BACHELOROPPGAVE

AUTENTISERING VED BRUK AV TPM (TRUSTED PLATFORM MODULE)

Forfatter(e): Per Øyvind Håvelsrud
David Ormbakken Henriksen

Dato: 25. mai 2009

Sammendrag

Tittel:	Autentisering ved bruk av TPM (Trusted Platform Module)
Dato:	25. Mai 2009
Forfattere:	Per Øyvind Håvelsrud og David Ormbakken Henriksen
Veileder:	Førsteamanuensis, Lasse Øverlier, NISlab, Høgskolen i Gjøvik
Oppdragsgiver:	Buypass AS
Kontaktperson	Mads Henriksveen
Nøkkelord:	TPM, Trusted Platform Module, Autentisering, C#
Antall sider:	54
Antall vedlegg:	6
Tilgjengelighet:	Åpen

Kort beskrivelse

Prosjektet er gjennomført på oppdrag fra Buypass AS. Prosjektoppgaven har gått ut på å se på hvordan TPM brikken kan benyttes for brukerautentisering mot en sentral webbasert-tjeneste. Herunder, kartlegge TPM brikken og utvikle en demonstrator/prototype.

Trusted Platform Module(TPM) er en sikkerhetsmodul, som leveres som en standardkomponent i et stadig økende antall solgte PC-er.

Rapporten har som formål å gi Buypass AS en mye bedre innsikt i TPM brikkens muligheter og begrensninger.

Utfordringen med prosjektet har vært at dokumentasjonen på TPM brikken har høy kompleksitet og dermed krever mye bakgrunnskunnskap. Siden TPM brikken er en relativt nyutviklet teknologi, er det også lite støttelitteratur.

Prototypen/demonstratoren er ikke ferdigutviklet, men mesteparten av arbeidet med grensesnittet ned mot TPM brikken er ferdigutviklet. Teorien rundt realiseringen av applikasjonen er beskrevet.

Utviklingen er gjort i Microsoft Visual Studio, i C# for Microsoft Windows Vista/Windows 7.

Forord

Prosjektoppgaven ble foreslått av Buypass AS ved Mads Henriksveen vinteren 2008. Vi hadde begge hørt om TPM brikken, men ingen av oss hadde noen stor forståelse for hva denne teknologien omfattet. Det vi viste fra før, var at denne teknologien var fremtidsrettet og kunne hjelpe til med å forbedre sikkerheten i PC-er.

Vi tok kontakt med oppdragsgiver i desember og avtalte et møte for å få avklart en del detaljer rundt prosjektbeskrivelsen. Etter dette møtet var vi begge to sikre på at dette var det prosjekt vi kunne tenke oss. Vi sendte inn en forespørsel til fagmiljøet og ble senere i desember måned tildelt prosjektet. I slutten av januar ble forprosjektrapporten levert og arbeidet startet for fullt.

Dette prosjektet har vært utfordrende med tanke på hvor mye kunnskap som har måte opparbeides og tilegnes under prosjektgjennomføringen. Kryptologi er blant annet et tungt emne som veldig mye av funksjonaliteten til TPM brikken er basert rundt og dermed krever en del innsikt i. Vi i gruppa har gått fra å kunne minimalt rundt denne teknologien, til å kunne veldig mye. Med andre ord, prosjektet har vært veldig lærerikt og gitt oss mye kunnskap og mange erfaringer som vi kan ta med oss videre.

Vi ønsker å takke vår veileder Lasse Øverlier for oppfølging og godt samarbeid. Vi ønsker også å takke vår oppdragsgiver Buypass AS ved Mads Henriksveen for all hjelp og for å ha foreslått dette prosjektet.

Gjøvik, 25.5.2009

X

Per Øyvind Håvelsrud

X

David Ormbakken Henriksen

INNHOLDSFORTEGNELSE

1. INNLEDNING

TPM brikken er en sikkerhetsmodul som blant annet kan benyttes for sikker oppbevaring av digitale nøkler, sertifikater og passord. Modulen har støtte for operasjoner som generering av nøkler, digital signering og kryptering. Denne teknologien vil kunne bidra til å rette opp mange sikkerhetsmessige svakheter i dagens PC-er.

Oppgavens formål er å kartlegge TPM brikken, se på hvordan TPM brikken kan brukes for å oppnå sikrere brukerautentisering og utvikle en prototype/demonstrator som bruker TPM brikken for brukerautentisering mot en sentral webbasert tjeneste.

1.2 MÅLGRUPPE

Målgruppe for denne rapporten er prosjektets veileder, sensor og oppdragsgiver. Rapporten har under skriving vært prøvd skrevet på en mest mulig forklarende måte. Til tross vil rapporten være tiltenkt en leser med over normal datafaglig forståelse og da spesielt innenfor områdene informasjonssikkerhet og programvareutvikling.

1.3 HVORFOR DETTE EMNET

Buypass AS ønsket med dette prosjektet å få en bedre innsikt enn det de har nå i TPM brikkens potensialer og begrensninger. De vil med resultatene som foreligger i dette prosjektet ha et større grunnlag for å bestemme seg for om TPM brikken er noe å basere fremtidige løsninger på. Prosjektet vil gi dem en teoretisk og praktisk plattform å bygge videre på, hvis de finner det hensiktsmessig.

Prosjektgruppa hadde mye av den samme motivasjonen som Buypass AS rundt valget av dette emnet. Vi hadde hørt om brikken, men det stoppet også der. Siden TPM brikken i dag er å finne i så mange PC-er(inkludert foresatte sine), men så få vet hva den er eller kan brukes til, ønsket vi å være en av de som hadde dypere innsikt i denne fremtidsrettede teknologien.

1.4 PROSJEKTGRUPPA SIN BAKGRUNN OG KOMPETANSE

Begge gruppemedlemmene har gått tre år ved Høgskolen i Gjøvik og har en bakgrunn med datatekniske fag. Per Øyvind har elektronikk bakgrunn fra videregående, i tillegg til jobberfaring med elektronikk fra Hapro og hadde derfor noe innsikt i kommunikasjon mot hardware fra ett softwarelag. Per Øyvind har de tre årene ved Høgskolen gått en programvareutviklingslinje og David har gått en informasjonssikkerhetslinje.

Behovet for tilegning av nye kunnskap i dette prosjektet, har for Per Øyvind sin del vært innenfor informasjonssikkerhetsrelaterte emner som blant annet kryptologi, mens for David har det vært innefor programvareutviklingsspråket C#. Begge har i stor grad måte oppfriske kunnskap tidligere lært og bygge på disse.

1.5 ARBEIDSMETODE OG GJENNOMFØRING

I dette prosjektet valgte vi å jobbe etter den inkrementelle utviklingsmodellen. Vi valgte denne modellen da det enkelt lot seg gjøre å dele opp arbeidet i moduler, som kunne utføres parallelt. Dette kommer tydelig frem i fremdriftsplanen(vedlegg A) som ble laget.

Da Per Øyvind hadde kunnskap i C# og programmering mot hardware fra før, falt det seg naturlig at han tok for seg den første modulen som var grensesnittet ned mot TPM brikken. David opparbeidet seg parallelt kunnskap innen C# og begynte å kartlegge TPM brikken. Fordelen med denne arbeidsfordeling var at vi fikk kunnskap rundt TPM brikken på begge plan(overordnet og underordnet). Hvis en av oss hadde en problemstilling som gikk den ene veien, kunne som regel den andre komplimentere og man kunne utarbeide et svar.

Det viste seg raskt at vi hadde tatt på oss mye mer arbeid, enn vi ville klare å gjennomføre. Dette da kompleksiteten rundt TPM brikken var mye større enn vi hadde forutsett. Dokumentasjonen var tyngre og krevde mer dyp bakkunnskap enn det vi hadde fra før. Det gikk derfor med en del arbeid på å opparbeide seg manglende/ikke-tilstrekkelige kunnskap. Dette var heller ikke alltid lett, da teknologien i seg selv er såpass ny og støttelitteratur ikke alltid var tilgjengelig/mulig å finne.

Vi valgte å fortsette prosjektet som oppsatt på fremdriftsplanen. Forskjellen var at David beveget seg etter hvert over til modul 2 og tok for seg teorien rundt hvordan realiseringen av modul 2 kunne oppnås. Med andre ord, utviklingsdelen av fremdriftsplanen har blitt forskjøvet fram til starten på modul 2. Modul 2 har ikke blitt utviklet, men en teoretisk beskrivelse av hvordan Modul 2 kan løses har blitt utarbeidet. Arbeidet som er satt opp før modul 2 har foregått som oppsatt.

1.6 ORGANISERING AV RAPPORTEN

Rapporten er delt inn i fire hovedkapitler. Disse fire kapitlene har en del avsnitt, noe som kommer tydelig frem utefra nummereringen. Vedleggene er plassert etter litteraturlisten og er henvist til i rapporten der de er aktuelle. Da mesteparten av dataspråket er basert på engelske ord, har vi så godt det lar seg gjøre oversatt til norske ord. Der vi ikke har funnet et norsk ord som kan beskrive det engelske ordet på en god nok måte, har vi valgt å bruke det engelske ordet.

Kort beskrivelse av kapitlene:

1. **Innledning**
   Dette kapitelet tar for seg prosjektet og en del praktiske opplysninger rundt det.
2. **Teori**
   Dette kapitlet er ment å gi en del grunnlegende kunnskap og konsepter rundt TPM brikken. Kapitelet er systematisk oppbygd for å gi en økende innføring i TPM brikken. Kapitelet avsluttes med å gi en løsning på hvordan modul 2 i utviklingsdelen kan løses.
3. **Utvikling**
   Dette kapitelet er bygd opp rundt en standard systemutviklingsmal for dokumentering. Kapitelet tar for seg kravspesifikasjon, design, implementering og testing.
4. **Avslutning**
   Dette kapitelet tar for seg drøftinger/diskusjoner rundt oppgaven og gjennomføringen, samt hva som gjenstår av videre arbeid. Kapitelet avsluttes med en konklusjon.

Kort beskrivelse av vedleggene:

- **Vedlegg A.**
  Dette vedlegget inneholder fremdriftsplanen, slik den var tenkt i januar.
- **Vedlegg B.**
  Dette vedlegget inneholder forprosjektrapporten som ble laget i januar.
- **Vedlegg C.**
  Dette vedlegget inneholder klassediagrammer for modul 1(grensesnittet mot TPM brikken).
- **Vedlegg D.**
  Dette vedlegget inneholder ett eksempel, på en av mange statusrapporter sendt til veileder.
- **Vedlegg E.**
  Dette vedlegget inneholder kopi av prosjektavtalen med arbeidsgiver.
- **Vedlegg F.**
  Dette vedlegget inneholder en skisse som forklarer hendelsesforløpet i TPM brukerautentiseringsløsningen.

2. TEORI

De første avsnittene i dette kapittelet er ment å gi en overordnet innføring i TPM brikken. Påfølgende avsnitt vil gi videre innføring i TPM brikken sin arkitektur og funksjonalitet. Teorien i disse avsnittene er hovedsakelig ment som en innføring i TPM brikken og som støtte litteratur. Støtte litteratur for det siste avsnittet som tar for seg brukerautentisering og kapitelet som omfatter utvikling.

Teori gjennomgått i dette kapittelet er primært hentet fra Trusted Computing Group(TCG) sin hjemmeside www.trustedcomputinggroup.org og da spesielt fra TPM spesifikasjonsdokumentet del 1[1] og TCG arkitekturoversikt dokumentet[2]. En bok utgitt av IBM kalt ”A Practical Guide to Trusted Computing”[3] har også vært brukt, foruten kunnskap opparbeidet under bachelor studiet. Andre kilder som har blitt brukt, vil være henvist til i teksten.

2.1 BAKGRUNN

I oktober 1999 ble Trusted Computing Platform Alliance(TCPA) opprettet. Dette var en industri allianse mellom Compaq, HP, Intel, Microsoft og IBM. Det overordnede målet til denne alliansen var å spesifisere og utvikle en fremtidig sikkerhetsløsning for PC-er som skulle være enkel å installere, bruke og håndtere. Da motivasjonen og intensjonen til de forskjellige partene bak dette prosjektet er mye diskutert andre steder og ikke er av stor relevans for dette prosjektet, vil vi ikke gå nærmere innpå det her.

For å oppnå målet sitt, så TCPA at det trengtes en ny komponent i systemet. En komponent som alltid ville være til stede i systemet og som kunne gå god for resten av systemet sin tilstand. Herav begrepene tiltrodd komponent og tiltrodd plattform. Den tiltrodde komponenten måtte være sikret mot fysiske angrep og kunne håndtere problematikken rundt de unngåelige sikkerhetshullene i operativsystemer og applikasjoner(software) som angripere vil benytte seg av. Dette var starten på det som resulterte i TPM spesifikasjonen og senere utviklingen av TPM brikken.

Tidlig 2001 slapp TCPA den første TPM spesifikasjonen, versjon 1.0. Den nyeste spesifikasjonen(versjon 1.1) og arbeidet så langt i prosjektet, ble i 2003 arvet av Trusted Computing Group(TCG) som var en omorganisering av TCPA under nytt navn. TCG kom samme året ut med versjon 1.2 av spesifikasjonen som fortsatt er siste versjon, men som har kommet i reviderte utgaver siden. Siste revisjon 103 ble publisert i juli 2007.

Selv som den første TPM spesifikasjonen ble publisert så tidlig som i 2001, tok det sin tid før den første TPM brikken ble produsert. Versjon 1.1 av TPM brikken finnes i et fåtall av bærbare PC-er solgt til forbruker. Det var først i 2006 med versjon 1.2 at brikken ble masseprodusert og inkludert i et stadig økende antall PC-er til dags dato.

2.2 STATUS I DAG

I dag er over 100 millioner PC-er utstyrt med en TPM brikke. Fleste parten av disse PC-ene er bærbare, men andelen stasjonære PC-er er økende. Blant de store produsentene som tilbyr denne teknologien i sine produkter finner vi Lenovo, Hewlett-Packard, Asus, Dell, Acer, Toshiba og Fujitsu Siemens. Det siste året har også flere produsenter av servere begynt å inkludere TPM brikken i sine løsninger, deriblant IBM og Dell. TPM brikken er forventet å bli en komponent inkludert i alle solgte PC-er i en ikke alt for fjern fremtid.

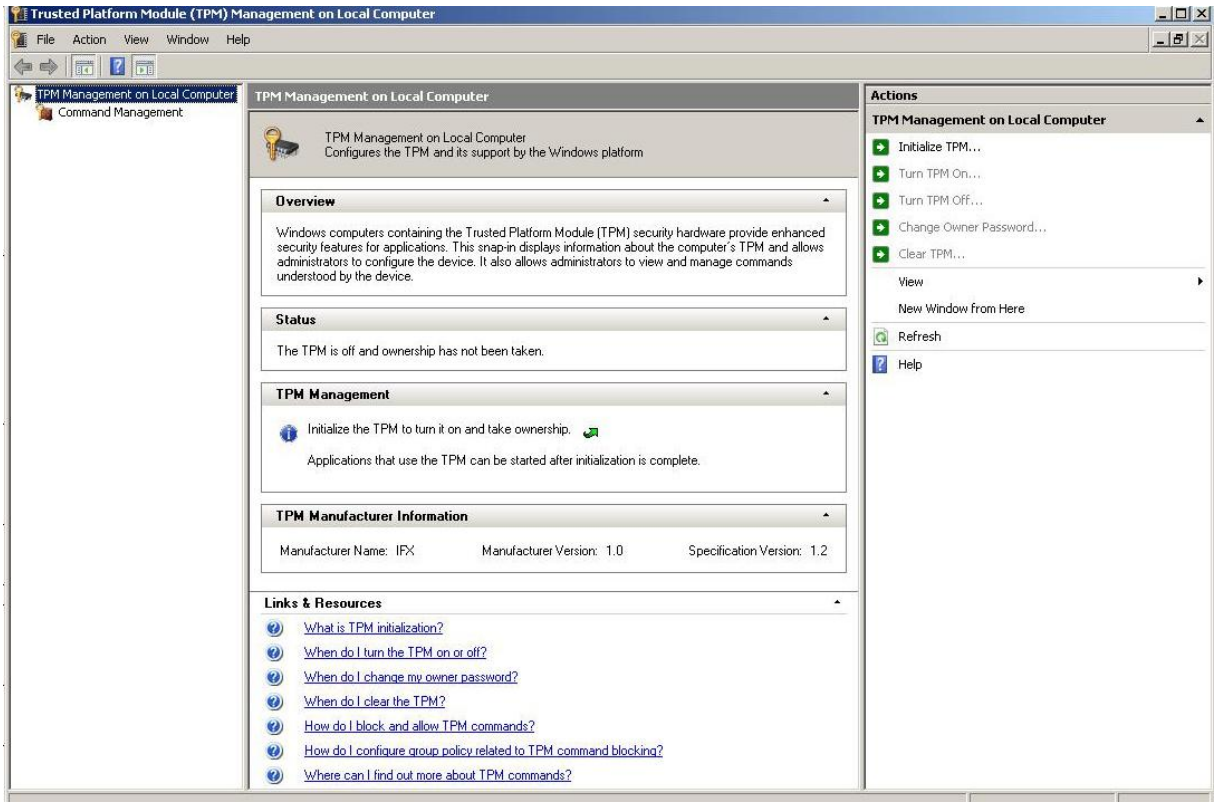
Dette betyr at veldig mange PC-brukere allerede i dag har denne teknologien tilgjengelig. Utfordringen i dag er derfor ikke å få distribuert TPM brikken, men å informere brukerne om mulighetene og fordelene ved å ta den i bruk. Dette er ikke en liten utfordring da de fleste brukerne i dag ikke vet at de er i besittelse av en TPM brikke og siden et enda lavere antall igjen ikke vet hva den kan brukes til. Av de som vet hva den kan brukes til, er det igjen enda færre som aktiverer den, da dette er noe brukeren må gjøre selv. Undersøkelse utført i 2008 viser at i underkant av 10 % av det totale antallet PC-er solgt med en TPM brikke, har en aktivert brikke. Hvis TPM brikken kan tilby forbedret sikkerhet, så er dette noe alle burde benytte seg av. På bakgrunn av dette har TCG startet en kampanje under mottoet «Turn it on!».

En annen utfordring er at TPM brikken i seg selv ikke kan tilby noe funksjonalitet til bruker uten spesiallaget programvare. Dette betyr at med mindre programvareutviklere benytter seg av TPM brikken og implementerer løsninger basert på den, så hjelper det lite om brukerne har aktivert brikken. BitLocker som ble lansert som en del av Windows Vista operativsystemet, er det mest kjente eksemplet på programvare som benytter seg av en TPM brikke. Stort sett alle PC-er solgt i dag leveres med produsentutviklet programvare, som kan ta i bruk basis funksjonaliteten i TPM brikken.

TCG sitt nyeste prosjekt går under navnet Mobile Trusted Module(MTM). Dette er en modifisert utgave av TPM brikken tilpasset håndholdte enheter og blir å finne i blant annet mobiltelefoner om noen år.

2.3 SETTE OPP BRIKKEN FOR BRUK

Før man kan begynne å utnytte funksjonaliteten TPM brikken tilbyr, må TPM brikken initialiseres. I Vista og kommende Windows 7 følger det med en applikasjon som kan brukes til dette. Denne applikasjonen startes fra run(Windows tasten + r) ved å skrive ”tpm.msc”.



Figur 1. Programvare som følger med Vista og Windows 7 for initialisering av TPM brikken.

Hvis ikke brikken har vært tatt i bruk tidligere, så vil man få opp et vindu tilsvarende det i figur 1. Applikasjonen krever først at PC-en slås av og så på igjen før man kan gjøre noe mer. Dette er en av sikkerhetsmekanismene innebygd i TPM brikken. Det er her brikken sjekker for fysisk tilstedeværelse. Dette gjøres ved å be bruker om et tastetrykk etter at TPM delen av BIOS er startet opp, men før hele BIOS er lastet inn. Dette gjøres for å sikre at eierskap av brikken ikke kan oppnås eksternt. Fysisk tilstedeværelse trengs også ved tilbakestilling. Etter oppstart må brukeren sette eier passordet til brikken. Brikken vil nå være aktivert og klar for bruk.

Utover initialisering av TPM brikken, kan denne applikasjonen brukes til å begrense kommando repertoaret til TPM brikken og tilbakestille TPM brikken til ukjent status igjen. For å ta i bruk annen funksjonalitet er det nødvendig med tredjeparts programvare.

Det som er viktig å merke seg er at alle som har fysisk tilgang til maskinen nå, kan uføre en tilbakestilling av brikken eller deaktivere brikken. Tilbakestilling vil medføre at all data lagret i brikken vil bli slettet og at TPM brikken må initialiseres på nytt. Kjenner man til eierpassordet, kan man utføre tilbakestilling og deaktivering av brikken uten fysisk tilgang til PC-en. Hvis eierpassordet blir ”borte”, vil administrasjons mulighet av brikken gå tapt(se kapittel: 2.5.4). Hvis eier og bruker ikke er en og

samme person, er det anbefalt å sette vise restriksjoner på tilbakestilling. Dette kan bli gjort ved hjelp av UAC(User Account Control) i Vista/Windows 7, som også kan brukes til mer enn å begrense rettigheter på bruk av TPM brikken.

For å utnytte mer av funksjonaliteten, er man avhengig av tredjeparts programvare. Foruten programvare som følger med fra leverandør ved kjøp av en ny PC med TPM, leverer blant annet Wave Systems, VeriSign, Phoenix Technologies og Softex programvareløsninger.

2.4 TPM ARKITEKTUREN

TPM brikkene produseres i dag av Atmel, Infineon, Broadcom, Sinosun, STMicroelectronics og Winbond. Krav som stilles til disse selskapene er at brikken som et minimum oppfyller de obligatoriske kriteriene som er spesifisert i den tredelte spesifikasjonen utgitt av TCG. Denne består av: Del 1: Designprinsipper[1], Del 2: TPM strukturer[4] og Del 3: TPM kommandoer[5]. Totalt fyller spesifikasjonene over 800 sider. Del 1 er den mest relevante når det kommer til arkitekturen. På noen av kriteriene er spesifikasjonen veldig konkret på hvordan løsningen skal implementeres, mens på andre kriterier igjen har produsent mer valgfrihet i henhold til implementering av løsningen. Det er også spesifisert løsninger som produsent kan velge å implementere, men som ikke er påkrevd.

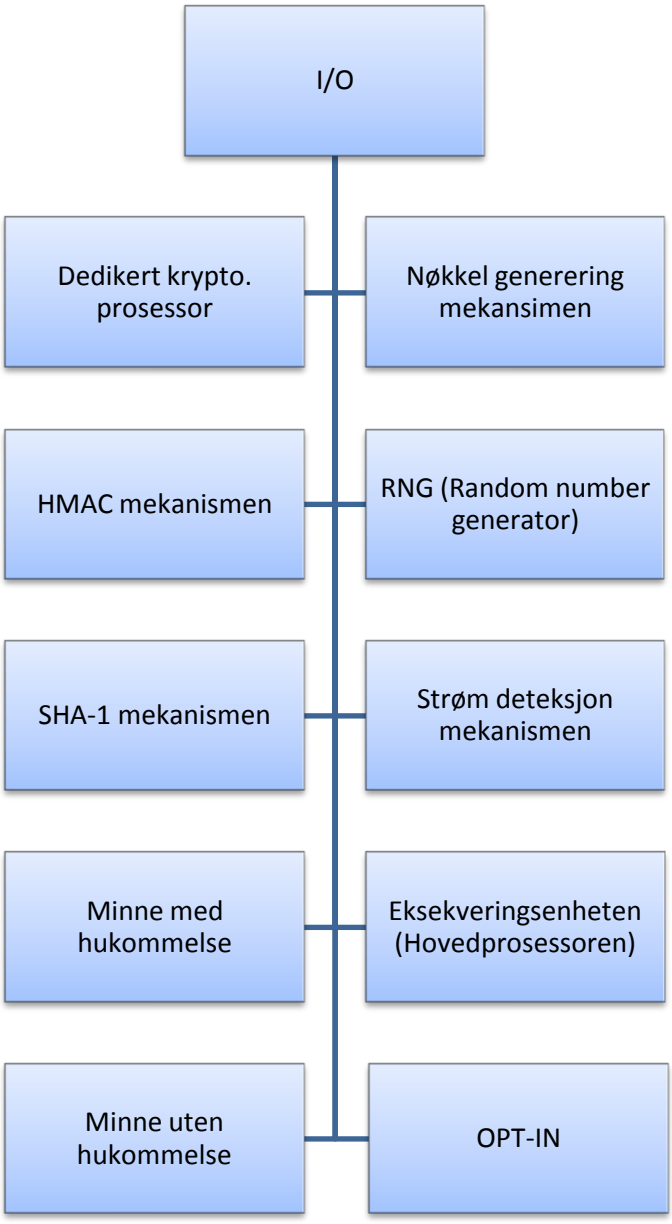
Produsentene står fritt når det kommer til valg av kommunikasjons grensesnitt og bus arkitektur. Til tross, så har LPC(Low Pin Count) bussen blitt å foretrekke av produsentene. Dette da denne bussen tilfredsstiller hastigheten som er nødvendig og gjør det lett å intrigere TPM brikken i plattformen, med tanke på kommunikasjon med blant annet BIOS.

Det stilles krav til at produsentene utstyrer brikkene med beskyttelse mot fysisk modifisering/tukling og beskyttelse mot ordbokangrep.

Figur 2. TPM brikken. Hentet fra: www.infineon.com.

2.4.1 KOMPONENTENE

En mikrokontroller består av en eller flere prosessorer, minne og diverse periferienheter. TPM brikken er en mikrokontroller og består derfor av komponenter i disse tre kategoriene. Figur 3 gir en oversikt over hovedkomponentene som utgjør TPM brikken(mikrokontrolleren). Under følger en enkel beskrivelse av hver enkelt komponent. Pågrunn av målsetningen om lav produksjonskostnad, ble en del komponenter som kunne vært ønskelig, men ikke nødvendig å inkludere, utelatt.



Figur 3. Hovedkomponentene i en TPM brikke. Hentet og modifisert fra: TCG spesifikasjonen del 1.

- **I/O (Input/Output)**
  Håndterer kommunikasjonen inn til og ut av TPM brikken. Denne komponenten dekoder/koder informasjonen den mottar og sender den videre internt/eksternt over databussen til riktig komponent. I/O komponenten fungerer også som tilgangskontroll og kan nekte informasjon å passere hvis fastsatte krav ikke er oppfylt.
- **Dedikert krypto. prosessor**
  Denne prosessoren er som navnet tilsier, dedikert til å utføre kryptografiske operasjoner. Den gjør operasjoner på data etter ønske fra de andre komponentene(SHA-1, RSA, HMAC, RNG).
- **Nøkkel generering mekanismen**
  Denne komponenten håndterer generering av asymmetriske nøkler, signering av data, asymmetrisk kryptering og asymmetrisk dekryptering. Den håndterer også symmetriske kryptering og dekryptering. Denne mekanismen bruker tall fra RNG komponenten i sine operasjoner.
- **HMAC(Hash Message Authentication Code) mekanismen**
  Denne komponenten lager MAC-verdier. MAC-verdier beregnes ved hjelp av en algoritme, som tar en nøkkel og en melding som input. MAC verdier er verdier som vil endre seg hvis innholdet i meldingen er endret og/eller hvis nøkkelen er endret. MAC-verdier brukes til å verifisere dataintegritet(at ingen modifikasjoner er gjort) og opphavet til meldinger(har autorisasjon). Eneste måten å få verifisert MAC-verdien, er ved å kjenne til nøkkelen som ble brukt. Denne komponenten bruker tall fra RNG komponenten og resultatet av SHA-1 operasjoner i sine operasjoner.
- **RNG(Random Number Generator)**
  Denne komponenten lager nonce (number used once). Den skal basere seg på diverse støy og lage så tilfeldige numre som i det hele tatt mulig. Det vil si numre som ikke følger noe bestemt mønster i forhold til hverandre. De fleste TPM brikkene i dag er utstyrt med en såkalt TRNG(True Random Number Generator) som er en forbedret utgave av RNG.
- **SHA(Secure Hash Algorithm)-1 mekanisme**
  SHA-1 er en algoritme som tar x-antall bit som input og lager en unik hash-verdi som resultat. Endring i en av bitene inn til algoritmen, vil resultere i en helt annen hash-verdi. Uansett lengde på data som blir kjørt igjennom SHA-1 algoritmen, vil resultatet alltid være en verdi på 160 bit(20 byte). En hash-verdi er altså en verdi som gjenspeiler en unik sammensetting av bit.
- **Strøm deteksjon**
  Overvåker strømtilstanden til plattformen TPM brikken er en del av. Denne komponenten ser også på ressursbruken av systemet i helhet og varsler hvis systemet er overbelastet. Sender statusrapporter til OPT-IN komponenten. Brukes blant annet for å oppdage forsøk på fysisk tukling.
- **OPT-IN**
  Denne komponenten er styringskomponenten i denne mikrokontrolleren. Den styrer modusene til TPM brikken(på/av, aktivert/deaktivert, påslått/avslått) og opprettholder tilstanden til data, samt håndterer flaggbit.

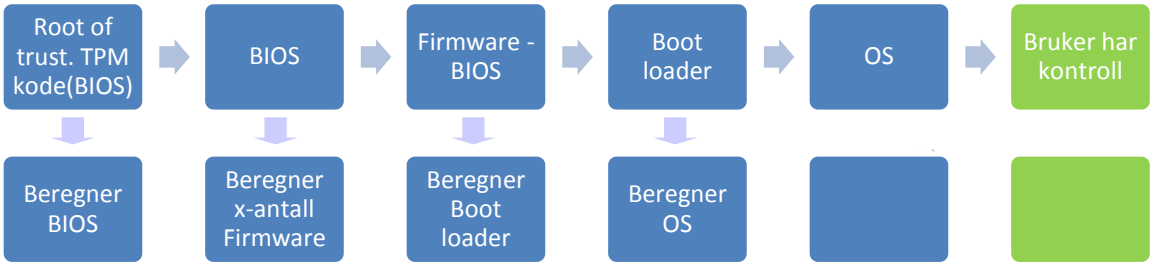
- **Eksekveringsenheten**
  Kjører programkode for og eksekverer alle TPM kommandoer som blir videresendt av I/O komponenten.
- **Minne med og uten hukommelse**
  Lagringsplass for midlertidig og permanent data.

2.5 TPM FUNKSJONALITET

TPM brikken tilfører i seg selv ikke så veldig mye ny funksjonalitet inn i datasikkerhetsverden. Det den derimot tilfører er et lavere nivå hvor funksjonaliteten kan bli tatt i bruk. Det som vanligvis blir gjort i software, er nå mulig å få gjort i hardware. Dette åpner opp for en del nye muligheter, som blant annet tiltrodd oppstart. I tillegg gir TPM brikken muligheten til å lagre informasjon på et sikrere sted. Det vil si i TPM brikken, istedenfor på en harddisk.

2.5.1 PLATTFORM INTEGRITET OG AUTENTISERING

Hver gang en PC med en aktivert TPM brikke starter opp, blir det utført en såkalt tiltrodd oppstart. Se figur 4 for en grafisk fremstilling. En tiltrodd oppstart, er en sekvensiell beregning, som har som mål å finne ut om tilstanden til maskinen er endret siden sist gang den ble startet. For å kunne oppnå en tiltrodd oppstart er det essensielt at starten på sekvensen av beregninger har full tillit og alltid vil ha det. Det er derfor utviklet en forbedret BIOS som må være tatt i bruk for å kunne utnytte all funksjonaliteten TPM brikken tilbyr. Det som er spesielt med denne BIOS-en er at istedenfor at hele BIOS blir startet med engang, er det kun den delen av BIOS som har med TPM brikken å gjøre som blir startet.



Figur 4. Sekvensen av beregninger i en tiltrodd oppstart.

Det første som skjer i en tiltrodd oppstart av en PC, er at TPM delen av BIOS gjør en beregning av resten av BIOS. Denne beregningen som kan bestå av x-antall bit, blir så kjørt gjennom en hash-funksjon(SHA-1) og resultatet av denne funksjonen blir lagt på en PCR(Platform Configuration Register). En PCR, er spesiell på den måten at man ikke kan legge en verdi direkte inn i registeret, eller slette data som ligger der fra før. Registeret vil inneholde følgende: SHA-1(PCR verdi + Beregning). PCR-verdien vil alltid være 0 i de første 16 registrene ved oppstart. Kontroll vil så bli overført til resten av BIOS, som utfører samme beregning på diverse firmware også videre. PCR-verdiene vil sekvensielt bli sjekket mot tidligere lagrede verdier. Stemmer alle verdiene overens, har man oppnådd en såkalt tiltrodd oppstart. Dette fungerer fordi beregningene blir utført av neste sekvensielle enhet før den i det hele tatt tillattes å eksekvere kode. Dette vil ikke forhindre ondsinnet kode fra å kjøre på systemet, men det vil nekte ondsinnet kode tilgang til sensitiv data lagret i TPM brikken. Dette da

TPM brikken kan låse nøkler til resultatet av en beregning lagt på en PCR. Med andre ord man låser nøkler til en tidligere kjent sikker system status. Hvis PCR-verdien ikke tilsvarer forventet verdi er ikke systemet sikkert og nøkkelen vil ikke bli frigitt.

TPM brikken benytter seg av 8 PCR-er(0-7) under beregningen av oppstarten. Utover disse 8 har alle TPM brikker minst 8 PCR-er til(8-15). Dette betyr at det er mulig å utføre beregninger av for eksempel diverse applikasjoner før de starter opp for første gang. Dette vil følge samme prinsipp som ved oppstart.

Metoden beskrevet over er såkalt statisk. Det vil si at den garanterer tilstanden til systemet og programmer kun under oppstart og ikke under kjøretid. Dette betyr at sårbarheter som blir utnyttet i kjørende programmer, ikke vil bli oppdaget. Den mest vanlige sårbarheten her er manglende inntastings sjekking i kode, som blir utnyttet av såkalte ”buffer overflow” angrep. TCG har derfor annonsert dynamisk tiltrodd oppstart som en arvtager til statisk tiltrodd oppstart.

Dynamisk tiltrodd oppstart er fortsatt i den teoretiske fasen og ikke dokumentert som en standard enda. Mye tyder på at dynamisk tiltrodd oppstart kun vil være mulig å realisere ved bruk av den nye teknologien som AMD og Intel kan tilby i noen av sine nye prosessorer. Teknologien er kalt SVM(Secure Virtual Machine) av AMD og TXT(Trusted Execution Technology) av Intel. Vi vil ikke gå noe nærmere innpå dette i denne rapporten.

2.5.2 NØKLER I TPM BRIKKEN

TPM brikken operer med flyttbare, ikke-flyttbare nøkler og sertifiserbare flyttbare nøkler. Ikke-flyttbare nøkler, er nøkler som er generert i TPM brikken og som ikke kan hentes ut under noen omstendigheter. Flyttbare nøkler derimot, kan eksporteres ut av brikken. Denne statusen settes når nøklene blir generert og er ikke mulig å forandre i etterkant. Nøkler laget i en TPM brikke, som blir importert inn til en annen TPM brikke, vil derfor aldri bli ansett som ikke-flyttbare. Sertifiserbare flyttbare nøkler derimot, er en blanding av de to andre nøkkel typene, som ble innført i TPM spesifikasjonen versjon 1.2. Disse nøklene blir sett på som ikke-flyttbare, men er mulig å flytte ut av TPM brikken under strengt definerte forutsetninger.

TPM brikken tilatter ikke at det samme nøkkelparet blir brukt til forskjellige operasjoner. Eksempel på dette er signeringsoperasjoner og krypteringsoperasjoner. Under signeringsoperasjoner brukes den private nøkkelen, mens under krypteringsoperasjoner brukes den offentlig. Det blir derfor vurdert som en sikkerhetsrisiko å tillate bruk av samme nøkkelpar til å utføre begge operasjonene. Hvilken operasjon nøkkelparet skal brukes til blir satt ved generering og er heller ikke mulig å forandre i etterkant. Bruker genererte nøkler faller inn under en av disse fire kategoriene: Lagringsnøkler, bindingsnøkler, signaturnøkler eller identitetsnøkler.

Ved generering av nye nøkkelpar, har brukeren muligheten til å sette et passord(authData) for bruk og et passord for eksportering. Dette passordet (hvis satt) vil bli krevd hver gang nøkkelparet blir tatt i bruk, eller nøkkelparet ønskes eksportert.

Fra produsent kommer alle TPM brikkene med et allerede innlagt RSA 2048 bit generert nøkkel par. Det er dette nøkkelparet som blir referert til som EK(Endorsment Key) og er et ikke-flyttbart nøkkel-

par. Dette nøkkelparet vil alltid være lagret i TPM brikken sitt minne med hukommelse, også etter tilbakestilling. Det er dette nøkkelparet som gjør TPM brikkene unike. Dette nøkkelparet er i tillegg signert av produsent i form av et sertifikat som også er lagret i brikken. Produsent sitt sertifikat er igjen signert av en tiltrodd tredjepart/CA. Ved hjelp av sertifikatet kan man påvise ekteheten til brikken og ved hjelp av EK kan flere TPM brikker og plattformer skilles fra hverandre. EK blir brukt en gang til å generere SRK(Storage Root Key) for å sikre unikheten, ellers er det kun i forbindelse med identitetsnøkler den blir tatt i bruk.

Den første nøkkelen som bruker generer er SRK. Denne nøkkelen blir generert automatisk som en del av prosessen for å ta eierskap av TPM brikken. Dette er også et RSA 2048 bit nøkkelpar. Nøkkelparet er ikke-flyttbart og er et krypteringsnøkkelpar som ligger under kategorien lagringsnøkler. SRK vil også alltid ligge i TPM brikken sitt minne med hukommelse og ikke være mulig å hente ut av TPM brikken.

LAGRINGSNØKLER

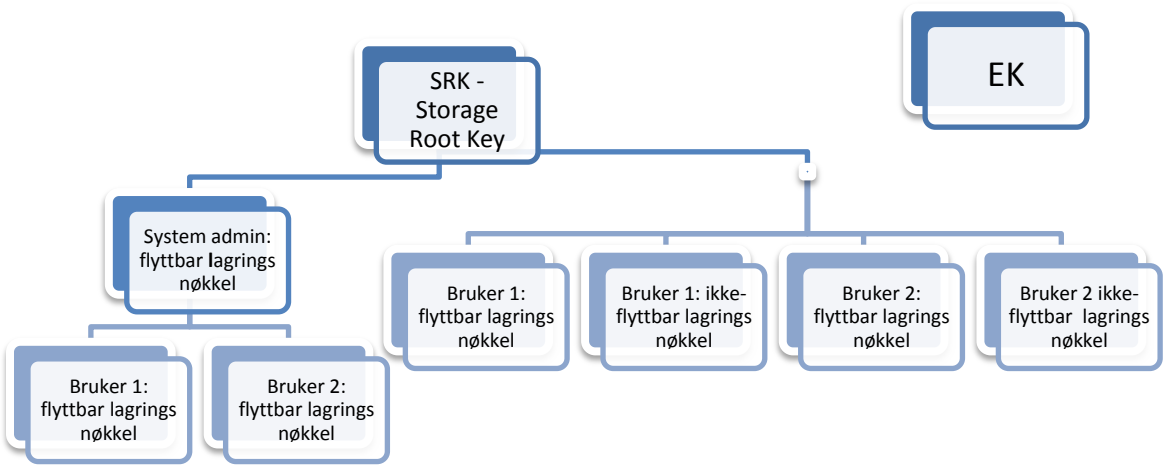
SRK nøkkelparet danner toppen av hierarkiet for alle andre nøkler generert senere. Dette betyr i praksis at hvis denne nøkkelen blir slettet, mister man tilgang til alle de andre nøklene også. Dette er hva som skjer ved tilbakestilling av TPM brikken.

Autorisasjon kan settes på alle lagrings nøkler nedover i hierarkiet, akkurat som på alle andre bruker genererte nøkler. Dette er en mulighet som er implementert for å gi fleksibilitet med tanke på flere brukere, se figur 5. Man kan begrense tilgang og tilpasse bruk til grupper og enkelt personer. For å bruke nøkler som ligger lavere i hierarkiet er det derfor nødvendig å kunne oppgi autorisasjon på lagrings nøkler som ligger over i hierarkiet, hvis autorisasjon er påkrevd.

Det er ikke satt noen begrensning på hvor mange lagrings nøkler det er mulig å opprette. Siden det er begrenset lagringsplass i TPM brikken, vil derfor lagringsnøkler ofte blir lagret utenfor brikken. Dette vil være sikkert da SRK alltid vil befinne seg i brikken og denne trengs for og dekryptere lagrings nøkler under i hierarkiet. Dekryptering vil kun foregå inne i TPM brikken. TPM brikken håndterer selv hvilke nøkler den til enhver tid trenger å ha dekryptert i minne for og dekryptere neste lagringsnøkkel. Lagringsnøkler er kryptert med den offentlige nøkkelen til lagringsnøkkelen over i hierarkiet og dekrypteres derfor med den private nøkkelen til lagringsnøkkelen over i hierarkiet.

Merk at alle flyttbare nøkler vil bli eksportert uten autorisasjon, hvis disse ligger under en annen flyttbar nøkkel som blir eksportert.

Det er altså hierarkiet av lagringsnøklene som gir fleksibiliteten til å møte de fleste løsningene en bedrift ønsker. Figur 5 viser et typisk oppsett.



Figur 5. Eksempel på organisering av den hierarkiske oppbygningen av nøkler i TPM brikken. Hentet og modifisert fra:[3].

I dette oppsettet vil alle nøkler som ligger under system administratoren sin flyttbare lagringsnøkkel kunne tas enkel sikkerhetskopi av. Som nevnt over, vil alt som ligger under en flyttbar lagringsnøkkel bli kopiert. SRK er som nevnt ikke eid av noen, så nøklene som ligger direkte under den er kun tilgjengelig for bruk og sikkerts kopiering av den respektive brukeren. Man kan som sagt lage så mange lagringsnøkler man vil. I en privat hjemme PC holder det som regel med en flyttbar og en ikke-flyttbar lagringsnøkkel. For en bærbar PC i en bedrift er brukere som regel begrenset. På en server derimot kunne det vært aktuelt å opprette x-antall lagringsnøkler. Istedenfor en bruker er det selvfølgelig også mulig å bruke grupper, hvor flere kjenner til autorisasjonshemmeligheten og deler nøkler seg imellom.

BINDINGSNØKLER

Bindingsnøkler er RSA 2048 bit nøkkelpar og utfører kun standard RSA operasjon. Akkurat som lagringsnøkler er disse nøklene brukt til kryptering, med den forskjellen er at disse nøklene ikke kan brukes til å kryptere andre RSA nøkler. Bindingsnøkler er primært flyttbare og brukes til lagring av symmetriske nøkler og til bindingsoperasjoner.

SIGNATURNØKLER

Signaturnøkler er som navnet tilsier brukt til signeringsoperasjoner. Signaturnøkler er RSA nøkler med en lengde på opptil 2048 bit. Disse nøklene er typisk lagret som flyttbare nøkler. Signering av dokumenter, meldinger, E-Mail og lignende er ofte ønskelig å få gjort fra flere PC-er. På den andre siden kan man ved hjelp av signeringsnøkler generert som ikke-flyttbare i tillegg til å bekrefte identiteten også bekrefte plattformen(se: Identitetsnøkler). Det er her fleksibiliteten kommer inn og hva bedriften finner som mest hensiktsmessig som bestemmer.

IDENTITETSNØKLER

Identitetsnøkler også kalt AIK(Attestation Identity Key(s)), vil alltid være ikke-flyttbare RSA 2048 bit nøkkelpar. AIK kan kun bli generert av eier av TPM brikken og kan kun brukes ved godkjent autorisasjon. AIK er signeringsnøkler brukt til å signere informasjon for å kunne verifisere tilhørigheten og integriteten. Eneste grunnen til at AIK ble opprettet i TPM spesifikasjonen og implementert i TPM brikken, var for å beskytte privatlivet til bruker(ene) av TPM brikken. Det AIK gjør eksisterer allerede i form av EK med tilhørende sertifikat. Problemet med EK er at det alltid vil finnes kun en i TPM brikken. Identitetsnøkler derimot kan man lage så mange man vil av. Se på AIK som muligheten til å bekrefte samme identitet, uten å bruke samme ID-kort. Dette kan for eksempel være aktuelt for å skille mellom jobb- og privatbruk av TPM brikken. Prosessen rundt generering av en AIK er avhengig av en tiltrodd tredjepart for å opprette et sertifikat.

AIK signerer kun informasjon generert internt i TPM brikken og har derfor følgende bruksområder:

Det første bruksområde er å signere ikke-flyttbare nøkler. Ikke-flyttbare nøkler signeres av en AIK for å verifisere integriteten. Med andre ord at nøkkelen er generert i en bestemt TPM, som igjen sitter i en bestemt maskin og ikke kan bli eksportert ut. Er det kun en bruker som har autorisasjon til den signerte ikke-flyttbare nøkkelen, kan man ved hjelp av AIK identifisere en bestemt bruker. Det samme gjelder hvis kun en bruker har autorisasjon til å bruke en bestemt AIK.

Det andre bruksområde for disse nøklene er å signere PCR verdier. Sitter man på en klient og ønsker å sjekke at oppstarten på en server har gått riktig for seg, kan man sende en etterspørsel til serveren sammen med et unikt tall. Serveren vil da signere PCR verdiene og det unike tallet med en AIK. Det unike tallet sendes med for å forsikre seg om at PCR verdiene er de nyeste og ikke tidligere verdier. Akkurat som over, vil AIK bekrefte tilhørigheten og integriteten til PCR verdiene.

2.5.3 SERTIFIKATER I TPM BRIKKEN

TPM brikken kommer med tre allerede innlagte sertifikater. Det første er allerede nevnt. Detter er EK sertifikatet som blir lagt inn av produsent av TPM brikken og verifiserer ekteheten til brikken. Det andre sertifikatet er Plattform sertifikatet. Dette sertifikatet blir lagt inn av PC/hovedkort produsent og verifiserer at en ekte TPM brikke har blitt montert på en korrekt plattform. Dette sertifikatet blir i likhet med EK sertifikatet brukt i forbindelse med identitetsnøkler. Det siste sertifikatet er kalt Conformance sertifikatet. Dette er et sertifikat som er lagt inn av en test lab og verifisere at TPM brikken er produsert på en tilfredsstillende måte og fungerer som den skal.

Utover disse tre sertifikatene, er det mulig å legge inn et sertifikat for hver enkelt hardware komponent som vil verifisere at denne komponenten har blitt montert på en korrekt plattform. Akkurat som EK sertifikatet, men litt mindre relevante.

Alle AIK nøkler vil ha et tilhørende sertifikat utestedet av en lokal eller ekstern TPM klar CA.

2.5.4 SIKKERHETSKOPIERING

Akkurat som all annen data, kan data som ligger lagret i TPM brikken gå tapt. Dette som en følge av en teknisk svikt på hovedkortet, at selve brikken blir defekt, eller rett og slett tilbakestilling av brikken ved en feiltakelse. I utgangspunktet er ødeleggelse av en TPM brikke det verste som kan skje, men svikt på komponenter som er en del av den tiltrodde oppstarten vil være like kritisk. Det er også viktig å merke seg at tapte passord som kreves for å autorisere bruk, ikke kan hentes tilbake eller tilbakestilles. Av disse passordene er selvfølgelig selve eier passordet det viktigste. Det kan derfor være smart å lagre dette passordet på en ekstern enhet.

Harddisken er den komponenten som blir sjekket i forbindelse med den tiltrodde oppstarten som svikter oftest. Skulle harddisken svikte og man allerede har speilet innholdet over på en annen disk, vil ikke dette medføre noe problem. Hovedkortet derimot kan ikke byttes ut med et annet kort av samme type. Kan ikke det opprinelige hovedkortet repareres er, også all data som det ikke har blitt tatt sikkerhetskopi av tapt.

I forbindelse med nøkler, ble det nevnt at alle flyttbare nøkler kan eksporteres ut av brikken. Ved å eksportere den flyttbare nøkkelen høyest i hierarkiet, vil automatisk alle de andre flyttbare nøklene under i hierarkiet bli eksportert. Dette er den primære sikkerhetskopi mekanismen i TPM brikken.

Det tilbys programvare som tar seg av denne sikkerhetskopieringen på en enkel og sikker måte. Programvaren krypterer all data lagret under flyttbare lagringsnøkler i TPM brikken og lagrer denne krypterte dataen til eier sin selvvalgte destinasjon. Programmet krypterer så dekrypteringsnøkkelen under et valgt passord av eier og lagrer den på en annen selvvalgt destinasjon. Ved hjelp av den krypterte dataen og dekrypteringsnøkkelen, kan programvaren gjenopprette data i den opprinelige TPM brikken, eller legge den i en ny TPM brikke.

Ikke-flyttbare nøkler derimot er en annen sak. Flyttbare nøkler er ment å kunne brukes i andre TPM brikker, men ikke-flyttbare nøkler er kun ment for bruk i en unik TPM. TCG har spesifisert en mulighet for å ta sikkerhetskopi av ikke-flyttbare nøkler. Det er derimot ingen av TPM brikke produsentene som har implementert denne muligheten, da det er valgfritt. Løsningen TCG har spesifisert går ut på at SRK nøkkelen blir kryptert med bruker sin offentlige nøkkel(ny flyttbar RSA nøkkel), for så å bli kryptert med produsent sin offentlige nøkkel. Den krypterte SRK nøkkelen kan så hentes ut av TPM brikken. Dette vil sikre at ingen har tilgang til SRK når den er utenfor TPM brikken. Alle de andre ikke-flyttbare nøklene er avhengig av SRK for å kunne dekrypteres og kan derfor også eksporteres ut. Etter å ha gitt produsent bevis på at den opprinelige TPM brikken er defekt, vil produsent dekryptere første del av den krypterte SRK nøkkelen, for så å kryptere den på nytt med den nye TPM brikken sin SRK nøkkel. Den fortsatt dobbelt krypterte SRK nøkkelen blir så sendt inn til den nye TPM brikken, hvor den nye SRK nøkkelen blir erstattet med den gamle. Alle de gamle ikke-flyttbare nøklene vil nå kunne dekrypteres og kunne tas i bruk igjen.

Kort oppsummert, utarbeid retningslinjer og tiltak for bedriften før TPM brikken blir tatt i bruk. Kun flyttbare nøkler kan bli sikkerhetskopiert og det er ikke mulig å hente ut passord eller endre passord, hvis det opprinelige passordet har gått tapt.

2.6 BRUKERAUTENTISERING

Dette avsnittet vil ta for seg hvordan en TPM brikke som en del av en plattform, kan brukes for å gjennomføre brukerautentisering mot en sentral webbasert tjeneste. I dette avsnittet vil også sikkerheten rundt webautentisering og hvordan TPM brikken kan bidra bli diskutert.

2.6.1 DEFINISJON

Brukerautentisering er det første som skjer når man etablerer en kommunikasjonskanal. Denne kommunikasjonskanalen kan opprettes over et lukket nettverk(typisk intranett), eller over et åpent nettverk(internett). Hensikten med brukerautentisering er å verifisere en bruker sin selvoppgitte identitet. Dette er viktig da identiteten igjen er tilknyttet rettigheter. Brukerautentisering må ikke forveksles med dataautentisering.

De tre fasene i en brukerautentiseringsprosess er:

1. **Identifikasjon:** Brukeren identifiserer seg selv.
2. **Autentisering:** Brukeren må bekrefte sin identitet. Brukeren kan bekrefte dette ved hjelp av noe han har, noe han er eller noe han vet.
3. **Autorisasjon:** Brukeren blir tildelt rettigheter

Ved å benytte seg av flere autentiseringsmåter, vil man kunne oppnå en sikrere brukerautentiseringsløsning.

2.6.2 VURDERING AV SIKKERHETSASPEKTENE

Den mest vanlige formen for brukerautentisering i dag foregår ved å oppgi et brukernavn og det tilhørende passordet. Dette foregår i tillegg ofte over en ukryptert kommunikasjonskanal. For noen webbaserte tjenester er dette en tilstrekkelig brukerautentiseringsløsning. For webbaserte tjenester som tilbyr tilgang til sensitiv informasjon, er dette langt ifra en tilstrekkelig brukerautentiseringsløsning. Problemet med en ukryptert kommunikasjonskanal, er at hvilken som helst person med tilstrekkelige datakunnskaper kan lytte på kanalen og se ”samtalen” i klar tekst, dette er på engelsk kalt eavesdropping. Dette unngås ved å kryptere kommunikasjonskanalen. For å kunne kryptere kanalen, må klient og server utveksle krypteringsnøkler. Under denne utvekslingen kan nøklene bli plukket opp av en tredjepart og man kan bli utsatt for ett mann i midten angrep(engelsk: man-in-the-middel). For å unngå mann i midten angrep, benytter man seg av PKI(Public Key Infrastructure). PKI baserer seg på en tiltrodd tredjepart, som går god for krypteringsnøklenes tilhørighet og man kan derfor være sikker på at man har mottatt de riktige nøklene.

De to angrepene over, er angrep på konfidensialiteten og integriteten på kommunikasjonskanalen. Utfordringen i dag ligger ikke her, da kryptering av kanalen og bruk av PKI er en god løsning. Utfordringen er elektronisk tyveri av innlogingsopplysninger i forkant av etableringen av kommunikasjonskanalen, noe som kan føre til identitetssvindel over kommunikasjonskanalen.

Det elektroniske tyveriet kan utføres ved å:

1. Komme seg inn på brukeren sin PC og hente ut opplysningene.
2. Lure brukeren til å utlevere opplysningene.

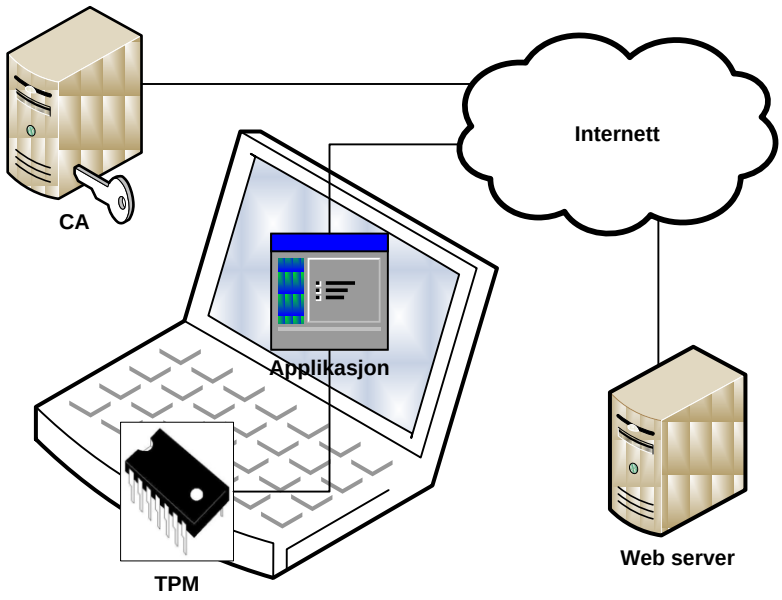
En angriper kan oppnå alternativ 1 ved for eksempel å installere ondsinnet programvare(engelsk: malware) på brukeren sin PC. Dette kan være programmer som logger tastaturtrykk eller tar skjermbilder, for så å sende dette tilbake til angriperen. Felles for alternativ 1, er at angriperen får tilgang til opplysningene ved å bryte integriteten til brukeren sitt system.

Den mest vanlige måten en angriper utfører alternativ 2 på, er ved å utgi seg for å være noe han ikke er og vente på at noen tror han, dette er på engelsk kalt phishing. Dette gjøres for eksempel ved å lage tilsynelatende identiske websider og lure brukere inn på disse sidene, dette er på engelsk kalt web spoofing. Brukerne klarer ikke å skille den originale websiden fra den falske og taster inn sine sensitive opplysninger. Opplysninger som angriperen senere bruker for å logge inn på den originale siden. Felles for disse angrepene er at angriperen får tilgang på opplysningene, fordi brukeren ikke har tilstrekkelig datakompetanse, eller er oppmerksom nok.

TPM brikken kan bidra på mange måter for å forbedre sikkerheten i henhold til truslene nevnt over. TPM brikken har som nevnt(se kapittel: 2.5.1) muligheten til å sjekke integriteten til systemet ved hjelp av statisk og etter hvert dynamisk tiltrodd oppstart. TPM brikken kan blant annet binde opplysninger til en kjent sikker plattformtilstand. Hvis ikke tilstanden er kjent, kan bruker nektes tilgang til opplysningene lagret i TPM brikken. Da ikke bruker får tak i opplysningene, vil heller ikke ondsinnet programvare gjøre det. Har en angriper først fått tilgang til en bruker sin PC, for eksempel ved hjelp av en trojaner. Vil opplysningene ligge lagret mye tryggere inni TPM brikken, enn alternativet på en harddisk beskyttet av software. Software vil ikke kunne beskytte seg mot blant annet ordbokangrep(engelsk: dictionary attack), mens TPM brikken har innebygd beskyttelse mot dette. Når det gjelder alternativ 2, kan man forhindre disse tyveriene ved å basere autentiseringen på at en bruker har tilgang til en privat nøkkel. Er dette en ikke-flyttbar nøkkel, vil ikke brukeren kunne utlevere den. Ikke engang en utro tjener ville klart å hente ut denne. En annen måte for å sikre seg mot tyveri er å autentisere plattformen som en del av brukerautentiseringsprosessen. En angriper kan ha alle de andre opplysningene, men vil ikke klare å etterligne plattformen til brukeren. For å ivareta personvernet, oppretter man en identitetsnøkkel til dette formålet.

2.6.3 BRUK AV TPM BRIKKEN I EN AUTENTISERINGSLØSNING MOT WEB

En mulig brukerautentiseringsløsning vi har sett for oss, er skissert i figur 6. Denne løsningen baserer seg på en applikasjon som kommuniserer med en webbasert tjeneste over internett. Denne applikasjonen installeres på klienten og skal i størst mulig grad automatisere autentiseringsprosessen for brukeren. Dette for å holde valgmulighetene og inntastingene fra bruker til et minimum, samt gjøre applikasjonen minst mulig avhengig av bruker sine ferdigheter. Applikasjonen kan for eksempel være et tilleggsprogram for en nettleser. Applikasjonen skal opprette et sertifikat som skal kunne identifisere bruker mot tjenesten, i tillegg til at tjenesten vil identifisere seg ved hjelp av et sertifikat(gjensidig autentisering). Applikasjonen vil derfor måte kommunisere med TPM brikken, i tillegg til å kommunisere med en CA over internett for sertifikatutstedelse.



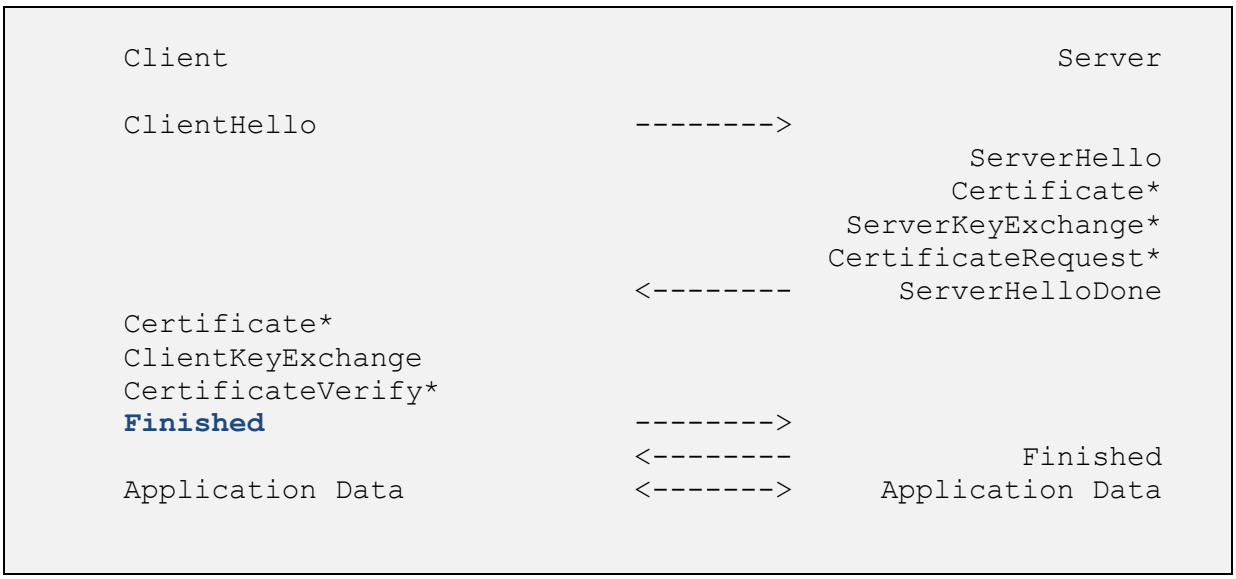
Figur 6. Grafisk fremstilling av løsningen på et overordnet plan.

Dette var den løsningen vi hadde sett for oss tidlig i prosjektet. Det viser seg derimot at det er en del utfordringer forbundet med å realisere denne løsningen. Disse utfordringene har opprinnelse i hvordan TPM brikken er spesifisert. I teksten under vil disse utfordringene komme fram og forutsetningene for realisering vil bli bevistgjort.

GJENNOMGANG AV LØSNINGEN

Se vedlegg F, for en skisse som viser hendelsesforløpet forklart i teksten under.

Ved sikker kommunikasjon mot en webbasert tjeneste, er SSL/TLS[6] den mest brukte protokollen. Denne protokollen tilbyr i tillegg gjensidig autentisering, noe som er ønskelig for løsningen vår. Dette tilbys i "handshake" delen av protokollen, ved etablering av kommunikasjonskanalen.



Figur 7. Hendelsesforløpet i handshake delen av TLS protokollen. Hentet fra: TLS spesifikasjonen.

Hendelsesforløpet til protokollen ved gjensidig autentiseringer er vist i figur 7. Serveren autentiserer seg selv og avlutter med en forespørsel om at klienten gjør det samme. Det er her ønskelig at applikasjonen skal ta over og generer et sertifikat for bruker. Eventuelt hente ut et allerede eksisterende sertifikat.

Eneste måten å generere et sertifikat som kan identifisere TPM brikken, er ved bruk av identitetsnøkler(se kapittel: 2.5.2). Det første som må gjøres er derfor å opprette et nytt AIK nøkkelpar. Prosessen for dette er ikke beskrevet i detalj i den nyeste TCG spesifikasjonen, men i TPM spesifikasjonen 1.1b er den det. Vi har hentet mye av denne forklaringen fra EAP-TPM [7] spesifikasjonen, som er en spesifikasjon under arbeid for hovedsakelig bruk av TPM autentisering over trådløse LAN.

Generering av AIK nøkkelparet gjøres ved hjelp av kommandoen TPM_MakeIdentity. Deretter brukes TSS_CollateIdentityRequest kommandoen for å samle sammen alle opplysningene en CA trenger for og utestede et sertifikat. Disse opplysningene omfatter blant annet alle sertifikatene som allerede ligger i TPM brikken, den offentlige AIK nøkkelen og den offentlige EK nøkkelen(brukes for å kryptere innholdet på vei tilbake). Når dette er gjort, oppretter applikasjonen en forbindelse med en CA og sender inn de innsamlede opplysningene. Denne CA tjenesten, vil være den eneste som vet om forbindelsen mellom EK og AIK(ivaretar personvernet). I dag eksisterer det bare en CA som kan håndtere utstedelse av sertifikat for en AIK. Dette er www.privacyca.com og denne tjenesten er i beta utgave. Sett at denne tjenesten var oppe og fungerte for fullt, ville applikasjonen få tilbake et AIK sertifikat. Applikasjonen må så kjøre kommandoen TPM_ActivateIdentity for å pakke ut og legge inn serti-

fikatet i TPM brikken. Applikasjonen har nå generert et nytt AIK nøkkelpar og fått utested et tilhørende sertifikat. Begge deler er lagret internt i TPM brikken og klare for bruk.

Som nevnt tidligere(se kapittel: 2.5.2), så kan AIK nøkler kun signere data generert internt i TPM brikken og kan derfor ikke brukes for TLS autentisering. Dette da TLS protokollen krever bekreftelse av eierskap av den private nøkkelen. Denne bekreftelsen utveksles over TLS protokollen ved å signere data allerede mottatt fra serversiden.

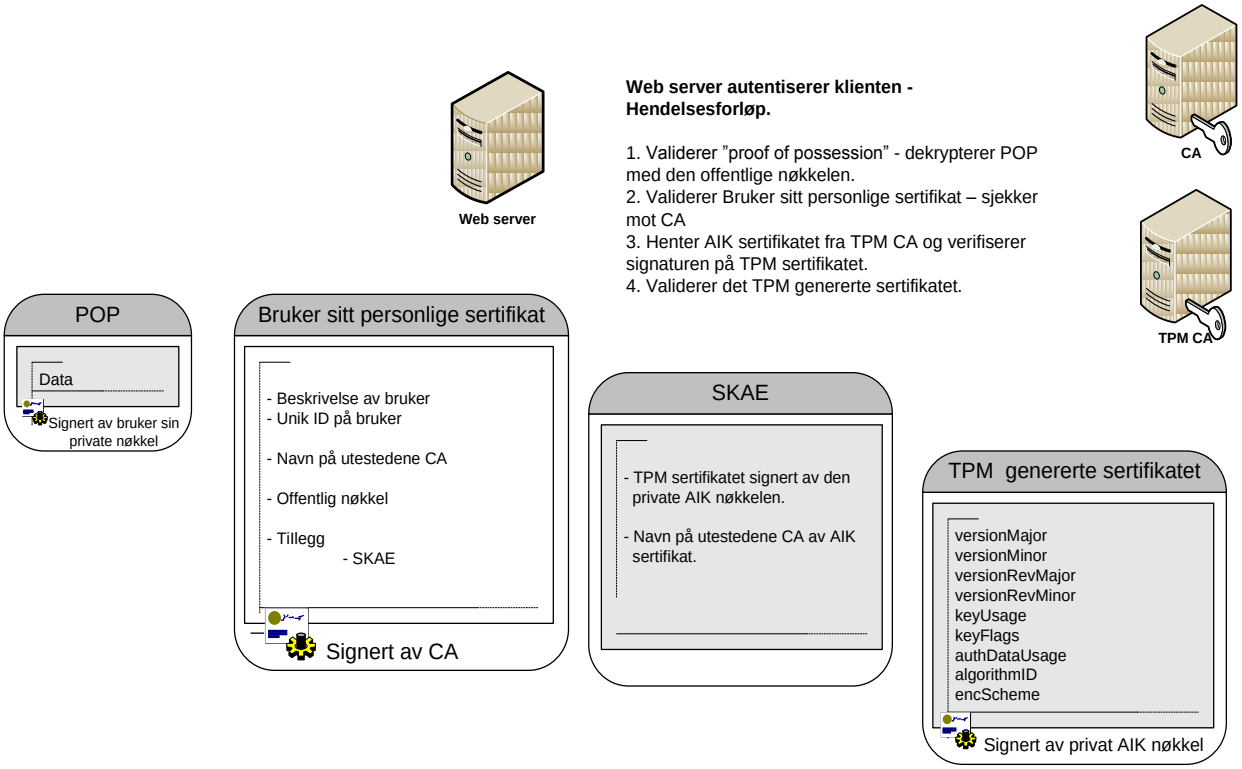
TPM brikken har en mulighet ved hjelp av kommandoen TPM_CertifyKey til å signere ikke-flyttbare nøkler og sertifiserbare ikke-flyttbare nøkler ved hjelp av en AIK. TPM_CertifyKey vil da lage et sertifikat inneholdende den offentlige nøkkelen og signere dette med den private AIK nøkkelen.

Det neste applikasjonen vår må gjøre da, er å opprette et nytt nøkkelpar ved hjelp av TPM_Key_Create kommandoen. Dette nye nøkkelparet må så få utestedt et sertifikat signert av AIK ved hjelp av TPM_CertifyKey. Det nye sertifikatet blir så generert. AIK sertifikatet som ble utested, er et X.509 sertifikat og dette er det eneste formatet TLS protokollen godkjenner. Sertifikatet som blir generert av TPM_CertifyKey er derimot ikke et X.509 sertifikat. TPM_CertifyKey vil resultere i et sertifikat, men altså ikke et som er godkjent for bruk med TLS protokollen. Det nye TPM genererte sertifikatet er lagret i en struktur kalt TPM_CERTIFY_INFO i TPM-en.

Vi trenger nå å få opprettet et X.509 sertifikat for bruk med TLS protokollen. TCG har spesifisert en løsning for dette problemet som går under navnet SKAE(Subject Key Attestation Evidence Extension)[8]. Vi ser også at TCG som et av forslagene [9] for forbedring i den neste TPM spesifikasjonen, har nevnt muligheten til å bruke AIK nøkler til å signere data generert utenfor TPM brikken. Dette vil da eventuelt være en mulighet spesifisert i de kommende spesifikasjonene, det vi har akkurat nå er SKAE.

SKAE baserer seg på muligheten X.509 sertifikat [10] har til å inkludere et tillegg(engelsk: extension). Dette tillegget går under navnet SKAE og vil inneholde sertifikatet som er signert av den private AIK nøkkelen og en referanse til utesteder(CA) av AIK sertifikatet. Ved hjelp av CreateSKAE vil applikasjonen kunne hente ut de nødvendige opplysningene og generere et såkalt SKAE tillegg. Applikasjonen vil så lage et personlig sertifikat forespørsel. Denne forespørselen vil inneholdende den ikke-flyttbare offentlige nøkkelen og SKAE tillegget, i tillegg til annen relevant informasjon. Applikasjonen vil sende dette inn til en hvilken som helst godkjent CA, som vil generere et X.509 sertifikat og legge ved SKAE før det blir signert. Applikasjonen vil motta sertifikatet og legge dette inn i TPM brikken.

Nå endelig vil applikasjonen ha et sertifikat å sende tilbake til webtjenesten, som svar på sertifikat etterspørselen. Applikasjonen vil nå også kunne bruke den private nøkkelen til å sende inn en POP(Proof Of Possession). Altså et bevis på eierskap av den private nøkkelen. Ulempen med SKAE, er at serversiden/webtjenesten må være klar over dette tillegget. Webtjenesten på server siden må vite hvordan SKAE er bygd opp og klare å verifisere informasjonen, noe som krever implementering av ekstra software. Sett at den webbaserte tjenesten har den nødvendige softwaren for å autentisere sertifikatet, så vil det foregå som forklart i figur 8.



Figur 8. Hendelsforløp for autentisering av sertifikat med SKAE tillegg.

Den webbaserte tjenesten kan nå autentisere brukeren på bakgrunn av det personlige sertifikatet. Ved å hente AIK sertifikatet, kan tjenesten autentisere plattformen. Det TPM generert sertifikatet inneholdt den offentlige nøkkelen signert av den private AIK nøkkelen. Siden AIK nøkkelen kun kan signere ikke-flyttbare nøkler generert internt i brikken, kan webtjenesten knytte brukeren til platt-formen. Vi har nå oppnådd en vellykket brukerautentisering mot den sentrale webbaserte tjenesten.

Det forutsettes selvfølgelig at brukeren kan tilknyttes rettigheter. Dette kan for eksempel gjøres ved å knytte den unike ID-en i det personlige sertifikatet opp mot et brukernavn ved registrering. Det er også mulig å benytte andre faktorer som passord for å gjøre autentiseringen enda sikrere. Ved å ge-nerere et sertifiserbart flyttbart nøkkelpar, istedenfor et ikke-flyttbart nøkkelpar, vil det være mulig å flytte nøklene og sertifikatet over på en annen TPM brikke, noe som tilfører fleksibilitet til løsningen.

2.7 OPPSUMMERING AV TEORI

Dette kapitelet startet med å ta for seg bakgrunnen til TPM brikken og hvilken status brikken har i dag. Det kommer tydelig frem at TPM brikken er en teknologi for fremtiden. Den største utfordringen er å informere brukere og utviklere om teknologien, så den blir tatt i bruk. Dette da den allerede er såpass utbredt.

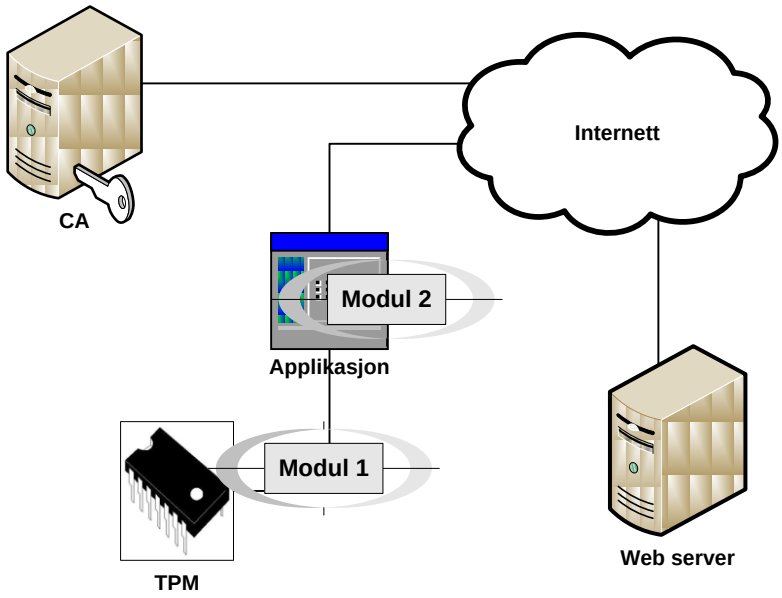
Videre blir arkitekturen nevnt og de fysiske komponentene beskrevet. Utefra arkitekturen, er det enkelt å se at dette er en komponent som kan bli brukt til å utføre det meste av kryptografiske operasjoner. Kryptografiske operasjoner er selve essensen i det å oppnå konfidensialitet, integritet, ikkebenekting og autentisering. Veldig mange av disse operasjonene blir i dag gjort i software. TPM brikken introduserer muligheten til å gjøre dette sikrere i hardware.

I neste avsnitt blir en del funksjonalitet beskrevet. Herunder hvordan TPM brikken forholder seg til nøkler, inkludert hva slags sertifikater den inneholder. Det mest spennende med TPM brikken, er at den har innført en mulighet til å sjekke integriteten til systemet og videre autentisere integriteten. Som tidligere forklart, er dette en statisk sjekk per dags dato og ikke en dynamisk sjekk. Blir dynamisk integritetssjekk en fungerende realitet, vil PC-sikkerheten kunne bli betydelig forbedret. Sikkerhetskopiering er også nevnt, da dette er et viktig aspekt innen datasikkerhet.

Siste avsnittet tar for seg hvordan TPM brikken kan brukes for brukerautentisering mot en sentral webbasert tjeneste. Det kommer her fram at TPM brikken, pågrunn av de strenge kravene til sikkerhet, ikke alltid er like enkel å ha med å gjøre. Utsteding av sertifikat for TPM brikker er ikke en tjeneste som er utbredt blant CA aktører. Det er heller ikke alt som var like gjennomtenkt under uformingen av spesifikasjonen. Dette blir kontinuerlig jobbet med av TCG og løsninger blir sluppet.

3. UTVIKLING

Figur 9, tidligere introdusert i kapittel 2, viser hvordan vi har valgt å dele opp utviklingsdelen av prototypen/demonstratoren i to moduler. Modul 1 er grensesnittet ned mot TPM brikken, mens modul 2 er selve applikasjonen med grensesnitt mot web. Vi kom aldri til Modul 2 i utviklingsdelen, så resten av dette kapitelet vil kun ta for seg modul 1.



Figur 9. Brukerautentiseringsmodellen mot Web, som viser Modulene i dette utviklingsprosjektet.

I modul 1 stod det mellom å videreutvikle et allerede eksisterende grensesnitt, eller å lage et nytt et fra bunnen av. Dette da vi ikke fant noen grensesnitt mot TPM brikken, som var i nærhet av hundre prosent ferdige. Vi valgte det siste, å utvikle grensesnittet fra bunnen av. TPM/J[11] var det mest komplette grensesnittet vi fant. I TPM/J er mye av funksjonaliteten implementert, men TPM/J mangler blant annet implementering av kommandoer som krever fysisk tilstedeværelse og har ikke definert bruk av brukerpassord og lignende. TPM/J er i tillegg et grensesnitt programmert i java. Java er ikke et programmeringsspråk noen av gruppemedlemmene har mye kjennskap til, eller erfaring med. Hvis vi hadde valgt å videreutvikle TPM/J, hadde det gått med mye tid på å sette seg inn i den allerede utviklede koden. Det hadde også gått med mye tid på å sette seg inn i java sin syntaks, finne et egnet utviklingsmiljø og finne ut av hvor forskjellig pakkene ligger i java sitt hierarki. Vi vurderte det derfor dit hen at videreutviklingen av grensesnittet TPM/J, ville ta nesten like mye tid som og programmer et grensesnitt fra bunnen av.

Når dette valget var tatt, så måte vi finne ut hvilket programmeringsspråk vi skulle utvikle modul 1 i. Programmeringsspråket vi valgte ble C#, framfor C/C++. Per Øyvind har erfaring med programmering mot hardware i C# fra før, i tillegg til gode språkkunnskaper. C# gjør i tillegg at vi blant annet slipper å tenke på sprengning av buffere(engelsk: buffer overflows). Sprengning av buffere er en av de vanligste sårbarhetene i programvare i dag. Skulle utviklingen vært gjort i C++ eller C, ville dette vært en av de største utfordringene. Dette siden alle strenger og lignende må sjekkes før de kan bli tatt i bruk. Siden C# og Java har mye til felles, vil en oversetting av C# kode til Java være relativt greit for en Ja-

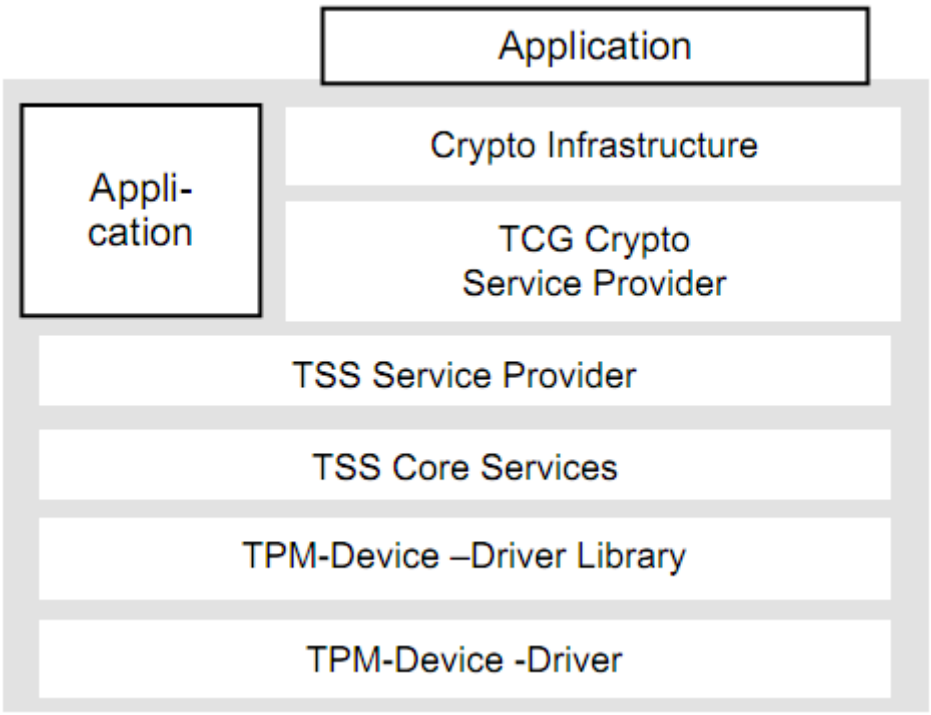
vautvikler. Dette da det er marginale syntaksmessige forskjeller. Buypass utvikler primært i java, så dette var også noe vi tok med i vurderingen når vi valgte C# framfor C/C++.

3.1 KRAVSPESIFIKASJON OG DESIGN

I forbindelse med utviklingen av modul 1, har det ikke blitt utviklet noen direkte kravspesifikasjon eller design dokument. Dette da spesifikasjonene til TCG er ganske omfattende og er laget for dette formålet, forklare hva modul 1 skal gjøre og hvordan modul 1 skal gjøre det.

3.1.1 KOMMENTARER TIL KRAV OG DESIGN

Som vist i figur 10 under, har TCG spesifisert flere lag ned mot TPM modulen. Hvert lag har et gitt abstraksjonsnivå. Desto høyere opp i lagene man kommer, jo mer abstrahert er man fra hardware. Desto høyere opp i abstraksjonslagene man kommer, jo høyere er sannsynligheten for at en applikasjonsprogrammerer kan lage en applikasjon, som kan utnytte potensialet i hardware best mulig. Windows Vista sitt grensesnitt tbs.dll har sådan lab abstraksjon, da denne ikke implementerer noen av TCG strukturene i seg selv, men kun gir et grensesnitt hvor man kan sende og motta bytes til og fra TPM modulen. For å øke abstraksjonene vil det derfor være fornuftig å implementere alle funksjonene til TPM brikken i ett eget grensesnitt, slik at funksjonaliteten blir lettere å få tak i når en applikasjon skal utvikles/designes.



Figur 10. Viser sammenhengen mellom lagene. Hentet fra: TCG spesifikasjonen.

Modul 1 skal ha et grensesnitt mot TPM modulen via tbs.dll. Modul 1 skal også ha ett klassegrensesnitt mot en programvaredel, hvor alle funksjoner spesifisert i del 3 av TCG sin TPM spesifikasjon skal være implementert i en klasse. Alle funksjoner skal være dokumentert slik at når man på ett senere tidspunkt skal benytte disse funksjonene, lett kan forstå hva de gjør og hva de benyttes til.

Siden tbs.dll ligger i TSS Core Services laget i spesifikasjonen, vil grensesnittet mot denne også ligge i dette laget. For så å øke abstraksjonen opp i nivået til applikasjonen skal det i modul 1 også implementere en TSS service provider klasse. Denne klassen velger vi å kalle TpmCommands, da dette er en klasse som er ment å implementere alle TPM kommandoene. Det vil her også legges inn en klasse for å håndtere ofte brukte kryptografiske funksjoner, dermed vil deler av dette grensesnittet løftes opp på TCG Crypto Service Provider laget i lagdelingsdiagrammet. For å kunne tolke dataene som kommer fra TPM modulen, vil det bli laget en klasse hvor alle data returnert fra modulen blir lagt inn. Denne vil på sett og vis ligge som en buffer, hvor man kan hente data forløpende utover i strømmen som blir returnert. Buffer klassen velger vi å kalle ByteArrayReadWriter og er inspirert av klassen med samme navn i TPM/J.

For å kunne tolke strømmene av data returnert fra TPM modulen, vil det bli laget et grensesnitt som alle klasser kan arve fra for å kunne konverteres til og fra rekker av bytes. Alle klasser spesifisert i del 2 av TCG sin TPM spesifikasjon implementerer dette grensesnittet, enten direkte eller indirekte via arv for å forenkle videre utvikling. For å kunne omforme verdityper til rekker av bytes vil det i tillegg bli utviklet en statisk klasse kalt TypeConverter, som skal stå for denne jobben. I C# finnes det allerede en klasse som gjør dette kalt BitConverter, men denne returnerer alltid strukturene i litle endian ,som har den minst signifikante byten sist. Siden TPM modulen alltid antar at bytestrukturene kommer i Big endian, som har den mest signifikante byten sist, må alle rekker retturnert fra BitConverter kunne reverseres, slik at leseretningen blir korrekt i forhold til hva TPM modulen benytter. Det er derfor ønskelig å lage en klasse som gjør denne jobben for oss, i stede for å måtte repetere denne jobben hver gang dette skal gjøres. TypeConverter sine funksjoner skal ha en boolsk verdi som spesifiserer hvorvidt rekken skal reverseres eller ikke. På denne måten skal man kunne benytte denne klassens grensesnitt uavhengig av ønsket retning på rekken. I tillegg skal denne klassen kunne konvertere fra byte rekker til verdityper via BitConverter. Denne skal også benytte samme grensesnitt som BitConverter men med unntak av den nye boolske verdien som spesifiserer om rekken må reverseres før konvertering. I tillegg skal TypeConverter inneholde funksjonalitet som sveiser sammen verdier til en sømløs rekke.

TPM/J har blitt benyttet til en del inspirasjon for designet. Selv om dette ikke er et ferdig grensesnitt, har det mye for seg og basere seg på allerede eksisterende løsninger. Dette hjelper med å få oversikt over hvilke klasser man trenger.

Alle grensesnitt som på en eller annen måte har blitt inspirert fra TPM/J har fått sammen navn, men med .NET navnekonvensjon. Det samme gjelder for strukturer spesifisert i del 2 av TPM spesifikasjonene.

3.2 IMPLEMENTERING

På grunnlag av kjennskap til Visual Studio som utviklingsverktøy fra før og etter at vi i gruppa kom fram til at vi skulle benytte Windows Vista som utviklingsplattform, ble dette ett klart førstevalg da det her er full støtte for .NET kjøretidsmiljøet. Utvikling i .NET gir også mulighet for å benytte C# Basic som kodeplattformer. C# Basic er ett scriptbasert programmeringsspråk, designet for å gjøre små enkle forhåndsdefinerte oppgaver i Windows miljøer.

Ved implementering av TPM modulen begynte vi med å kartlegge grensesnittet mot maskinvaren, for å åpne et grensesnitt fra programmeringsspråket vi benytter. I all enkelhet baserer vi oss på Windows Vista sitt grensesnitt mot TPM 1.2 modulen. Dette grensesnittet heter TBS og er implementert i C. For å benytte dette grensesnittet måtte vi i C# lage en klasse som åpnet for kommunikasjon mot DLL filen. Her benyttet vi marshaling, ved hjelp av funksjoner som i C# kalles DllImport, samt en del andre metoder. For å beskrive implementeringen vises et eksempel nedenfor. Beskrivelse på de forskjellige attributtene følger.

```
[MethodImpl(
      MethodCodeType = MethodCodeType.Runtime),
      SuppressUnmanagedCodeSecurity,
      DllImport("tbs.dll",
      EntryPoint = "Tbsip_Context_Close",
      CallingConvention
            = CallingConvention.Winapi,
      SetLastError = true)]
private static extern unsafe uint CloseContext(
      [In, Out]void* context);
```

MethodImpl er en klasse som benyttes for å fortelle kompilatoren hvordan maskinkoden det kommuniseres med er optimalisert. MethodCodeType er en enumerasjon med mulighetene IL, Native, OPTIL og Runtime. IL benyttes for kode som er kompilert av andre .NET språk, Native benyttes når koden man kaller er i ren maskinkode, OPTIL spesifiserer optimalisert IL kode og Runtime spesifiserer at implementeringen skal bestemmes ved kjøring. Her valgt vi Runtime, slik at uavhengig av senere implementeringer fra Microsoft sin side, har vi et grensesnitt som er korrekt implementert. Klassen SuppressUnmanagedCodeSecurity benyttes for å gi kompilatoren beskjed om at det som kommer, ikke nødvendigvis er kode opprettholdt av søppelinnsamleren i .NET, men at den ikke skal bry seg om dette, da vi vil opprettholde dette internt i klassen. DllImport klassen benyttes for å gi kompilatoren informasjon om hvilken DLL/Exe fil jeg har tenkt å implementere kall fra. I DllImport spesifiseres Inngangsområdet i filen, samt hvordan metodekall skal utføres. Det kan også spesifiseres hvordan tekst skal tolkes ved bruk av dette. For å forsikre om at jeg ved kall til funksjonen, alltid får ett rent kall uavhengig av utfallet av tidligere kall, antas det at siste kall gikk galt. Når så funksjonen deklareres, settes denne til å være privat for klassen, slik at denne ikke er direkte eksponert. Det benyttes offentlige funksjoner som kaller funksjonene implementert via DllImport. Disse sørger blant annet for at det ikke er ubrukte minneområder som ikke er slettet. Grunnen til at funksjonen er markert med syntaksen unsafe, er at man i C# ikke kan benytte pekere utenfor et område som er markert med

denne syntaksen. Til slutt benyttes det en void peker hvor data som går inn kan være endret når funksjonen returnerer. Returverdien er spesifisert av Microsoft som ett heltall, men siden dette skal vurderes som feilkoder har vi valgt å benytte en ikke-signert heltallsverdi. Dette har ingen spesiell begrunnelse, annet en at man da kan sjekke mot bare positive heltall. Når denne funksjonen skal kalles, har vi valgt å abstrahere oss fra pekersyntaksen med dens kompleksitet med en gang, slik at kall kan gjøres mest mulig intuitivt. Som nevnt over benyttet vi klassen SuppressUnmanagedCodeSecurity og av den grunn bør klassen vi har kreert arve fra grensesnittet IDisposable. IDisposable beskriver ett grensesnitt søpleinnsamleren kan benytte seg av for å flytte på og/eller slette klassen. Her blir kode for å rydde opp eksekvert og dermed er sikkerheten til kjøremiljøet opprettholdt globalt. Når en instans av TbsProxy klassen blir skapt, åpner denne for kommunikasjon mot TPM modulen og holder på data relevant til den åpne konteksten.

For å lette kommunikasjonen mot TPM modulen har det blitt laget en type konverteringsklasse for å omforme verdityper til arrayer av byter. Denne klassen benytter C# sin egen BitConverter klasse for å hente ut verdiene av typen eller en byte, men i stedet for å returnere byte arrayen direkte, kan denne klassen reversere arrayen til big endian hvis nødvendig. I TypeConverter klassen har det i tillegg blitt lagt inn metoder for å smelte sammen flere typer til en sømløs array med alle verdiene. Denne funksjonaliteten er veldig grei å benytte for å legge sammen verdier som skal benyttes for generering av SHA1 eller HMAC verdier. Denne funksjonaliteten gjør at man kan sende med utallige verdier som så blir omgjort til en liste med bytes og smeltet sammen til en array med alle verdiene sekvensielt i rekkefølgen de ble lagt inn. For å gjøre om tekst til bytes, er det lagt inn 2 forskjellige funksjoner som er tenkt benyttet. Den ene returnerer en array av Unicode bytes, den andre ASCII bytes. Til slutt har det i denne klassen blitt lagt inn metoder for å skrive ut heksadesimale strenger som representerer en gitt byte array og selvfølgelig parsing av denne strengen hvis nødvendig. Funksjonaliteten i TypeConverter klassen er inspirert fra TPM/J[11] prosjektet, hvor de benyttet en klasse som lignet BitConverter, men som alltid returnerte Big Endian byte arrayer i stede for Litle Endian som er standard i C#. I stede for noen av disse valgte vi å lage en versjon hvor programmereren kan spesifisere returtypen for å øke fleksibiliteten. Denne klassen er statisk og kan derfor ikke arves eller skapes instanser av.

Alle TPM strukturer spesifisert i plugin-mainP2Structrev103[4] er implementert som klasser som kan konverteres til arrayer av bytes via funksjonene ToBytes og FromBytes for å kunne serialiseres til en TPM modul. En del av grensesnittene her er også inspirert av TPM/J[11], som benytter et noenlunde likt grensesnitt. Der det er funnet fornuftig har klassene fått tilleggsgrensesnitt for å kunne opprettes på andre måter, slik at de kan opprettes av programmereren ved behov. Eksempler på dette finnes i TpmNonce strukturen, som benyttes for ikke-gjentagelses beskyttelse. Her benytter både TPM modulen og programmet denne for par/odde nonce. Par verdien blir generert i TPM modulen, mens odde verdien blir generert i programmet, dermed vil en ny instans av TpmNonce alltid være 20 bytes.

Ved kommunikasjon med TPM modulen kan det oppstå flere unntak gjennom programmet, dette er her tenkt at alle slike situasjoner skal ha sitt eget unntak som senere programmerere kan forholde seg til, i stede for å måtte teste eventuelle returverdier. Foreløpig er det kun laget unntak for bruk i forbindelse med tbs.dll, hvor returverdiene fra en gitt funksjon blir sjekket mot 0. Hvis den ikke er 0 blir det kastet en unntak som forflytter seg til første punkt, som håndterer unntaket eller en av dets overordnede i arvestrukturen.

For å håndtere kryptografi er det skapt en egen klasse som tar hånd om alt fra å generere et nøkkelpar, til å skape en spesifisert mengde tilfeldige bytes. Her er det implementert algoritmer for å kryptere ved hjelp av X-OR og RSA og for å skape SHA1 og HMAC verdier ut fra predefinerte mønstre. Denne klassen benytter mye fra Microsoft sin egen krypto leverandør for å gjøre jobben. Dette er gjort blant annet fordi det ikke er hensiktsmessig å implementere algoritmene selv og fordi dette er vedlikeholdt av Microsoft, slik at eventuelt oppdagede feil vil bli rettet av disse.

Alle konstanter det har blitt behov for å bruke i forbindelse med TPM strukturene, har blitt omformet til enumerasjons medlemmer. Dette gjør at det blir enklere å generere korrekt verdi i konstantene ved en senere anledning. Blant annet ved at programmereren ikke får beskjed om å sende med ett heltall, men en verdi av en enumerasjon hvor hver verdi i enumerasjonen er kommentert med hva den gitte verdien betyr.

```
/// <summary>
/// Represents the Ek type.
/// </summary>
public enum TpmEkType : short
{
      /// <summary>
      /// The blob must be
      /// TpmBlobEkActivate
      /// </summary>
      Activate = 0x0001,
      /// <summary>
      /// The blob must be TpmEkBlobAuth
      /// </summary>
      Auth = 0x0002,
}
```

Her er TpmEkType benyttet som eksempel da det her kun er 2 lovlige verdier. Hvis en programmerer, eller en bruker skulle måtte forholde seg til verdiene som en short, ville vedkommende person måtte slå opp i en eller annen oversikt for å få vite hvilke verdier som er lovlige. Hvilke verdier av de 65 536 mulighetene som ett 16 bits signert heltall gir mulighet for å benytte.

Etter alle strukturer og konstanter fra plugin-mainP2Structrev103[4] ble implementert, begynte vi på implementering av alle funksjoner i plugin-mainP3Commandsrev103[5]. Når vi hadde kommet hit begynte ting straks å bli vanskeligere. Dokumentet beskriver oppbyggingen av en kommando ganske rett frem. Det beskrives dog ikke hvordan forskjellige attributter i kommandoene skal beregnes. Dette kommer allikevel frem av dokumentet mainP1DPrev103[1] seksjon 13.2. Her beskrives blant annet hvordan autorisasjonsdataene skal beregnes, før de sendes til TPM modulen ved autorisasjons økter. Mange av funksjonene beskrevet i plugin-mainP3Commandsrev103 har en relativt høy kompleksitet, noe som har gjort at det har blitt benyttet mye tid for å nøste opp i kommandoene. Noen av kommandoene har fått egne klasserfor å holde på returnert data. Dette gjelder blant annet ved opprettelse av en OIAP økt. Her lagres pekeren til økten og de tilfeldige par/odde strengene med bytes. Koden i rammen under viser innholdet i klassen. For å få nok plass, er kommentarer fjernet.

```
class TpmOiapSession
{
      #region Private members
      protected RNGCryptoServiceProvider rng =
                                    new RNGCryptoServiceProvider();
      int authorizationHandle;
      TpmNonce authNonceEven;
      TpmNonce authNonceOdd;
      #endregion
      #region Public members
      …

      public TpmNonce AuthNonceOdd
      {
            get
            {
                  byte[] nonceBytes = new byte[20];
                  rng.GetBytes(nonceBytes);
                  this.authNonceOdd = new TpmNonce(nonceBytes, 0);
                  return authNonceOdd;
            }
      }
      public static TpmEncAuth GenerateEncAuth(string key,
            params object[] hValues)
      {
            byte[] data = TypeConverter.ToByteArray(true, hValues);
            HMACSHA1 hmacer = new HMACSHA1();
            hmacer.Key = TypeConverter.StringToUnicodeBytes(false, key);
            return new TpmEncAuth(hmacer.ComputeHash(data), 0);
      }
      public static TpmDigest GenerateParamDigest(
            params object[] sValues)
      {
            SHA1CryptoServiceProvider sha1 =
                  new SHA1CryptoServiceProvider();
            byte[] value = TypeConverter.ToByteArray(true, sValues);
            return new TpmDigest(sha1.ComputeHash(value), 0);
      }

      #endregion
      #region Constructor
      public TpmOiapSession(int authHandle, TpmNonce lastNonceEven)
      {
            this.authorizationHandle = authHandle;
            this.authNonceEven = lastNonceEven;
      }
      #endregion
}
```

3.3 TESTING

I forbindelse med utviklingen av modul 1 har det for det meste blitt benyttet whitebox testing. Hver klasse har blitt konvertert til og fra byte rekker, for å forsikre at denne funksjonaliteten virker tilfredsstilende. For første del av modul 1, som baserte seg på å skape kommunikasjon mot allerede eksisterende kode, har det blitt skapt en blackbox test på forhånd, som kaller et grensesnitt som denne modulen skal implementere. Her opprettes kommunikasjonen og det har blitt sendt en rekke av bytes hvor vi vet at vi får et feilfritt svar ved korrekt kommunikasjon. For å teste funksjoner som foregår internt i TPM modulen, har det blitt laget tester som sjekker om en rekke gir samme resultat på innsiden av TPM modulen, som på utsiden. Dette innebærer at alle TPM funksjoner blir verifisert av allerede eksisterende kryptografiske funksjoner, i .NET kjøretidsmiljøet. Hvis funksjonene returnerer en gyldig verdi betyr dette at sjekken har blitt godkjent. Selv om det har blitt benyttet whitebox testing betyr ikke dette at kvaliteten på testene har blitt kompromittert, da en test ikke går pass før den kan verifiseres av innebygde moduler i .NET. Selv om RSA funksjonene ikke har blitt ferdige, er testen på disse allerede klare. Her skal det skapes ett nøkkelpar inne i TPM modulen, hvor offentlig nøkkel blir hentet ut. Deretter blir en klartekst kryptert med og uten OAEP padding, for så å sendes til TPM for dekryptering. Hvis de dekrypterte verdiene samsvarer med opprinnelig klartekst, vil denne testen gå pass, hvis ikke vil testen feile. Hver verdi vil bli verifisert mot opprinnelig klartekst hver for seg, slik at eventuelle feil som forekommer med padding, ikke påvirker resultatene uten og vis versa. Hvis bare en av dekrypteringene feiler, vil det også henvises til hvilken av disse som feiler, slik man slipper å gå over mer kode enn nødvendig for å finne eventuelle feil.

4. AVSLUTNING

4.1 EVALUERING AV OPPGAVEN

Nå når vi sitter her på slutten av prosjektet og reflekterer over oppgave, ser vi at vi var litt vel optimistiske i januar, med tanke på omfanget av hva vi skulle klare å realisere i løpet av prosjektperioden. Grunnen til denne overvurdering, var at vi på forhånd ikke hadde nok innsikt i teknologien. Da det å få innsikt i teknologien var en del av oppgaven, føler vi at dette hadde vært vanskelig unngå. Vi kunne brukt mer tid i starten på å sette oss inn i TPM brikken, men da det tok såpass lang tid for oss å oppnå denne forståelse gjennom prosjektet, ville ikke en ekstra uke eller to hjulpet. Når det gjelder utviklingsdelen, kunne vi valgt å basere oss på å videreutvikle et allerede eksisterende grensesnitt. Dette kunne resultert i at utviklingen av modul 2 hadde blitt påbegynt. Vi valgte her å utvikle grensesnittet mot TPM brikken fra bunnen av. Vi føler fortsatt at dette var det korrekte valget. Hvis vi skulle fortsatt utvikling av et allerede påbegynt grensesnitt, hadde vi måte benyttet mye av tiden for og neste opp i eksisterende dokumentasjon og kildekode, for å revalidere at dette var korrekt før vi kunne gå videre. Når det gjelder teoridelen og kartleggingen av TPM brikken, kunne kartleggingen vært mindre omfattende og vi kunne heller valgt å se på det som kun var relevant for å gjennomføre autentiseringen. I møte med oppdragsgiver, var det derimot en del ønsker om mest mulig teori. Da veldig mange av aspektene ved TPM brikken er knyttet sammen på en eller annen måte, så var det på mange måter uansett unngåelig.

4.2 EVALUERING AV GRUPPAS ARBEID

Arbeidsfordelingen har vært god innad i gruppa. Siden vi kun er 2 stykker har det vært relativt greit å forholde seg til hverandre uten å bli uvenner. Vi føler at ansvarsforhold har vært klart avgrenset hele tiden og at resultatene er noe vi begge kan stå for. Selv om vi kan stå for resultatene, betyr ikke dette at vi er strålende fornøyde. Vi skulle begge to ønske at vi hadde hatt bedre tid til å få i land alle målene vi hadde satt oss for prosjektet i januar. Vi ser at dette har vært et tidskrevende prosjekt og for å komme helt i mål ville vi trengt et par måneder til. Detaljene på dette dokumentet, viser at vi har brukt hele 122 timer på bare rapportskriving.

4.3 VIDERE ARBEID

Det som gjenstår på implementeringen av grensesnittet mot TPM brikken, er per i dag stort sett bare å skrive ferdig funksjonene som kommuniserer med TPM brikken i TpmCommands klassen. Det er her snakk om maks 600 kodelinjer som skal inn i en klasse. Grensesnittet/Modul 1 vil da være ferdig utviklet.

Ved hjelp av teoridelen, vil utfordringene og fremgangsmåten for å utvikle modul 2 komme tydelig frem. Det er klart at omfanget man velger på prototypen/demonstratoren som gjenstår i modul 2, vil bestemme hvor tidsomfattende denne delen vil bli å utvikle. Er det noe vi har lært, så er det i hvert fall at det alltid tar lengre tid enn man først tror, spesielt når det er snakk om TPM brikken.

4.4 KONKLUSJON

Dette prosjektet har vært det første store prosjektet vi har gjennomført. Det har vært veldig lærerikt og har uten tvil gitt oss mange erfaringer som vi vil ta med oss inn i fremtidige jobbrelaterte prosjekter. Erfaringer som blant annet viktigheten rundt det å planlegge godt før man begynner, at ting alltid tar lengre tid en det man tror og at dokumentering av rapporten burde skje kontinuerlig.

Vi har fått brukt kunnskap opparbeidet gjennom vårt treårige studie. Kunnskap fra fag som blant annet arkitektur, operativsystemer, programmering og systemutvikling. Brukt denne kunnskapen på tvers av hverandre til å forstå informasjon, tilegne ny informasjon, utvikle løsninger og dokumentere disse.

Oppgaven har gitt oss betydelig innsikt i TPM brikken. Vi kommer begge til å ha denne brikken i bakhodet hvis vi engang i fremtiden får i oppdrag å utvikle nye løsninger. Løsninger der blant annet bruk av kryptering, sikker lagring eller autentisering er aktuelt.

Vi konkluderer med at selv om vi ikke kom helt i mål med det som var ønskelig, har vi hatt stor nytteverdi av dette prosjektet. Vi har lært arbeidsprosessen i prosjektgjennomføring, dokumentering av denne prosessen og ikke minst fått oversikt i en veldig fremtidsrettet sikkerhetsteknologi.

LITTERATURLISTE

1. **Trusted Computing Group.** TPM Specification Version 1.2 Revision 103: Part 1 - Design Principles. [Online] Juli 9, 2007. [Cited: Mai 10, 2009.] http://www.trustedcomputinggroup.org/resources/tpm_specification_version_12_revision_103_part_1__3.

2. —. TCG Specification Architecture Overview Revision 1.4. [Online] Trusted Computing Group, August 2, 2007. [Cited: Mai 10, 2009.] http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14.

3. **Challener, David, et al.** *A Practical Guide to Trusted Computing.* Indianapolis : IBM Press, 2008. 978-0-13-239842-8.

4. **Trusted Copputing Group.** TPM SpecificationVersion 1.2 Revision 103: Part 2 - Structures of the TPM. [Online] Oktober 26, 2006. [Cited: Mai 10, 2009.] http://www.trustedcomputinggroup.org/resources/tpm_specification_version_12_revision_103_part_1__3.

5. **Trusted Computing Group.** TPM SpecificationVersion 1.2 Revision 103: Part 3 - Commands. [Online] Trusted Computing Group, Oktober 26, 2006. [Cited: Mai 10, 2009.] http://www.trustedcomputinggroup.org/resources/tpm_specification_version_12_revision_103_part_1__3.

6. **Internet Engineering Task Force.** The Transport Layer Security (TLS) Protocol Version 1.2. [Online] Internet Engineering Task Force, August 2008. [Cited: Mai 10, 2009.] http://tools.ietf.org/html/rfc5246.

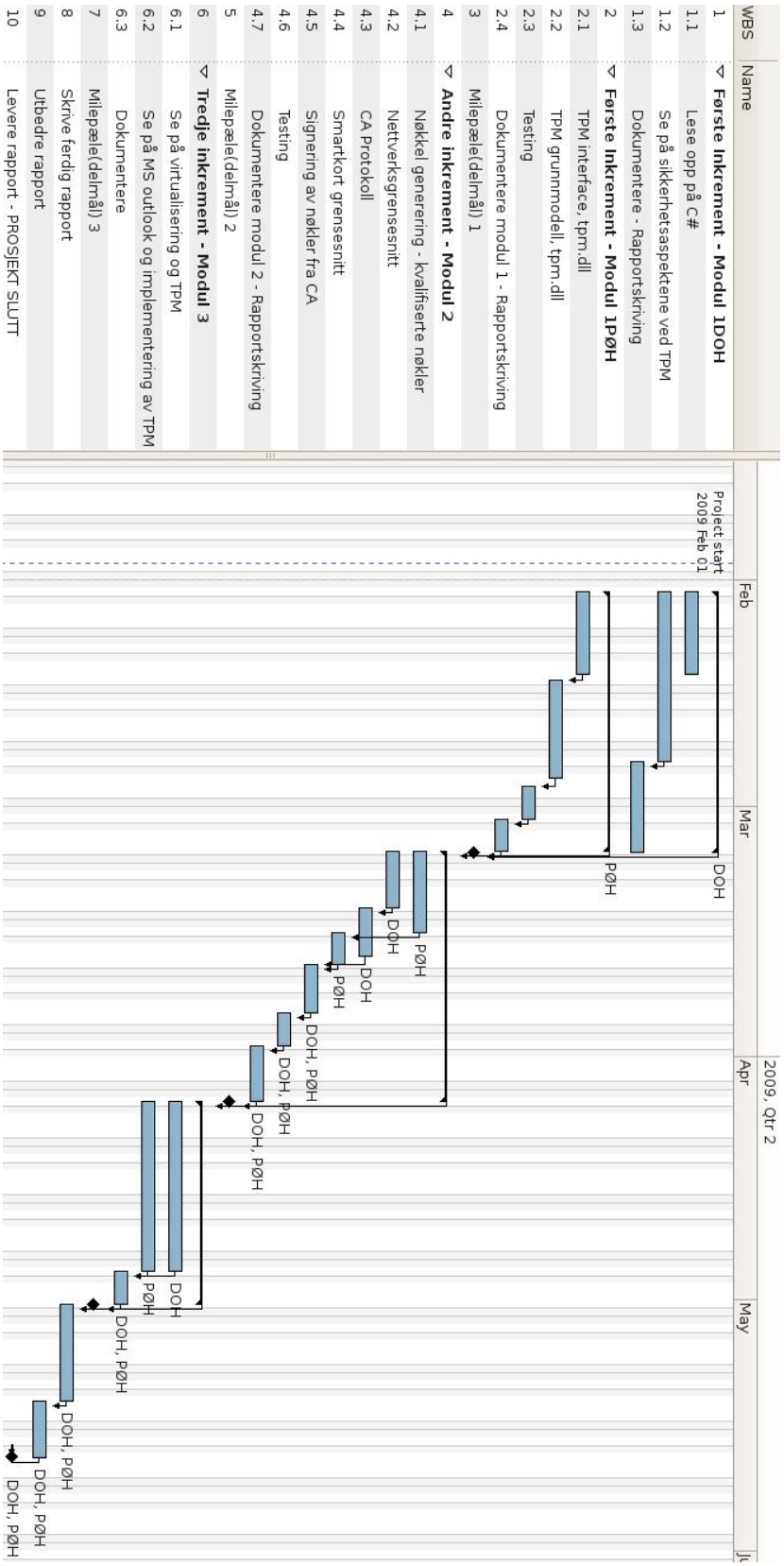
7. —. Extensible Authentication Protocol Method for Trusted Computing Groups. [Online] Internet Engineering Task Force, Mars 3, 2009. [Cited: Mai 10, 2009.] http://tools.ietf.org/html/draft-latze-emu-eap-tpm-00#ref-TCGMainSpec.

8. **TCG Infrastructure Workgroup.** Subject Key Attestation Evidence Specification Version 1.0. [Online] Trusted Computing Group, Juni 16, 2005. [Cited: Mai 10, 2009.] http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_subject_key_attestation_evidence_extension_version_10/.

9. **Group, Trusted Computing.** *Summary Of Features Under Consideration For The Next Generation Of TPM.* s.l. : Trusted Computing Group, 2009.

10. **Wikipedia.org.** X.509. [Online] April 30, 2009. [Cited: Mai 10, 2009.] http://en.wikipedia.org/wiki/X.509.

11. **Sarmenta, Luis.** TPM/J Java-based API for the Trusted Platform Module(TPM). [Online] April 3, 2007. [Cited: Mai 10, 2009.] http://projects.csail.mit.edu/tc/tpmj/.


WBS
Name
1 Første inkrement - Modul 1DOH
1.1 Lese opp på C#
1.2 Se på sikkerhetsaspektene ved TPM
1.3 Dokumentere - Rapportskriving
2 Første inkrement - Modul 1PØH
2.1 TPM interface, tpm.dll
2.2 TPM grunnmodell, tpm.dll
2.3 Testing
2.4 Dokumentere modul 1 - Rapportskriving
3 Milepæle(delmål) 1
4 Andre inkrement - Modul 2
4.1 Nøkkel generering - kvalifiserte nøkler
4.2 Nettverksgrensesnitt
4.3 CA Protokoll
4.4 Smartkort grensesnitt
4.5 Signering av nøkler fra CA
4.6 Testing
4.7 Dokumentere modul 2 - Rapportskriving
5 Milepæle(delmål) 2
6 Tredje inkrement - Modul 3
6.1 Se på virtualisering og TPM
6.2 Se på MS outlook og implementering av TPM
6.3 Dokumentere
7 Milepæle(delmål) 3
8 Skrive ferdig rapport
9 Utbedre rapport
10 Levere rapport - PROSJEKT SLUTT
Project start
2009 Feb 01
Feb
Mar
Apr
May
2009, Qtr 2
DOH
PØH
DOH
PØH
DOH
PØH
DOH, PØH
DOH, PØH
DOH, PØH
PØH
DOH
DOH, PØH
DOH, PØH
DOH, PØH

Forprosjektrapport

Autentisering ved bruk av TPM(Trusted Platform Module)

Hovedprosjekt ved Høgskolen i Gjøvik våren 2009

Per Øyvind Håvelsrud

David Ormbakken Henriksen

INNHOLD

1. MÅL OG RAMMER

1.1 BAKGRUNN

Trusted Computing Group er en organisasjon som dedikerer sitt arbeid til å forbedre mange aspekter ved informasjonssikkerhet. Denne organisasjonen utarbeidet spesifikasjonene som senere ble implementert og resulterte i brikken kalt Trusted Platform Module(TPM). TPM(versjon 1.1) ble fra og med året 2006 inkludert i PC-er solgt til forbruker. Et år senere, i 2007 kom versjon 1.2 som til dags dato fortsatt er den nyeste versjonen. TPM brikken har siden 2006 blitt å finne som standardkomponent i et stadig økende antall produserte PC-er.

TPM er en sikkerhetsmodul som blant annet kan benyttes for sikker oppbevaring av digitale nøkler, sertifikater og passord. Modulen har støtte for operasjoner som generering av nøkler, digital signering og kryptering. Denne teknologien vil kunne bidra til å rette opp mange sikkerhetsmessige svakheter i dagens PC-er. Microsofts operativ system Vista har for eksempel støtte for bruk av TPM(versjon 1.2) på operativsystemnivå i form av applikasjonen Bitlocker.

Buypass AS er et firma som eies i fellesskap av Norsk Tipping og Posten Norge. Buypass er en ledende leverandør av brukervennlige og sikre løsninger for elektronisk identifikasjon, elektronisk signatur og betaling. De har blant annet levert en identifiseringsløsning til Norsk Tipping sine tjenester over internett. Denne løsningen er basert på en smartkortleser, et smartkort og en PIN(autentisering). Foruten Norsk Tipping, kan denne løsningen blant annet brukes hos Altin, Posten og Komplett.

1.2 PROSJEKT MÅL

1.2.1 EFFEKTMÅL

Dette prosjektet vil gi Buypass en mye bedre innsikt enn de har nå i TPM brikkens potensial og begrensninger. Prosjektet vil gi dem en plattform å bygge videre på som viser løsninger i praksis på utviklings siden. Dette prosjektet vil også avdekke utbredelser og løsninger som allerede eksisterer. Buypass vil forhåpentligvis kunne basere fremtidige beslutninger i prosjekt hvor TPM modulen kan være aktuell på resultatet som vil foreligge i dette prosjektet.

1.2.2 RESULTATMÅL

Resultatet av dette prosjektet vil være en prototype/demonstrator programmert fra bunnen av, som utnytter TPM til brukerautentisering mot en sentral web basert tjeneste, samt en skriftlig rapport.

1.2.3 LÆRINGSMÅL

Læringsmålene med prosessen:

- Jobbe selvstendig med tanke på sikkerhet under utvikling av programvare.
- Få dypere innsikt i prosjektarbeidsprosessen og dokumentasjon av denne.
- Lære å bruke opparbeidet teori i praksis samt å tilegne seg ny kunnskap og opparbeide nye erfaringer.
- Få dypere innsikt i en relativt ny og fremtidsrettet sikkerhetsløsning.

1.3 RAMMER

Alle retningslinjer for prosjektet er å finne på:
http://www.hig.no/student/bacheloroppgave/retningslinjer

Disse er satt av Høgskolen i Gjøvik og må følges for å kunne oppnå en god karakter i faget.

Tidsfrist for prosjektet er satt til 20.05.09 da sluttrapporten må være inne hos kopisentralen innen klokka 12:00 og produktet må leveres. Prosjektpresentasjon skal holdes fjerde, femte eller sjette juni 2009

2. OMFANG

2.1 OPPGAVEBESKRIVELSE

Oppgavens formål er å se på hvordan TPM modulen kan bidra til å forbedre sikkerhet i allerede eksisterende løsninger og hvordan den kan bli benyttet i fremtidige løsninger. Det skal utvikles en prototype/demonstrator som utnytter TPM til brukerautentisering mot en sentral webbasert tjeneste. I forbindelse med dette må tjenesten det er snakk om kunne kommunisere med TPM modulen i brukerens pc og kunne kalle TPM modulen for nøkkelgenerering. Det må også sees på hvordan man kan bekrefte nøkkelens tilhørighet ved hjelp av andre former for autentisering i forkant. Det må velges/lages en metode for hvordan en bruker kan identifiseres første gang en generert nøkkel benyttes slik at denne nøkkelen entydig kan identifisere brukeren i senere tid. Dette er ment og lages slik at brukerautentiseringen blir sikrere ved at brukeren ikke bare identifiserer seg selv, men samtidig identifiserer datamaskinen han/hun sitter på. Det må her også avklares hvorvidt dette lar seg gjøre slik at en bruker (Alice) vil kunne bekrefte at hun er hvem hun gir seg ut for å være første gang hun autentiserer seg slik at bruker nummer to (Bob) vet at det er Alice som er sender av en melding og ikke en person som bare utgir seg for å være Alice.

Mer info om TPM finnes på: https://www.trustedcomputinggroup.org

Oppgaven kan deles inn i 3 deler:

1. TPM og administrasjon av denne.
   Vi vil se på hvilke muligheter modulen tilbyr, hvilke utbredelser som finne, hvilke eksisterende løsninger som utnytter modulen i dag og så videre. Lage en administrasjonsmodul som en bruker kan benytte for å få tilgang til funksjonalitet i TPM brikken. Brukeren kan for eksempel bruke denne administratormodulen til å generere nøkler (RSA-nøkler) internt i TPM brikken og importere nøkler generert eksternt for å nevne noe.
2. Sertifikatutstedelse
   Her kan vi tenke oss ulike varianter:
   - Bruk av ekstern CA - for eksempel Buypass CA som utsteder kvalifiserte sertifikater (også for testformål) Her vil det være krav til slik utstedelse som må tilfredsstilles (legitimasjonskontroll og bruk av kvalifisert ID)
   - Lage egen CA - bruke Windows standard produkter eller utvikle noe eget. Her vil man kunne stå friere med henhold til krav til utstedelse.
3. Bruk
   Her kan vi tenke oss ulike brukssenarioer:
   - autentisering mot Web-tjeneste
   - signering/kryptering av e-post (Outlook)
   - signering av dokumenter (Adobe).
   - Avhengig av studentenes interesser så kan man vektlegge ulike varianter.Her kan bruk av Microsoft Cryptographic API New Generation(CNG) være noe å kikke på. I CNG er det tilsynelatende støtte for TPM.

Det som skal gjøres:

- Utvikle en demonstrator som viser hvordan nøkkelgenerering kan gjøres, hvordan sertifikater kan godkjennes av en CA: Dette bør innbefatte både kvalifiserte sertifikater (sertifikater hvor man ikke har mulighet til å kopiere/hente ut privat nøkkel og dermed ligger isolert i TPM modulen som benyttes) og portable sertifikater(privat og offentlig nøkkel kan begge hentes ut).
- En applikasjon for PC som kan kommunisere med TPM modulen for nøkkelgenerering.
- En web-tjeneste for kommunikasjon med brukeren, hvor brukeren autentiseres og den genererte PKI nøkkelen kan signeres for senere bruk mot andre tjenester slik at brukeren kan benytte denne for å bekrefte sin identitet. Eventuelt benytte seg av noe som allerede eksisterer.
- Utvikle ett administrasjonsgrensesnitt mot TPM for blant annet å generere RSA nøkler og importere nøkkelpar fra andre TPM-brikker.
- Kartlegge bruk av TPM. Se på forskjellige sikkerhetsaspekter. Hvilke muligheter har TPM til å forebygge og beskytte seg mot relevante trusler? Stikkord: Phishing, man-in-the-middel, malware, trojanere, key-loggere, utro tjenere, tjenestenekt(DOS). Se på hvordan tilgjengelighet, integritet, ikke-benekting og konfidensialitet kan oppnås(blir ivaretatt). Hvis ikke TPM modulen har implementer noen løsninger, eller har løsninger som ikke

fungerer optimalt, hvordan kan disse forbedres ved hjelp av eksterne løsninger. Sammenligne funn gjort mot alternative løsninger som eksisterer og se på hvordan TPM modulen eventuelt kan bidra til økt sikkerhet i disse. Vurdere fordeler og ulemper.

- Fortløpende ta vurderinger rundt personvernhensyn og sikkerhetshensyn. Spesielt tenk sikre løsninger ved utviklingen av prototypen.
- Se på støtte for TPM i et virtuallisert miljø
- Se på muligheter for å implementere et grensesnitt i MS Outlook

Det må i forbindelse med utviklingen av prototypen avklares hvordan brukeren skal kunne identifiseres, hvorvidt det eksisterer et entydig grensesnitt for kommunikasjon mot forskjellige TPM moduler og eventuelt hvorvidt dette kan benyttes mot forskjellige operativsystemer(Hovedvekt mot Windows).

2.2 AVGRENSNING

På grunn av begrenset tid, vil utviklingsdelen av prosjektet kun foregå mot operativsystemet Windows Vista. Da kommende Windows 7 sies å være fullstendig kompatibel med Vista, vil denne også være støttet.

Tidsperspektivet vil også begrense oss til å utvikle prototypen i kun et programmeringsspråk. Dette blir C#(se punkt 5.1 for begrunnelse av dette valget).

3. PROSJEKTORGANISERING

3.1 ANSVARSFORHOLD OG ROLLER

Prosjektgruppa vil bestå av David Ormbakken Henriksen og Per Øyvind Håvelsrud.

David har fordypning i informasjonssikkerhet og vil ha hovedansvaret for sikkerhetsaspektet i prosjektet. David vil også være prosjektleder og ansvarlig for strukturering av dokumentasjon.

Per Øyvind har fordypning i programvareutvikling og vil være den som har hovedansvaret for utviklingsaspektet i prosjektet. Per Øyvind vil også ha ansvaret for web-siden og oppdatering av denne etter behov.

Alle i prosjektgruppa vil ha ansvar for å dokumentere arbeid og skrive rapport underveis i prosjektet.

Styringsgruppa vil bestå av Lasse Øverli og Mads Henriksveen.

Lasse vil være veileder og ha det overordnede ansvaret for å veilede gruppa i riktig retning og komme med konstruktiv kritikk underveis.

Mads er kontaktperson for oppdragsgiver og vil ha muligheten til å komme med innspill og ønsker underveis.

3.2 RUTINER OG REGLER I GRUPPA

Dager og tidspunkter for samling er følgende:

Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Sum
10:00 – 17:00		10:00 – 13:30	12:00 – 17:00	10:00 – 17:00	
7 timer		3,5 timer	5 timer	7 timer	22,5 timer

Det vil utover dette forventes 6,5 timer selvstendig jobbing. Totalt utgjør dette 30 timer jobbing med prosjektet i uka. Dette er veiledende og kan naturlig nok i perioder utgjøre mer. Oppmøte på ovenfor nevnte dager med oppgitte tidspunkter er obligatorisk hvis ikke annet er avtalt innad i gruppen på forhånd. Oppmøtested er utenfor skolens bibliotek. Forsinkelser samt fravær skal varsles så raskt som mulig.

Logg føres av begge gruppemedlemmene individuelt. Det har til dette formålet blitt utviklet en enkel applikasjon som logger tid og notater/kommentarer.

Statusrapporter skrives i fellesskap hver mandag og sendes på e-post til Lasse. Statusrapporten vil følge en fastsatt mal.

4. PLANLEGGING, OPPFØLGING OG RAPPORTERING

4.1 HOVEDINNDELING AV PROSJEKTET

Etter å ha diskutert en del frem og tilbake innad i gruppa, falt valget til slutt på den inkrementelle utviklings- modellen. Dette betyr at vi kommer til å jobbe sekvensielt med oppdelte moduler.

Utviklingen av dette prosjektet kan relativt enkelt deles opp i moduler. Fordelen ved å gjøre det sånn er at vi får fullført de mest kritiske modulene først. De kritiske modulene er de modulene som vi forventer å bli ferdig med innen endelig tidsfrist og som kan by på en del utfordringer i design og kode fasen. Dette er henholdsvis modul 1 til og med 2. Med tid som den mest kritiske faktoren, vil denne modellen bidra til å gi oss en bedre oversikt over progresjonen versus tidsfrister. Den inkrementelle utviklingsmodellen er også fleksibel når det kommer til endringer. Det er ganske enkelt å gå tilbake og utføre endringer i ettertid, noe som kan være aktuelt.

Vi ser at denne utviklingsmodellen vil gi oss noe “overhead” i form av ekstra testing. Denne modellen kan også gi oss noen utfordringer når det kommer til sammensettingen (integrasjonen) av de enkelte modulene, men dette vil vi ha i bakhode under utviklingsfasen.

Pågrunn av tidsrammen pålagt prosjektet og usikkerheten rundt tidsbruk ved gjennomføringen av enkelte punkter under modulene, vil modul 3 være en fleksibel modul med henhold til gjennomføringen. Dette vil også være en modul hvor tilbakemelding/ønsker av veileder og/eller oppdragsgiver til å styre i en annen retning kan gjøres. Disse tilbakemeldingene/ønskene vil i så fall være basert på funn gjort i tidligere faser av prosjektet, da TPM modulen i seg selv er en såpass ny løsning med mange vinklinger og muligheter.

4.2 PLAN FOR STATUSMØTER OG BESLUTNINGSPUNKTER

I samtale med veileder har vi blitt enig om å ha en fleksibel møteløsning etter behov. I utgangspunktet baserer vi oss på å ha et statusmøte annenhver uke. Vi har også muligheten til å kontakte veileder ved hjelp av e-post, telefon, msn og skype. Veileder vil motta en skriftlig statusoppdatering hver mandag fra og med uke 7.

Et møte med arbeidsgiver vil bli gjennomført etter at milepælene som er satt opp er passert. Dette for å holde oppdragsgiver oppdatert og gi mulighet for eventuelle innspill og ønsker.

Statusoppdatering innad i gruppa vil bli tatt under hver samling.

5. ORGANISERING AV KVALITETSSIKRING

5.1 UTVIKLINGSMILJØ

Med mål om høy sikkerhet i ferdig utviklet applikasjon, er C# og Java de programmeringsspråkene som best vil hjelpe oss å oppnå dette. Grunnen er ganske enkelt at disse språkene har et kjøretidsmiljø med referansetellere til allokerte minneresurser og passer på sletting av ikke lenger brukte minneområder, slik at en programmerer slipper feil med tanke på dette. Siden Buypass ikke har noen preferanser på valg av språk og Per Øyvind har kjennskap til både Java og C# som utviklingsspråk fra før og av disse foretrekker C# vil dette bli valgt som programmeringsspråk i prosjektet. C# sitt kjøretidsmiljø vet vi med sikkerhet at håndterer både buffer og stack overflow for brukeren og passer på at man ikke får skrevet til minneområder som ikke er allokert av en gitt peker. En rask titt i CVE(Common Vulnerabilities and Exposures) sin database viser at sårbarhetene ovenfor(spesielt buffer overflows) er såpass vanlig, så vi har derfor bevist valgt å styre unna C/C++ så langt det lar seg gjøre.

Utviklingsmiljøet vi retter oss mot er i all hovedsak er Microsoft Visual Studio, dette miljøet har full støtte for C# siden både språk og verktøy er utviklet av samme leverandør.

5.2 STANDARDER OG KILDEKODE

All kode som blir skrevet skal dokumenteres. Kildekoden skal benytte Pascal case i offentlige funksjoner, felter og lignende. Dette er i samsvar med .NET sin egen navnekonvensjon for interne funksjoner i klassene som ikke er eksponert ut til andre programmerere enn dem som skriver koden. Programmereren kan selv bestemme hvordan han vil gjøre det, det er allikevel anbefalt av Microsoft at man her benytter notasjon hvor første ord skrives med liten forbokstav, resten har stor.

Eksempler på Pascal notasjon:

TrustedPlatformModule, Integer, Bool

Eksempel på Privat notasjon

trustedPlatformModule, integer, bool

Mer informasjon om dette finnes Her: http://www.irritatedvowel.com/Programming/Standards.aspx

All dokumentasjon av kildekode vil bli satt over den gitte klasse/funksjonsdeklarasjonen, det vil her bli benyttet C# /// notasjon som gjør at det lett kan konverteres til XML. Fra XML kan dette så kjøres ved hjelp av XSLT til html, som så kan inkluderes som vedlegg til prosjektrapporten.

5.3 KONFIGURASJONSSTYRING

SVN(Subversion) vil bli brukt som konfigurasjonsstyringsverktøy for kildekode. Server og repository leveres og driftes av IT-Tjenesten ved Høgskolen. SVN er allerede oppe og går. Dette vil være vår sikkerhet ved eventuelle tap av data. I kildekodefiler skal det ved editering legges inn kommentarer på hva som er gjort.

Dokumenter vil bli lastes opp til gruppas hjemmeområde ved jevne mellomrom og dette vil fungere som backup, foruten at samme dokumentasjon vil være på begge gruppedeltakere sine PC-er, som kjører forskjellig operativsystem(Linux og Windows). Versjonskontroll vil skje manuelt.

5.4 RISIKOVURDERING

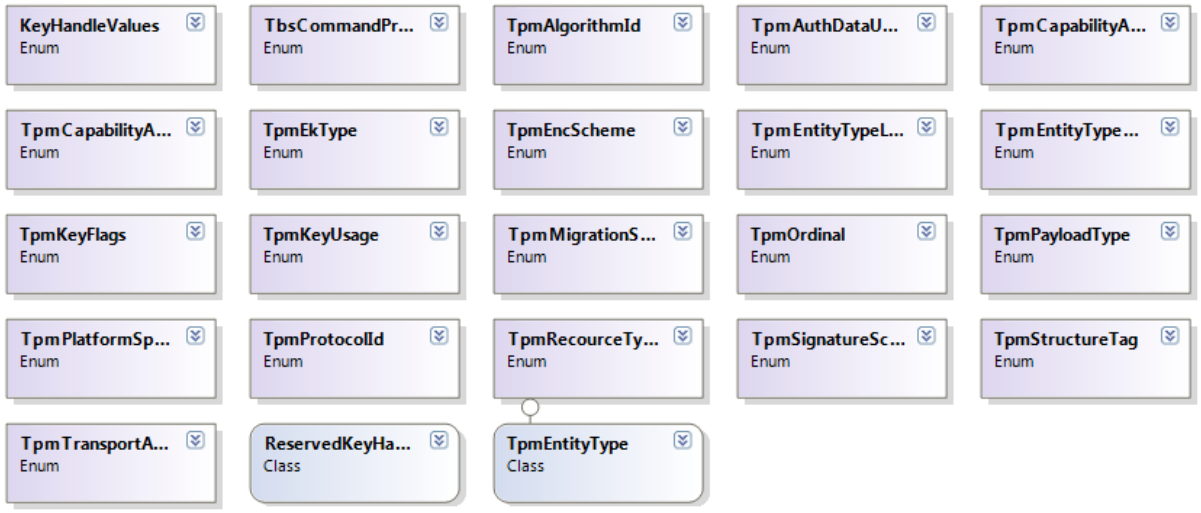
Følgende risikoer er avdekket:

Mulig hendelse	Sann-syn-lighet	Konsekvens	Årsak	Tiltak
Prototypen blir ikke ferdig innen avsatt tidsramme.	Lav	Resultatmålet blir ikke oppfylt.	Dårlig prosjektstyring. Lav arbeidsinnsats. (Uforutsett teknisk vanskelighetsgrad)	Være bevist på fremdriften i prosjektet. Være forberedt på å legge inn et ekstra gir hvis nødvendig.
Tap av prosjektdata(kildekode, dokumentasjon osv.)	Veldig lav	Fra: Har ikke noe produkt å levere. Til: Kun tap av noen arbeidstimer.	Virus, maskinsvikt, program krasj, glemme å lagre, uhell.	Ta backup regelmessig, både lokalt og eksternt.
Veiledningsproblemer	Lav	Progresjonen i prosjektet stopper opp.	Sterk uenighet, får ikke tak i veileder	Ha alternative ressurser å gå til. Holde jevn kontakt. Ha flere måter for å oppnå kontakt.
Samarbeidet innad i gruppa svikter.	Lav	Moralen synker. Fremdriften saktes. Kvaliteten på pro-	Uenighet. Konflikter.	God kommunikasjon. Bruke veileder som

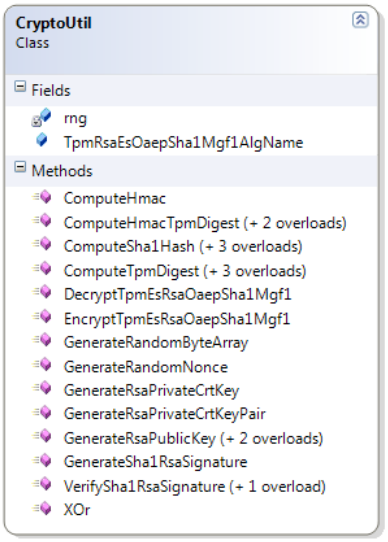
		sjektet faller.		konfliktmegler.
Fravær	Mode-rat	Forsinkelser.	Sykdom eller andre ukontrollerbare fakto-rer.	Tilpasse situasjonen. Jobbe for å ta igjen forsinkelser. Prøve til enhver tid å ligge fo-ran skjema kontra ligge bak skjema.
Maskinvareproble-mer	Lav	Forsinkelser. Tap av data.	Komponentsvikt, ty-veri, uhell.	IT-Tjenesten vil ha utstyr som er mulig å låne. Eventuelt høre med Buypass.

6. PLAN FOR GJENNOMFØRING

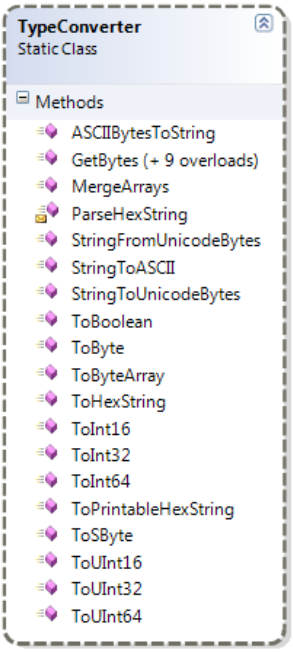
Gjennomføringen av prosjektet sånn vi har sett det for oss kommer frem i vedlegg 1. Ettersom tids-bruken på hver enkelt modul er vanskelig å forutse, vil dette diagrammet naturlig nok være veile-dende og mest sannsynelig se annerledes ut ved prosjekt slutt. Til tross, dette diagrammet vil være hva vi jobber etter og hva vi vil bruke til å måle fremdriften i prosjekt etter. Det er lagt opp tre mile-pæler som vil være våre viktigste indikatorer på hvordan vi ligger an. Disse milepælene er lagt til henholdsvis 06.03.09, 06.04.09 og 01.05.09. Den viktigste av disse vil være 06.04.09. Det vil bli lagt ut en fremdriftsplan som viser fortløpende progresjon på prosjektområdet vårt. Url: http://hovedprosjekter.hig.no/v2009/imt/in/tpm-authentication/status.html



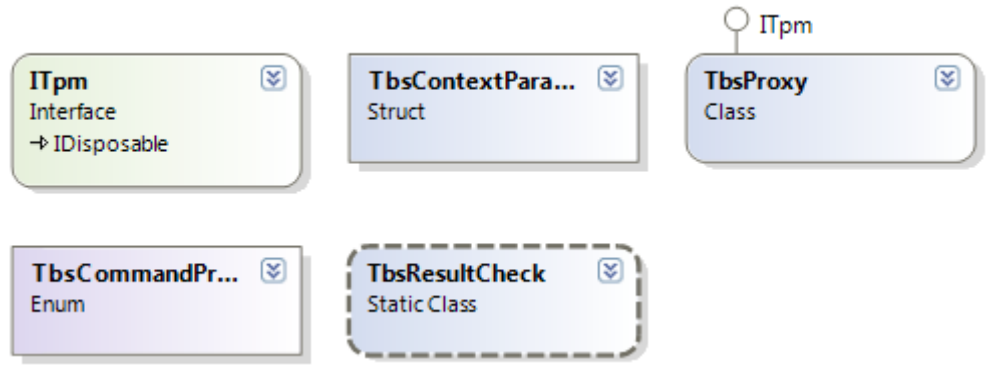
Figur 2 Klasser og enumerasjonen av konstanter definert i strukturdokumentet



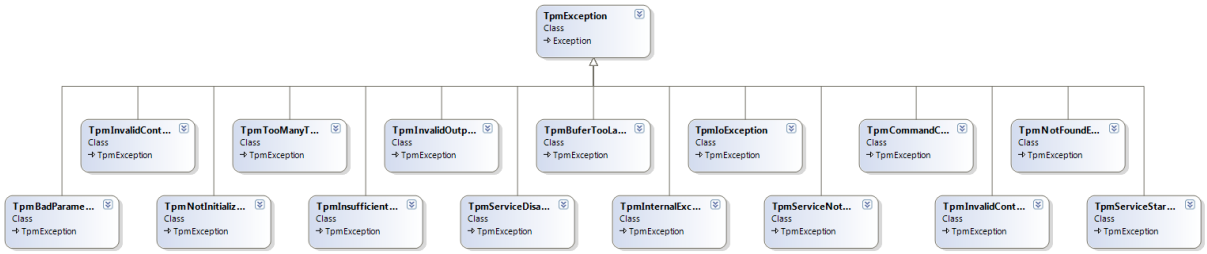
Figur 3 CryptoUtil benyttes for å håndtere ofte brukte kryptografiske funksjoner.



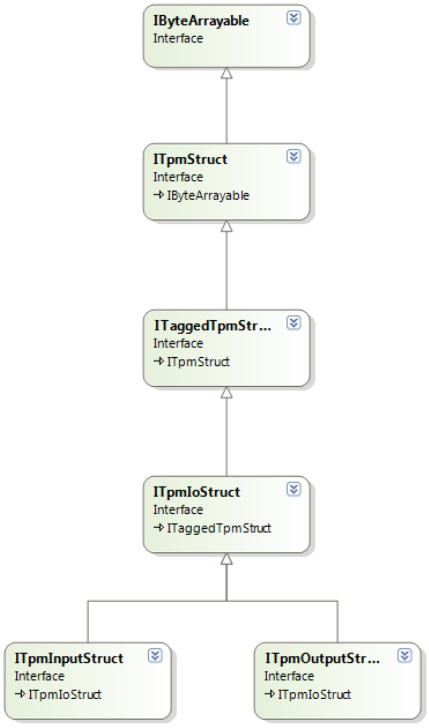
Figur 4 TypeConverter benyttes for å konvertere til og fra byte arrayer der dette er relevant, i motsetning til BitConverter kan man her spesifisere Big/Litle endian struktur.



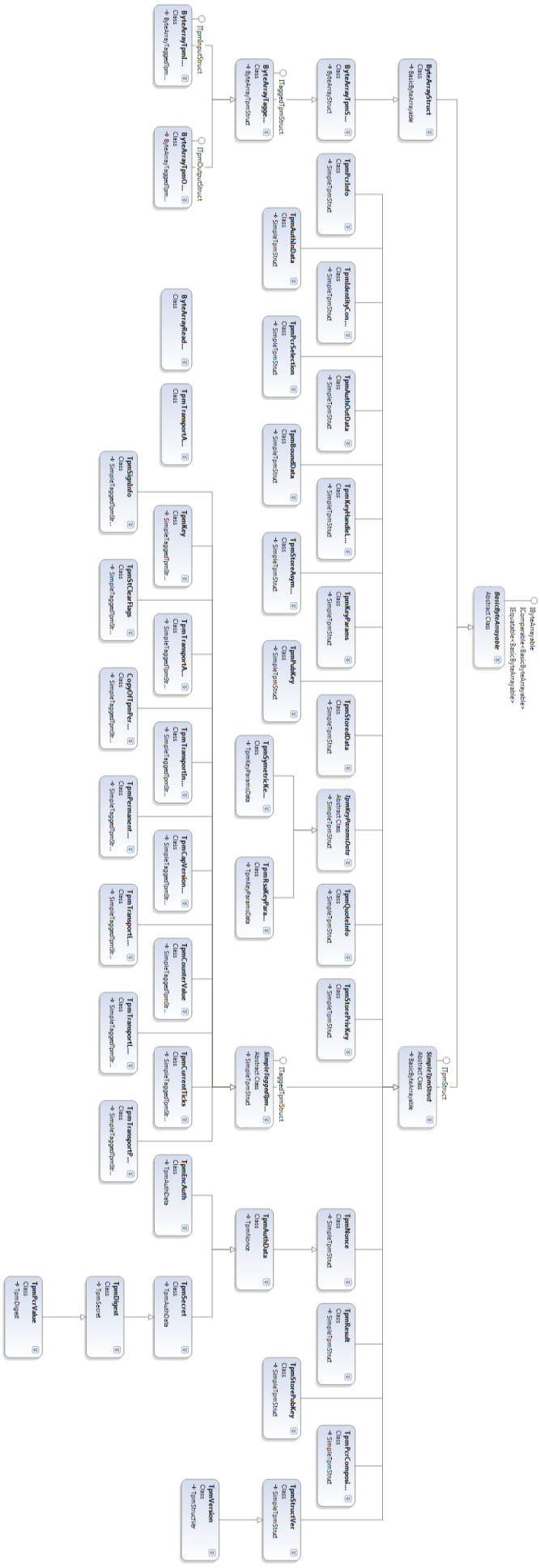
Figur 5 klasser og liknende benyttet for å kommunisere mot tbs.dll



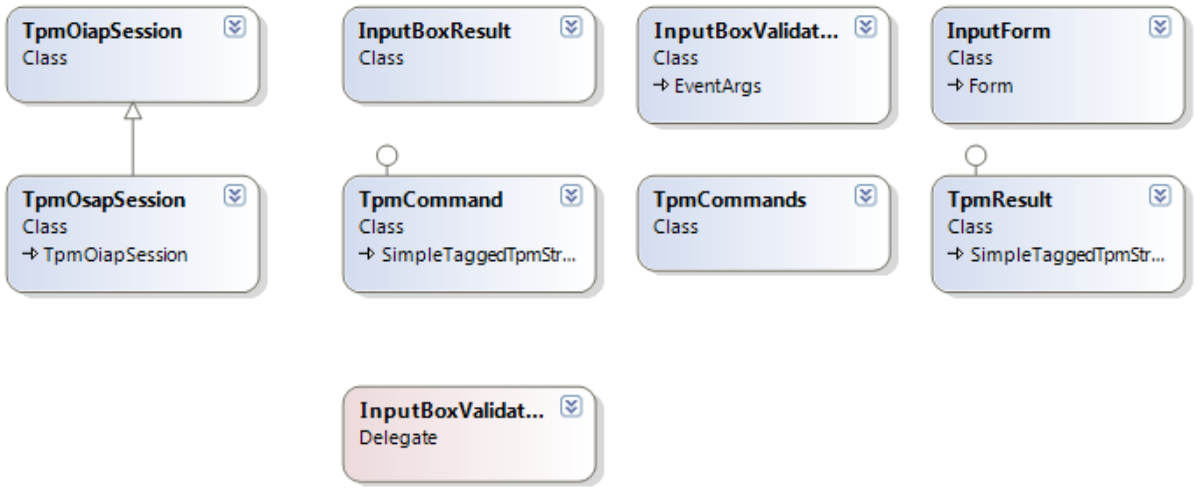
Figur 6 Alle unntakene utviklet for bruk mot TPM biblioteket



Figur 7 Grensesnitt benyttet for å gi mulighet for å jobbe med klasser uavhengig av hverandre.



Figur 8 Klasser benyttet for å tolke TPM strukturer.



Figur 9 Diverse andre klasser benyttet. Merk: TpmOsapSession arver fra TpmOiapSession da en OSAP økt er en utvidelse av en OIAP økt.

Statusrapport (Eksempel)

for(gruppe): Autentisering ved bruk av TPM

Dato: Sammendrag

1. STATUS

1.1 PLANLEGGING

David er ferdig med opplesing på C# og har begynt på å se på sikkerhetsaspektene ved TPM. Foreløpig så ser det ut som dette skal bli ferdig innen avsatt tidsfrist 6. mars(ferdig med rapportskriving av funn).

Per Øyvind programmerer forsatt på dll filen(wrapperen) og har implementert alle strukturene fra plugin-mainP2Structrev103.pdf, og har nå begynt med ett eksternt grensesnitt utad fra Dll filen, dette blir gjort på en slik måte at en programmerer kan kalle funksjoner fra klassene og få tilbake mest mulig standard C# klasser og biblioteker der dette kan benyttes.

1.2 ORGANISERING AV GRUPPENS ARBEID OG ORGANISERING

Arbeidsfordelingen følger fortsatt hva som er satt opp i den opprinnelige fremdriftsplanen.

1.3 RAPPORTSKRIVING

Ikke påbegynt enda, men all kildekode er dokumentert.

2. TOTALSTATUS FOR PUNKTENE OVER(OPPSUMMERING)

David begynner å lese seg opp på TPM og sikkerhetsaspektene nå. Per Øyvind har begynt med et eksternt grensesnitt.

3. MULIGHETER? TRUSLER/PROBLEMER?

Ingen store problemer så langt.

4. HVA ER AVSLUTTET? HVILKE OPPGAVER ER AVSLUTTET?

Opplesing på C# under Modul 1DOH.

Implementering av strukturer POH.

5. HVA ER UNDER ARBEID

- Se på sikkerhetsaspektene ved TPM, samt. Rapportskriving under modul 1DOH.
- Programmering av TPM interface og grunnmodell under modul 1PØH. (Grensesnitt mot selve modulen er klargjort i C#, klasse for kommunikasjon er under arbeid, ser nå på mulige eksterne grensesnitt for å få et mest mulig intuitivt bibliotek.

6. ER TIDSFRISTENE:

6.1 OVERHOLDT

Ja

6.2 OVERSKREDET

Nei

6.3 KRITISKE

Modul 1PØH kan ta noe lenger tid, foreløpig er det 1 uker til igjen til tidsfristen for disse, dette kan gå over fristen med opptil 1 uke. Etter ferdigstilt første del av oppgaven ser det ut til at dette ikke skal bli det store problemet.

7. HVA MED MOTIVASJONEN:

7.1 GRUPPENS SAMARBEID

Fungerer greit.

7.2 FORHOLD SOM OPPMUNTRER ELLER FRUSTRERER

Ingen spesielle. Det er selvfølgelig oppmuntrende å skape kode og å se at dette virker slik det er tenkt.

8. HVORDAN OPPLEVES VEILEDERKONTAKT:

Fornøyd

HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

Bypass A/S VED/ MADS HENRIKSVEEN

(oppdragsgiver), og

DAVID O. HENRIKSEN

PER ØYVIND HÅVELSRUD

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 15.01-09 til 20.05-09.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra HiG å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra HiG. Studentene dekker utgifter for trykking og ferdigstillelse av den skriftlige besvarelsen vedrørende prosjektet.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. HiG står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode, disketter, taper mv. som inngår som del av eller vedlegg til besvarelsen, gis det en kopi av til HiG, som vederlagsfritt kan benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av HiG til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved HiG og/eller studenter har interesser.

   Besvarelser med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens Internett-sider. Dette avhenger av

at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer 3 - tre - eksemplarer av oppgavebesvarelsen med vedlegg til Studenttorget. I tillegg leveres et eksemplar til oppdragsgiver. HiG kan stille til disposisjon ytterligere eksemplar(er) for oppdragsgiver mot at denne godtgjør produksjonskostnadene.

8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene.

   Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale, skjer dette uten HiG som partner.

10. Når HiG også opptrer som oppdragsgiver trer HiG inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

HiGs veileder (navn): LASSE ØVERLI

Oppdragsgivers kontaktperson (navn): MADS HENRIKSVEEN

Student(er) (signatur): _____ dato 25.01-09

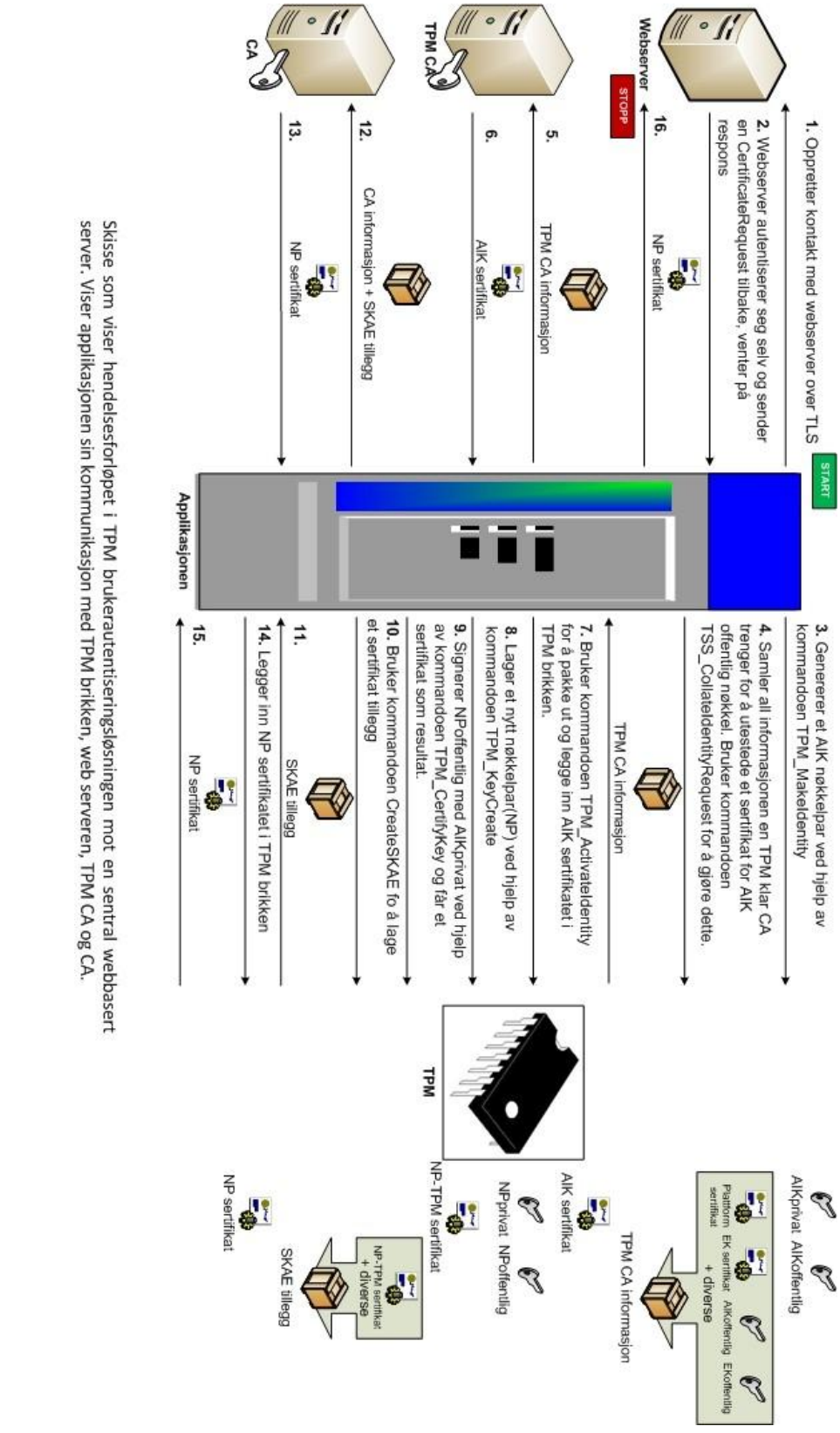
_____ dato 26/01-09

_____ dato _____

_____ dato _____

Oppdragsgiver (signatur): _____ dato 26/1-2009

~~Dekan (signatur):~~ BUD. DIR. _____ dato 27/1.2009



Skisse som viser hendelsesforløpet i TPM brukerautentiseringsløsningen mot en sentral webbasert server. Viser applikasjonen sin kommunikasjon med TPM brikken, web serveren, TPM CA og CA.