test av nøkklepar).

Nøklene er representert i objekter (RSAPrivateCrtKey), men må initialiseres med nøkkeldata fra de tilsvarende filene FID=4301-4303 før de kan tas i bruk. Dette er nødvendig, da nøklene ikke blir generert i EID applet'en. Formatet på denne filen, er kodet etter PKCS#1 standarden for private RSA nøkler, med ASN.1 syntaks. Selve initialiseringen valgte vi å gjøre automatisk ved første gangs bruk av nøkkelen, så det er noe både kortutsteder og kortholder slipper å tenke på. Alternativet ville være å sende en APDU som setter i gang initialiseringen, men det vurderte vi som en dårlig løsning med unødvendig kommunikasjon og potensielle sikkerhetsrisikoer. Innhentingen av nøkkeldataene som initialiserer de private RSA nøklene har vi kodet som en egen rutine og den henter ut følgende data som initialiseres med nøkkelen i henhold til PKCS#1:

- Primtall 1, p
- Primtall 2, q
- Eksponent 1
- Eksponent 2
- Koeffisient

De private nøklene tilhører en nøkkel par, der den offentlige (public) nøkkelen ligger i tilsvarende sertifikat (FID=4401-4403). Denne hentes ut til terminalapplikasjonen, slik at terminalapplikasjonen og EIDs nøkler tilhører samme nøkkelpar. Sertifikatene inneholder også persondata om kortholder.

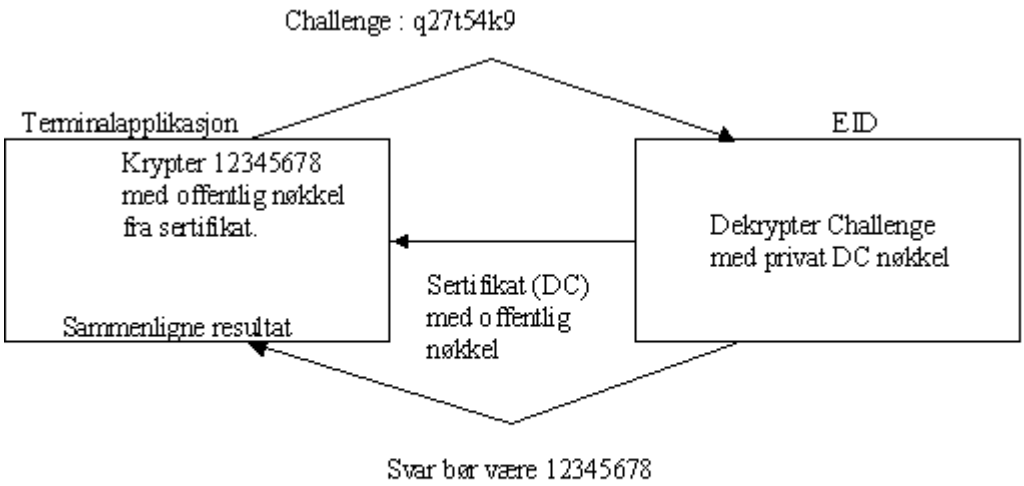
De private nøklene er "hemmelige" og skal kun brukes av kortholder. De fungerer som en bekreftelse på at kortholder er den personen han utgir seg for å være. Derfor blir de beskyttet av hver sin PIN kode, som må valideres før bruk av den tilsvarende nøkkelen.

For nærmere og mer detaljert informasjon, se dokumentasjon på cd-rom, **[\Dokumentasjon\Dokumentasjon – eid – EID applet.pdf]**.

3.2.6 Challenge dekryptering

"Challenge" er et ord som brukes i forbindelse med testing av nøkkelparene mellom EID applet'en og terminalapplikasjonen. Denne testingen utføres med hensikt for å finne ut om terminalapplikasjonen og EID applet'en kan stole på hverandre med hensyn på bruk av RSA nøklene. Challenge dekryptering er en av sikkerhetsrutinene EID støtter, og brukes sammen med Challenge kryptering før en digital signatur kan lages. Det er påkrevd å ha både ID PIN og DC PIN validert før denne metoden kan brukes. I tillegg har vi valgt å kreve at fingeravtrykket også skal være godkjent, for å øke sikkerheten omkring bruk av de kryptografiske funksjonene.

Prinsippet for en dekryptering er som følger. Terminalapplikasjonen lager en datamengde på noen bytes (16-32). Terminalapplikasjonen krypterer så denne med offentlig nøkkel hentet ut fra DC sertifikatet (FID=4401) . Dette blir da en ”Challenge” (utfordring) som sendes til EID applet’en. EID dekrypterer denne og fjerner eventuell PKCS1 padding. Siden nøklene er 1024 bits (=128 bytes) må derfor datalengden på data som skal krypteres eller dekrypteres, være 128 bytes. Svaret sendes så tilbake til terminalapplikasjonen. Dersom returdataene er de samme som den opprinnelige datamengden, kan det da konkluderes med at nøkkelparet stemmer overens og terminalapplikasjonen krypterer riktig. Se illustrasjon figur 3.2. Vi hadde ikke så mange muligheter for å lage denne metoden på, og ble derfor nødt til å følge API’et. For nærmere informasjon, se dokumentasjon av koden på cd-rom, **[\Dokumentasjon\Dokumentasjon – eid – EID applet.pdf].**



Figur 3.2: Prinsippet for Challenge dekryptering.

Pseudokode over dekrypteringsprosessen i EID:

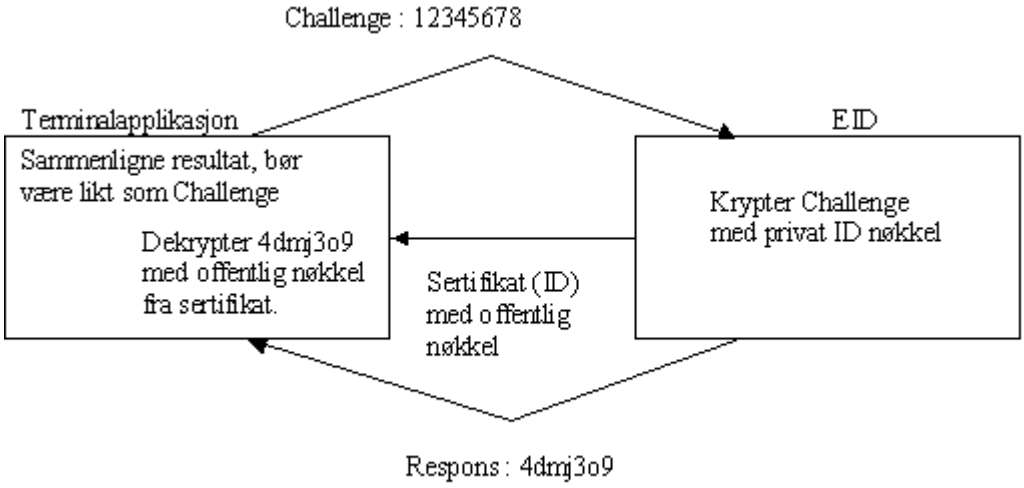
```
DekryptereChallenge()
  {
    if(dc pin og id pin og fingeravtrykk != validert)
      return feil;
    if(privat DC nøkkel != initialisert )
      initialisernøkkel(DC);
    dekrypter(128 bytes APDUdata med DC key og RSA algoritme);
    fjernPKCS1Padding(dektrypterte APDU data);
    send(dekryptert data uten padding);
  }
```

3.2.7 Challenge kryptering

Dette er også en testing av nøkkelpar. I stedet for DC nøkkelparet, som i Challenge dekryptering, testes her ID nøkkelparet. Også denne sjekkes før det kan lages en digital signatur, for å forsikre at nøkkelparet som brukes mellom terminalapplikasjonen og EID er i samsvar med hverandre. Kravet til å bruke denne funksjonen, er at ID PIN og DC PIN er validert. I tillegg har vi også her valgt å kreve at fingeravtrykket må være godkjent før funksjonen kan brukes.

Prinsippet for Challenge kryptering er som følger:

Terminalapplikasjonen lager en datamengde på ca 16-32 bytes. Denne sendes til kortet. Kortet legger til PKCS1 padding for at datalengden skal bli akkurat 128 bytes. Deretter krypteres dataene med den private RSA ID nøkkelen, og sendes tilbake til terminalapplikasjonen. Her blir de mottatte dataene (128 bytes) dekryptert med den offentlige RSA nøkkelen fra ID sertifikatet. Dersom de dekrypterte dataene samsvarer med de originale sendt til EID, hører nøkkelparet sammen og terminalen og EID gjennomfører kryptografiske funksjoner riktig. Etter at både ID og DC nøkkelparene stemmer, blir det antatt at også DC nøkkelparet stemmer, og det er klart for å lage en digital signatur. For illustrasjon av Challenge kryptering, se figur 3.3. Også her, er programmereren veldig knyttet til å følge API'ets måter å gjennomføre en dekryptering på. Alternativer måter å gjøre dette på er derfor meget begrenset.



Figur 3.3: Prinsippet for Challenge kryptering.

Pseudokode for krypteringsprosessen i EID:

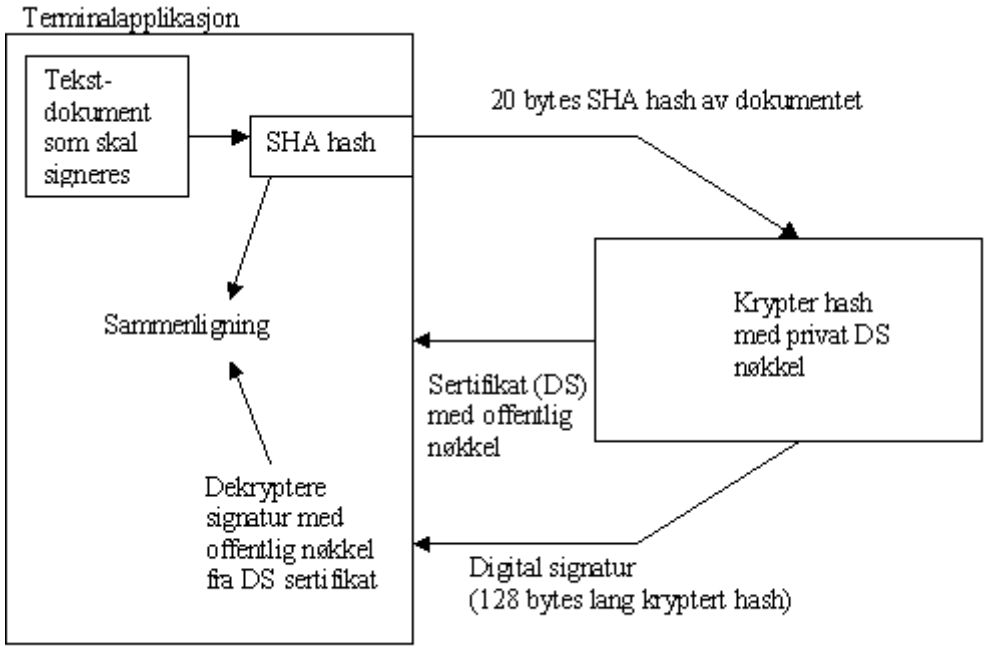
```
KryptereChallenge()
  {
   if(dc pin og id pin og fingeravtrykk != validert)
     return feil;
   if(privat ID nøkkel != initialisert )
      initialisernøkkel(ID);
   leggtilPKCS1Padding(APDU data);
   krypter(128 bytes ferdigpaddet APDUdata med ID key og RSA algoritme);
   send(128 bytes kryptert data);
  }
```

For nærmere informasjon, se dokumentasjon av koden på cd-rom,
[\Dokumentasjon\Dokumentasjon – eid – EID applet.pdf].

3.2.8 Lage digital signatur

Dette er hovedoppgaven til EID, nemlig å generere en digital signatur ut fra en hash sendt til kortet. Her brukes den private RSA nøkkelen DS for å lage signaturen. Denne er beskyttet av DS PIN koden, ID PIN koden, samt at Challenge kryptering og Challenge dekryptering må være gjennomført, før signaturen kan lages. I tillegg er det som vanlig påkrevd at ID PIN og fingeravtrykket er validert, før det gis tillatelse til å lage signaturen.
Prosessen skjer som følger. Terminalapplikasjonen genererer en 20 bytes lang SHA hash ut fra dokumentet, eller teksten som skal signeres. Denne sendes til kortet og EID vil da kryptere denne med den private RSA DS nøkkelen i en signeringsfunksjon. Hash'en må som vanlig være 128 bytes, grunnet lengden på RSA nøkkelen. Paddingen følger PKCS1 standarden. Den krypterte hash'en, som da er selve signaturen, returneres til terminalapplikasjonen.

Ved en verifikasjon av den digitale signaturen, dekrypteres denne hash'en med DS sertifikatets offentlige nøkkel, og paddingen fjernes. Dersom resultatet stemmer overens med en SHA hash av dokumentet, vil dette si at signaturen er riktig, og dokumentet er på sin originale form fra forfatteren (kortholderen). Selve prosessen er illustrert i figur 3.4.



Figur 3.4: Prinsippet for digital signatur

Pseudokode for prosessen som genererer den digitale signaturen i EID:

```
LageDigitalSignatur()
  {
   if(id pin og ds pin og fingeravtrykk != validert)
     return feil;
   if(privat DS nøkkel != initialisert)
     initalisernøkkel(DS);
   signer(APDU data med private DS key og SHA hash algoritme og RSA
                                                          kryptering);
   send(128 bytes digital signatur);
  }
```

For nærmere informasjon, se dokumentasjon av koden på cd-rom,
[\Dokumentasjon\Dokumentasjon – eid – EID applet.pdf].

3.2.9 APDU respons fra terminalen

Enhver APDU kommando sendt til kortet returnerer minst en statuskode. Dersom data skal returneres til terminalapplikasjonen, kommer disse dataene før statuskoden. Statuskoden er delt på 2 bytes, og bør alltid behandles fra en terminalapplikasjon. Det er verdt å merke seg at alle kommandoer som sendes til EID, som kjører på normal måte uten feil, returnerer 90 00, som betyr ingen feil. For detaljert oversikt over returdata og status koder, refereres det til EID spesifikasjonen **[Vedlegg D]** samt dokumentasjonen til EID kildekoden på **[\Dokumentasjon\Dokumentasjon – eid – EID applet.pdf]**.

3.3 Inkrement 2: Biometri

3.3.1 Innledning

Denne delen omhandler hvordan vi har løst problemet angående bruk av fingeravtrykk som identifiseringsmetode for kortholderen. Av kravspesifikasjonen hadde vi to mulige løsninger for å identifisere kortholderen å velge mellom:

- En *oncard* løsning, der fingeravtrykktemplate'ene sammenlignes i selve EID applet'en på smartkortet.
- En *offcard* løsning, der sammenligningen skjer i terminalapplikasjonen, ved å sammenligne mot en originaltemplate som hentes fra en fil på kortet.

Etter en del undersøkelser, viste det seg at vi ble tvunget til å velge offcardløsningen av følgende årsaker:

- API metodene for sammenligning av template'er lar seg ikke implementere i Javacard JRE på kortet, grunnet begrenset støtte for datatyper.
- Javacard 2.11 API'et støtter ikke biometri.
- Det finnes lite/intet ferdig kode for sammenligning av template'er på Javacard som vi kunne bruke. Dette medfører vi måtte ha laget en egen algoritme/funksjon for sammenligning av template'er ut fra en matematisk beskrevet biometrialgoritme, som lar seg implementere i Javacard. Ikke nok med det, men vi måtte også laget en metode for å lage template'en ut ifra et fingeravtrykksbilde fra fingerscanneren. Noe som var uaktuelt, grunnet kompleksitet og tidsbruk, siden rammeverket vårt er å holde seg til ferdigdefinerte API.
- Precise Biometrics, leverandøren av fingeravtrykks-API'et, ville ikke ut med hvordan template'en som vi bruker, er bygd opp. Derfor kunne vi ikke lage en tilsvarende sammenligningsalgoritme for kortet, siden vi ikke kunne få informasjon om hvilke data vi eventuelt skulle sammenligne.
- Biometri er "big-business", og meste av teknologien beskyttes.
- Egen pakker for behandling av fingeravtrykk på smartkort kan kjøpes, men dette var uaktuelt for vår del.

For nærmere informasjon om dette temaet, henvises det til **[Vedlegg B]**.

3.3.2 Offcard løsningen

Denne løsningen innebærer at kortholder identifiserer seg med fingeravtrykk. Fingeravtrykket er en rimelig sikker måte å identifisere seg på, dersom fingeravtrykket blir sjekket og behandlet på en sikker og god måte. Offcard løsningen kan ikke karakteriseres som en sikker måte, da kortholderens originale fingeravtrykkstemplate'et må leses fra kortet før en identifisering kan gjennomføres. Deretter må en APDU sendes til kortet og fortelle EID at fingeravtrykket er godkjent. Dette er ikke noen god måte, da det ikke er noen garanti for at fingeravtrykket er sjekket i det hele tatt. Det beste ville være oncard løsningen, der fingeravtrykkstemplate'et sendes til kortet, og kortet sammenligner og verifiserer dette selv samt oppdaterer statusen for godkjent fingeravtrykk selv. Siden dette ikke er mulig i dette prosjektet valgte å bruke fingeravtrykket som et supplement til de 3 PIN kodene (DC, DS og ID). Vi følte at offcard løsningen alene ikke gir tilstrekklig sikkerhet for å beskytte de private RSA nøklene mot uautorisert bruk.

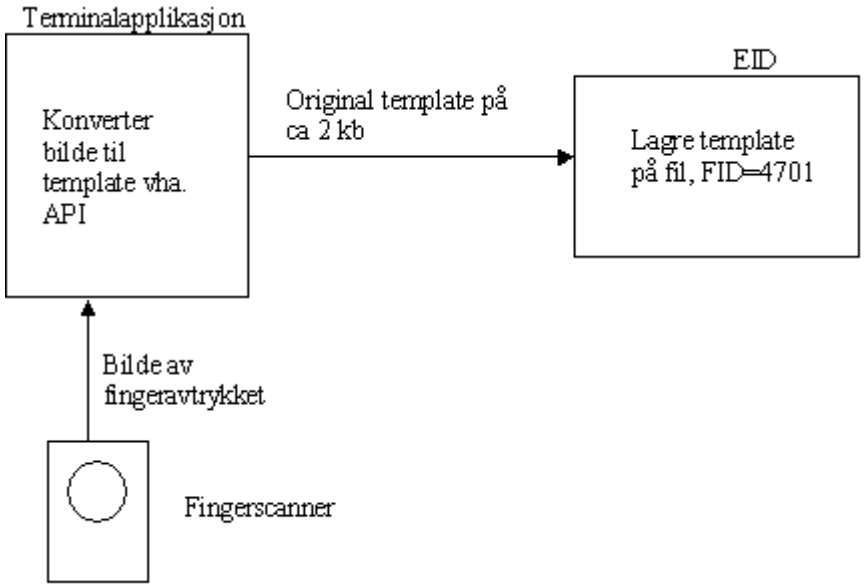
For detaljert informasjon angående biometri, henvises det til Biorapport **[Vedlegg B]**.

3.3.2.1 Generering av original template

Når EID installeres og gjøres klart til kortholder, må også kortholderens originale fingeravtrykk registreres. Det er dette som anses som kortholderens originale fingeravtrykk som det under hver identifisering sammenlignes mot. Denne registreingen må skje fra terminalapplikasjonen, der vi har laget en egen klasse, Biometric,(se **[Dokumentasjon\Dokumentasjon – biometric.pdf]** på cd rom for detaljer), som inneholder kommunikasjonen med fingerscanneren, samt generering av template. Selve koden for genereringen er skjult i Precise Biometrics' API'et. Template'en er lagret som et objekt i minnet. Dette har vi valgt å råskrive direkte som en byte array av to grunner:

1. Innholdet i File objektene i EID, lagrer data som byte arrays.
2. Det blir lettere å lese template'en fra kortet, direkte som objekt fra byte array ved sammenligningsprosessen.

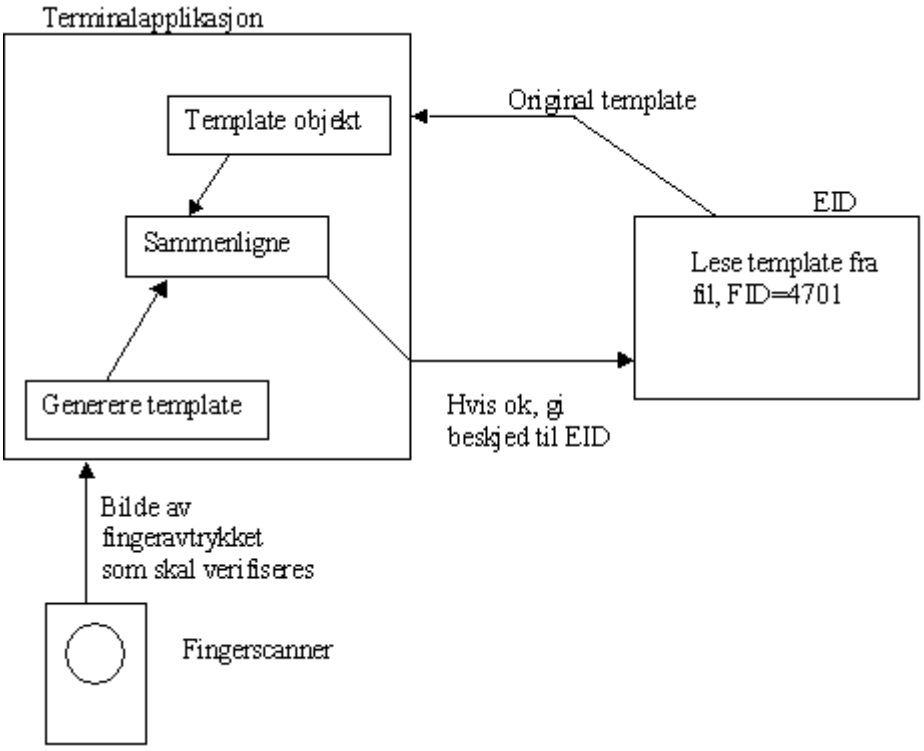
Template'en, som byte array, er på ca 2 kb og skrives til filen med FID=4701 på filsystemet i EID. Denne filen er ikke definert i EID spesifikasjonen, men vi har altså valgt å bruke denne som fil for original fingeravtrykktemplate. Kortholder må ha leserettigheter på denne filen, da den må hentes fra kortet under hver eneste identifikasjon. Det er kortutsteder som skriver denne filen til kortet, og derfor må IS PIN være validert. Det brukes standard filsystemkommandoer for dette, (Select File, Write Binary).
Prosessen i å generere template, og lagre dette på smartkortet, er illustrert i figur 3.5.



Figur 3.5: Registrering av kortholders fingeravtrykk og lagring av dette på kortet.

3.3.2.2 Verifisering av fingeravtrykk

Før kortholder kan benytte seg av EID applet'ens kryptografiske funksjonalitet, må bl.a. fingeravtrykket sjekkes og godkjennes. Siden vi måtte velge offcardløsningen, skjer selve sammenligningen i terminalapplikasjonen. Dette innebærer at den originale fingeravtrykktemplate'en på EID, FID=4701, må leses inn fra kortet og sammenlignes mot et nytt fingeravtrykk fra fingerscanneren. Selve lesingen skjer ved hjelp av EIDs filsystem kommandoer Select File og Read Binary. Siden godkjenning av fingeravtrykket ikke har noe med kortutsteder (og derav IS PIN), har 4701 filen lesetilgang fra kortholder, men dette krever også at ID PIN er validert før fingeravtrykket sjekkes. Etter innlesing av fingeravtrykk, sjekkes dette mot den originale template'en. Dersom dette blir godkjent, må EID fortelles at fingeravtrykket er godkjent. Dette gjøres ved hjelp av en APDU, (se tabell 3.2), og dette er ikke noen spesielt god løsning. Kortholder kan, dersom terminalapplikasjonene tillater det, sende denne APDU'en ved å bygge den opp selv, uten å tatt på fingerscanneren engang. EID, når den mottar denne APDU'en, tror at fingeravtrykket er godkjent. Likevel er dette eneste måten å i hele tatt få sagt ifra til EID at fingeravtrykket er godkjent. Dette må da betraktes som en "nødløsning" inntill en oncard løsning kan implementeres. Vi har av denne grunn vært svært nøye på at APDU'en fra tabell 3.2 ikke sendes før fingeravtrykket er verifisert og godkjent, slev om dette bare er en "falsk trygghet".
Prosessen som godkjenner kortholders fingeravtrykk, er illustrert i figur 3.6.



Figur 3.6: Prosessen for en verifisering av et fingeravtrykk.

3.3.2.3 Biometri i terminalapplikasjon

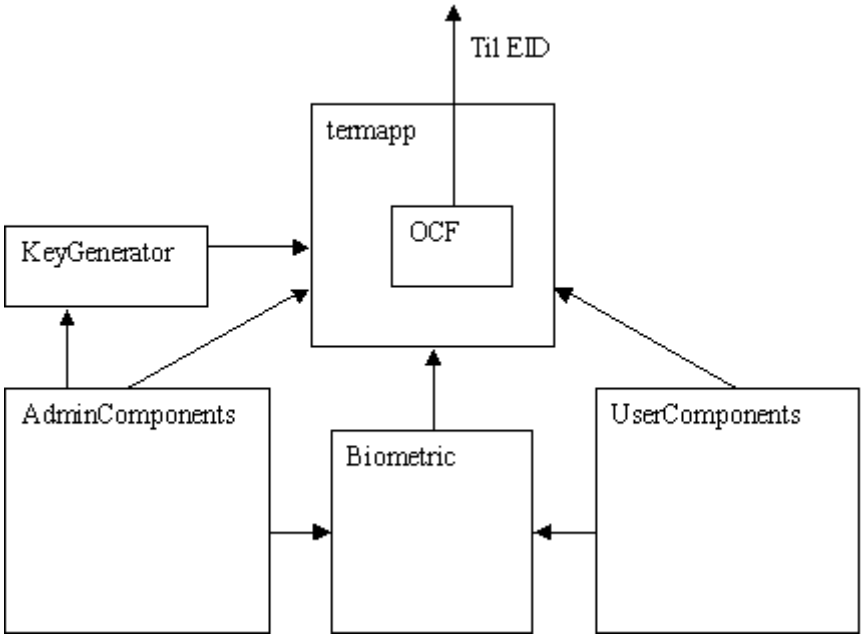
Siden vi har laget en offcard løsning, skjer all fingeravtrykksbehandlingen i selve terminalapplikasjonen. Av den grunn valgte vi å legge resten av arbeidet innen biometriinkrementet til inkrement 3,Demonstrasjonsapplikasjonen. Vi følte det passet bedre å jobbe med resten av inkrementet under utviklingen av terminalapplikasjonen. Her valgte vi å legge all funksjonalitet i forbindelse med behandlingen av fingeravtrykk i en egen modul eller klasse, Biometric. Denne tar seg av selve lesing av fingeravtrykk, generering av template'er, lesing og skriving av fil fra EID, samt sammenligning av template'er. Se dokumentasjonen på cd-rom for mer detaljerte opplysninger om denne klassen.
[\Dokumentasjon\Dokumentasjon – terminal – biometric.pdf].
Se ellers delkapittel 3.4.4.

3.4 Inkrement 3: Demonstrasjonsapplikasjon

3.4.1 Innledning

Dette er applikasjonen, også kalt terminalapplikasjonen, som skal kjøres på en PC. Hovedoppgaven til denne, er å kommunisere med EID og demonstrere funksjonaliteten til denne. Vi har valgt å lage denne så modulbasert som mulig av mer eller mindre selvstendige klasser. Dette kan være ønskelig dersom terminalapplikasjonen skal spesialtilpasses etter behov. Grunnet dårlig tid, har vi ikke lagt vekt på å perfeksjonere det grafiske brukergrensesnittet. I stedet har vi lagt mye vekt på å få vist frem funksjonaliteten som EID kan tilby.

Terminalapplikasjonen har vi bygd opp av moduler som kommuniserer med hverandre, som vist på figur 3.7. Hvordan hver av dem er designet, gåes nærmere inn på i den neste delkapitlene.



Figur 3.7: Oversikt over de viktige modulene i demonstrasjons-Applikasjonen, og hvordan de kommuniserer.

3.4.2 Modul: termapp

Denne modulen har hovedansvaret for koordinering av kommunikasjon mot de andre modulene, da den er hovedklassen som programmet kjører. Denne modulen styrer den viktigste delen av hele terminalapplikasjonen, OCF (OpenCard Framework). Det er denne delen som står for selve kommunikasjonen med smartkortet. Vi valgte å legge OCF i termapp, da alle andre moduler kan bruke denne for sending av sine APDU'er til smartkortet. Dette gjør at OCF koden kun trenger å implementeres en gang. Dette letter også arbeidet i å finne eventuelle feil i koden.

I tillegg administrerer modulen terminalapplikasjonenes tre offentlige RSA nøkler (DC, DS og ID), som brukes til å dekryptere og kryptere data til og fra EID applet'en. DS brukes for eksempel når en digital signatur skal verifiseres. Modulen inneholder metoder for initialisering av disse nøklene. For detaljerte beskrivelser av denne modulen, se Dokumentasjonen på cd-rom **[\Dokumentasjon\Dokumentasjon - terminal – termapp.pdf]**.

3.4.3 Modul: KeyGenerator

Dette er en støttemodul for AdminComponents og termapp modulene. Denne tar seg av generering av 1024 bits RSA nøkkelpar. Denne ble utviklet da det ble klart at vi ikke fikk verken RSA nøkler eller sertifikater fra ErgoSolutions. Denne modulen tar seg av generering av nøkkelpar, samt initialisering av offentlige RSA nøkler. I tillegg har den støtte for å skrive nøkkelpar til fil, kodet etter PKCS1, ASN1 syntaksen. For detaljerte beskrivelser av denne modulen, se Dokumentasjonen på cd-rom **[\Dokumentasjon\Dokumentasjon - terminal – KeyGenerator.pdf]**.

3.4.4 Modul: Biometric

Dette er en fortsettelse av Inkrement 2. Grunnet offcard løsning, fant vi ut at det var mest hensiktsmessig å legge behandlingen av fingeravtrykk i en modul i terminalapplikasjonen. Resultatet er denne modulen som tar seg av følgende:

- Lese fingeravtrykksbilde fra fingerscanner
- Sjekker at kvaliteten på bildet er tilfredstillende.
- Trekker ut egenskapene av bildet og genererer en template ut av det.
- Sammenligne to template'er om de er like. (Dvs godkjenne et fingeravtrykk)
- Konvertere template til byte array og sende det til fil 4701 på EID filsystemet.
- Lese inn en byte array fra fil 4701 på EID filsystemet og gjøre denne om til en template som kan sammenlignes mot en annen.
- Viser bildet av fingeravtrykket til bruker i et eget vindu. Se figur 3.8.

Figur 3.8: Biometric klassens bilde av et fingeravtrykk hentet fra fingerscanneren.

Siden all funksjonalitet i forbindelse med fingeravtrykksbehandling ligger i en modul, spares mye arbeid med å slippe å implementere dette i hver klasse som bruker biometri.
Selve koden for lesing av bilder fra fingerscanner, generering av template'er, og sammenligning av dem, er API funksjoner fra Precise Biometrics.

Pseudokode for hvordan et fingeravtrykk sjekkes mot originalen på EID applet:

```
SjekkFingeravtrykk()
   {
     originalfinger= hentTemplateFraKort();
     do
       {nyfinger=lagTemplateFraBilde(hentBildeFraFingerscanner());
       }
     while(nyfinger != OKkvalitet);
     if(sammenlignTemplater(originalfinger,nyfinger)==true)
       {sendAPDU(90 00);      //fingeravtrykk stemte, fortell EID
        return ok;
       }
     else
       return feil;
  }
```

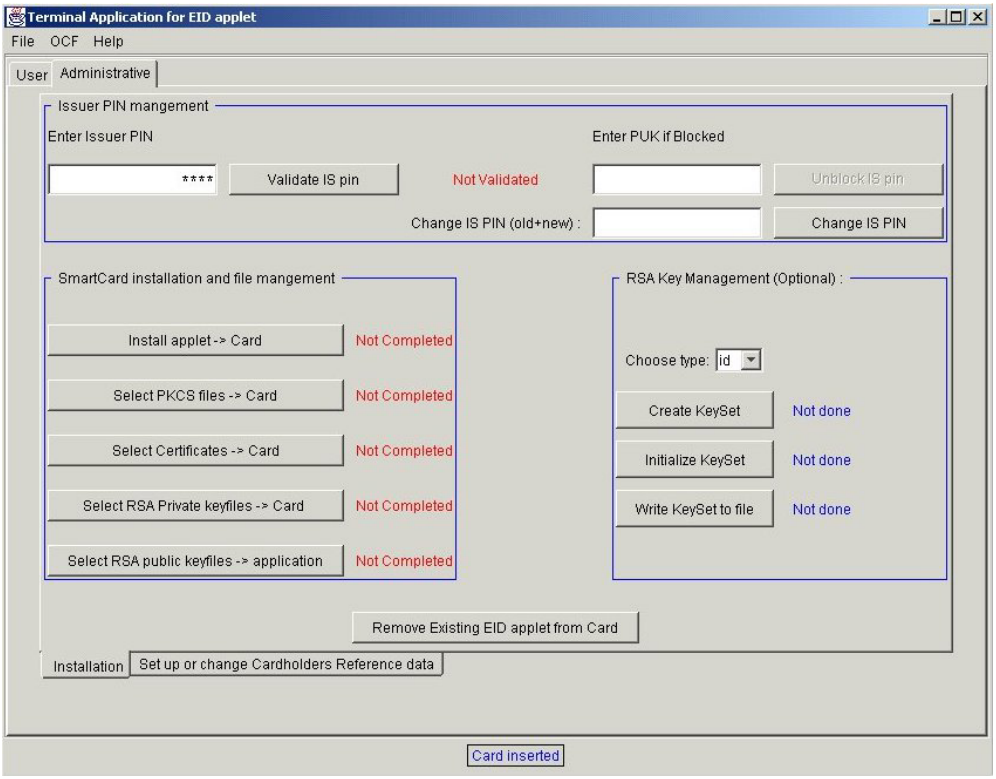
For detaljerte beskrivelser av denne modulen, se Dokumentasjonen på cd-rom **[\Dokumentasjon\Dokumentasjon - terminal – Biometric.pdf]**.

3.4.5 Modul: AdminComponents

Denne modulen administrerer alt angående det kortutsteder trenger for å klargjøre smartkortet med EID for kortholderen. Funksjonalitet støttet i denne modulen, som vi mener er det nødvendige for å få EID til å fungere:

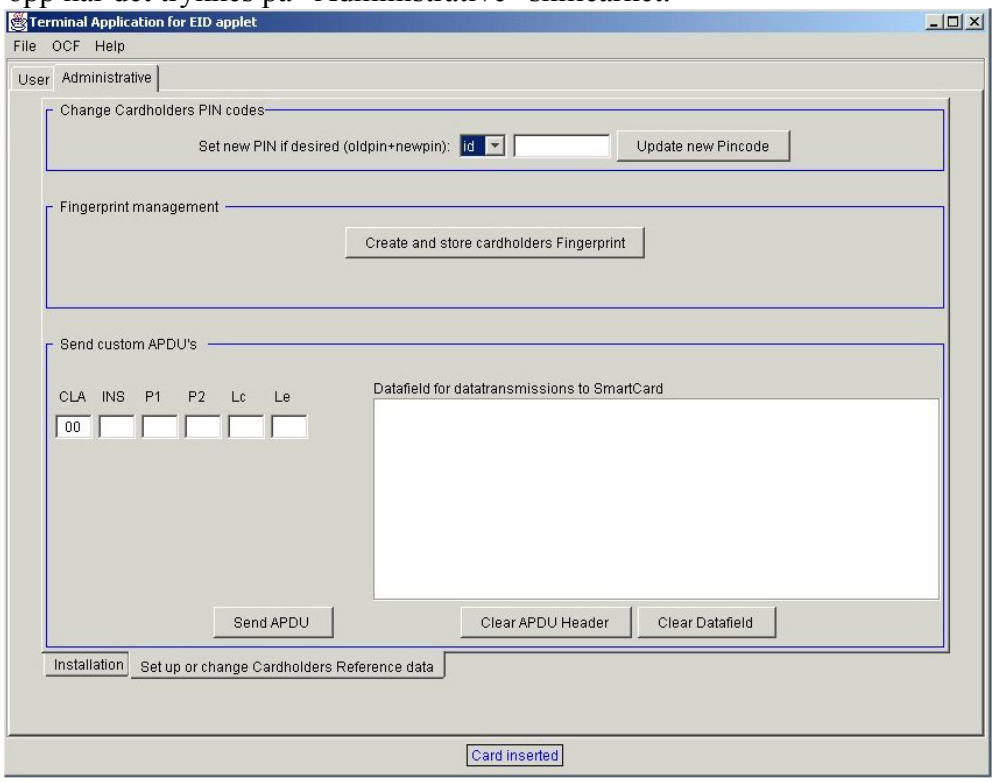
- Validering av IS PIN kode, og opplåsing av IS PIN.
- Installering av EID applet på kortet.
- Skrive filer til kort:
  - nøkkelinformasjon om privat RSA nøkler.
  - Sertifikater med offentlig RSA nøkkel.
  - PKCS#15 filer.
- Endre kortholders PIN koder.
- Lage original template av kortholders finger avtrykk.
- Slette EID fra smartkortet.
- Generere RSA nøkkelpar (tilleggsfunksjon).
- Sende egne APDU kommandoer til EID (tilleggsfunksjon).

Modulen har også sitt eget grafiske brukergrensesnitt, som er kodet inn i klassen. Dette har vi gjort for at klassen skal være selvstendig og modulær. Se figur 3.9 for skjermbilde av AdminComponents modulen.



Figur 3.9: Grensesnittet som møter kortutsteder.

Vi har valgt å passordbeskytte tilgangen til de administrative funksjonene for kortutsteder, siden kortholders funksjoner kun ligger et skilleark unna. Dette skjer ved at en passordboks kommer opp når det trykkes på ”Administrative” skillearket.



Figur 3.10: Andre del av grensesnittet som møter kortutsteder.

Grensesnittet for AdminComponents er delt i et skilleark. Første del vises i figur 3.9, mens andre del vises i figur 3.10.

For detaljerte beskrivelser av denne modulen, se Dokumentasjonen på cd-rom **[\Dokumentasjon\Dokumentasjon - terminal – AdminComponents.pdf]**.

3.4.6 Modul: UserComponents

Dette er den viktigste delen for å kunne vise at EID applet'en fungerer. Dette er den delen kortholder bruker for å lage en digital signatur. Som AdminComponents modulen, er også denne mest mulig selvstendig og har sine egne grafiske komponenter. Poenget er at en av disse to modulen kan fjernes etter valg, og resten av programmet vil fungere like greit. Det er bare å fjerne ønsket *.class* fil før programmet kjøres.

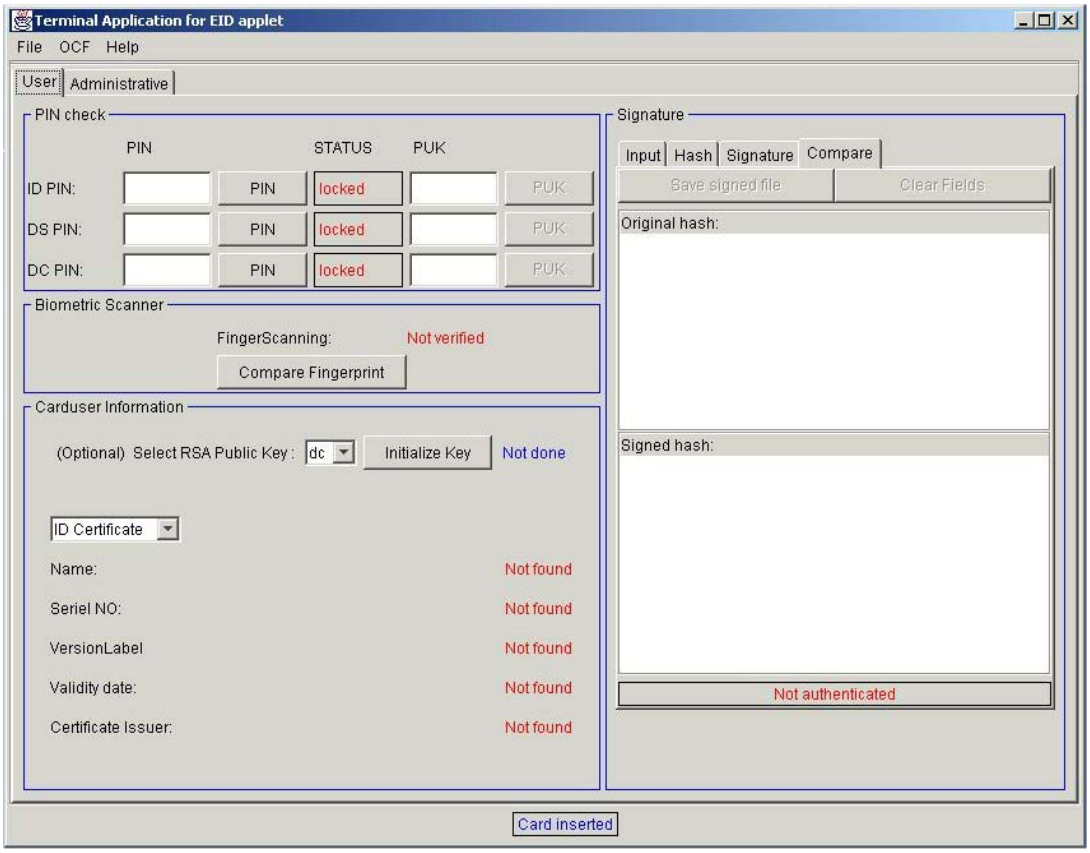
UserComponents modulen inneholder den funksjonaliteten som behøves for at kortholderen skal kunne lage en digital signaturen:

- Validering av PIN koder (ID, DC og DS), samt mulighet for å endre PIN kodene og låse opp blokkerte PIN koder med PUK koder.
- Validere fingeravtrykk mot originalen som ligger på kortet.
- Hente inn sertifikatet ved validert PIN, og initialisere den offentlige RSA nøkkelen i dette, med terminalapplikasjonens nøkler. I tillegg vises kortholder informasjon hentet fra tilsvarende sertifikat.
- Mulighet for å skrive en tekst, eller laste en inn fra fil.
- Lage SHA hash av denne teksten.
- Få EID til å signere denne hash'en.
- Se hvordan signaturen ser ut.
- Sammenligne dekryptert signert hash med originalen for å se om signaturen er korrekt.

Vi har valgt å lage UserComponents på denne måten, da dette best demonstrerer hvordan EID fungerer. Det er viktig å visuelt se at den dekrypterte signaturen er den samme som den originale hash'en av tekstdokumentet, selv om det er 20 tall som ikke gir noen mening annet enn at de skal være like. Vi har, både i AdminComponents og i denne modulen, lagt vekt på tolking av statuskodene returnert fra kortet. Alle feilmeldinger er prøvd å være så forklarende som mulig, i stedet for en 4 sifferet feilkode som. Den er umulig å skjønne for vanlige brukere dersom de ikke har detaljerte kunnskaper om hvordan EID fungerer. Se figur 3.11 for grensesnittet som kortholderen møter ved vanlig bruk.

Det er denne modulen som vi har valgt å legge koden for Challenge kryptering (også kalt encipher) og Challenge dekryptering (også kalt decipher). I tillegg har vi også lagt koden som omhandler den digitale signaturen. Dette er ganske naturlig da denne modulen er den eneste som har med kryptografisk kommunikasjon med EID.

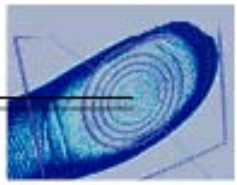
For å kunne ta i bruk RSA algoritmen i terminalapplikasjonen, måtte vi skaffe oss tilgang til den gjennom en tredjeparts leverandør (Security Provider). Dette koster ofte penger, men vi kom over en prøveversjon av en API pakke, JCSI 2.2 fra Wedgetail Communications, som virker i 30 dager. Vi valgte å implementere denne pakken, da den også er kompatibel med Sun's JCE krypteringspakke.



Figur 3.10: Grensesnittet kortholderen møter

For detaljerte beskrivelser av denne modulen, se Dokumentasjonen på cd-rom **[\Dokumentasjon\Dokumentasjon - terminal – UserComponents.pdf]**.

IMPLEMENTASJON



Java SmartSign 2002

4 Implementasjon

4.1 Innledning

Dette kapittelet omhandler hvordan vi har valgt å bygge løsningen av oppgavene kodemessig med tanke på utviklingsverktøy, dokumentasjon, kommentering o.l., samt eksempler på hvordan koden er realisert.
Etter systemutviklingsmodellen, faller det naturlig å dele inn delkapitelene etter hvert inkrement, da disse ble utviklet hver for seg.

4.2 Generelt for alle inkrement

4.2.1 Valg av utviklingsverktøy

All programmering i dette prosjektet er lagt til Java plattformen, etter krav fra arbeidsgiver. Derfor er Inprise's (Borland) JBuilder 4.0 et naturlig valg, som også arbeidsgiver har mye erfaring med. All nødvendig programvare og API har blitt installert på en PC, som ErgoSolutions har stilt til vår disposisjon.

4.2.2 Oppsett av kildekode

Vi har valgt å bruke engelsk som basespråk for variabelnavn samt kommentering av kode. Dette har vi gjort fordi engelsk er et språk som de fleste kan, og dersom det skal arbeides videre på kildekoden vår av andre enn nordmenn, er det lettere for dem å sette seg inn i koden.

Navngivning av variabler og funksjoner har vi prøvd å begrense slik at lengden helst ikke overstiger 12 tegn. De skal ha et mer eller mindre forklarende navn etter hvilken funksjon de har. Vi har prøvd så godt som mulig å følge "Java standarden" for bruk av små og store bokstaver. Det vil si start med liten bokstav, og bruke stor forbokstav for hvert nye del-ord. Navn på statiske variable benyttes med kun med store bokstaver. Klammeparenteser settes alltid på en ny linje med et tabulator innrykk (2-3 tegn). Avsluttende klammeparentes skal befinne seg på samme vertikale kolonne som startende. Parametere til funksjoner skilles med et mellomrom etter komma. Lange kodesetninger brytes helst opp. Vi har prøvd å etterleve dette så godt som mulig, men det kan nok forekomme små avvik noen steder. Det er tross alt rundt 8000 linjer kode å holde styr på, og sannsynligheten for at noen tegn kan være litt feil plassert, er temmelig stor.

4.2.3 Kommentering og dokumentering av kildekode

Vi har vært nøye med å kommentere koden godt. Spesielt på de mer kompliserte kodedelene, som for eksempel kryptografifunksjoner (spesielt!), kortkommunikasjon, filsystembehandling, parsing- og paddingfunksjoner og bit-operasjoner. "Triviell" GUI kode og vanlige, enkle Java kodesnutter er ikke fult så detaljert kommentert.
Som nevnt tidligere, baserer også kommenteringen seg på engelsk språk. Dette skjer konsistent.

Vi har valgt å skrive egen dokumentasjon av kildekoden. Det er mulig å bruke programmer til dette ut fra kildekoden, eksempelvis JavaDoc, men vi synes det ikke gir en grundig nok forklaring på hva som skjer i de ulike metodene. Denne dokumentasjonen skrev vi på norsk, siden den er ment for å være et innføringsdokument for ErgoSolutions for senere videreutvikling av koden. Hver klasse har sitt egen dokument som inneholder forklaring til hva klassen gjør, og hvilken funksjoner den inneholder, samt en forklaring til disse. Dokumentasjonen er for omfattende til å ta med i denne rapporten, men den ligger vedlagt på cd-rom under **[\Dokumentasjon].**

4.3 Inkrement 1: EID

4.3.1 Valg av utviklingsverktøy

Utviklingsverktøyet til EID applet'en ble valgt av arbeidsgiver. Siden EID skal ligge på smartkortet med begrenset kapasitet, må spesialtilpasset utviklingsverktøy brukes. JBuilder 4.0 har ikke støtte for å lage Javacard applets, noe som EID er. Derfor har leverandøren av smartkortet, Gemplus, utviklet en tilleggspakke (plug-in) som integreres i JBuilder 4, GemXpresso RAD III. Denne ble anskaffet av arbeidsgiver, og alt ble installert på PC'en som vi har fått låne, sammen med smartkortet og fingerscanner/kortleseren. GemXpresso pakken har gjort at vi kan bruke JBuilder som kodingsplattform, selv for Javacard applets.
Vi brukte ingen nyere versjon av JBuilder grunnet at nåværende versjon GemXpresso pakka kun fungerer med JBuilder 3.5-4.0 og JDK versjon 1.2.2.

Under kompilering av Javacard applets, klarer ikke JBuilder automatisk å lage kompilert kode som Javacard JRE'et på smartkortet skjønner. Derfor brukes GemXpresso plug-in til å konvertere JBuilders .class filer til en .jar fil inneholdende .cap filer. Disse er konvertert kode som Javacard forstår. .jar filen brukes under lasting av EID applet'en ned på kortet.

GemXpresso pakken inneholder også en smartkortsimulator samt muligheter for administrasjon av Javacard applets på kortet (JCardManager).

4.3.2 Bruk av API

Programmering av Javacard applets krever et eget API, grunnet at smartkort har begrensede ressurser og en spesialtilpasset versjon av Java er derfor påkrevd. Sun har utviklet API'et vi bruker, Javacard 2.11. Dette er støttet av JRE på selve smartkortet. Gemplus' pakke inneholder også dette API'et sammen med kryptografiske algoritmer. I tillegg har vi valgt å bruke VOP (Visa Open Platform), et tilleggs-API for sikker kommunikasjon og administrasjon av applet. Dette er også levert sammen med Gemplus pakken. For hvilke API pakker som importeres, se utdrag av kode 1.

```
// specific import for Javacard API access
import javacard.framework.*;
import javacard.security.*;
import javacardx.crypto.*;
// specific import for OpenPlatform API access
import visa.openplatform.*;
```

Utdrag av kode 1: importerte pakker fra Javacard og VOP API'ene.

4.3.3 Kodeprinsipper

Siden smartkort har begrensede ressurser med tanke på lagringskapasitet og regnekraft, måtte vi legge oss på en linje som krever:

- En mest mulig effektiv, kompakt og rask kode.
- Minimalisere bruk av variable, grunnet å lage applet'en så liten som mulig.
- Siden Javacard JRE på kortet ikke har en Garbage Collector, må gjenbruk av objekter og variabler benyttes. Ellers vil kortets lagringskapasitet sakte, men sikkert bli oppspist av ferdigbrukte varaibler.
- Midlertidige variabler begrenses maksimalt, spesielt arrays.
- Exception behandling for eventuelle feilsituasjoner som oppstår under run-time.

4.3.4 Eksempel på kildekode

```
/**
   The method creates a digital signature. The terminal applicataion
   creates a hash from a document and sends it to the card (apdu),
   which signs the hash using the prk_ds key and the signatureobject.
   The signed hash is returned (apdu) to the terminal.
*/
  private void pso_create_dig_sign(APDU apdu) throws ISOException
    {
     byte[] buffer=apdu.getBuffer();
     short nrbytestosign;
     //checks if id pin, fingerprint and ds pin have been validated
    //(ds= digital signature)
     if(!pin_id.isValidated() || !validBIO || !pin_ds.isValidated())
       ISOException.throwIt(ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED);
     //ready to recieve
     apdu.setIncomingAndReceive();
     // Decrypt APDU buffer
```

```
//unwrapAPDU(apdu);

//checks if key has been initialized
if(!prk_ds.isInitialized())
  //try to initialize key
  if(initialize_RSAkey(prk2,prk_ds)!=(short)0x9000)
    ISOException.throwIt(ISO7816.SW_COMMAND_NOT_ALLOWED);

//gets datalength to copy and sign
nrbytestosign=(short)(buffer[ISO7816.OFFSET_LC] & 0xFF);
//copies data to be signed to avoid overvriting during signing process
//note:key has been reused from initializeRSAkey()method to save
       space
Util.arrayCopyNonAtomic(buffer,ISO7816.OFFSET_CDATA,key,(short)0,
                                                    nrbytestosign);
try
 {
  //initializes signature object with right key (ds) to sign
  sign.init(prk_ds,Signature.MODE_SIGN);
  //sign the data (hash) and return to apdu buffer(offset 0)
   sign.sign(key,(short)0,nrbytestosign,buffer,(short)0);
  }
catch(CryptoException ce)
  {//catches any error if objects involved have been used or
   // iniitialized wrong
     ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
  }
//dig.sign. pin always should be reset after 1 time use
pin_ds.reset();
//resets array used for storing data to sign
Util.arrayFillNonAtomic(key,(short)0,(short)key.length,(byte)0x00);
// send the signed data (hash) back to terminal
apdu.setOutgoing();
apdu.setOutgoingLength((short)128);
apdu.sendBytes((short)0,(short)128);
} //end pso_create_dig_sign
```

Utdrag fra kode 2: Metoden som genererer den digitale signaturen i EID applet.

4.4 Inkrement 2: Biometri

4.4.1 Valg av utviklingsverktøy

Siden vi ikke kunne lage ”oncard” løsningen, ble eneste programmering i forhold til biometri på EID applet’en, støtte for templatefilene 4701 (og fremtidig 4702), samt variabeloppdatering. Dette ble implementert under EID inkrementet.
Resten av biometrikoden flytett vi over som en modul i terminalapplikasjonen. Utviklingen av denne modulen beskrives her.

Precise Biometrics, leverandør av fingerscannerteknologien, har laget at API for programmeringsspråket C, og et for Java. Siden all programmeringen vår baserer seg på Java, var det naturlig å velge Java API’et. Dette kan importeres direkte inn i JBuilder 4.0 og vi har derfor også brukt den utviklingsplattformen her.

4.4.2 Bruk av API

All behandling av fingeravtrykk er definert i Precise Biometrics Java API. Dette tilbyr de funksjoner vi trenger for å lage og sammenligne template’er ut fra fingeravtrykk. Dette API’et fungerer kun på PC, og kan dessverre ikke implementeres på smartkortet.
Se Utdrag fra kode 3 for importerte API pakker.

```
// these packages are from precise biometrics
import com.precisebiometrics.fingerprint.event.*;
import com.precisebiometrics.fingerprint.*;
```

Utdrag av kode 3: Importering av API for fingeravtrykksbehandling i Biometric klassen.

4.4.3 Kodeprinsipper

Siden denne klassen kjøres på PC’ trenger ikke koden være så trimmet som på EID. Viktige forutsetninger i denne klassen, er å vise brukeren bilde av fingeravtrykket. Vi har også lagt vekt på at fingeravtrykket skal ha en tilfredsstillende kvalitet, før det gjøres om til template. Et krav til klassen, er at den behandler oppbygning og klargjøring av APDU for lesing og skriving av template’er til smartkortet.
Ellers følges standard kodeoppsett fra 4.2 og tilsvarende krav til kommentering og dokumentering.

4.4.4 Eksempel på kildekode

```
/**
 * This function gets a fingerprint from the reader.
 * If a finger is not present it waits with the reading.
 * And when a image is captured the drawing on the frame is updated.
 */
  public void getFingerPrintFromReader() throws Exception
    {
     SetStatusBar("Place finger on reader");
     //If no finger is present then sleep 80 milliseconds and check again
     do{
       System.out.println("Place finger on reader");
       Thread.sleep(80);
       }while(!fingerPrint.hasFingerPresent());
     try
       {
       //when a finger is present then capture it and store it in fpReader
       fingerPrint.captureOneImage(fpReader);
       //repaint the image to frame
       if(fpReader.getDisplayableImage() != null)
         {this.getGraphics().drawImage(fpReader.getDisplayableImage(),0,0,
                        fpReader.getDisplayableImageSize().width,
                        fpReader.getDisplayableImageSize().height,this);
         }
      }
    catch(Exception e)
      {
      throw new Exception("Could not capture image"+e.getMessage());
      }

    SetStatusBar("Image captured");
    System.out.println("Image captured");
    }//end getFingerPrintFromReader()
```

Utdrag fra Biometric class: getFingerPrintFromReader()

Utdrag fra Biometric class viser hele funksjonen for å hente et fingeravtrykk fra leseren. For hele kildenkoden for Biometric klassen se **[\Dokumentasjon\Dokumentasjon - terminal – Biometric.pdf]**.

4.5 Inkrement 3: Demonstrasjonsapplikasjon

4.5.1 Valg av utviklingsverktøy

Demonstrasjonsapplikasjonen utvikles på Java plattformen, og det er også her det mest fornuftige å bruke JBuilder 4.0,som vi også etter hvert har blitt ganske så vant med. Den har en innebygd design komponent for å lage GUI komponenter, som vi håpte skulle spare oss for en del arbeid. Ellers har vi hatt gode erfaringer med JBuilder 4.0 programvaren så langt.

4.5.2 Bruk av API

Demonstrasjonsapplikasjonen skal kommunisere med smartkortet. Til dette finnes det ingen støtte for i standard JDK. De største leverandører av smartkort har i samarbeid utviklet et API for standardisering av koden som kommuniserer med kortet. Dette API'et kalles OCF (OpenCard Framework), og er ment for å lette programmeringen for smartkort kommunikasjon. Gemplus har lagt med dette API'et i GemXpresso Rad III pakken, noe tilpasset for kortet og kortleseren vår. Det er blant annet implementert VOP (Visa Open Platform) API støtte for sikker kommunikasjon mellom kortet og terminalapplikasjonen. Denne API pakken importeres direkte i JBuilder prosjektet. Se Utdrag av kode 5.

```
// standard OCF imports
import opencard.core.service.*;
import opencard.core.terminal.*;
import opencard.opt.applet.AppletID;
import opencard.core.util.*;
// gemplus VISA imports
import com.gemplus.opencard.service.op.*;
import com.gemplus.opencard.service.op.vop.*;
import com.gemplus.opencard.service.op.vop.vop211.*;
import com.gemplus.opencard.service.op.vop.vop200.*;
// gemxpresso spesific imports
import com.gemplus.tools.gemxpresso.*;
import com.gemplus.tools.gemxpresso.util.GxpSystem;
```

Utdrag av kode 5: Importering av OCF og VOP for smartkortkommunikasjon.

Ellers har vi valgt å bruke Javas Swing pakke for oppbygging av det grafiske brukergrensesnittet. Av de på gruppen som har vært borti Java før, har hatt en liten innføring i denne pakken. Derfor falt naturlig å basere GUI på disse komponentene.

For de kryptografiske funksjonene på terminalapplikasjonen måtte vi skaffe en 3. parts leverandør av kryptografi API'et. RSA algoritmen er ikke implementert i Java JDK, og derfor valgte vi, etter noe leting på Internett, API'et JCSI 2.2 fra leverandøren (også kalt Security Provider) Wedgetail Communications (http://www.wedgetail.com) . Dette valgte vi av følgende grunner:

- Det er kompatibelt med Sun's datatyper angående kryptografi.
- Det er kompatibelt med Sun JCE.
- Det er gratis å bruke i 30 dager, nok for vår del. Mange av de andre leverandørene krevde penger.
- Det dekker akkurat våre behov: Støtte for RSA algoritmer både for kryptering/dekryptering, og signatur behandling. I tillegg støtter den PKCS1 padding og SHA/SHA1 hashing algoritme.
- Det er relativt enkelt å implementere i Java JDK 1.2.2, som vi bruker.

4.5.3 Kodeprinsipper

Som vanlig, har vi også her fulgt kravene for kommentering og dokumentering. Vi har også prøvd å bygge demonstrasjonsapplikasjonen så modulær som mulig. Det vil si å bygge funksjonene som logisk hører sammen i en egen klasse. Da kan API pakkene (package) importeres i klassen etter hvilke funksjonalitet den har. Dette gjør senere endring lettere, samt det bli mye lettere å luke ut feil. Koden blir også mer oversiktlig og lettere å holde orden på.

4.5.4 Eksempel på kildekode

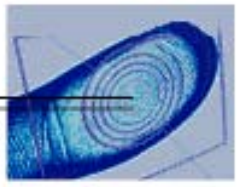
```
//parse the decrypted sign's signature , tlv encoded
     try
       {int totalLen=0;
        int len=0;
        int offset=0;
        //there must be data to parse
        if(returnedBytes==null)
           throw new Exception("No data to parse.");
        //first byte should be 0x30 (sequence)
        if((returnedBytes[offset]&0xFF)!=0x30)
          throw new Exception("No sequence byte 0x30 ("
                              +Integer.toString((int)returnedBytes[offset]))
                         ;
        offset++;
        totalLen=(returnedBytes[offset]&0xFF);
        //get total length of response
        if(totalLen>128)
         throw new Exception("Unlogical long sequence,probably bad
                              response.");
        offset++;
        //next sequence should be object identifier
        if((returnedBytes[offset]&0xFF)!=0x30)
```

```
 throw new Exception("No sequence byte 0x30 ("
             +Integer.toString((int)returnedBytes[offset])+
                ")\nShould be object identifier (Sha1).");
   offset++;
   //get length of objectidentifier
   len=(returnedBytes[offset]&0xFF);
   offset++;
   //do not read this(possibly check of sha1 could be implemented
      here)
   offset+=len;
   //get the original hash from response
   if((returnedBytes[offset]&0xFF)!=0x04)
     throw new Exception("No digest byte 0x04 ("
                  +Integer.toString((int)returnedBytes[offset])+
                             ")\nShould be original hash here.");
   offset++;
   len=(returnedBytes[offset]&0xFF);
   if(len!=0x14)
     throw new Exception("Invalif hash lenth ("
        +Integer.toString((int)returnedBytes[offset])+
                          ") Should be 20 bytes long.");
   offset++;
   //get the hash from response
   System.arraycopy(returnedBytes,offset,originalHash,0,len);
   }
catch(Exception e)
   {//if parsing errors occour
    e.printStackTrace();
    throw new Exception("Parse Error! Unable to get hash from"+
                        "cardresponse.\nReasom: "+e.getMessage()) ;
   }
```

Utdrag fra kode 6: Fjerning av padding etter terminalapplikasjonen har dekryptert signatur fra kortet med den offentlige RSA nøkkelen DS.

TESTING

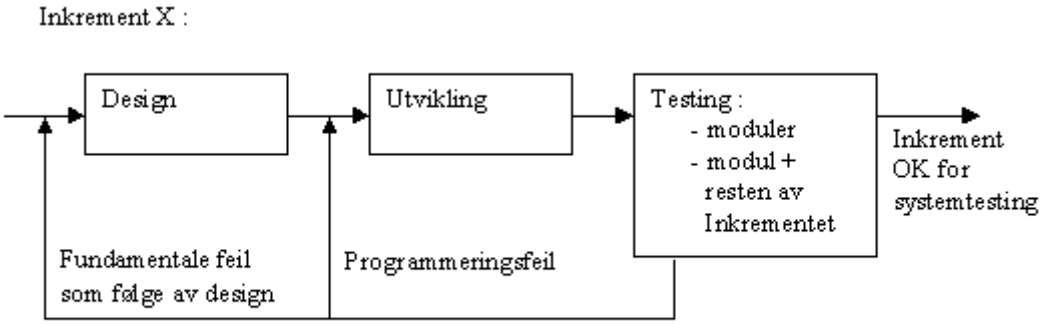
Java SmartSign 2002



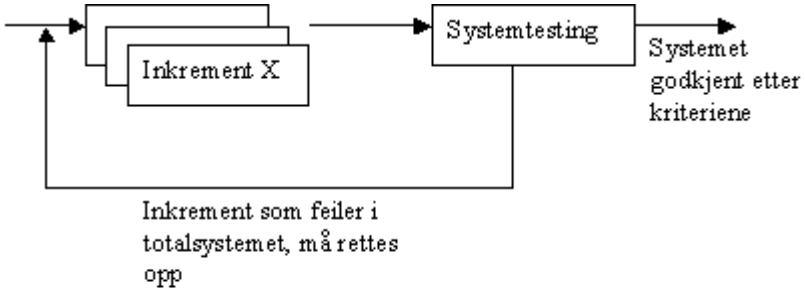
5 Testing

5.1 Innledning

Dette kapittelet omhandler hvordan vi har lagt opp testing og kvalitetssikring av systemet for å få et så godt resultat som mulig. I tillegg presenterer vi de viktigste testresultatene og hvilke konsekvenser disse fikk for eventuelle endringer på det totale systemet. Siden vi har valgt inkrementell systemutviklingsmodell for dette prosjektet, var det naturlig for oss å teste hvert inkrement før det ble innlemmet i det totale systemet. Deretter testet vi inkrementet sammen med resten av det ferdige systemet. Se illustrasjon av dette i figur 5.1 og figur 5.2. Resultatet av tester, har ført til at noe har blitt endret på, både av større og mindre betydningsfull grad. Testingen av modul/inkrement skjer først og fremst av programmerer selv, men under testing av hele systemet, skal alle involverte innen programmeringen være med på avdekke problemer, for å få inn flere synsvinkler på applikasjonen. Tilslutt skal en utenforstående være med på testingen, for å finne ut hva mennesker som ikke har kjennskaper til systemet synes.



Figur 5.1: Oversikt over testfasen for et inkrement.



Figur 5.2: Prinsippet for hvordan hele systemet testes med alle inkrementene.

Vi har valgt å baserer testingen etter ”black-box” og ”white-box” prinsippene. Alle API funksjoner skjuler hvordan data endres gjennom behandling. Derfor blir ”black-box” testing utført på disse. Det er kun mulighet til å sjekke om output fra funksjonen stemmer logisk overens med input. Dette kan da avsløre eventuelle feil på input, eller feil bruk av selve funksjonen. Programmereren må derimot stole på at selve funksjonen behandler dataene på riktig måte uansett. Samme input flere ganger bør også gi den samme outputen.
Mye av den egenlagde koden bruker vi ”white-box” testing på. Her kan det følges med på hva som skjer under behandlingen av data, instruksjon for instruksjon. Dette passer bra for blant annet parser-funskjoner og padding-behandling, der testdata kan bl.a. skrives til skjerm under funksjonens løp.

Selve testingen av moduler og funksjoner , valgte vi tidlig å teste etter som de ble utviklet og lagt til inkrementet. Dette valget gjorde vi bevisst for å enklere kunne finne, og fjerne feil. Vi visste fra tidligere erfaringer med programmering, at å kode ferdig alt før testing, gjør alt vanskeligere med å forstå hva som er feil.

5.2 Testing av inkrement 1: EID

5.2.1 Testmetoder

Gemplus, leverandøren av smartkortet, har i sin GemXpresso Rad III pakke laget en kortsimulator som simulerer hvordan applet’en vil oppføre seg på GXP211 smartkortet. Denne har vi basert testingen av EID applet’en på. Den har også har muligheter for å vise innhold av variabler, samt at den viser hvilke exceptions som blir kastet fra applet’ene som kjøres på kortet. I tillegg viser den APDU trace, dvs den viser innhold av APDU’er som sendes til og fra kortet. EID applet’en kan ikke testes fult ut alene, da den er ment for å kommunisere med demonstrasjonsapplikasjonen. Alle kryptografiske funksjoner blir derfor ikke mulig å teste skikkelig, før terminalapplikasjonen fungerer.

Vi valgte å teste EID i moduler, ettersom de ble utviklet. Hver ny modul, eller funksjon, ble testet sammen med de ferdig utviklede delene av EID. Denne metoden gjorde det lettere å finne feil, samt en fikk mye bedre oversikt over hva som faktisk var gjort ferdig, hvilken deler som ikke virket skikkelig samt hvilke deler som stod igjen.

EID valgte vi å teste på denne måten:

- EID med filsystem
- EID med filsystem og PIN koder
- EID med filsystem, PIN koder og kryptografiske funksjoner

5.2.2 Testkriterier

Følgende krav satte vi under testing for at inkrementet kunne sies å være ferdig og klar for testing sammen med resten av systemet:

- Filsystemet skal fungere i henhold til EID spesifikasjonen.
- PIN kodene skal kunne sjekkes, endres, blokkeres og låses opp. Opplåsing skjer med PUK kodene.
- EID skal kunne bruke RSA nøklene sine til å:
    - Gjennomføre en kryptering (encipher).
    - Gjennomføre en dekryptering (decipher).
    - Lage en digital signatur.
- PIN koder og fingeravtrykk skal begrense tilgangen til kortets funksjonaliteter som skal beskyttes mot bruk av uvedkommende.
- Alle APDU'er mottatt på kortet, skal sjekkes grundig mot eventuelle feil.
- Alle exceptions kortet kaster skal behandles, samt eventuelle feilsituasjoner.

5.2.3 Testresultater

Her presenteres de endelige resultatene etter testingen. Testingen har foregått fra Gemplus' JCardManager, et program for administrering av Javacard applets på kortet, men her har vi jobbet mot simulatoren i stedet. JCardManager brukes for å bygge opp APDU'er som sendes til kortet, dvs. simulatoren.
Det er verdt å merke seg at under testingen ble mange mindre feil oppdaget, men disse ble rettet fort opp og nevnes derfor ikke her i testresultatene.

EID med filsystem:

- Alle filsystemfunksjonene fungerer. (Select File, Read Binary, Write Binary, Delete File, Create File).
- APDU'ene brukt til filsystemkommandoene tolkes riktig og feilsituasjoner behandles som de skal.
- Offset for lesing og skriving av filer fungerer, slik at data ikke blir feil.

EID med filsystem og PIN koder:

- PIN kodene kan verifiseres riktig.
- PIN kodene kan endres.
- PIN koden blokkeres riktig etter 3 mislykkede forsøk.
- PIN koden kan låses opp med gyldig PUK kode.
- PUK koden blir blokkert etter 3 ugyldige forsøk.
- APDU'er sjekkes, og feil detekteres.
- Filtilgang fungerer i henhold til rettigheter definert i EID
- spesifikasjonen, slik at riktig PIN kode må settes før filtilgang tillates.

EID med filsystem, PIN koder og kryptografiske funksjoner:

- tilgang til de kryptografiske funksjonene (encipher, decipher og digital signatur), nektes dersom ikke de riktige PIN kodene er satt.
- Feil i APDU detekteres og returnerer feilmelding i henhold til EID spesifikasjonen.
- RSA nøklene blir riktig initialisert ut fra parsing av filene 4301-4303 på kortet før nøklene brukes for første gang.
- Encipher returnerer, etter en del trøbbel, kryptert data. Dekryptering av disse må testes i demonstrasjonsapplikasjonen.
- Decipher returnerer data, men går ikke å sjekke at dataene er riktige, da testdataene sendt til kortet ikke er krytpert med offentlig RSA nøkkel. Dette må vente til terminalapplikasjonen er klar.
- Digital signatur blir returnert. Datalengde stemmer, men sjekking at det går an å verifisere denne signaturen, må vente til testing i terminalapplikasjonen.

5.2.4 Problemer som testingen avdekket

Problemet:

Encipher funksjonen, som krypterer data sendt til kortet for sjekking av nøkkelpar, slet vi lenge med før vi klarte å finne ut hva som var galt. Denne funksjonen kastet exception med grunn at koden for initialisering av et objekt brukt til kryptering/dekryptering var brukt på feil måte: `cipher.init(prk_id,Cipher.MODE_ENCRYPT);`

Hva er gjort:

Vi sjekket at de private nøkkelfilene er kodet etter PKCS#1, ASN.1 syntaksen, og at funksjonen som henter ut nøkkeldataene fra denne fungerer som den skal. Vi har sett nøye på koden og dokumentasjonen for å avsløre eventuell feil bruk. Dessuten har vi brukt Internett til å lete i diskusjonsforumer og på andre sider som omhandler kryptografi og Javacard. Dessverre uten hell.

Løsningen:

Etter hjelp fra arbeidsgiver, skulle det vise seg at det var `Cipher.MODE_ENCRYPT` parameteren som var gal. All logikk tilsa at siden funksjonen skulle kryptere, burde parameteren stemme (encryption). Men det viste seg at API funksjonen forventet en offentlig RSA nøkkel ved `MODE_ENCRYPT`, ikke en privat som i dette tilfellet. Derfor måtte `Cipher.MODE_DECRYPT` brukes som parameter, ettersom `init` forventer en privat RSA nøkkel ved denne modusen. Siden vil lager PKCS1 paddingen selv, og objektet som krypterer/dekrypterer (Cipher) er initialisert med RSA algoritme uten padding, har modusen egentlig ingenting å si. Ved kall til funksjonen vil en RSA transformasjon skje likevel. Inntil videre så dette ut til å løse problemet.

5.3 Testing av Inkrement 2: Biometri

5.3.1 Testmetoder

Siden det ble sammenligning av fingeravtrykk i terminalapplikasjonen i stedet for på smartkortet, måtte vi legge koden for behandling av fingeravtrykk i en egen klasse i demonstrasjonsapplikasjonen. Denne klassen kommuniserer med fingerscanneren på kortleseren. Derfor testes denne først for seg for å luke ut feil i forbindelse med fingeravtrykksbehandlingen.

Biometridelen som skal implementeres på EID, innebærer støtte for en ekstra APDU som forteller at fingeravtrykket er godkjent. Testing av denne, valgte vi å vente med til hele systemet skulle testes, grunnet at denne avhenger av at EID, demonstrasjonsapplikasjonen fungerer og biometrimodulen i demonstrasjonsapplikasjonen.

5.3.2 Testkriterier

Det ble sattopp følgende kriterier for at biometri modulen i terminalapplikasjonene skulle være tilfredsstilt før den kunne testes sammen med hele systemet:

- Lese et fingeravtrykk fra fingerscanneren.
- Kunne vise bilde på skjermen av fingeravtrykket.
- Sjekke at fingeravtrykket har tilfredstillende kvalitet.
- Generere en template fra fingeravtrykket.
- Kunne sammenligne 2 template'er og finne ut om de stemmer overens.
- Konvertere en template til en byte array, og omvendt for tilpasning til EID filsystemet.
- Bygge opp APDU'er for lesing og skriving av template'er til EID applet.

5.3.3 Testresultater

Testingen ble foretatt fra JBuilder 4.0, med eget vindu for visning av fingeravtrykk. Ellers er systemvinduet mye brukt til byte array output. Alle mindre småfeil rettet opp, er for mange og ubetydelige til å nevnes her. Likevel tar også slike feil kan ta lang tid.
Resultatet av testingen:

- Fingeravtrykk vises i eget vindu.
- Kvaliteten på fingeravtrykket er tilfredstillende (OK) i følge API konstantene til Precise Biometrics. Flere mulige verdier, men perfekt kvalitet som krav, gjorde det svært vanskelig å få et godkjent fingeravtrykk. Derfor måtte kvalitetskravet legges på OK.
- Template'er genereres korrekt i følge API'et, og sammenligning er mulig.
- Behandling av APDU'er som leser eller skriver til 4701 filen på kortet ser ut til å være riktig. Det må imidlertid klargjøres for flere lese/skrive operasjoner, da template'en som byte array er på nesten 2 kb.

5.4 Testing av inkrement 3: Demonstrasjonsapplikasjonen

5.4.1 Testmetoder

Testingen av terminalapplikasjonen vil stort sett innebære å få modulene til å virke sammen, samt prøve å få kontakt med kortet via OCF laget. Det meste av funksjonaliteten til terminalapplikasjonen avhenger å ha kommunikasjon med EID applet, dessuten bruk av biometri modulen. Derfor blir testingen av selve applikasjonen noe begrenset, og resten må testes i systemet som helhet.

5.4.2 Testkriterier

For at den delen av demonstrasjonsapplikasjonen som kan testes for seg selv, skal kunne klassifiseres som klar for implementering i det totale systemet, har vi satt følgende krav:

- OCF laget må fungere, dvs. terminalapplikasjonen må kunne få kontakt med smartkortet.
- GUI komponentene til administrerende funksjoner for kortutsteder (AdminComponents), og for signeringsfunksjonen for kortholder (UserComponents) skal fungere skikkelig.
- All oppbygning av APDU'er i tilsvarende komponenter skal være riktige.
- Feilmeldinger skal gis når brukeren gir data til systemet som lager APDU med feil innhold i forhold til det EID krever.

5.4.3 Testresultater

Testen ble foretatt med JBuilder 4.0, og all sjekk av at APDU data er riktig vises på skjerm ved hjelp av dialogbokser, som for øvrig fjernes etter testing er fullført.

Resultatet av testingen:

- OCF ville ikke fungere, det vil si, vi fikk kontakt med kortet, men under opprettelse av sikker kommunikasjonskanal mellom terminalapplikasjon og smartkortet, fikk vi exception om en ikke støttet algoritme (les mer i 5.4.4).
- GUI komponentene fungerte tilfredsstillende. Alle knapper og input felt fungerte som planlagt. Styring av skilleark og innlasting av GUI komponenter fra AdminComponents og UserComponents fungerte også som antatt.
- Alle APDU som bygges opp, stemmer i henhold til EID spesifikasjonen. Feilsituasjoner behandles med feil APDU data og/eller header returnerer forklarende feilmeldinger. Tolking av feilmeldinger både av gal input fra bruker, og respons fra EID, er noe vi har vært nøye med å behandle på en forklarende måte.

5.4.4 Problemer som testingen avdekket

Problemet:

Det viste seg å være umulig å få opprettet en sikker kommunikasjonskanal mellom terminalapplikasjonen og smartkortet. Den sikre kommunikasjonskanalen er en del av VOP (Visa Open Platform) API koden, og brukes som et ekstra lag for sikkerhet i forbindelse med lasting av applet'er til kort, sletting og aktivisering av applets (Select). Problemet ser ut til å ligge i terminalapplikasjonen, men akkurat hva som gjør at vi får et exception her, er uvisst. API'ets begrunnelse er: `CardServiceVOPException: ???: VOP.error.OFFCARD.no such algorithm:DESede/ECB/No Padding.`

Hva er gjort:

Vi har lagt ned et betydelig antall timer i å prøve å finne ut hvorfor vi få denne feilmelding. Vi har eksperimentert med endringer av kode på nesten alle mulige måter, og også sett på eksempler fra leverandøren Gemplus, uten at dette har hjulpet. Internett har også blitt brukt for å hente inn mer informasjon angående dette problemet. Diskusjonsforum har blitt flittig brukt, men det finnes svært lite informasjon om dette emnet, siden innholdet av koden er av en slik type som ikke er så utbredt. Vi har også vært i kontakt med arbeidsgiver om dette problemet, uten at de hadde noen forslag til løsning. Heller ikke importøren av programpakken har vært borti problemet, som vi har fått høre om. Vi antar at problemet ligger i konfigurasjonen av GemXpresso RAD pakken et eller annet sted, men dette blir bare spekulasjoner.

Løsningen:

Siden vi ikke kom noen vei, og vi hadde dårlig tid, ble det i samråd med arbeidsgiver avgjort å fjerne VOP fra OCF laget. Dette førte til følgende konsekvenser:

- All VOP kode i EID måtte kommenteres ut for å kunne kommunisere med applet'en.
- Deler av VOP koden i OCF laget måtte også kommenteres ut.
- Installerings- og slettingsrutinene virker ikke lenger og vi får heller ikke testet om de virker, da disse er avhengig av en fungerende VOP og sikker kanal mellom kort og terminalapplikasjon. Installering av EID på smartkortet må heretter skje fra Gemplus' JCardManager.
- Siden det nå ikke kan brukes en sikker kanal (i følge VISA standarden), blir systemets totale sikkerhet mye dårligere. For mer informasjon om sikkerheten på Javacard se **[Vedlegg A]**.

Problemet er nok ikke umulig å løse, men det krever mye ekstra tid å sette seg detaljert inn i VOP koden, samt hvordan GemXpresso systemet fungerer i detalj. API kode er vanskelig å debugge, da mye av detaljene rundt koden er skjult for programmerer. Dette gjør det også mye vanskeligere å finne feilen, dersom dette da er et kode problem.

Etter ny test uten VOP kode for opprettelse av sikke kanal (authentication), viste det seg at det var mulig å få kontakt med kortet. Da vi sendte en "Warm reset" på kortet, fikk vi returnert ATR (Answer To Request) bytene fra kortet. Dette var ett sikkert tegn på at vi hadde fått kommunikasjon med kortet, og gleden var til å både ta og føle på.

5.5 Testing av hele systemet

5.5.1 Testmetoder

Her skal hele systemet testes om modulene virker sammen. Dette inkluderer EID på smartkortet, biometrimodul i demonstrasjonsapplikasjonen, samt demonstrasjonsapplikasjonen selv. EID kjøres her installert og klart til bruk på smartkortet. Dette må gjøres fra Gemplus' JCardManager, siden installeringsrutinen vår ikke fungerer uten VOP, og kan derfor heller ikke testes. Selve testen går ut på å prøve å kunne kommunisere med EID og se hvordan kommandoene sendt fra terminalapplikasjonen responderes av EID.

5.5.2 Testkriterier

For at systemet skal kunne sies å være ferdig i følge våre prosjektmål, har vi satt opp følgende krav som må fungere:

- Demonstrasjonsapplikasjonen må kunne sende kommandoer til EID, og få respons tilbake.
- De nødvendige filene som brukes for initialisere smartkortet med kortholderen, må kunne lastes ned på kortet. Dette gjelder sertifikater, PKCS#1, ASN.1 kodede filer for initialisering av de private RSA nøklene på kortet. PKCS#15 filene er ikke nødvendige for at systemet skal fungere, derfor setter vi ikke som krav at disse må være tilstede på kortet. Disse filene er kun for kompabilitet av ISO7816-4 filsystemet og PKCS#15 standarden. I tillegg må kortholders originale fingeravtrykktemplate lagres på kortet.
- Kortholderen må kunne identifisere seg mot EID ved å få godkjent PIN koder samt fingeravtrykket.
- Kortholder skal kunne lage en digital signatur.
- Den digitale signaturen skal også verifiseres av terminalapplikasjonen for å vise at det finerer å dekryptere denne med offentlig DS RSA nøkkel.

5.5.3 Testresultater

I testresultatene her har vi utelatt alle de mindre småfeil som ble oppdaget. I stedet tar vi for oss problemene som var av en slik karakter at det ble bruk mye tid og krefter på oppretting, og at problemene var av vesentlig karakter. Løsninger på ting som feilet, diskuteres i neste delkapittel, 5.5.4.

Resultatet av testingen:

- EID mottar og behandler APDU'er slik som antatt.
- De nødvendige filene som skal skrives til kortet, blir skrevet riktig, og er klare til bruk.
- Kortholder får verifisert både PIN koder og fingeravtrykk dersom de er riktige.
- Biometridelen ser ut til å fungere tilfredsstillende med hensyn på de oppgaver den skal ta seg av.
- Det viser seg at lesing av filer fra kortet ikke fungerer. Se mer i delkapittel 5.4.4.
- Demonstrasjonsapplikasjonen klarer ikke å gjennomføre en suksessfull decipher kommando mot kortet. Derfor kan heller ikke en digital signatur lages.
  For løsning, se neste delkapittel,. 5.5.4.

5.5.4 Problemer som testingen avdekket

Problem 1:

Det ble kastet et ScardTransmit exception under sending av Read Binary APDU. Denne APDU kommandoen brukes for å lese fingeravtrykk og sertifikater.
Grunnen til exception var at et grunnsystem (PCSC) i OCF som kommuniserer med kortleseren via operativsystemet, ikke klarte å sende APDU'en. Systemet påsto at APDU'en var ugyldig.

Hva ble gjort:

Vi gikk nøye inn på Read Binary APDU'en generert fra terminalapplikasjonen for å se om vi hadde laget denne gal. Det som var merkelig, var at denne kommandoen fungerte utmerket i simulatoren brukt under testingen av EID. Det var noe undrende at vi prøvde å endre på APDU'ens oppbygning, for å se om vi fikk bedre resultater. Vanligvis sendes ikke noen databytes eller Lc byte i Read Binary. Antall bytes som skal leses legge i Le byten, og denne skal legges sist, noe vi også gjorde. Vi prøvde å sende med Lc byte med 0x00 og ingen som datafelt. Dette løste exception problemet, men et nytt problem dukket opp i kjølvannet av dette. EIDs metode som leser filer ville ikke skjønne Le byten. Den tolket innholdet alltid til å være 0x00 uansett hva som var Le verdien.

Løsningen:

Siden arbeidsiver ikke hadde noen gode indikasjoner på hva som eventuelt kunne være galt, måtte vi på grunn av tidspress bestemme oss for en løsning som fungerte. Det viste seg nemlig at ved å legge innholdet av Le byten som en byte i datafeltet, kunne denne brukes til å representere antall bytes som skulle leses. Dette innbar også at litt av EID koden (`read_binary(APDU apdu)`) måtte endres fra å hente inn antall bytes som skulle leses fra datafeltet i stedet for i Le byten. Se tabell 5.1 for detaljer. Dette blir et avvik fra hvordan APDU'en er definert i EID spesifikasjonen, men vi følte at dette måtte gjøres for å få kommandoen til å virke. Dessuten hadde vi ikke mer tid til å bruke på dette problemet. Denne bør ses nærmere på ved videre utvikling av systemet.

CLA	INS	P1	P2	Lc	Data	Le
00	B0	Msb offset	Lsb offset	01	Kopi av Le	Antall bytes å lese

Tabell 5.1: Oversikt over Read Binary APDU'en sånn som den funger i dette systemet.

Problem 2:

Før en digital signatur kan lages, må ID og DC nøkkelen sjekkes mot de nøklene som terminalapplikasjonen har hentet ut fra sertifikatene. Encipher, som bruker ID nøkkelparet, fungerte som den skulle. EID krypterte data fra terminalapplikasjonen, og terminalapplikasjonen dekrypterte dataene og fant ut at de stemte. I decipher, der EID skal dekryptere krypterte data fra terminalapplikasjonen, fungerte ikke. EID returnerte akkurat det samme APDU innholdet som ble sent ned til kortet.

Hva ble gjort:

Vi brukte lang tid på å prøve å forstå dette problemet. Det virket noe ulogisk hvorfor EID ikke gjorde noe med dataene den fikk tilsendt. Spesielt siden encipher fungerte som den skulle. Vi prøvde å endre `pso_decpiher()` metoden i EID, ved å skrifte modus på kryptering/dekrypterings objektet (Cipher) til både `MODE_ENCRYPT` og `MODE_DECRYPT` , uten hell. Dessuten gikk vi igjennom krypteringskoden i terminalapplikasjonen uten å finne synlige feil. Diskusjonsforum på Internett ble flittig brukt for å lete etter mulig hjelp som kunne sette oss på riktig ledetråd. Dessverre uten hell også her. Det endte med at vi snakket med arbeidsgiver, og tok med PC'en dit for videre assistanse.

Løsningen:

Etter utallige forsøk for å prøve å få decipher til å fungere, både kodemessig og sjekking av nøkler måtte vi se oss slått av API koden. Arbeidsgiver antydet derfor å bruke `pso_encipher()` metoden på EID, i stedet for `pso_decipher()`. Siden at `pso_encipher()` er "hardkodet" til å bruke ID nøkkelen, måtte derfor den offentlige ID nøkkelen brukes fra terminalapplikasjonen sin side. Dette innebærer at DC nøkkelparet ikke testes idet hele tatt, noe som ikke er med på å øke sikkerheten.

Å bruke `pso_encipher()` på EID fungerte utmerket. Grunnen til det, er at de kryptere dataene fra terminalapplikasjonen er 128 bytes lange, og EID vil derfor ikke legge til PKCS1 padding. Siden Cipher objektet på kortet ikke er initialisert med padding, er det likegyldig om det kjøres `MODE_ENCRYPT` eller `MODE_DECRYPT`, siden RSA transformasjonen vil bli gjennomført likevel. Svaret som terminalapplikasjonen mottok, var paddet med tall forskjellige fra 0x00, og disse måtte fjernes manuelt. Det viste seg nå at en deciher kommando fungerte, dog ikke helt etter EID spesifikasjonen. For fremtidige løsninger, bør nok dette brukes tid på for å få decipher til å fungere som den er ment til.

Problem 3:

Samtidig som decipher ikke var løst, kommenterte vi ut den, for å få testet å lage en digital signatur. Det viste seg imidlertid at noe var galt med denne. EID klarte å lage en digital signatur ut fra SHA hash'en som ble sendt som data. Problemet var at terminalapplikasjonen ikke fikk de samme dataene som forventet, etter dekryptering av signaturen med den offentlige DS nøkkelen. Dekryptert signatur skulle i teorien stemme med SHA hash'en fra tekstdokumentet. Dette stemte ikke.

Hva ble gjort:

Mye tid gikk med for eksperimentering og endring av kode både på terminalapplikasjonen og i EID. Resultatet av dekryptert signatur ville uansett ikke stemme med opprinnelig SHA hash. Diskusjonsforum og dokumentasjon på API koden, ble flittig benyttet, men ledetrådene vi fant viste seg å ikke gi resultat. Det endte med dette, som med decipher problemet. Vi måtte få assistanse av arbeidsgiver.

Løsningen:

Løsningen på dette problemet, viste seg å være enklere enn man skulle tro, dog ikke særlig logisk. Det problemene demonstrerte også problemet med å bruke API. Dersom dokumentasjonen er dårlig, kan det være vanskelig å finne ut hva som er galt, siden innholdet av API koden stort sett er skjult.

Etter testing sammen med arbeidsgiver, viste det seg at API koden i EID applet'en som lagde selve signaturen, også gjennomførte en SHA hash av mottatte data. Disse dataene var allerede en SHA hash av tekstdokumentet som skulle signeres. Siden API koden ikke kan endres, måtte vi derfor endre på terminalapplikasjonens metode for å verifisere signaturen.
I stedet for å bruke krypto-API'ets verify() kommando, måtte vi lage en egen. Denne dekrypterte den digitale signaturen med offentlig DS nøkkel. Resultatet av dekrypteringen, måtte parses i en egenlaget funksjon for å hente ut den originale hash'en. Dekryptert data er PKCS#1, ASN.1 kodet. Etter at den originale hash'en ble hentet ut, måtte den sammenlignes med en SHA dobbelt-hash'et 20 byters datamengde fra originaldokumentet. Siden EID også hash'er hash'en som sendes til kortet, må resultatet også sammenlignet med en dobbelhash av teksten som skulle signeres, og ikke en enkel hash som vi gjorde i utgangspunktet. Etter disse forandringene viste det seg at signaturen kunne verifiseres, og selve hovedmålet med systemet var oppnådd.

Detaljer:

Testene avdekket også noen små skjønnhetsfeil angående det grafiske brukergrensenittet. Dette gjelder oppdatering av statusinformasjon, spesielt i forbindlese med fingeravtrykket. Vi har rettet opp på noe av det bl.a. egen status linje i fingeravtrykksvinduet, men siden vi føler dette kommer litt utenfor prosjektets mål, og tiden var knapp, valgte vi å ikke konsentrere oss så mye om dette. I fremtidige versjoner kan det være lurt å se litt mer på dette.

5.6 Generelle forsinkelser før testing

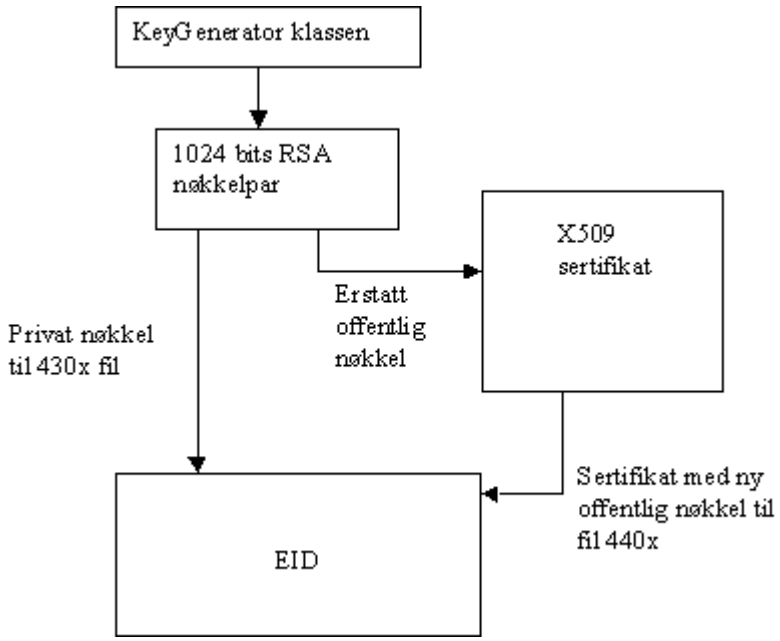
Grunnet at vi ikke fikk sertifikater og nøkkelpar av arbeidsgiver til å bruke sammen med EID og demonstrasjonsapplikasjonen, måtte disse lages selv. Dette førte til at vi lagde en egen klasse for nøkkelgenerering og administrasjon av skriving av nøkler til(se også kapittel 3). Dette bremset opp fremdriften et par dager, siden klassen for nøkkeladministrasjon ikke var medberegnet i vår tidsplan, og heller egentlig ikke en del av oppgaven.

For å lage sertifikater, brukte vi Sun's keytool program for generering av selvsignerte X509 sertifikater med 1024 bits RSA nøkkelpar. Dette er "dummy" sertifikater kun met for testing. Eksempel på bruk av keytool: generering, så skriving av sertifikat til fil:

$>keytool -genkey -v -alias crt_ds -keyalg "RSA" -keysize1024 -keypass 22222222 -sigalg "SHA1withRSA" -dname "cn=Ola Nordmann,ou=Hovedprosjekt, o=HiG,c=NO" -keystore kstore -storepass 12211221

$>keytool -export -alias crt_ds -file crt_ds.cer –keystore kstore -storepass 12211221

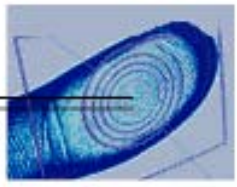
Problemet her var imidlertid at vi ikke fikk lov av keytool til å hente ut den private nøkkelen, som vi er avhengig av å initialisere i EID. Derfor var egentlig sertifikatet ubrukelig. Derfor ble keyGenerator klassen laget for å lage RSA nøkkelpar. For kunne bruke dette sammen med sertifikatet, måtte vi gå inn i sertifikatet og erstatte den eksisterende offentlige nøkkelen med vår egen genererte. Etter leting med en hex-editor, fant vi ut at den 162 bytes lange offentlige nøkkelen skulle legges inn i byte posisjon 0x5d (93) og utover, i sertifikat filen.
Dette gjør at sertifikatets offentlige nøkkel kan brukes mot den private som da legges på kortet. Se figur 5.3 for illustrasjon. Selv endringen av sertifikatet gjøres utenfor demonstrasjonsapplikasjonen.



Figur 5.3: Tilpasning av selvsignerte sertifikater med egenlagde RSA nøkkelpar.

KONKLUSJON

Java SmartSign 2002



6 Konklusjon

6.1 Evaluering i forhold til kravspesifikasjonen

Allerede fra starten av visste vi at dette var et hovedprosjekt med en del usikkerheter. Arbeidsgiver visste ikke så mye om plattformen som skulle brukes, noe som gjenspeiler seg i at vi, når oppgaven ble gitt, fikk oppgitt at kortet hadde plass til 32 kb. Like før levering av kravspesifikasjonen fikk vi imidlertid oppgitt at dette hadde plass til kun 24 kb data, mens enda senere i prosjektet ble dette nedjustert til 19 kb. I tillegg var det uvisst om Javacardplattformen støttet den standardiserte filsystemet som skulle brukes og en del andre faktorer dokumentert i kapittel 5.5.4. At det ble en del usikkerhet rundt oppgaven er for så vidt naturlig, da en større del av oppgaven skulle være og se på hvilke muligheter Javacardplattformen hadde å tilby. Dette førte imidlertid til en del utfordringer og problem, som i de fleste tilfeller ble løst. Likevel finnes det enkelte avvik fra kravspesifikasjonen som dokumentert under.

6.1.1 EID-applikasjon

I tråd med kravspesifikasjonen skulle det her implementeres en Elektronisk ID-applikasjon på Javacard med mulighet for generering av elektronisk signatur. Dette skulle gjøres etter gitt spesifikasjon fra arbeidsgiver, se **[Vedlegg D]**, noe som begrenset mulighetene ved design noe. Resultatet er blitt en EID-applikasjon som i all hovedsak fungerer som den skal, og oppfyller kravene som er satt. Likevel var det visse ting som ikke lot seg gjennomføre, da først og fremst fordi disse ikke ble støttet i Javacard, siden gitt spesifikasjon gjaldt et MultOSbasert system. I tillegg var det også noen problemer under koding som ikke har latt seg løse, slik at det har måttet finnes alternative løsninger, som diskutert i kapittel 5.4 og 5.5**.** Likevel føler vi at dette punktet i kravspesifikasjonen i all hovedsak er oppfylt.
Avvikene fra kravspesifikasjonen er:

6.1.1.1 Filsystem

Som nevnt over baserte spesifikasjonen fra ErgoSolutions seg på et MultOSbasert smartkort. MultOS har en innebygget støtte for et ISO-7816-4 basert filsystem, som nevnt i **[Vedlegg A]**. Versjon 2.0 av Javacard API'et hadde en tilpasning til et slikt filsystem, problemet vårt var imidlertid at Javacard versjon 2.11 som ble brukt i prosjektet, ikke har støtte for et slikt tradisjonelt filsystem. Dette er i stor grad på grunn av at Java baserer seg på lagring av data i objekter i stedet. For ikke å bryte med standarden måtte derfor gruppen designe og implementere et eget filsystem. Dette fungerer, med unntak av at read_binary funksjonen, der det måtte justeres et parameter i henhold til spesifikasjonen, se kapittel 5.5.4 .

6.1.1.2 Kryptering

I henhold til spesifikasjonen fra oppdragsgiver skulle det utvikles tre kryptografioperasjoner. To av dem sjekker at nøkkelparene til kort og kortleser stemmer, mens den tredje foretar selve signeringen. Det ble imidlertid problemer med den ene av dem. Det ble gjort flere iherdige forsøk på i løse dette, og arbeidsgiver fant heller ingen løsning på problemet. Dette ble løst ved å benytte den andre nøkkelverifiserings-funksjonen to ganger etter hverandre. Se ellers 3.2.6 og 5.5.4 for nærmere informasjon.

6.1.1.3 PIN/PUK-koder

Implementasjonen av PIN- og PUK-koder har gått etter planen, og fungerer som den skal etter spesifikasjonen.

6.1.2 Biometridel

Fingeravtrykkslesing er implementert, men denne er imidlertid offcard, noe som vi visste kunne bli utfallet på forhånd, men som ikke var ønskelig da dette går utover sikkerheten. Det ble dermed klart at det måtte implementeres PIN-koder i tillegg for å få en noenlunde sikkerhet på applikasjonen. Måten godkjenningen av biometridelen overføres til kortet på er lite tilfredsstillende, men som vedlagt dokument viser **[Vedlegg B]**, er problemene rundt implementasjon av oncard løsning for så vidt store at vi ikke har hatt muligheter til dette innenfor prosjektets rammer.

6.1.3 Demoapplikasjon

Demoapplikasjonen er laget, og denne fungerer som den skal iht. kravspesifikasjonen.
Den demonstrerer at systemet virker, og kan signere et dokument og verifisere dette, selv om applikasjonen naturligvis er noe mer statisk enn i en fullt utviklet system. Applikasjonene er i tillegg oppdelt i moduler på en slik måte at det vil være lett og utvikle systemet videre, med egne moduler for testing, administrering og biometri, så langt dette har vist seg mulig. Se ellers punkt 5.5.4.

6.1.4 Sikkerhet

Sikkerheten er naturligvis et viktig aspekt ved en Elektronisk ID-applikasjon.
Selv om dette ikke har vært et direkte hovedmål ved prosjektet, har det så langt det har vært mulig, forsøkt å gjøre applikasjon og kode sikker. Likevel må det tas i betraktning at gruppen ikke har noen erfaring i å skrive sikker kode fra før, med unntak av sjekk av buffere, APDU-data og lignende. I de tilfeller da det måtte tas et valg mellom å få ting til å fungere eller gjøre det sikkert, har funksjonaliteten blitt prioritert. Et eksempel på dette er VOP, der det etter lang tids feilsøking, ble valgt å droppe dette for å få løsningen til å fungere, og komme videre i prosjektet.

6.1.5 Sammenligning av sikkerheten på Javacard i forhold til MultOS

Rapporten ble skrevet som spesifisert i kravspesifikasjonen, og har så vidt vi kan bedømme, ingen større mangler eller avvik i forhold til oppgitt oppgave. Se **[Vedlegg A]** for resultatet fra denne rapporten.

6.2 Forslag til forbedringer for fremtidig bruk

Slik systemet er nå, er det ikke klart for vanlig praktisk bruk. Som vi har nevnt er dette en prototype, som må videreutvikles en del før den kan tas i bruk. Spesielt gjelder dette terminal/demonstrasjonsapplikasjonssiden. Hovedvekten i dette prosjektet er lagt på kortapplikasjonen, mens demonstrasjonsapplikasjonen på PC er kun for å demonstrere bruken av EID'applikasjonen på smartkortet. Som følge av dette er derfor PC-applikasjonen den som er mest "uferdig". Den består strengt tatt av fire deler, Det er kommunikasjon mot kortet fra terminalapplikasjonen, dette skjer gjennom OCF, og tar seg av å starte og initialisere rammeverket for å få til kommunikasjon med selve smartkortet. I tillegg fås det gjennom dette tilgang til en del funksjonalitet som forenkler det å jobbe mot smartkortet sett fra programmererens side. Videre er det en brukerdel, hvor kortholderen autentiserer seg mot kortet, og foretar selve signeringen. Som en sidemodul til denne, brukes biometrienheten som tar seg fingeravtrykklesing i autentiseringsprosessen. Administrasjonsenheten er siste del. Denne tar for seg de oppgaver som en utsteder av kortet vil måtte betjene. Delene kommuniserer tett, men det ble lagt vekt på å skille disse mest mulig for å få en så modulbasert løsning som mulig.

6.2.1 Kommunikasjonsdel

Her har den grunnleggende kommunikasjonen som trengs mellom smartkortet og terminalapplikasjonen blitt etablert, men det har ikke blitt tid til å implementere en såkalt CardListener. Denne vil i praksis sørger for å gi beskjed til OCF laget om når et smartkort blir satt inn , og når det blir fjernet fra kortleseren. Dagens implementasjon fungerer ved at applikasjonen søker etter et smartkort ved oppstart og setter enten at kort eksisterer, eller at dette ikke finnes. Det er ikke tatt hensyn til at kort kan fjernes mens det jobbes mot det, som bør gjøres i en videreutvikling av systemet.

6.2.2 Bruker og administratordel

Når det gjelder utseende er dette utviklet med hensyn på funksjonalitet og ikke spesielt for brukervennlighet. Derfor bør det ved fremtidig bruk, implementeres et mer brukervennlig grensesnitt. Slik det er i dag er dette noe sparsommelig, og det bør legges til en utvidet menylinje for å øke brukervennligheten, samt med knapper, ikoner, snarveier etc. I tillegg bør det implementeres en hjelpefunksjon, og skrives en hjelpefil hvor brukeren kan søke etter informasjon han lurer på i forbindelse med praktisk bruk av applikasjonen. Selve designet og plassering av komponenter bør nok også sees på før systemet lanseres til vanlige brukere, og det bør sørges for konsistens når det gjelder utseende og innhold i feilmeldinger. Vinduet som

viser fingeravtrykket bør innlemmes i selve applikasjonens hovedvindu, da dette ved dagens løsning, er frittstående i forhold til resten av applikasjonen. Ytterligere perfeksjonering av sjekking av input fra bruker kan foretas.

Dagens implementasjon har heller ikke noen nettverksfunksjonalitet. Denne vil sørge for at CA kontaktes og at sertifikat for personen du har mottatt signering fra kan hentes ned og verifiseres automatisk, i stedet for at bruker må hente ned sertifikat selv og velge dette manuelt i programmet.

6.2.3 Biometridel

Etter hvert som kortteknologien utvikles bør det forsøkes å få implementert fingeravtrykksjekken på smartkortet (oncard). For at dette skal la seg gjøre må kort med større minne og prosessor benyttes, noe som nok vil komme på markedet innen kort tid. Har man et kort som kan takle selve prosesseringen, står valget mellom å utvikle en løsning selv, ut fra en biometrisk algoritme, eller å bruke eventuell API-funksjonalitet. Slik API-funksjonalitet er blant annet foreslått i spesifikasjonen for Javacard 2.2 API'et.
Et annet forbedringspunkt for biometridelen er å lagre mer enn ett fingeravtrykk på kortet. Dette er forholdsvis enkelt, men krever plass til en ekstra template på kortet, og en mindre omskriving av koden. Poenget med dette er at kortholder da ikke er avhengig av en enkelt finger, hvis denne skulle bli skadet av rifter o.l.

I tillegg har kan det fås en marginal forbedring av fingeravtrykket på kortet hvis det brukes et "best-av-tre" system ved scanning av fingeravtrykket som brukes til lagring på kortet. Dette har ingen sikkerhetsmessig betydning, da man uansett har satt et minstekrav på kvalitet, men vil kunne gi et litt bedre utgangspunkt for gjenkjenning, slik at det blir færre feil-scanninger av fingeren.

6.2.4 EID-applikasjon

Videreutvikling av EID-applikasjonen på smartkortet vil i første omgang være og se på `pso_decipher()`, som er kryptografioperasjonen som ikke fungerer i dag. Vi har imidlertid ingen klar løsning for å få denne funksjonen til å fungere, etter gjentatte forsøk, men det antas at dette vil løse seg ved videre utvikling.
Neste skritt vil være å implementere fingeravtrykkssammenligningen på kortet som nevnt over. Et annet punkt som bør gjennomføres et at IS og DC PIN nullstilles før kasting av eventuelle isoexception i koden, for økt sikkerhet. Når det gjelder filsystemet bør det ses på om bug'en ved `read-binary()` funksjonen kan rettes, se 5.5.4. Samt se på muligheten for å erstatte variabelnavn på filer i filsystemet, ved å sette alle i en array av objekter, for lettere traversering.

Det kan også ses på muligheten for å effektivisere koden, samt at denne bør gjennomgå en sikkerhetsvurdering, da systemet bygger på sikkerheten ved EID'en .
En av forbedringene som bør få høy prioritet er at det bør sørges for at dataendringer av array innhold skjer atomisk. Slik løsningen ble implementert i første omgang fungerte dette i simulator, men ved testing mot reelt smartkort fungerte ikke denne metoden, slik at løsningen

som brukes i dag måtte bruke en annen metode som ikke ivaretar atomisitet ved oppdatering av arrayer. Dette vil i første omgang ha noe å si hvis kortet fjernes uventet mens det jobbes mot det.

6.3 Sikkerhet

Sikkerheten er imidlertid delvis et problem. Det viste seg at Javacard-plattformen ikke har noen klart definert løsning på hvordan applikasjonene på kortet skal administreres. Dette medfører at applikasjonen i utgangspunktet kan slettes av hvem som helst som har investert i en utviklerpakke for kortet. Dette er lite tilfredsstillende rent sikkerhetsmessig sett. Det kan imidlertid ikke fås tilgang til koden for EID-applikasjon, eller de data som denne inneholder (nøkler, verdier etc.), men den kan slettes og erstattes. For å bøte på dette er en utviklet et tilleggsrammeverket av Visa kalt Visa Open Platform (VOP). Bruk av denne hadde ikke arbeidsgiver krevd ved prosjektets start, men det ble etter hvert innsett av gruppen at denne burde brukes. Dette systemet sørger for at det fås en sikker applikasjonshåndtering ved hjelp av krypteringsnøkler, der en spesiell nøkkel må brukes for å få lov til og administrere applikasjoner på kortet. Implementasjonen av dette viste seg imidlertid å være svært vanskeligere enn antatt. Først måtte man sette seg inn i hvordan VOP fungerte, noe som gikk greit, men tok en del tid. Deretter måtte det tas fatt på selve implementasjonen. Etter å ha programmert denne, og begynt testingen ble det funnet ut at dette ikke fungerte. Av en ukjent grunn fikk terminalapplikasjonen ikke autentisert seg mot kortet. Dette gjorde at kortet i praksis ikke kunne brukes, da det på grunn av dette ikke gis tillatelse til å benytte funksjonalitet eller applikasjoner på kortet av sikkerhetsgrunner. Koden ble gjentatte ganger endret for å prøve å få den til og virke og arbeidsgiver og produsent ble kontaktet, samt at det ble foretatt intensiv leting på Internett etter løsninger. Til tross for dette ville ikke VOP funksjonene fungere, og gruppen valgte derfor i samråd med arbeidsgiver å droppe VOP for å ta fatt på andre oppgaver. Dette påvirker ikke funksjonaliteten ved løsningen, og er heller ikke avgjørende for systemet da dette er en prototype, men en vesentlig svekkelse av sikkerheten.

6.4 Kritikk av oppgaven

Alt i alt er gruppen fornøyd med resultatet. Det har blitt utviklet et system som fungerer, med tanke på at det i utgangspunktet skulle være en prototype, ikke et ferdig system. Likevel finnes det enkelte ting som nok kunne vært gjort annerledes, hvis prosjektet skulle gjøres på nytt.

Noe av det første som det burde ses nærmere på, er implementeringen av VOP. Hadde gruppen visst mer om dette systemet ved prosjektets start, og vært klar over de svakheter Javacard hadde på administrasjon av applikasjoner, ville det nok ha blitt satt av noe mer tid til dette.
Likevel føler gruppen at tidsforbruket på å løse problemet, er forsvarlig . Siden arbeidsgiver og de hjelpekildene vi har hatt tilgjengelig ikke hadde noen ledetråder på problemet, kom vi derfor ikke videre med dette. Det bør imidlertid være løsbart i et litt større tidsperspektiv. VOP var imidlertid ikke et krav i forhold til kravspesifikasjonen.

Ganntskjemaet som var satt opp ved prosjektets start, viste seg å ikke stemme helt overens med den praktiske fremgangen. Dette skyldes nok vår manglende erfaring innen arbeid i prosjekter av denne størrelsen. Se ellers **[Vedlegg F]** for forklaringer på avvikene i henhold til planen.

Når det gjelder feilsøking og forventet tidsforbruk på denne, hadde man nok også undervurdert dette noe. Det var klart at problemer ville oppstå, og at disse ville ta noe tid å løse, men det viste seg at enkelte av problemene, henholdsvis VOP og en del problemer rundt EID'en, tok lengre tid enn beregnet. Dette er imidlertid vanskelig og beregne på forhånd, og vil nok også bli bedre med økende erfaring.

Vi ser i ettertid at man med fordel kunne hatt minst en maskin ekstra å teste mot. Testingen av systemet, og dels også utvikling fungerte i dette prosjektet ved at man måtte skrive kode individuelt, og kopiere denne over på maskinen utlånt fra ErgoSolutions for kompilering og simulering/testing. Grunnen til dette er i hovedsak lisensproblematikken rundt bruk av programvaren.

Biometridelen av prosjektet er implementert etter de rammer og begrensninger vi hadde, både når det gjelder kort, og tid vi hadde til rådighet. Slik løsningen er nå har den et lite tilfredsstillende sikkerhetsnivå, men fungerer som den skal.

6.5 Konklusjon

Oppgaven vår var å utvikle EID på en relativt ny smartkortplattform, og med mulighet for fingeravtrykk som autentiseringsform. Dette var teknologi verken vi eller oppdragsgiver hadde særlig kunnskap om på forhånd. På grunn av dette, har oppgaven også gått ut på en del ”forskning” og undersøkelse, snarere enn en ren implementasjon. På bakgrunn av dette sier gruppen seg rimelig fornøyd med resultatet, og føler at vi har nådd målene vi har satt oss i forkant av prosjektet. Selvsagt kunne man likevel ønske at enkelte ting hadde blitt bedre, men da man ikke var sikker på hva man kunne få til på forhånd, var dette forventet.

Arbeidet innad i gruppen føler vi har fungert svært godt. Dels kan dette nok tilskrives at vi kjente hverandre godt på forhånd. Det har også vært en stor fordel at vi bor på samme sted, på denne måten har det lett å kalle sammen til møter og ta kontakt når problemer oppsto.

I løpet av prosjektet har vi nå fått muligheten til å sette oss i en teknologi vi tror vil spille en stor rolle i fremtiden. Dette har vært spennende, og vi har også fått et innblikk i hvordan PKI-teknologien fungerer, og kan brukes for å oppnå større sikkerhet i et digitalt samfunn.

Litteraturliste

Følgende har vært brukt som støttelitteratur og oppslagsverk til dette prosjektet :

”Smart Card Handbook” Second Edition”
W. Rankl og W. Effing
John Wiley & Sons, Inc., 2000

“Java Card technology for smartcards –Architecture and programmer’s guide”
Chen, Zhiqun
Sun Microsystems
Addison-Wesley, 2000

”Smart Card Security and Applications” Second Edition
Mike Hendry
Artech House, Inc., 2001

”Smart Card Application Development Using Java”
Uwe Hansmann, Martin S. Nicklous, Thomas Schäck, Frank Seliger
Springer-Verlag Berlin Heidelberg, 2000

”Java How to program” Third Edition
H.M. Deitel, P.J. Deitel
Prentice-Hall, Inc., 1999

”GemXpresso RAD III User Guide v. 3.1”
Gemplus, 2001

”GemXpresso RAD III Getting Started v. 3.1”
Gemplus, 2001

”PKCS#15 EID”
Morten Johansen, 2002

”ISO/IEC 7816-4”
ISO/IEC, 1995

“PKCS#1 v.2.1 : RSA Cryptography Standard”
RSA Laboratories, 2001

”PKCS#15 v.1.1 : Cryptographic Token Information Syntax Standard”
RSA Laboratories, 2000

Vedlegg

Vedlegg A - *Sammenligning av Javacard mot Multos*

Tar for seg angrepsformer mot smartkort generelt. Informasjon, generelt og teknisk, om Javacard og Multos. Tilslutt En sammenligning av sikkerheten på disse.

Vedlegg B - *Biometrirapport*

Tar for seg tankegangen ved bruk av fingeravtrykk, både generelt og prosjektmessig.

Vedlegg C - *En kort introduksjon til kryptering og PKI*

Forklarer hvordan prosessene for de forskjellige krypteringsmetodene fungerer. Synkron-, asynkron kryptering, i tilegg elektronisk signatur. Hva er og hvorfor PKI?

Vedlegg D - *EID spesifikasjon*

EID spesifikasjon gitt av arbeidsgiver, ErgoSolutions. Inneholder de standarder som måtte følges under utviklingsprosessen.

Vedlegg E - *Brukerveiledning*

Forklarer de forskjellige knappene og hva de gjør for en bruker av systemet.

Vedlegg F - *Evaluering av gannt skjema*

Diagrammer over tenkt og endelig arbeidstid av forskjellige delene i prosjektet. I tillegg en evaluering av forskjellene.

Vedlegg G - *Eksempler på gruppemøtereferat, statusrapport og statusmøterapport*

Viser strukturen og innholdsmessig hva som ble tatt opp ved møter

Vedlegg H - *Grupperegler*

Regler som gruppemedlemmene ble enige om før prosjektstart.

Vedlegg I - *Prosjektavtale*

Avtale mellom HIG, ErgoSolutions og gruppens medlemmer (her ikke underskrevet).

Vedlegg J - *Innhold på vedlagt cd-rom*

Strukturen på medfølgende CD.