SIKRING AV POSISJONSDATA – BLUE FORCE TRACKING

av

ANDRÉ NORDBØ
NAN BARÅS

HOVEDPROSJEKTOPPGAVE 2007

FORSVARETS INGENIØRHØGSKOLE/
JØRSTADMOEN

FORSVARETS INGENIØRHØGSKOLE

HOVEDPROSJEKTOPPGAVE 2007

Kandidatenes navn: Nan Barås og André Nordbø

Oppgavens tittel: Sikring av posisjonsdata – Blue Force Tracking

Oppgavens tekst:

Posisjonsbestemmelse av egne og alliertes styrker er et viktig hjelpemiddel, bl.a. for å unngå blue-on-blue situasjoner. Et system kan realiseres ved å utstyre den enkelte soldaten eller avdelingen med en GPS-mottaker hvis posisjonsdata overføres som meldinger via et eget samband til et sentralt punkt hvor dataen kan synliggjøres i et KKIS. Posisjonene til egne styrker er imidlertid sensitiv informasjon og vil utgjøre en sikkerhetsrisiko hvis de ikke beskyttes. Ved overføring over radiosamband eller andre ubeskyttede nettverk bør informasjonen derfor krypteres. I tillegg til konfidensialitetsbeskyttelse må meldingene også integritetsbeskyttes slik at meldinger ikke kan manipuleres eller sendes flere ganger. Selv om enkeltvise posisjonsmeldinger i seg selv kan betraktes som informasjon med moderat beskyttelsesbehov (f.eks. gradert BEGRENSET), vil den aggregerte informasjonen med posisjonene til store styrkegrupper ha større verdi og følgelig ha høyere gradering. Samtidig er det ønskelig at disse dataene kan inngå et KKIS som normalt også vil ha et høyt beskyttelsesbehov (f.eks. System High, HEMMELIG). Kravet til kryptoutstyr blir dermed gitt av behovet for å beskytte det totale kommando-kontroll-systemet, og ikke de enkelte posisjons-meldingene.

Thales har under utvikling en AES-versjon av kryptoapparatet TCE 621, som kan være egnet for å beskytte trafikk inn og ut av et slikt KKIS. Ved å lage en mobil kryptoenhet som er interoperabel med denne, vil det være mulig å realisere et system for beskyttet posisjonsrapportering.

Oppgave:

- Sette seg inn i de relevante delene av den AES-baserte sikkerhetsprotokollen for TCE 621 og lage et design for den ønskede funksjonaliteten.

- Implementere en enhet som tar i mot posisjonsdata fra en ekstern GPS-mottaker og sammen med en unik ID og en tidsangivelse kryptere denne informasjonen for overføring over et IP-basert nettverk. De mottatte meldingene skal så dekrypteres av en AES-basert TCE 621.

  Enheten kan realiseres i form av av en PC hvor de nødvendige delene av sikkerhetsprotokollen og AES krypteringsfunksjon realiseres i programvare under Wiondows. Posisjonsdata fra GPS-mottakeren leses inn over et standard grensesnitt på PCen. De krypterte posisjonsmeldingene sendes ut over PCens normale nettverksgrensesnitt.

Oppgaven gitt:

Besvarelsen levert: 19.11.07

Omfang: 15 Studiepoeng

Utført ved: FK KKIS
Veiledere: Thales

Sikring av posisjonsdata
-Blue Force Tracking-

Hovedprosjektoppgave av

Nan Barås og André Nordbø

Forsvarets Ingeniørhøgskole 2007

Forord

Denne rapporten er besvarelsen på vår hovedoppgave, som ble utført ved Forsvarets Ingeniørhøgskole, FIH, høsten 2007. Oppgaven ble gitt på oppdrag fra Thales Norway og omhandler i hovedtrekk programmering av en mobil enhet i Blue Force Tracking senariet, som er kompatibel med deres egen stasjonære variant.

Denne prosjektperioden har gitt oss muligheten til å gå inn i dybden i et system vår arbeidsgiver er avhengig av, men som vi ansatte kun er brukere av. Det har gitt oss en forståelse av hvordan slike systemer er bygd opp og hvor mye små enkelt komponenter har å si i et system som dette.

Oppgaven kan i sin helhet deles inn i 4 deler:

- Teorien systemet er bygd opp rundt
- Vår fremgangsmåte for å nå Thales' mål
- Implementering og testing av vår metode
- Vedlegg som inneholder materiale med gradering K

Vi benytter samtidig anledningen til å takke våre veiledere Odd Arne Samdahl, Øystein Buer, Hans Petter Moen og alle andre ved Thales som har bidratt til at vi har nådd målet ved oppgaven. Og mens vi er inne på Thales vil vi gjerne takke kjøkkenet i kantina for usedvanlig gode lunsjer og ukens store høydepunkt; vafler på fredager.
Vi takker også faglæreren vår, Einar Skjellerudsveen, for tilbakemeldinger på rapporten underveis og fremskaffing av diverse utstyr som det har vært behov for under ferden mot målet.

Jørstadmoen 19. november 2007

Nan Barås

André Nordbø

Sammendrag

Begrepet Blue Force Tracking (BFT) er et konsept som blir brukt om en type kommunikasjonssystemer i militære sammenhenger. Intensjonen er at systemet skal kunne overvåke og vise posisjonene til egne styrker og avdelinger, og på den måten unngå såkalte "blue-on-blue" situasjoner der en skader sine egne. Med denne form for kommunikasjonssystem vil Forsvarets avdelinger kunne få en sanntids oversikt over egne styrkers posisjon ute i felt. Systemets fysiske oppbygging vil som oftest bestå av en GPS tilkoblet en mobil enhet som håndterer kryptering og videre transmisjon av dataene i jevne mellomrom, samt mottak og dekryptering i kommandoplassen.

Prosjektet gikk ut på å lage programvare for denne mobile enheten. For å klare dette var det behov for å sette seg inn i Thales sikkerhetsprotokoll rundt TCE621 AES. Det er et krypteringsapparat utviklet for å kryptere nettverkstrafikk mellom datanettverk over usikre nett som for eksempel Internett, og er mottaker/ dekrypteringsenheten tiltenkt brukt her.

Sikkerhetsprotokollen benyttet er basert på IPsec og er en variant av ESP i tunnelmodus. Dette innebærer at IP-pakker fra det interne sikre nettet blir kryptert som de er og innkapslet i en ekstern IP-pakke for ruting over det eksterne nettverket. Krypteringen skjer ved bruk av AES, også kjent som Rijndael algoritmen. AES er benyttet i counter mode. I tillegg til kryptering må den interne IP-pakken autentiseres, noe som er løst ved å bruke CMAC algoritmen.

Hovedsakelig fungerer sikkerhetsprotokollen til Thales som de offentlige standardene tilsier. Unntakene fra dette er at ESP formatet er noe omdefinert, telleren i counter mode benytter et tilbakekoblet skiftregister med et udokumentert polynom, samt at TCE621 er little endian i motsetning til big endian som er standarden for kommunikasjon mellom nettverksenheter.

Videre ble det satt opp et labmiljø som skulle simulere kommunikasjonsprosessen slik det ville blitt brukt i felt. En GPS mottaker ble tilkoblet PC-en som simulerte den mobile enheten. PC-en var satt opp med Windows XP og ved bruk av den utviklede Javaprogramvaren ble det oppnådd enveis interoperabel med TCE621 AES.

Innholdsfortegnelse

Figurliste

1. Innledning

Begrepet Blue Force Tracking (BFT) er et konsept som blir brukt om en type kommunikasjonssystemer i militære sammenhenger. Intensjonen er at systemet skal kunne overvåke og vise posisjonene til egne styrker og avdelinger, og på den måten unngå såkalte "blue-on-blue" situasjoner der en skader sine egne. Med denne form for kommunikasjonssystem vil Forsvarets avdelinger kunne få en sanntids oversikt over egne styrkers posisjon ute i felt. Dette gir Forsvaret en stor fordel i form av å hele tiden ha kontroll på sine egne, og på så måte unngå situasjoner med katastrofale følger.

Per i dag finnes det mange ulike systemer for overvåking av kjøretøy og personell som både Staten og sivile bedrifter benytter seg av. Eksempler på dette er Røde Kors som benytter DISKO[1] (Digitale samhandlingsverktøy i Kommandoområde) til å følge søks og redningslagene sine fra en sentral base. Store transportselskaper innen shipping benytter overvåkingssystem for flåtestyring, noe som gir dem oversikt over skipenes lokasjon på havet. Flere og flere drosjefirmaer har tatt i bruk tilsvarende systemer som viser hvor deres enkeltbiler befinner seg i områdene de opererer i. Dette for å minske søk etter fører og bil dersom de skulle blitt utsatt for en ulykke eller en kriminell handling.

I dag er det flere større selskaper som satser på ulike kommersielle overvåkingssystemer, og et av disse er Systematic. Deres SitaWare[2] er i dag et fullt kommersielt og ugradert system, men som er satt til testing i militære situasjoner av det Danske Forsvaret. Også det Norske Forsvaret har fulgt med på utviklingen av denne typen systemer, og har foreløpig tatt i bruk et kommunikasjonssystem som går under navnet BMS (Battlefield Management System). I Irak benytter USA et Blue Force Tracking system som går under FBCB2[3] (Forces XXI Battle Command, Brigade and Below System).

Militærets bruk av slike systemer går derfor hovedsakelig ut på å sende posisjonskoordinater med tilhørende identifikasjonsdata tilbake til en kommandoplass. Systemets fysiske oppbygging vil da ofte bestå av en GPS som er koblet til en enhet som håndterer kryptering og videre transmisjon av dataene i jevne mellomrom, samt mottak og dekryptering i kommandoplassen. Muligheten til å legge til tilleggsfunksjoner kan bidra til et utvidet system med flere funksjonaliteter. Dette kan for eksempel være muligheten til å sende korte tekstmeldinger og tilleggsinformasjon til en kartapplikasjon. Disse funksjonene kan ofte være behjelpelig for andre i avdelingen, dersom lagene ute i felt kommer over uforutsette hindringer som sprengte broer, ødelagte veier eller fiendtlige styrker.

Den informasjonen et BFT system kan formidler har frem til moderne tid kun vært løselig ved hjelp av tale over radiosamband. Dette er fremdeles en tidkrevende metode hvor avsender har måttet lese av sine egne koordinater, kryptere dem ved hjelp av en forhåndsbestemt metode, kalle opp kommandoplassen, for så å få overlevert meldingen. Deretter skal meldingen dekrypteres manuelt og koordinatene bli tegnet for hånd på et kart. Men med dette moderne systemet vil styrkens posisjon bli automatisk hentet fra en GPS og kryptert sammen med identifiseringsdata, før det sendes tilbake til kommandoplassen. Her vil dataene bli dekryptert, for så kunne vises i et K2IS system ved hjelp av en grafisk kartapplikasjon.

---

1 DISKO: hjelpekorps.org/disko/ av Norges Røde Kors
2 SitaWare – The C4I framework: systematic.dk/UK/Products/SitaWare/ av Systematic
3 Bigger Role for Blue Force Tracking: military-information-technology.com/article.cfm?DocID=504 av Mickey McCarter

2. Hoveddel

2.1 Situasjonen i dag

Ideen bak Blue Force Tracking er som tidligere nevnt å unngå ”blue on blue” situasjoner.
I en ideell verden kunne vi dratt analogier til hvordan enkelte strategispill[4] til PC fungerer. Ledelsen, det vil si spilleren, har der full oversikt over hvor alle egne enheter befinner seg og kan koordinere angrep og forsvar med all informasjon tilgjenglig om sine styrker. Kunnskap om fienden blir automatisk lagt inn i situasjonsbildet vårt og automatiserer innhentings- oppgavene til etterretning. Der flere styrker kjemper i lag styrt av flere ledelseselementer (spillere) blir situasjonsbildene sammensmeltet til ett stort komplett bilde for å unngå å skade samarbeidspartnere.

Militær ledelse har alltid forsøkt å strekke seg mot den ideelle situasjonsforståelse. Dette ved å organisere forsvaret i et hierarkisk system der informasjonsflyten tilpasses det aktuelle nivået. Informasjonsbehovet vil på mange måter styres av hvor stor grad av tillit og selvstendighet ledelsen sender nedover i hierarkiet. Herunder vil stikkordet intensjon være sentralt. For at dette systemet skal virke så godt som mulig, er hierarkiet bygget opp ved å dele inn området i små teiger med hver sin sjef. Sjefer over sjefer dekker stadig større området og ser mer og mer av krigsspillet, men mindre og mindre av detaljene. Blue on blue situasjoner blir dermed redusert ved at teigene vet om seg selv og de rundt seg. Denne tankegangen gir mening i tradisjonell symmetrisk krigføring, mens den kanskje ikke er fullt så ideell i dagens usymmetriske krigsopperasjoner der multinasjonalt samarbeid og kombinasjon av ulike enheter blir viktigere og viktigere. En vesentlig problemstilling her vil da være å finne ut *på hvilken måte blue on blue situasjoner skaper utfordringer i dagens trusselbilde?*

Det Norske luftforsvaret har kommet et godt stykke på vei innen det å bruke teknologi til å skaffe oversikt over sine egne styrker. Linker mellom fly[5] og bakke sørger for at Forsvarets operative hovedkvarter til en hver tid har full kontroll over hvor alle landets operative fly befinner seg. Sjøforsvaret har noe tilsvarende men dog noe tregere system da kravet til sanntidsinformasjon ikke er fullt så høyt til sjøs.

I dagens bruk av militærmakt er det spesielt to faktorer som forverrer ledelsens oppgave i å lede operasjoner i forhold til mer tradisjonell krigføring. Det ene er multinasjonalt samarbeid og det andre er koordinering mellom militærgrenene land, sjø og luft. Disse to faktorene har mye til felles. Flere faktorer som uklart fiendebilde og operasjoner utført utenfor egne landegrenser på ukjent plass er også viktige, men sett opp mot temaet i oppgaven har de ikke stor relevans.

Forsvarsgrenene luft, sjø og land opererer naturligvis på forskjellig vis. Fly er svært raske og har stort ødeleggelsespotensial. De er dyre per enhet og krever høy grad av sanntidsinformasjon for å kunne ledes optimalt.
I den andre enden består landstyrker av svært stasjonære enheter best utrustet til mer ”delikate” operasjoner som ikke kan løses med bomber og raketter. Landstyrker preges av veldig mange enheter, soldater og diverse kjøretøy. Dette enorme antallet er trolig grunnen til at presis styring ovenfra og at informasjonsflyten ikke blir like omfattende hos landstyrker.

---

[4] Age of Empires 2 og Empire Earth
[5] For eksempel Link 16

Det er noen unntak som for eksempel spesialjegere som ofte har veldig kritiske oppdrag og opererer isolert på egenhånd. Dette skaper et nytt interessant spørsmål: *Hva kan vi si om kravet til sanntidsinformasjon, eller sagt med et annet ord, oppdateringshastigheten til informasjonen?*

På grunn av forskjellig operasjonsmåte på tvers av forsvarsgrenene oppstår det kulturelle forskjeller mellom grenene, som igjen skaper skiller. Når flere grener skal samarbeide innenfor de samme fysiske områdene får sjefene behov for å koordinere seg imellom. Dette på et nasjonalt plan. Trekker vi fenomenet videre, mot multinasjonalt samarbeid ser vi også at kommunikasjon blir en enda større utfordring. Og i tillegg må vi introdusere noen filter for å begrense hvilken informasjon vi deler. Vi ønsker ikke å dele all informasjon med hvem som helst. Dette av sikkerhetsmessige årsaker.

For luft og sjø har vi systemer som er godt på vei mot målsetningen om å gi et automatisert situasjonsbilde. Dette hovedsakelig fordi de har relativt få, dyre og teknologisk avanserte enheter. Landstyrker på den andre siden har ikke disse samme fordelene. Til en viss grad er systemer laget for de tyngre enhetene som stridsvogner og kjøretøy[6], men for den gjennomsnittlige fotsoldat/lag er det fremdeles teknologiske vanskeligheter som må overkommes.
I tillegg har vi aspektet rundt Kommando Kontroll Informasjonssystemene (KKIS) Forsvaret benytter. Situasjonen i dag, er at vi har mange frittstående systemer som ikke utveksler data på en tilfredsstillende måte. Det jobbes med integrasjon, og det jobbes med å utvikle nye systemer. Forsvaret er i en startfase og veien frem til et komplett helhetlig system som virker i praksis er enda lang å gå.

Tilbake til vårt første spørsmål angående problemstillingen rundt blue on blue:
Krig er kaotisk og usikkert. En ting er slik vi har det i dag der fienden vår, rent bortsett fra å være vanskelig å definere, er svært teknologisk underlegne oss. Verre blir det om fienden er på lik linje med oss og kan skade oss der vi i dag sitter trygt. En måte å tenke på, er at alle soldater skal kunne vite hvor alle andre allierte soldater befinner seg i nærområdet. Dette ville gi soldaten bedre oversikt, og minimert sjansene for ildretting mot egne. Et slikt system måtte være ekstremt teknisk avansert og ha kapasiteter langt utover rimelige grenser per dags dato**.**

Informasjonen er dermed ikke først og fremst interessant for den gjennomsnittlige soldat. Situasjonen er noe annerledes for spesialjegere som jobber langt unna hovedstyrken, innenfor fiendens territorium og observerer nøkkelpunkter.
En annen situasjon der blue on blue faktisk er et problem, er situasjoner som innebærer ildstøtte fra for eksempel fly. Hvis ikke ledelsen vet om sine egne innenfor faresonen kan slik støtte få fatale følger for våre egne på bakken.
Avgjørelsene om slik støtte taes på høyt nivå, så det er mest naturlig å konkludere med at det viktigste er å få informasjonen oppover i systemet, og ikke horisontalt ut.

Når det gjelder oppdateringshastigheten vil den primært avhenge av to ting: Hvor store muligheter enheten har til å bevege seg mellom hver innrapportering og hvem som skal benytte informasjonen. Tenker vi oss dette benyttet i SIBO[7] miljø (horisontalt) setter det store krav til nøyaktighet og sanntidsinformasjon. I slike situasjoner vil det som nevnt før være uhensiktsmessig å måtte bearbeide mer informasjon enn nødvendig og det kreves drill og automatikk fremfor avanserte systemer.

---

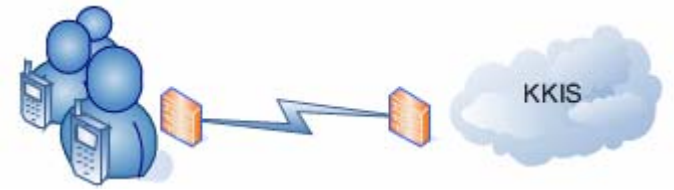
[6] BMS – Battlefield Management System
[7] Strid i bebygd område

For høyere ledelse (oppover) er det ikke interessant å vite hvor alle soldatene befinner seg til enhver tid. Det ville være overkill av informasjon. Det som er mest interessant er avvikene. De som ikke befinner seg der hovedstyrkene er, slik at ikke disse blir truffet av tyngre skyts.

Over til det tekniske er det flere faktorer som påvirker hvordan et slikt system kan realiseres. Vi har enheter ute i operasjonsområdet, og vi har en kartapplikasjon i et KKIS som lar ledelsen planlegge oppdrag. Mellom disse må det være en kommunikasjonsbærer for å overføre posisjonene til enhetene.

Figur 1: Enheter og et KKIS med en link mellom

Nøyaktig posisjon er lett å få tak i ved å utstyre hver soldat med en GPS mottaker, og fordi vi ønsker å beskytte vår posisjon, må vi implementere systemer som forhindrer at fienden kan misbruke vårt system. Det innebærer at fienden ikke skal kunne lese hva vi sender fra oss eller tukle med innholdet i meldingene. Vi må også sørge for at vi kan verifisere at meldingen kom fra en av våre. Disse egenskapene kalles konfidensialitet, integritet og autentisering. En fjerde situasjon som er uønskelig er om fienden kan ”ta opp” en melding sent av en av oss, for så å ”spille den av” på nytt på nettet. Fienden trenger ikke forstå innholdet, og meldingen var opprinnelig autentisert. Her benyttes en motfunksjonalitet som kalles ”anti-replay”.



Figur 2: GPS og sikkerhet må være med

Egenutvikling er dyrt og tar lang tid. Dette har vi erfart igjennom store prosjekter Forsvaret har hatt opp igjennom tidene. Trenden i dag går derfor mot å kjøpe ”Hyllevarer” og tilpasse disse til de kravene vi må sette til funksjonalitet og sikkerhet. Kommandoplasser er satt opp med standard x86 datamaskiner med den samme grunnmuren vi finner på maskinen hjemme, altså Windows. Nettverket som knytter maskinene sammen er også standard TCP/IP nett slik vi finner på det globale Internett.

2.2 Sikkerhetsprotokollen

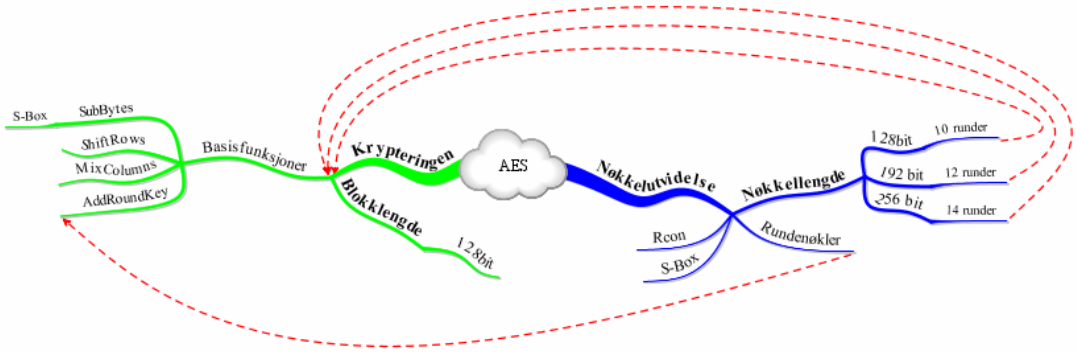
Posisjonsmeldingene samlet sett vil ha en enorm verdi samlet i et KKIS system og får derfor høyt beskyttelsesgradering. Thales TCE621 er derfor tenkt på som et mellomledd mellom den ytre usikre nettverksverdenen, og sikre miljø i kommandoplassene. For å sikre at informasjonen på veien ikke blir kompromittert eller misbrukt på annen måte bruker vi kryptografi på bestemte måter med delte hemmeligheter, altså nøkler. De to hovedmetodene av kryptografi er asymmetrisk og symmetrisk. Ved bruk av asymmetrisk algoritme må hver enhet ha hver sin private og offentlige nøkkel, samt at det må være et system der mottakeren vet om de offentlige nøklene til alle kommunikasjonsparter. Asymmetrisk krypto er i størrelsesforhold 1000 ganger tregere enn symmetriske algoritmer, som benytter kun én nøkkel per forbindelse. Thales har valgt en symmetrisk algoritme som bruker et nøkkelpar per forbindelse. Fordi det er ønskelig med en åpen algoritme som i seg selv ikke er gradert, har Thales valgt å satse på algoritmen AES i counter mode. Rammeverket rundt som sørger for å administrere sikkerheten er protokollen ESP i tunnel mode.

2.2.1 AES

Advanced Encryption Standard (AES) er en symmetrisk krypteringsalgoritme som er blokkorientert. Den ble utviklet av Joan Dermen og Vincent Rijmen. Sammen kom de opp med navnet Rijandael som er det egentlige navnet til algoritmen. Dette skjedde i 1998. På grunn av at algoritmen DES ble utdatert pga for korte nøkler, ble det startet en konkurranse for å komme opp med alternativer til den nye AES standarden. Rijandael ble vinneren av konkurransen og ble AES i 2002. Siden da har algoritmen vært utprøvd og forsøkt knekt. Ingen metode er enda funnet for å bryte den raskere enn et bruteforce angrep, med unntak av sidekanalsangrep som går på implementeringen og ikke algoritmen i seg selv.

AES fungerer slik at den deler opp det den skal kryptere, meldingen, inn i faste deler hver bestående av 128bit. Dette utgjør blokkstørrelsen. Det skal nevnes at Rijandael algoritmen i utgangspunktet ikke låser denne størrelsen, men kan bruke høyere blokkstørrelser. Algoritmen har også en symmetrisk nøkkel på 128, 192 eller 256bit. Dette er den delte hemmeligheten begge ender av kommunikasjonskanalen må være i besittelse av. Nøkkellengden avgjør hvor mange runder algoritmen skal gå igjennom. Dette vil bli klarere ved nærmere gjennomgang av AES.

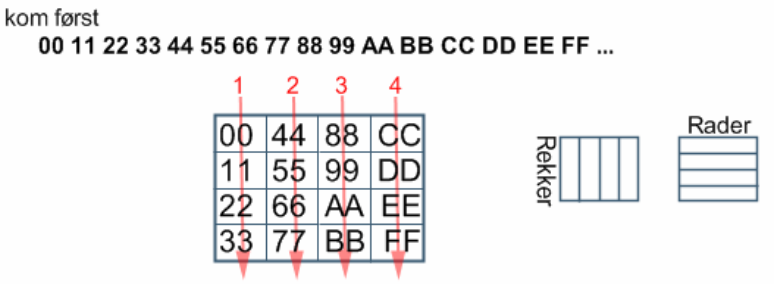
Algoritmen har to hovedfunksjoner. Den ene er å omstokke på meldingen ved bruk av 4 hovedfunksjoner. Den andre er å utvide nøkkelen. Dette er illustrert i **figur 3.**



Figur 3: Illustrasjon av hovedfunksjonaliteten til AES

Blokkene er organisert i et mønster på 4*4 ruter med 2 heksadesimale tall i hver rute. Dette gir da 8bit per rute * 4 * 4 = 128bit. Denne organiseringen av bit går igjennom AES algoritmens steg, og de gjeldende verdiene benevnes med *tilstanden* i teksten som følger. Verdier fylles innledningsvis inn i matrisen slik vist i **figur 4:**

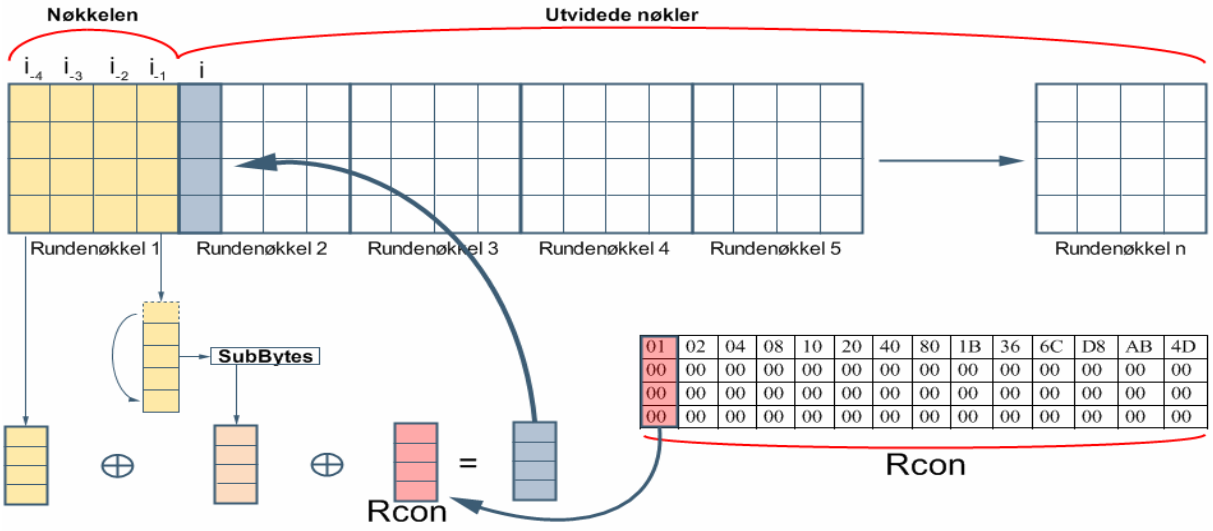
For å kunne referere til bestemte deler av tilstanden, benyttes *rekker* om vertikale sammenhengende verdier, og *rad* om horisontale verdier.



Figur 4: Innfyllingsmatrise for en AES tilstand

2.2.1.1 Nøkkelutvidelse

Nøkkelutvidelsen har som hovedmål å unngå å benytte samme nøkkel i AddRoundKey funksjonen for hver runde. Skal en 128bit nøkkel benyttes i 10 runder, må den utvides til 1280bit. Dette gjøres på følgende måte som vist i **figur 5**: Illustrasjonsideen er hentet fra Internett[8]

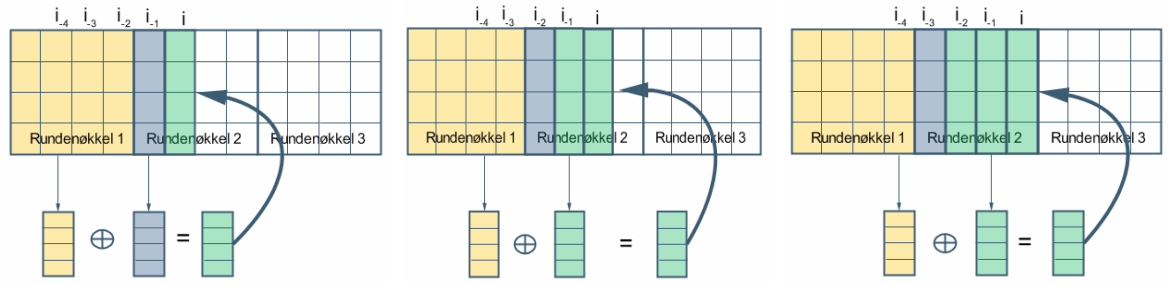


Figur 5: Utvidelse av rundenøkler

Her tas det utgangspunkt i en 128bit nøkkel brukt på 128bits blokker.
Første rekke av hver nye rundenøkkel beregnes ved å angi en variabel i, som sier hvilken rekke det skal beregnes for nå. Foregående rekker vil da bli merket med i_{-1}, i_{-2}, osv. Forrige rad i_{-1} blir kopiert til et buffer, rotert slik at øverste boks havner nederst for så å føres inn i *SubBytes* steget, og tilslutt XOR-et med i_{-4} og *Rcon*. Resultatet blir så kopiert til plass i. Det er så igjen 3 rader før vi igjen utfører denne opperasjonen på nytt, da med ny plassering for i.

[8] AES flash illustrasjon: www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael_ingles2004.swf av Enrique Zabala

Det som skjer med de neste 3 rekkene innen samme rundenøkkel er noe enklere.
Her blir i_{-1} XOR i_{-4} beregnet og lagt til i. Dette vises i **figur 6**.

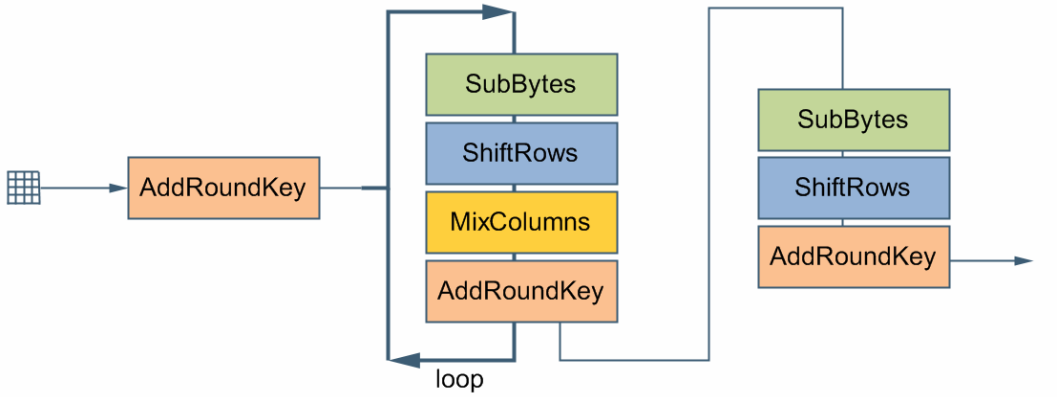


Figur 6: Resterende rader innenfor gjeldende rundenøkkel

Nå som den andre rundenøkkelen er beregnet, og den tredje skal beregnes, benyttes metoden vist i **figur 5** på nytt. Slik beregnes alle rundenøklene til alle aktuelle runder har sin egen rundenøkkel. SubBytes steget som benytter en S-boks, forklares ved gjennomgang av krypteringsprosessen. Rcon er en tabell med konstater der rekke 1 benyttes ved beregning av rundenøkkel 2, rekke 2 benyttes ved rundenøkkel 3 osv.

2.2.1.2 Krypteringsprosessen

Krypteringsprosessen består som nevnt av fire hovedmoduler som gjentas etter et fast mønster. Dette mønstret vil variere med nøkkellengden. Her kommer mønstret til 128/128 versjonen:

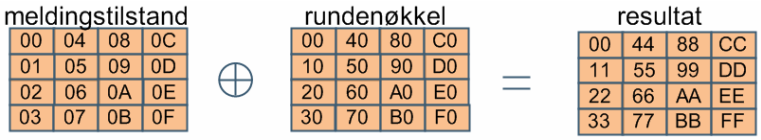


Figur 7: Meldingens gang igjennom krypteringsfunksjonene

En 128bits tilstand er her vist helt til venstre i **figur 7**. Den går først igjennom *AddRoundKey*. Så begynner den på en loop den må igjennom 9 ganger. Til slutt har den 3 steg som utgjør den 10. runden. Merk at *MixColumns* ikke benyttes det siste runden.
Hver av de fire stegene blir beskrevet i detalj i de neste underpunktene.

2.2.1.2.1 AddRoundKey

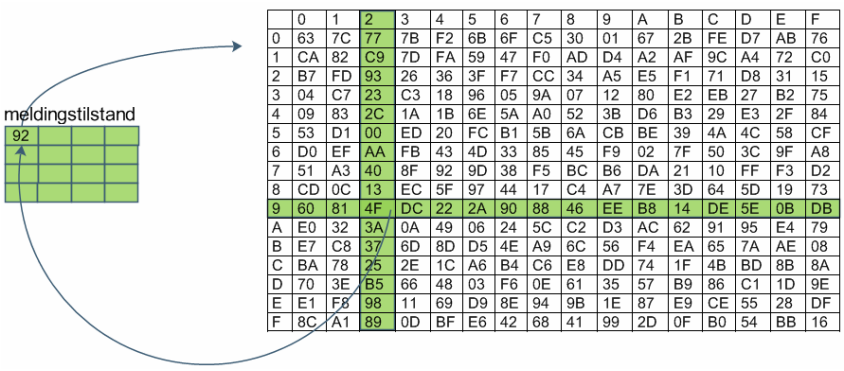
En funksjon som tar rute for rute XOR med tilsvarende rute i rundenøkkelen. Hvilken rundenøkkel som skal benyttes, avgjøres av hvilken runde algoritmen jobber med for øyeblikket.



Figur 8: Melding XOR rundenøkkel utgjør AddRoundKey operasjonen

2.2.1.2.2 SubBytes

En funksjon der hver rute av tilstanden blir byttet ut med et annet 2siffret heksadesimalt tall gitt av en substitusjonsmatrise (*S-box*). Denne substitusjonstabellen er vist i figuren under.



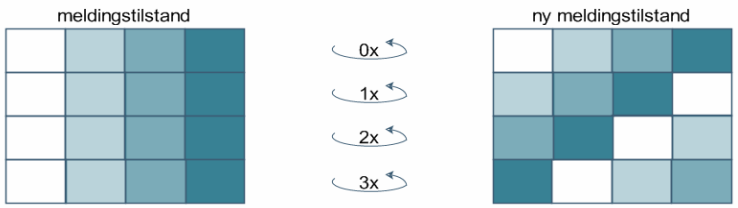
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figur 9: S-box benyttet for ombyttingsoppslag

Måten dette gjøres på er at det for hvert sett av 2 heksadesimale tegn, for eksempel 92, letes det opp på den vertikale aksen for det første tallet, og på den horisontale aksen for det neste. Resultatet blir krysning mellom linjene der dette tallet blir den nye verdien.

2.2.1.2.3 ShiftRows

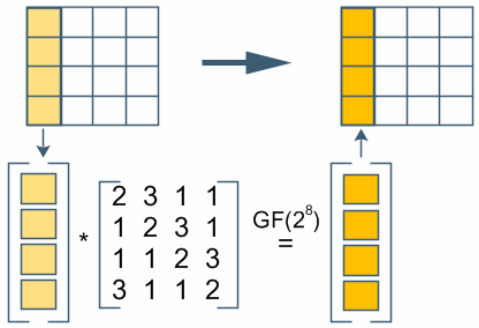
Denne funksjonen roterer tilstanden med forskjellige antall plasser avhengig av hvilken rad det er. Første raden roteres ikke, andre raden én plass til venstre, tredje to plasser og fjerde tre plasser. Dette er illustrert på figuren under.



Figur 10: ShiftRows: før og etter

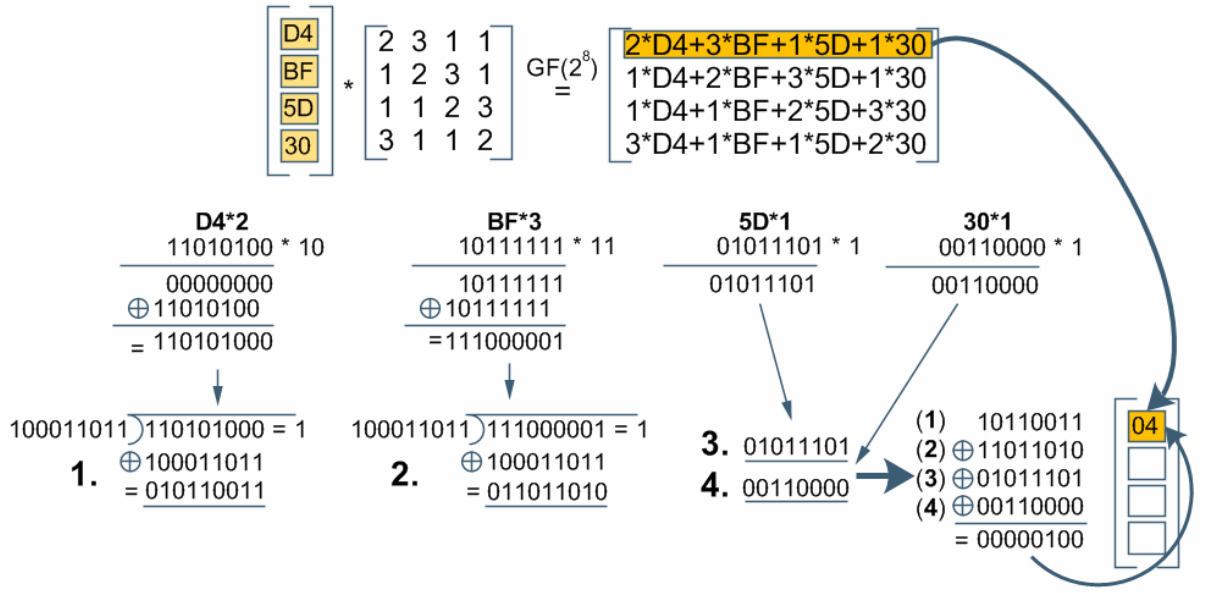
2.2.1.2.4 MixColumns

Denne funksjonen utfører en multiplikasjon rekke for rekke med en vektor: Multiplikasjonen utføres i et endelig felt, og medfører at pluss og gange ikke kan utføres på vanlig vis. Et endelig felt vil si at vi har et endelig lovlig verdier resultatet kan bli, uansett hva slags regneoperasjoner som blir utført. Det er 8 bit i hver rute i tilstanden, og det gir lovlige verdier mellom 0 og 255 desimalt (00 og FF heksadesimalt). Dette håndheves ved moduloregning.



Figur 11: MixColumns behandler én og én rekke over et endelig felt

For å vise hvordan dette blir gjort på et detaljert nivå, kommer her et eksempel på hvordan dette kan regnes ut for hånd. Matematikken bak dette er svært avansert, så kun fremgangsmetoden vises. Produktene blir alltid tatt modulo 100011011.



Figur 12: Utregning av én rute i MixColumn steget

Som vist i **figur 12**, blir det benyttet matriseregning for å gjøre en [1,4] og en [4,4] matrise om til en ny [1,4] matrise. Neste steg er å multiplisere sammen enkeltparene, der XOR erstatter PLUSS. Det blir tatt modulo binært 100011011, her vist som divisjon, og restene blir tilslutt XORet sammen.

2.2.1.3 Invertert AES

Det er noen tilpassninger for å kunne benytte 192 og 256 bit versjonene, men i prinsippet er det samme fremgangsmåte. For å kunne gå andre veien, det vil si fra en kryptert melding og tilbake Dette gjøres ved at meldingen blir sent motsatt vei av det den ble beskrevet tidligere i **figur 7**.
I tillegg må de fire operasjonene omdefineres:

AddRoundKey: XOR er seg selv invertert når benyttet 2 ganger, så denne forblir lik.
SubBytes: Her benyttes en invertert S-Box for oppslag. Viser ikke denne da det viser seg at den ikke er relevant for resten av oppgaven.
ShiftRows: Omstokking av ruter til høyre, istedenfor venstre, like mange ganger som før.
MixColumns: Her benyttes en annen matrise: **figur 13.**

$$\begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix}$$

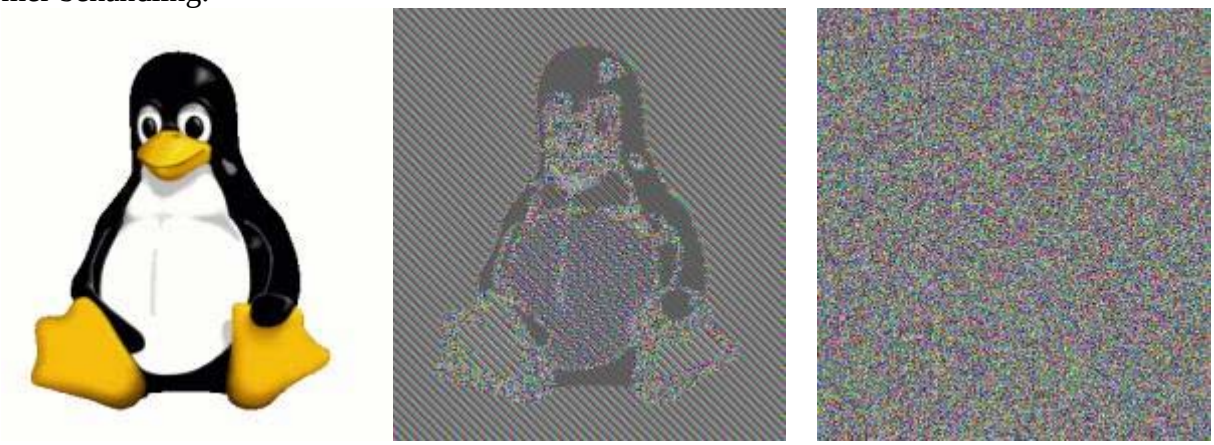
Figur 13: Invertert MixColumn matrise

2.2.2 Counter Mode

For å kunne bruke en blokkorientert krypteringsalgoritme må det være et rammeverk rundt den som behandler innmating av data. Algoritmen bearbeider som kjent hele tiden med blokker på 128bit, og det er veldig sjeldent det kun er128bit nyttedata. Det kan være mindre, men oftest er det mye mer. Da må meldingen deles inn i 128bits deler, og siste rest fylles slik at alle de 128 bitene i siste meldingsdel har en verdi.
Den enkleste bruksformen ville være å mate en og en blokk av 128 meldingsbit inn i AES og anse resultatet som kryptert. Denne måten å bruke blokkoder på blir kalt ”elektronisk kodebok” eller ECB-mode og er den enkleste av flere modus tilgjenglig. Men hvorfor er ikke ECB godt nok?

Det en blokkorientert algoritme gjør, er i prinsippet bare å utvide alfabetet benyttet for å representere meldinger. Med ASCII som benyttes for å skrive denne rapporten benyttes 8bit for å representere et tegn. 8bit er som kjent 256 mulige kombinasjoner, men alle kombinasjoner benyttes ikke. De mest brukte er a-z, A-Z, 0-9, pluss diverse spesialtegn. Disse utgjør om lag ~100 av de 256 kombinasjonene. (De mest brukte tegnene ligger innenfor de første 7 bitene) AES samler så opp flere tegn i en pakke (her 16 tegn), og omformer det til ett nytt tegn som er 128bit, tilsvarende 3,4e+38 kombinasjoner. Dette tallet er så stort at det er svært vanskelig å forholde seg til det. Det finnes ingen sammenheng i transformasjonen, så ingen uten nøkkelen forstår det nye alfabet. Analyse av frekvensen til enkeltbokstaver blir også meningsløs da denne variasjonen blir utjevnet i denne prosessen. Det reelle antall kombinasjoner på de nye alfabetet blir naturligvis ikke fullt så høyt som nevnt her i stad, fordi vi i utgangspunktet ikke benyttet oss av alle kombinasjoner i originalalfabetet.

Problemet med frekvensanalyse er dessverre ikke helt borte. Enhver lik kombinasjon av 128bit gir samme krypterte 128bits melding ut hver gang. Et eksempel som viser hvorfor dette er skadelig er klippet ut fra en artikkel på Wikipedia[9] der bildedata er kjørt gjennom AES uten noe mer behandling.



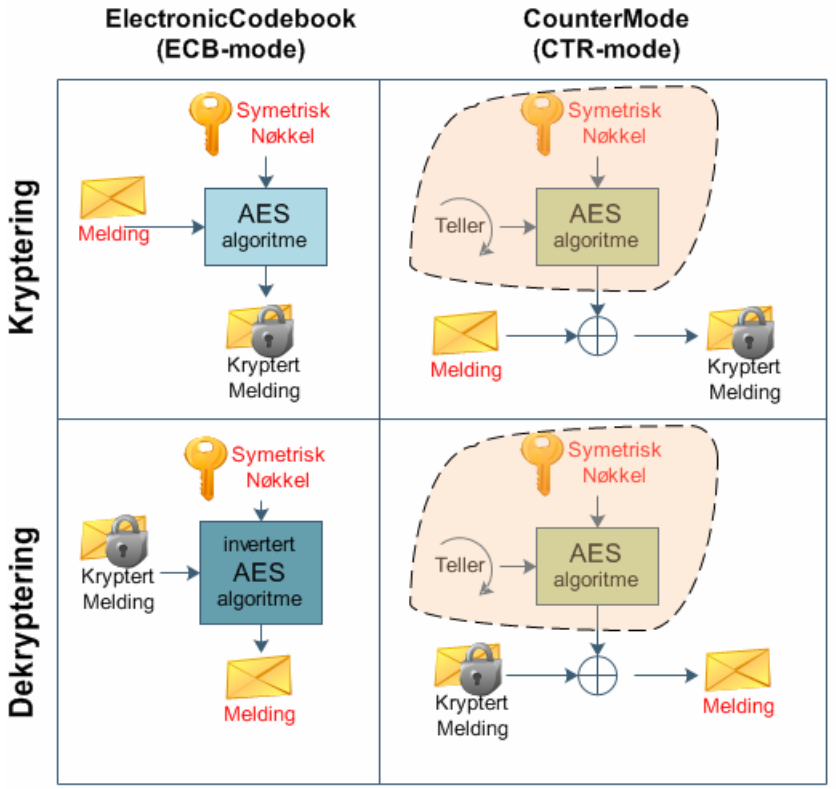
Figur 14: Originalbilde, kryptert ”feil” med AES, kryptert riktig med AES

Bildet i midten av **figur 14** er kryptert med AES, men det er fortsatt mulig å ”se” bildet selv om hver farge/fargeovergang ikke lenger har riktig verdi. Dette fordi hvitt alltid blir oversatt med samme krypterte ordet hver gang. Det samme gjelder naturligvis for alle andre farger i bildet.

[9] Block cipher modes of operation: en.wikipedia.org/wiki/Modes_of_operation av 13.Oktober 2007

Bildet helt til høyre har implementert en operasjonsmodus som bruker resultatet fra sist blokk, en teller eller en annen form til tilbakekobling, inn i neste blokkberegning. Slik blir resultatet ren støy. Støy er nøyaktig det vi ønsker at resultatet skal se ut som for alle andre enn oss.

Det er kommet opp med mange forskjellige operasjonsmåter**.** Den Thales har valgt å bruke i sin TCE621 er en som heter counter mode (CTR). Prinsippet for denne er vist i **figur 15.**



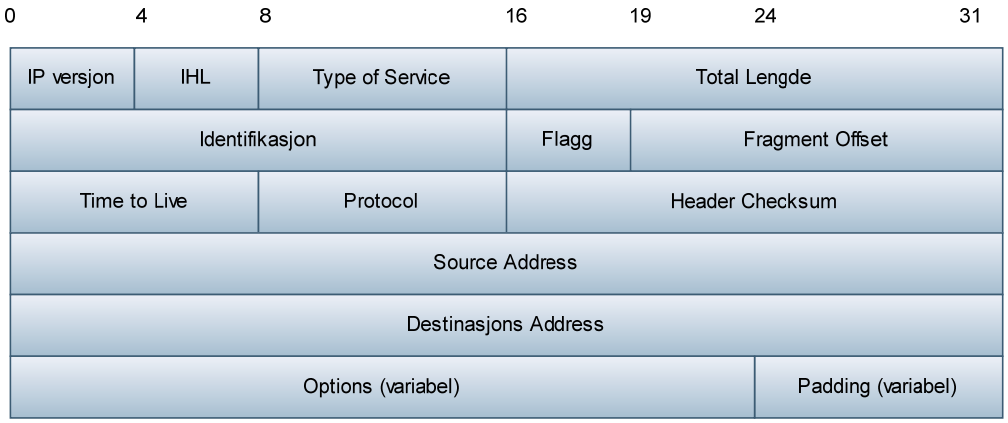
Figur 15: Hvordan ECB- og CTR-mode fungerer ved kryptering og dekryptering

Istedenfor å mate algoritmen med meldingen, maters den med en verdi som telles oppover for hver blokk. Resultatet av krypteringen XOR-es med meldingen, gir oss det vi sender til motparten. Det vi med andre ord gjør her, er å generere en form for one-time-pads[10]. Det er kritisk at det aldri blir gjenbruk av verdier telleren har benyttet før. Telleren må derfor være tilstrekkelig stor (antall kombinasjoner). Fordelene med denne operasjonsmodusen er at det ikke er behov for en invers AES algoritme for dekryptering. Dette igjen medfører at algoritmen ikke er avhengig av meldingen og det kan forhåndsberegnes materiale på begge sider av kanalen som kan benyttes når data skal sendes eller mottas. Ulempen er at telleren må koordineres på begge sider av kanalen, samt at det er fint mulig å implementere dette svært usikkert, noe WEP[11] algoritmen er et eksempel på.

[10] Det perfekte krypteringssystemet. Umulig å dekryptere. Ulempen er at den krever like mye tilfeldig nøkkeldata som nyttedata den skal kryptere, og dette nøkkelmaterialet må overføres til mottakeren på en trygg måte.
[11] WEP står for Wireless Equalent Privacy og er en sikkerhetsfunksjon for å beskytte trådløse nettverksforbindelser. Den har i etterkant vist seg å være særdeles svak ved at den gjenbruker nøkkeldata om igjen.

2.2.3 IPsec

IPsec[12] [13] (Internet Protocol Security) er et rammeverk av protokoller som sørger for konfidensialitet, integritet og autentisering av data mellom to eller flere parter over et IP basert nettverk. IPsec er plassert i nettverkslaget i OSI modellen, noe som medfører at den kan beskytte protokoller, som UDP og TCP, i transportlaget og over.
IP er som kjent protokollen som ser til at data på et IP basert nett kommer frem til riktig mottaker. Figurene under viser hvordan IP versjon 4 er bygd opp. IPv4 er fremdeles den som er mest brukt og det er også denne versjonen vi vil konsentrere oss om videre.



Figur 16: IPv4 header

IPsec er i hovedsak implementert med et sett av kryptografiske protokoller. Dette for å skape sikker pakkeflyt, gjensidig autentifikasjon og for å etablere kryptografiske parametere. Nærmere forklart er arkitekturen bak IPsec bygd på et konsept som innebærer bruken av algoritmer og nøkler for å kunne kryptere og autentisere datastrømmer mellom kommunikatorer. IPsec gir muligheten til å benytte flere forskjellige typer krypteringsalgoritmer og autentiseringsmetoder. Det er opp til den enkelte implementasjon hvilke alternativer som vil være tilgjenglige.

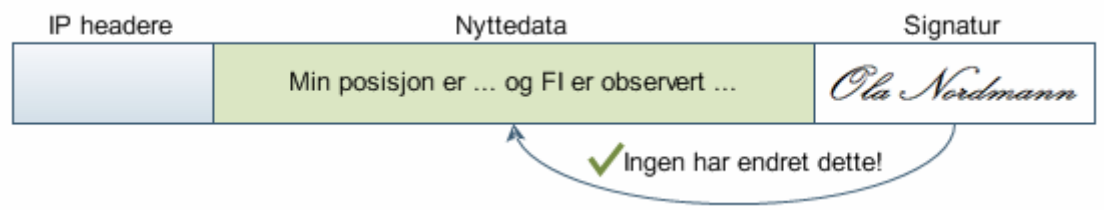
12 An Illustratated Guide to IPsec: www.unixwiz.net/techtips/iguide-ipsec.html av Steve Friedl
13 RFC 2406

2.2.3.1 AH og ESP

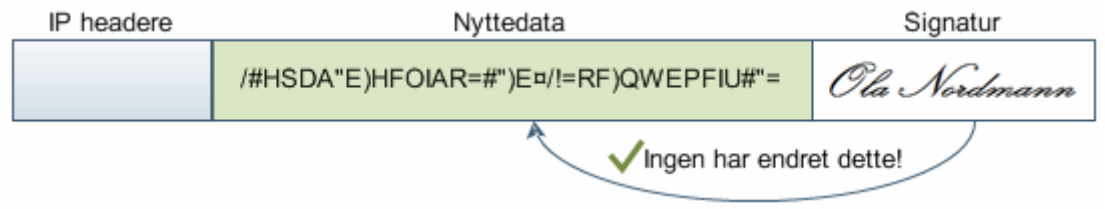
IPsec består av to hovedprotokoller:

1. AH (Authentication Header)
2. ESP (Encapsulating Security Payload)

AH tilbyr autentisering, integritet og anti-replay, mens ESP i tillegg til de nevnte egenskapene også tilbyr konfidensialitet (kryptering) for IP payloaden. Payload tilsvarer nyttedata.

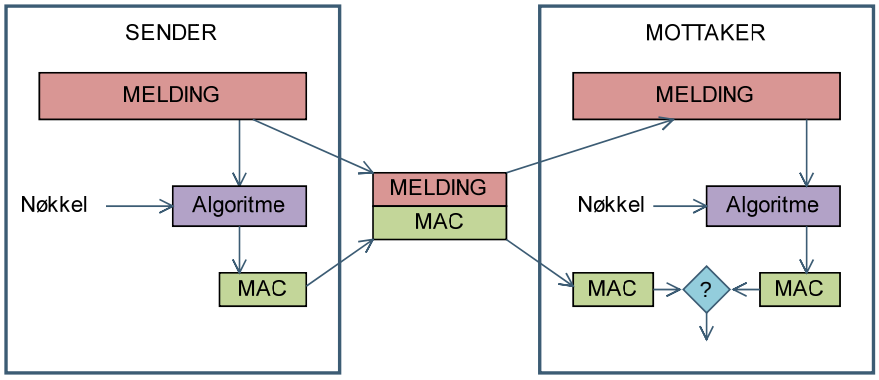


Figur 17: Authentication Header



Figur 18: Encapsulating Security Payload

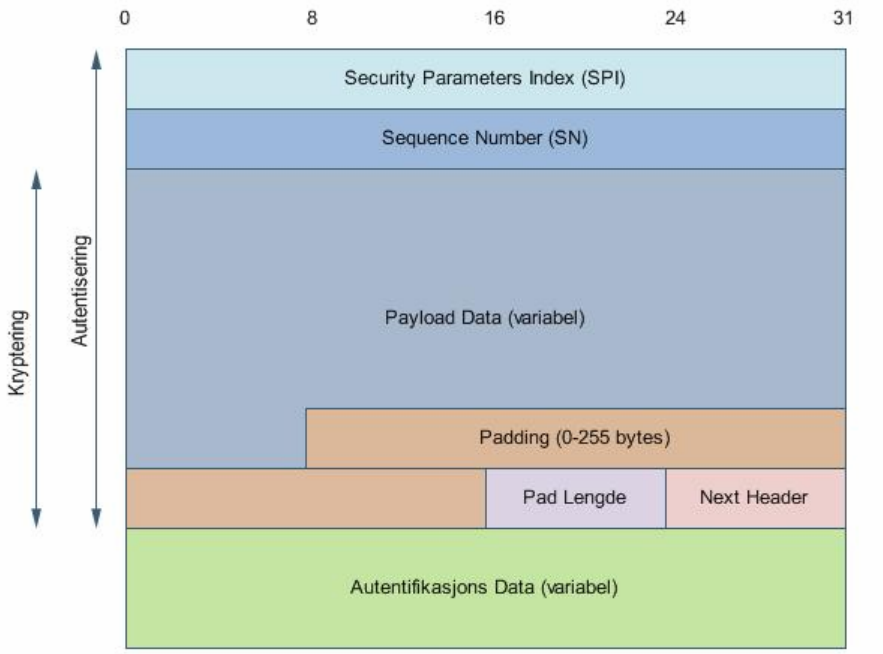
MAC[14] blir benyttet for å autentisere motpartens meldinger. MAC står for Message Authentication Code, og utfører selve autentiseringen. Det vil si at det blir benyttet en hemmelighet (nøkkel) mellom de involverte partene som blir benyttet til å generere en liten blokk med data som er basert på meldingen, hemmeligheten og en hash funksjon. Den lille blokken vil så bli lagt til den originale meldingen før den sendes over nettet. Mottakeren utfører samme prosedyre som senderen, og sjekker om resultatet blir det samme. Dersom de er identiske vet mottaker at meldingen ikke er blitt tuklet med og at meldingen kommer fra den påståtte senderen.



Figur 19: MAC beregning

[14] Network Security Essentials, Third Edition, William Stallings, Pearson Prentice Hall, 2007: Kapittel 3

ESP krypterer payload, padding, pad length og next header, se **figur 20**. Dersom den krypteringsalgoritmen som blir benyttet krever kryptografisk synkroniseringsdata, som for eksempel en initialiserings vektor (IV), så vil denne dataen bli plassert før payloadfeltet. Dersom en slik IV er inkludert vil den ikke være kryptert. Krypteringen blir gjennomført ved bruk av en symmetrisk krypteringsalgoritme i samsvar med en hemmelig nøkkel. IPsec er kompatible med mange ulike algoritmer, men de aller vanligste er 3DES, RC5, IDEA, CAST, AES og Blowfish.



Figur 20: Ipsec ESP header

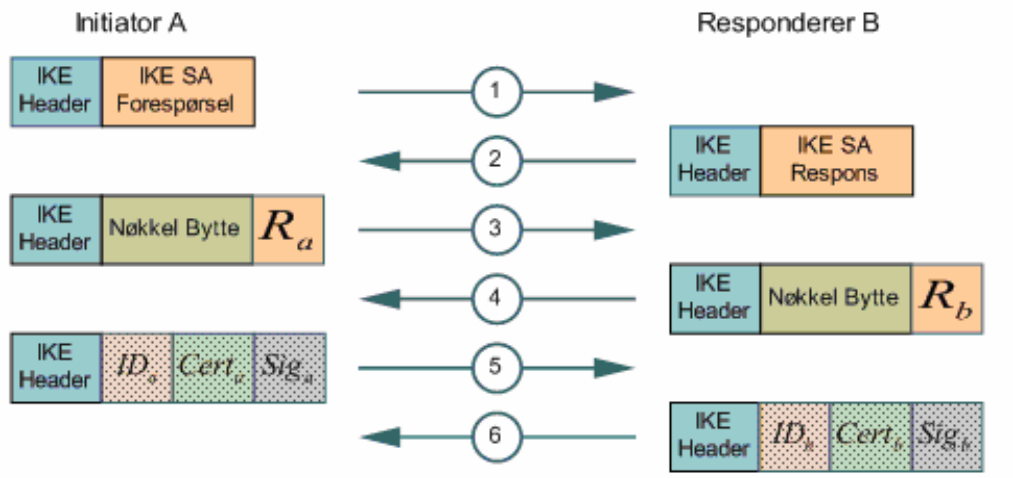
- **SPI**: Identifiserer sikkerhets parameterne i kombinasjon med IP adressen.
- **SN**: Et tall som øker jevnt for hver sendte pakke, fra en gitt posisjon, for å unngå re-play.
- **Payload data**: Dataen som skal sendes.
- **Padding**: Da dataen som skal sendes ikke alltid fyller ut hele blokken vil det her bli lagt til ekstra nullere ved behov.
- **Pad length**: Størrelsen på paddingen.
- **Next header**: Henviser til protokollen som blir sendt og beskyttet.
- **Autentifikasjons data**: Inneholder en MAC som er nødvendig for og kunne autentisere pakken.

Som nevnt benyttes det hemmelige nøkler for å få utført hash funksjoner og krypterings-algoritmer. En metode å fordele hemmelige nøkler på er at en administrator genererer et sett med nøkler for så å manuelt distribuere dem til de andre involverte partene. Dette kan fort bli en tidkrevende og tung metode dersom de involverte er spredt utover et stort geografisk område.

En annen metode, som i dag er mer brukt på Internett, er å benytte IKE som står for Internet Key Exchange. Måten dette gjøres på er at IKE protokollen hos hver part sender noen meldinger frem og tilbake for bli enige om hemmeligheter. Dette gjøres trygt ved bruk Diffie-Hellman[15] algoritmen. Avtalt informasjon utgjør en security association (SA) i IPsec protokollen.

[15] En algoritme der hver part genererer to tall. Et privat og et offentlig. Det offentlige tallet blir utvekslet mellom partene på et usikkert nettverk. Ved bruk av hverandres offentlige og sin private nøkkel kan hver part komme frem til en felles hemmelig nøkkel bare disse to partene vet om.

ISAKMP (Internet Security Association Key Management Protocol) er rammeverket IKE benytter for at de involverte partene skal kunne etablerer kompatible SA i begge ender. **Figur 21** viser hvordan IKE oppretter kontakt og avklarer delte hemmeligheter ved bruk av Diffie-Hellman. Videre benyttes denne hemmeligheten for å utveksle andre datatyper om ønsket. Dette kan være sertifikater, signaturer, ID-er og liknende.
Denne utvekslingen skjer i forkant av all videre kommunikasjon med ESP eller AH.



Figur 21: Utveksling av hemmelig data ved hjelp av IKE[16]

I denne oppgaven vil ikke denne form for nøkkelfordeling bli gjennomført, da TCE621 i operativ tilstand får tildelt de nøklene den trenger fra et management center. I denne oppgaven blir nøklene manuelt satt som forhåndsdelte nøkler.

2.2.3.2 Tunnel mode og Transport mode

IPsec tilbyr to forskjellige metoder for sending av IP pakker:

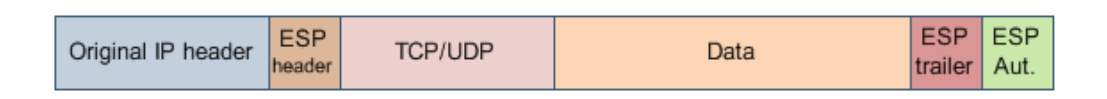
- Transport modus
- Tunnel modus.

Transport modus, **figur 22**, blir vanligvis brukt mellom to brukere. Her blir bare payloaden kryptert og/eller autentisert. På bakgrunn av det vil ikke rutingen bli berørt. Med denne metoden vil transport- og applikasjonslaget alltid være beskyttet i henhold til AH eller ESP.

Figur 22: Transport modus

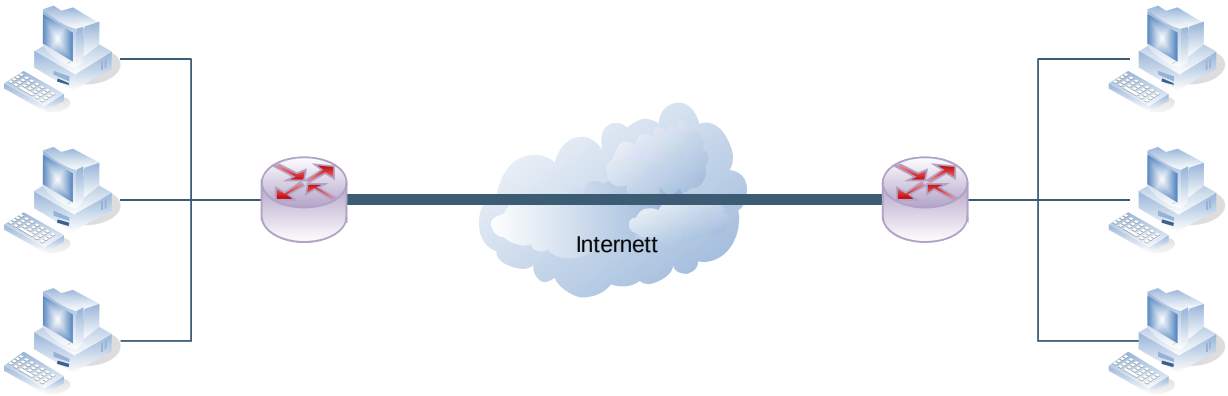
[16] RFC 2409

Ved bruk av ESP i transport modus, **figur 23**, vil bare payloaden (data) bli kryptert mens den originale IP headeren forblir inntakt. Det blir lagt til en egen ESP header mellom den originale IP headeren og payloaden. Etter payload kommer en ESP trailer som også inneholder autentiseringsdata for ESP og payload.



Figur 23: ESP i transport modus

Tunnel modus, se **figur 24**, blir vanligvis benyttet mellom to gatewayer, altså mellom to nettverk. Her blir hele IP pakken kryptert, og derfor blir det lagt til en ny IP header utenpå, slik at ruting over det eksterne nettverket blir mulig.



Figur 24: Tunnel modus

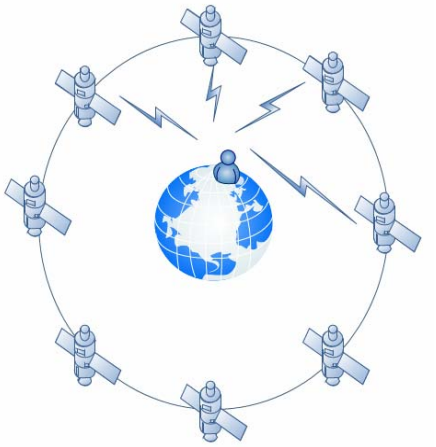
Figur 25: ESP i tunnel modus

Når det skal utføres både kryptering og autentisering på IP pakker er det krypteringen som blir utført først. Autentiseringsdata blir så beregnet på grunnlag av det krypterte resultatet. Hos mottaker blir autentiseringen først etterprøvd, og dersom den ikke godtas vil pakken bli kastet. På denne måten unngås opphoping av falske pakker, og dette minimerer muligheten for resurstømmende DoS[17] angrep da det ikke blir behov for å kjøre pakken igjennom et dekrypteringsteg.

[17] Tjenestenektingsangrep er en eller flere enheters resurser blir overbelastet og ikke lenger er i stand til å yte service til gyldige brukere.

2.2.4 GPS kommunikasjonsprotokoll

GPS står for Global Positioning System[18] og er et satellittbasert navigasjonssystem. Dette systemet består av minst 24 satellitter som går i bane rundt jorden, og er mest kjent under navnet NAVSTAR. Satellittene kretser to runder rundt jorden i løpet av et døgn mens de kontinuerlig sender radiosignaler i lysets hastighet. Alle satellitter har en innebygd atomklokke. Og det er ved hjelp av blant annet disse klokkene at posisjoner blir kalkulert. En GPS-mottaker tar i mot radiosignalene og kalkulerer en så nøyaktig posisjonering av mottakers plassering som mulig. Dette gjøres ved å bruke tiden satellittene sender, posisjonen til satellittene, forsinkelsene i signalet og sendetidsforholdet mellom de respektive satellittene. Atomklokkene i satellittene synkroniseres til GPS time, mens de fleste klokkene her på jorden synkroniseres mot UTC (Coordinated Universal Time). Det er en den lille forskjellen i tid mellom klokkene på grunn av jordens rotasjon, men denne er det tatt høyde for i utregningene.
Nøyaktigheten avhenger av antallet satellitter GPS-mottakeren er i kontakt med. For å kunne kalkulere lengdegrad (øst/vest) og breddegrad (nord/sør) er det behov for tre satellitter, og ved kontakt med en fjerde er alle tre dimensjonene på plass.



Figur 26: GPS (NAVSTAR)

2.2.4.1 NMEA 0183

NMEA 0183[19] [20] er en standard som spesifiserer et kommunikasjonsformat mellom ulike enheter. Det kan være navigasjonsutstyr, dybde målere og GPS mottakere.
Kommunikasjonen NMEA 0183 beskriver går ut på at det er en sender ”talker” som sender informasjon ved bruk at setninger ”sentences” til en eller flere mottakere ”listeners”.
Disse setningene er formatert som ASCII kode på seriell form, og inneholder den informasjonen som produsenten ønsker gjort tilgjenglig. Det er ingen standard på hva setningene skal inneholde, ettersom NMEA 0183 benyttes i flere typer produkter, med selve formatene på de forskjellige setningene er standardisert.

[18] Global Positioning System: en.wikipedia.org/wiki/Global_Positioning_System av 18.November 2007
[19] The NMEA 0183 Protocol : fort21.ru/download/NMEAdescription.pdf
[20] NMEA 0183: en.wikipedia.org/wiki/NMEA_0183 av 8.November 2007

Det er tre grunnleggende former for disse setningene:

- Talker Sentences
- Proprietary Sentences
- Query Sentences

Talker Sentences er den setningen sender avgir til mottaker. Hver setning starter med $ tegnet, og avsluttes med <CR><LF>. Hele meldingen kan inneholde opptil 83 tegn.
Setningens generelle format utover dette er bygd opp på følgende måte:

$ttsss,d1,d2,…<CR><LF>

De to første bokstavene etter $ tegnet er "talkers" ID mens de neste tre er setningens ID.
Den påfølgende dataen følger så etter atskilt med komma.

Proprietary Sentences er standarden som tillater den enkelte produsenten å legge til egen informasjon. Disse setningene starter med "$P", med tre påfølgende bokstaver for å identifisere produsenten og deretter den dataen produsenten ønsker å ta med. Resten av setningen følger standarden til "talkers" setning.

Query Sentences er en type spørresetning fra "listener" til "talker" hvor mottakeren kan be om en bestemt type setning fra sender. Et eksempel kan være en datamaskin som ønsker koordinater fra en GPS, da vil setningen se følgende ut:

$CCGPQ,GGA<CR><LF>

Her viser de to første bokstavene "CC" til hvem som spør, de neste to "GP" til hvem som skal svare. Den femte bokstaven i setningene er alltid Q, dette er for å definere at setningen er en spørresetning. "GGA" (Global Positioning Fix Data) er så et eksempel på type data som blir etterspurt.

PLGR (Precision Lightweight GPS Receiver) er en håndholdt GPS med fem kanaler og singel frekvens, og er den GPS-en som det Norske Forsvaret benytter seg av. Den tilbyr posisjon, hastighet og tid, samt innebygd anti-spoofing og anti-jamming.

Figur 27: Forsvarets GPS mottaker (PLGR)

2.2.4.2 GPS tilkobling

GPS benyttet i denne oppgaven er typen PLGR som nevnt ovenfor. Enhver GPS bruker noe tid på å synkronisere med satellittene og finne sin posisjon etter lengre tid avslått. Selve GPS mottakeren må konfigureres for å sende ut informasjon til PC. Først må den kobles til PC. Her er det benyttet COM1 porten med tilhørende kabel mot GPS. Videre oppsett må utføres på GPS mottakeren.

Under MENY velges SETUP. 5 sider ned er det en meny som heter SETUP I/O. Ved å velge *Custom* i stedet for *Standard* på valget SERIAL, får man muligheten til å konfigurere ekstern tilkobling etter eget ønske. Dette valget medfører at man får opp ekstra sider under gjeldende side.

På den første siden velges formatet på informasjonen som skal overføres og til hvilken hastighet den skal ha inn og ut fra GPS-en. I det andre stiller man inn hvilken informasjon man ønsker å få overført. Dette gjøres etter NMEA egne protokoller.

Side 1 *Side 2*

I vårt tilfelle trenger vi GLL og ZDA, som henholdsvis står for posisjon og tid.
Disse to NMEA setningene inneholder den informasjonen som er aktuelle for dette prosjektet.

GLL er representert på følgende måte:
(Geographic Position – Latitude/Longitude)

```
        1       2 3       4 5         6 7
        |       | |       | |         | |
$--GLL,llll.ll,a,yyyyy.yy,a,hhmmss.ss,A*hh
```

1) Breddegrad
2) N eller S (Nord eller Sør)
3) Lengdegrad
4) E eller W (Øst eller Vest)
5) Tid (UTC)
6) Status A – gyldig data, Status V – ugyldig data
7) Checksum

ZDA er representert på følgende måte:
(Time & Date – UTC, Day, Month, Year and Local Time Zone)

```
        1          2  3  4    5  6  7
        |          |  |  |    |  |  |
$--ZDA,hhmmss.ss,xx,xx,xxxx,xx,xx*hh
```

1) Tid (UTC)
2) Dag, 01 til 31
3) Måned, 01 til 12
4) År
5) Tidssone forskyvning i minutter
6) Tidssone forskyvning i timer, 00 til +/- 13 timer
7) Checksum

Figur 28 viser den informasjonen som nå dukker opp via COM porten. NMEA setningen GGA vises også, og inneholder noe av den samme informasjon som GLL og ZDA. Bokstavene ”GP” foran hver setning beskriver hvilken type enhet ”talker” vi har kontakt med, hvor ”GP” er forkortelsen for en GPS enhet.

```
$GPGGA,180911.00,6108.738,N,01022.981,E,1,04,24.50,138.7,M,-40.9,M,,*44
$GPGLL,6108.738,N,01022.981,E,180911.00,A*06
$GPZDA,180912.00,12,11,2007,00,00*63
```

Figur 28: Utsnitt fra GPS overføring i NMEA lest fra PC

2.3 Arkitektur og Design

2.3.1 Dokumentasjonen

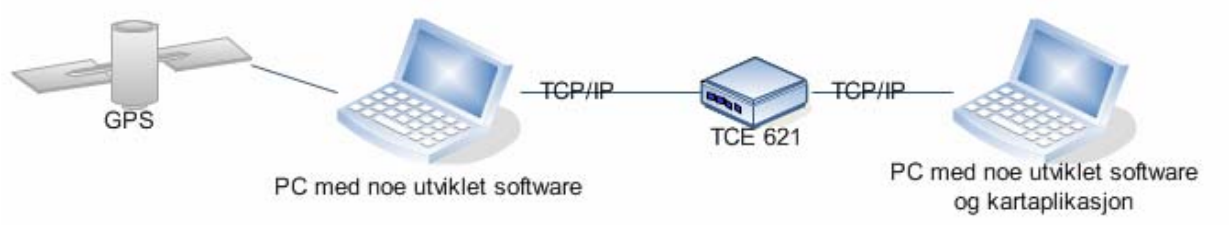
Sikkerhetsprotokollen Thales benytter i TCE621 AES er beskrevet i et eget konfidensielt dokument av 29.08.2007. Dette er grunnlaget for design av løsning og følgende referanser er oppgitt for sikkerhetsprotokollen:

FIPS 197 – Advanced Encryption Standard (AES)
NIST 800-38A – Recommendation for Block Cipher Modes of Operation
NIST 800-38B – The CMAC Mode for Authentication
RFC 2406 – IP Encapsulating Security Payload

Det er forklart at en form for ESP er benyttet i tunnel mode. Det offisielle designet på ESP pakkeformatet er beskrevet i detalj under **punkt 2.2.3.1** og er grunnlaget for de videre endringene Thales har gjort med denne protokollen. AES i counter mode benyttes etter den offisielle standarden, der nøkkelen har lengden 256bit. InitialVektor, forkortet IV, blir brukt for å igangsette counter mode, der et LFSR (Linear Feedback Shift Register) sørger for å håndtere måten IV blir telt opp på. For å beregne MAC/ICV koden, blir CMAC benyttet. CMAC er ikke endret og følger den offisielle standarden.

2.3.2 Realiseringsplan

Målet med prosjektet er ikke å designe en enhet tenkt benyttet ute i felt, men å konstruere et laboratoriemiljø som simulerer nettverkstrafikk som om det skulle ha kommet fra en slik enhet. Thales har stilt deres TCE621 AES versjon tilgjenglig for testing, og prosjektet har som mål å ved bruk av en datamaskin koblet til en GPS, generere trafikk over et TCP/IP nettverk som TCE621 skal kunne forstå og sende videre til en datamaskin som simulerer et KKIS.
Dette er vist i **figur 29**.



Figur 29: Oppgavens laboratoriemiljø

Prosjektgruppen har valgt å dele implementeringsdelen av prosjektet inn i tre delmål. Gjennomføringen av delmålene vil bli utført i kronologisk rekkefølge, hvor hvert delmål bygger på det foregående og av den grunn må ferdigstilles før neste delmål iverksettes.

1. Være i stand til å sende trafikk fra et program ut på nettet, samt å kunne ta imot trafikken på en annen maskin og vise denne samme informasjonen. Dette vil være grunnfunksjonaliteten som alt annet bygger på og er derfor ikke et mål i seg selv.

2. Oppnå kryptering fra sendermaskin på trafikken sent til mottakermaskinen. TCE621 tar seg av dekrypteringen for mottakermaskinen. Dette undermålet anses for å være det mest krevende.

3. Overføre posisjonsmeldinger fra GPS til systemet vårt, og videre få dette kryptert og sent automatisk.

I tillegg til dette har det vært nevnt å få vist posisjonene i et kartsystem, men på grunn av tilgjenglig tid, samt at dette vil være helt ut mot grensen av hva kjernen av oppgaven er, har gruppen valgt å ikke bruke tid på denne funksjonaliteten.

2.3.3 Plattform og designvalg

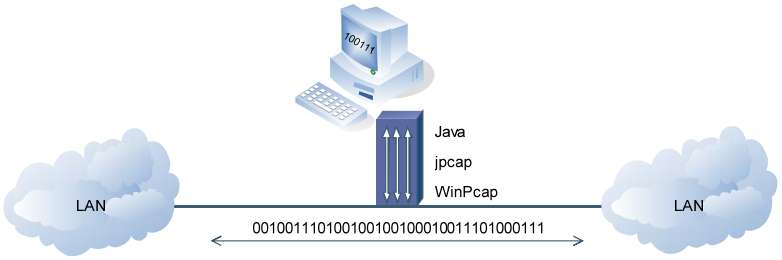
For å realisere et miljø hvor det er mulig å nå de respektive målene, må det avklares hvilken plattform som skal benyttes. Valgene det endte opp med ble operativsystemet Windows XP med programmeringsspråket Java. Videre er funksjonaliteten til programmet delt inn i et egenprodusert rammeverk bestående av moduler. Modulene er delvis egenproduserte, og delvis integrert fra eksisterende kode.

Java har ikke muligheten til å sende vilkårlig data ut på nettverkskortet. Den kan bare sende TCP og UDP pakker. Ønskes det å sende pakker på et lavere nivå i OSI modellen (lag 4) må ekstern mellomvare benyttes. Måten dette løses på er å laste ned en kerneldriver

Rammeverket er derfor avhengig av direkte tilgang til nettverkskortet, og det stod mellom to valg her. Kerneldriverne TUN/TAP[21] og Pcap[22]. TUN/TAP drivere oppretter virtuelle nettverkskort i et operativsystem. TAP er driveren som kobler seg til datalinklaget og fungerer som en bro, mens TUN driveren kobler seg til transportlaget, som regel IP laget, og kan brukes for å lage IP tunneler.

Valget endte på Pcap, et API (Application Programming Interface) som tillater kapring og sending av nettverkspakker direkte opp til userspace programmer ved å gå utenom nettverksstacken i operativsystemet på maskinen. Pcap er grunnsteinen i mange open source og kommersielle nettverksverktøy som for eksempel IDS-er, tcpdump og Wireshark.

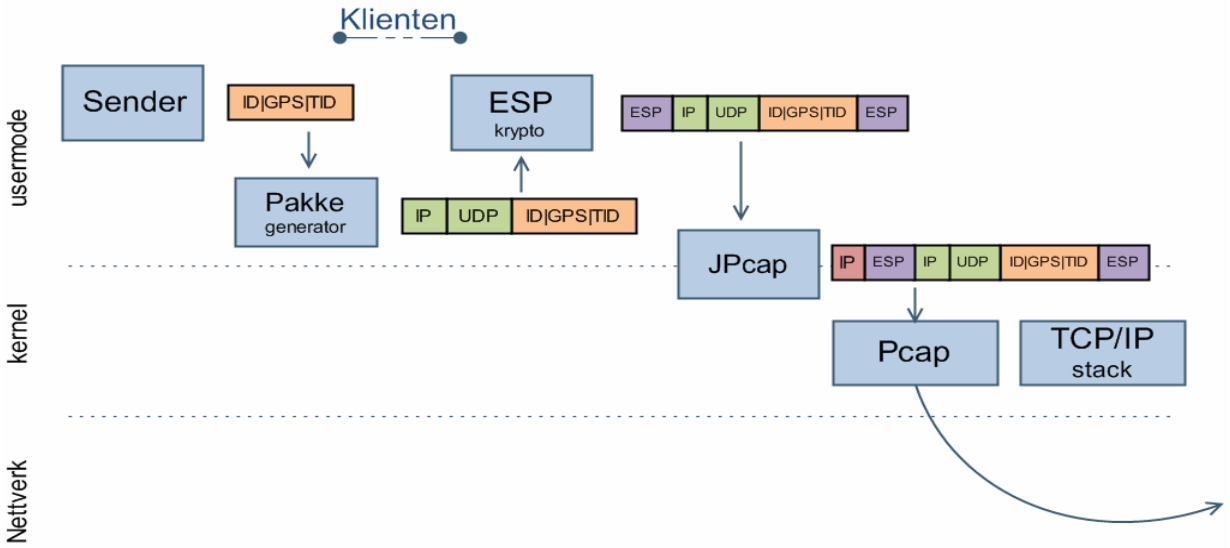
Videre trengs et bindeledd mellom Pcap og Java. Dette bindeledd heter Jpcap[23] og tilbyr et API for bruk i javakode. Dette er illustrert i **figur 30**.



Figur 30: Hvordan Pcap fungerer

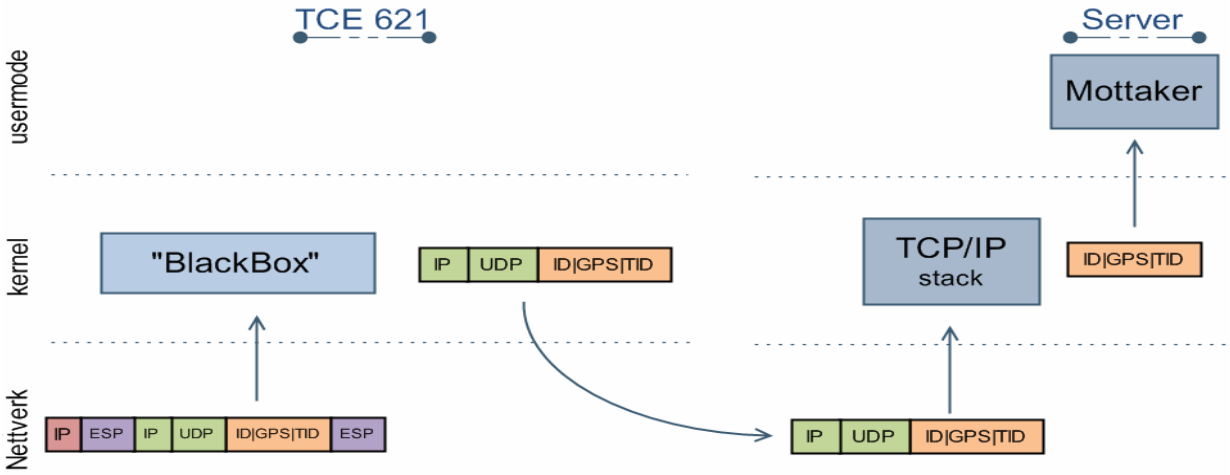
Som **figur 31** viser blir ulike innpakkingsoppgaver utført i separate steg. For å forklare figuren er usermode det nivået brukeren har tilgang til, typiske brukerprogrammer. Kernel er kjernen i operativsystemet, og håndterer maskinvareressurser. Vanligvis har ikke brukeren tilgang på dette nivået, noe Pcap i dette tilfellet overkjører.

[21] Universal TUN/TAP driver: vtun.sourceforge.net/tun/faq.html av Maxim Krasnyansky
[22] Pcap: http://en.wikipedia.org/wiki/Pcap av 7.November 2007
[22] The Windows Packet Capture Library: www.winpcap.org/
[23] Jpcap – a Java library for capturing and sending network packets: netresearch.ics.uci.edu/kfujii/jpcap



Figur 31: Oppgavene senderen må utføre

Det som skjer videre er at TCE621 mottar nettverkstrafikken Pcap sender ut, dekrypterer den og sender informasjonen til mottakermaskinen på sikker side. Dette vises i **figur 32**.



Figur 32: TCE621 mottar data, dekrypterer og sender det videre til mottakeren

I vedlegg B er det illustrert med et enda høyere detaljnivå forklarer hvordan headere er satt sammen. Den forklarer også hvordan ESP pakkeformatet til Thales er bygget opp. Deler av dette er konfidensielt og derfor skilt ut.

2.4 Implementasjon

På den konfidensielle laptopen ble det installert *Windows XP med SP2*. Java-versjonen benyttet heter *Development Kit versjon 6*, som også inneholder Java Runtime Environment for kjøring av java-programmer. *WinPcap 4.0.1* ble benyttet som bindeledd ut mot nettverkskortet, samt *Jpcap 0.7* som bindeledd mellom *WinPcap* og Java.
Videre ble *TextPad 4.7* benyttet for å programmere koden. Dette programmet fargelegger koden og forenkler kompilering og kjøring ved hurtigtaster inne i programmet.
For å sjekke hva som faktisk blir sent ut fra maskinen ble *WireShark 0.99* benyttet. Det er en meget god netverkssniffer med god protokollforståelse og fint grensesnitt. Det er verd å nevne i denne sammenheng at nettverksgrensesnittet må ha tilkoblet en last for at Wireshark skal starte og lytte etter pakker.

All kode er skilt ut i **vedlegg A - Implementasjonskode** med bakgrunn i av deler av koden er gradert konfidensielt, samtidig som at denne separasjonen skaper en større oversikt i dokumentet. Forklaringer på koden er å finne i punktene som følger.

2.4.1 Grunnfunksjonalitet

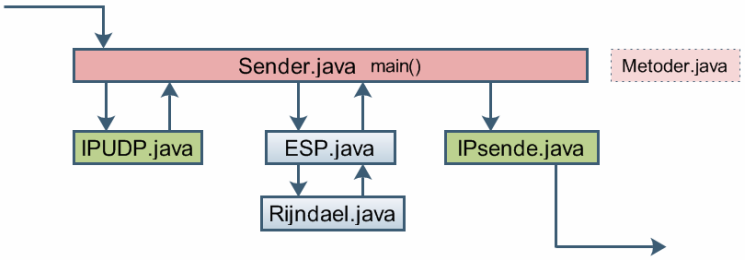
Det første som må på plass er en server og en klient som henholdsvis kan sende og motta data mellom hverandre. Fordi det ikke skal være noen returkanal, benyttes UDP som transportprotokoll. Den er en del enklere enn TCP med tanke på mindre administrasjonsdata, samt at den er mindre resurskrevende for operativsystemet.

Serveren, også kalt mottakeren, skal sitte på rød side av TCE621 og vil derfor være et veldig enkelt program. Klienten, altså senderen, skal etter hvert endres på, men den er likevel med her for å vise hvordan den kan brukes direkte om det hadde vært benyttet to TCE621 maskiner mellom senderen og mottakeren.

2.4.2 Rammeverk

Senderen må være i stand til å sette sammen protokollene i riktig rekkefølge og dytte dette ut på nettverksgrensesnittet. Utgangspunktet hittil i prosessen er at posisjon, tid, id og annen tilsvarende nyttedata bare settes til noen tilfeldige faste verder da de ikke har noen reel betydning for krypteringsprosessen. Nyttedataen blir så sendt mellom tre steg:

1. Generere et UDP og IP lag rundt nyttedataene
2. Beregne ESP rammeverket samt kryptere og lage sjekksum
3. Pakke resultatet enda en gang inn i IP for så å fysisk sende innholdet.

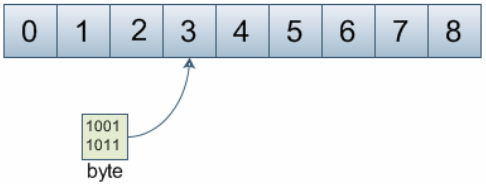


Figur 33: Klassene (javafilene) programmet består av, og måten de samhandler

Av **figur 33** er dette illustrert ved *IPUDP.java*, *ESP.java* og *IPsende.java*. Rundt de tre stegene er det et rammeverk som styrer data mellom stegene. Denne klassen er kalt *Sender.java*, og inneholder *main()* metoden som er Javas kjørbare metode. Den skal kunne sette de viktigste parametrene som trengs i de forskjellige stegene, slik at dette er samlet på et sted. En del metoder

er benyttet på tvers av klassene og er derfor samlet i en egen klasse *Metoder.java*. Dette for å unngå at disse kodebitene blir repetert på flere steder.

Hovedfunksjonaliteten til dette programmet går ut på å manipulere binære data som skal plasseres eksakt på sine respektive plasser i henhold til definerte standarder. Av den grunn er den gjennomgående datarepresentasjonsformen valgt til å være et *array* av den primitive datatypen *byte*. Et array er en konteiner for primitive datatyper og kan sammenlignes med et arkiv med mange skuffer i. Hver skuff kan åpnes ved å kalle på dets nummer. Hver byte, som tilsvarer 8 bit, blir plassert i hver sin skuff. På denne måten kan vi manipulere rekkefølge og innhold på bitnivå akkurat som det passer oss. Når informasjon skal sendes ut på nettverket er det særdeles viktig at alle parter er enige om rekkefølge informasjonen sendes ut. Slik det gjøres her, er ved at posisjon 0 blir sent ut på nettverket først i tid, etterfulgt av posisjon 1, 2 og 3 ...



Figur 34: En illustrasjon av et array.

Dette bytearrayet blir så sent frem og tilbake mellom de tre stegene for å stadig legge på seg nye headere. Første steget legger på IP og UDP header manuelt. Andre steget legger på ESP header, krypterer data og beregner CMAC/ICV (integritets sum). Det tredje steget bruker Jpcap biblioteket, pakker det inn i enda en IP adresse, og sender datastrømmen ut på nettet. Dette er uavhengig av hva den faktiske IP adressen til maskiner er.

2.4.2.1 Sender.java (Main)

Sender.java er som nå kjent administratoren som håndterer informasjonsflyten. For at programmet skal fortsette å sende mer enn én gang, er det benyttet en loopløkke kalt *while()* med input *true*. Løkken vil da loope evig, helt til brukeren velger å avslutte programmet.
Det er også lagt til en sovefunksjon som settes til tid i millisekunder. På denne måten bli tempoet begrenset til ønsket antall pakker per tidsenhet. Utenom dette gjør ikke klassen annet enn å sette parametere.

2.4.2.2 Metoder.java (Hjelpefunksjoner)

Under programmeringen har det vært behov for å konvertere mellom ulike datarepresentasjoner. Funksjonene har vært benyttet på tvers av klassene, og er derfor lagt i sin egen fil for å hindre og ha flere kopier av dem. Metodene er kort fortalt;

public static byte[] intToByte(int verdi, int antall)
Tar imot et heltall (int) *verdi* og et valgfritt (int) *antall*. Den omformer så *verdi* om til et bytearray med lengde bestemt av parametrer *antall*. Det er viktig å være oppmerksom på at et int er en 32bits primitiv datatype som aldri kan beskrive noe større enn 4 byte.

public static String display(byte[] a)
Denne metoden tar imot et vilkårlig bytearray og omgjør det til en stringrepresentasjon i heksadesimalt. På denne måten kan innholdet i en bytearray vises ved å skrive den til skjerm.

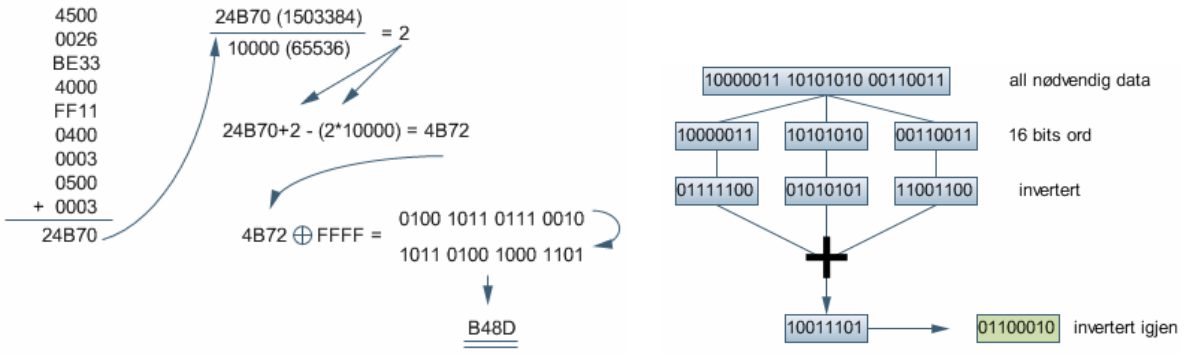
public static byte[] tilBytes(String hexStr)
Den omgjør en tekstlig heksadesimal Streng til et bytearray. Motsatt av display().

2.4.2.3 IPUDP.java (IP og UDP generering)

Denne klassen har flere oppgaver. Først setter den sammen en IP pakke med felter hentet fra main() metoden, og andre felter som den bare manuelt setter. Så setter den sammen et UDP diagram. Tilslutt legger den til nyttedata den fikk som input fra *sender.java.* De fleste felter er konstante, med unntak av sjekksummer, IDnummer og pakkestørrelsesfeltene.
Sjekksummen til IP beregnes av informasjon som finnes i IP headerene, mens UDP sjekksummen også henter noe informasjon fra IP headeren samt av nyttedataen.
IDnummeret genereres her ved bruk av en tilfeldighetsgenerator som bare velger et tall mellom 1 og maksverdi. Størrelsesfeltene beregnes ved å se på lengden av alle inputer og kjente variable som at UDP alltid er 8byte og IP er 20Byte. Når koden leses er det viktig å ha i bakhodet at det foregår en god del triksing ved å gå mellom dataformater. Et eksempel på dette er der det står (byte) 69: Dette skrives i bufferet som 0100 0101 binært, og betyr IPversjon=4 og headerlengde=5 (5 ord av 32bit tilsvarer 20 byte).

Sjekksumgenerering er skilt ut i en egen metode som benyttes av både IP og UDP. Alt annet i koden er triksing med variable for å få plukket ut det nødvendige og satt det riktig sammen igjen. Her brukes primært byteArrayOutputStreams som forenkler utvidelse av bytestrømmer som i arrays er låste til initialstørrelse.

HeaderToSjekksum()



Figur 35: To fremgangsmåter for å illustrere sjekksumberegning for IP og UDP.

Sjekksummen beregnes ved å dele opp all nødvendig data inn i ord av 16bit. Disse ordene blir så invertert, addert sammen og summen invertert på nytt før den endelige sjekksummen er funnet. Eventuelt droppes mente. Den bitvis inverteringen kalles one's kompliment i standarden[24]. (se **figur 35** på høyre side).
Av mangel på tilgang til enkeltbit i bytearrayet, er dette løst på en litt annen måte: Først blir det antatt at input er delelig med 16bit. Dette håndheves utenfor metoden. Videre blir de heksadesimale verdiene konvertert til desimalt (eller heksadesimalt slik vist i **figur 35** på venstre side). Desimalt blir de summert sammen. For å fjerne mente beregnes det hvor mange ganger summen har overskredet *maksimalverden + 1* av 16 bit, samt at det må korrigeres for ikke å invertere før summeringen skjer. Antall overtredelser blir trukket fra summen, og tilslutt invertert. Igjen står den korrekte sjekksummen.

[24] RFC 1071

2.4.2.4 IPSende.java (Bruk av jpcap)

Denne klassen gjør mye av det samme som IPUDP klassen gjør, men gjør det ved bruk av Jpcap sin API. Vi lager først en IP pakke. Setter de rette parametrene på den, legger datafeltet vårt som nå er *ESP+IP+UDP+Data* inn i denne nye IP pakken. Så genereres en Ethernet pakke. Den blir så satt til å ha next header til IP. Vi legger til vår egen MAC adresse automatisk, men må manuelt gi den adressen til motparten, her TCE621 apparatet. Så legges Ethernet pakken til IP-pakken og sendes.

Jpcap API

Jpcap består av to pakker: jpcap og jpcap.packet. Disse lastes inn ved å importere *jpcap.** og *jpcap.packet.**. Førstnevnte inneholder metoder for å sniffe pakker på nettet og sende egne pakker ut. I tillegg har den funksjoner for å skrive til og lese fra filer. Sistnevnte har metoder for å generere pakker, samt å hente informasjon ut fra pakker sniffet på nettverket.
Selve API-en er å finne på deres hjemmesider[25].

Implementasjonen her er fullstendig uavhengig av den originale protokollstacken på maskinen. Det er ikke implementert noe system for å oversette mellom IP og MAC adresser noe ARP protokollen normalt gjør. Av den grunn må MAC adresse til TCE621 manuelt tastes inn.

2.4.3 Sikkerhetsprotokollen (ESP)

Frem til nå har ESP.java metoden kun returnert det den fikk, og ikke tilført noe som helst. I dette steget skal IP/UDP pakken fra *IPUDP.java* krypteres, og det skal beregnes en sjekksum for integritetskontroll. I tillegg må ESP pakkens administrative parametere settes.

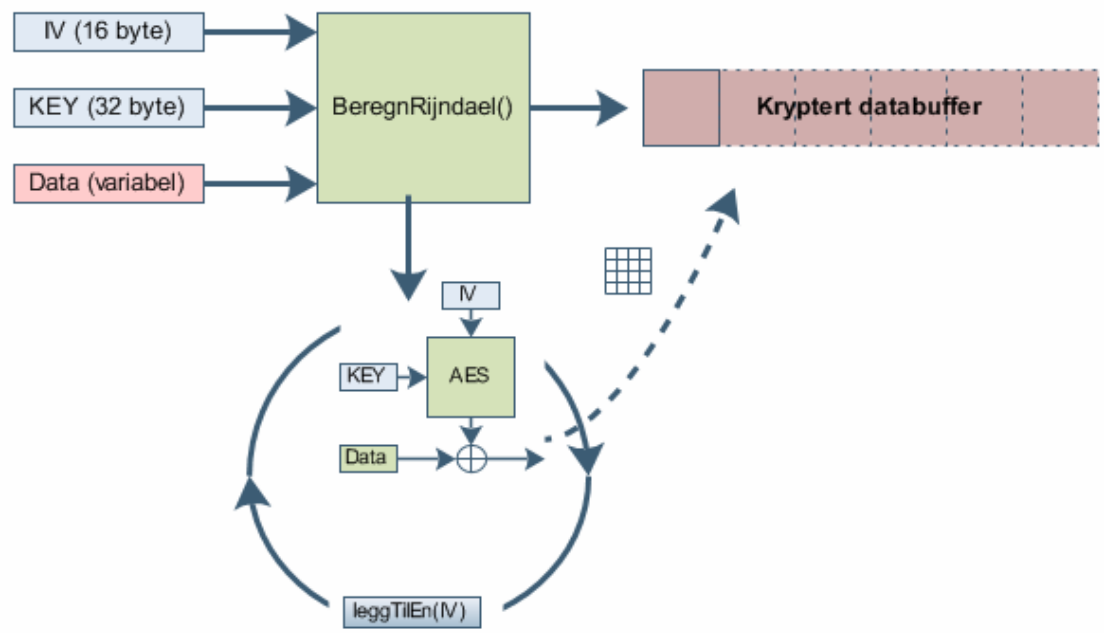
2.4.3.1 ESP.java

Metoden Beregn() i ESP.java har som hovedmål å sette sammen Thales ESP format korrekt, samtidig som at den skal administrere krypteringen av den røde IP pakken og beregningen av ICV.

2.4.3.1.1 BeregnRijndael() - AES counter mode

Countermode benyttes nøyaktig slik beskrevet i den offisielle dokumentasjonen. Likevel må metoden ta hensyn til lengden på nyttedata. AES jobber på bolker av 16 byte av gangen, men når det går tomt for nyttedata å XOR-e med, stopper datastrømmen. Dette håndteres ved variabelen ”*mangler*” som teller opp hvor mange ganger det ikke var mer data igjen den siste runden.

[25] Dokumentasjon for Jpcap: http://netreserch.ics.uci.edu/kfujii/jpcap/doc/javadoc/index.html



Figur 36: Hvordan funksjonen BeregnRijndael() opererer.

IV blir gitt inn i funksjonen, men metoden kaller selv på leggTilEn() metoden for og teller IV opp for neste gjennomgang. Resultatet for hver loop er at databufferet kryptertData blir matet med stadig mer kryptert data.

Denne metoden, samt neste, bruker aktivt en egen klasse Rijndael.java[26] funnet på Internett. Den er laget av en professor ved navnet *Paulo Barreto* paulo.barreto@terra.com.br
Klassen har et ganske enkelt grensesnitt og vi benytter den kun som AES oppslagsbok.
Den kalles ved å lage et ny instans av klassen:

Rijndael aes = new Rijndael();
Videre må nøkkelen mates til vår nye instans:
aes.makeKey(key, 256);
Med dette på plass krypterer og dekrypterer instansen ved å bruke:
aes.encrypt(plaintext, ciphertext);
aes.decrypt(ciphertext, plaintext);

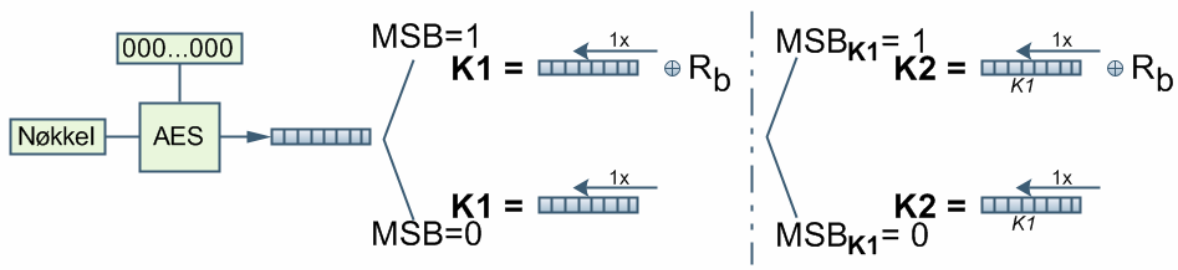
Dokumentasjon på AES og implementasjon av CTR-mode finnes ved å søke opp **FIPS 197** og **NIST 800-38A**. Den offisielle standarden har kun eksempler som sier at **leggTilEn()** funksjonen skal telle kronologisk 1,2,3,4..

2.4.3.1.2 BeregnICV()

Integritetssummen benyttet her kjører alt som skal autentiseres igjennom AES kodeboken med samme nøkkel som for krypteringsprosessen. Resultatet etter hver kjøring XOR-es med neste bolk av nyttedata før det igjen blir dyttet igjennom AES kodeboken. Dette loopes inntil hele meldingen som skal autentiseres er blitt behandlet. Resultatet er en 16byte/128bit sjekksum der kun de første 64 bitene blir benyttet.

[26] Mappen der Rijndael.java er å laste ned: http://www.cs.ucdavis.edu/~rogaway/ocb/ocb-java/ av *Paulo Barreto*

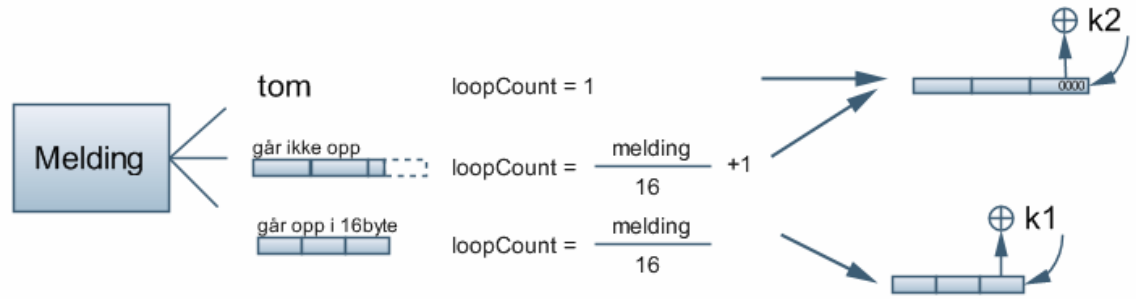
Det første som må gjøres her, er å kalkulere to nye nøkler. Disse utledes av hovednøkkelen etter følgende metode:



Figur 37: Utregning av K1 og K2

Her er R_b en konstant som er 15 byte med 0'ere etterfulgt av 10000111 som til sammen tilsvarer 16byte, noe den må være for å overensstemme med det AES genererer.

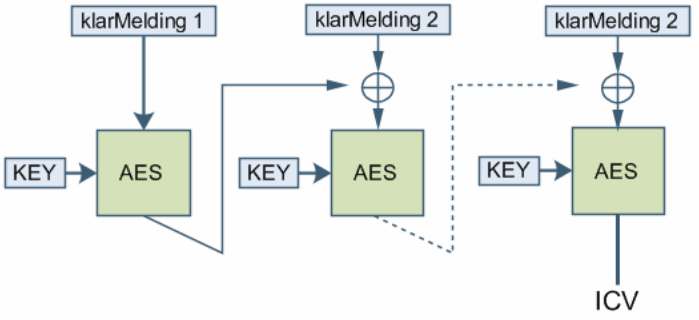
Det beregnes først en AES transformasjon med bare 0'ere som input samt den hemmelige nøkkelen. Hvis så det mest signifikante bittet (plassen lengst til venstre) er 1, skiftes hele resultatet en plass til venstre og XOR-es med R_b. Hvis MSB var 0, bestemmes K1 kun ved å skifte en gang til venstre. Videre beregnes K2 på samme måte, men med utgangspunkt i K1.



Figur 38: K1 og K2 benyttes avhengig av lengden på meldingen.

Metoden som utfører rotering til venstre heter **shift()** og er plassert utenfor selve BeregnICV() metoden. Videre må det taes hensyn til om hvorvidt hele meldingen går opp i bruddstykker av 16byte, eller om det må paddes. K1 skal XOR-es med siste blokk av meldingen hvis meldingen går opp. Hvis ikke må det paddes ved å legge til binært 1, etterfulgt av så mange 0'ere som nødvendig for å gå opp i siste blokken. Tilslutt må siste blokk av meldingen XOR-es med K2.

Så gjenstår å loope den nye variabelen *"klarMelding"* igjennom AES algoritmen etter følgende mønster som vist i **figur 39**. Som nevnt tidligere er det bare den første 64bit av resultatet som benyttes.



Figur 39: Beregning av ICV igjennom AES algoritmen

Dokumentasjonen for dette, samt diverse melding-sjekksum par, heter ***NIST 800-38B***.

2.4.4 GPS tilkobling

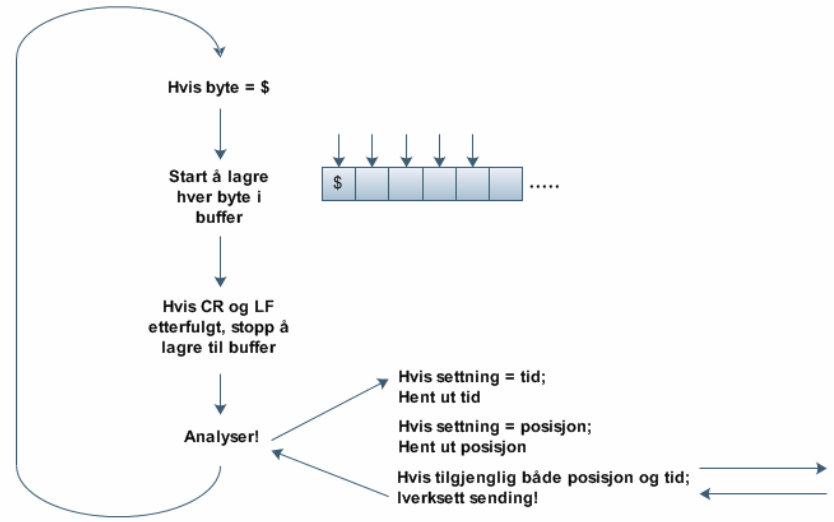
GPS er oppsatt til å sende informasjon på COM1 porten. Det som gjenstår er å implementere kode i Java som er i stand til å hente inn og sortere ut relevant informasjon. Til dette er biblioteket javax.comm lastet inn i Java.

Fremgangsmåten[27] for å integrere denne i Java er å laste ned javacomm20-win32.zip som er versjon 2.0 av biblioteket. Versjon 3 støttes ikke i Windows. Videre må filene pakkes ut og legges direkte i JDK mappen til Java lokalt på maskinen. Filene **win32com.dll**, **javax.comm.properties** og **comm.jar** må flyttes fra **\commapi** til henholdsvis mappene **\bin**, **\jre\lib** og **\jre\lib\ext.** Med dette på plass er javax.comm API klar for bruk. Som en del av biblioteket er det lagt ved eksempelfiler for bruk av javax.comm API. Disse er plassert i mappen "samples" tilhørende biblioteket. Filen **SimpleRead.java** er her benyttet som utgangspunkt.

Første endring var å endre port til COM1, samt å sette farten til 4800 bps. I utgangspunktet henter programmet inn 10 byte om gangen og bare printer dem ut.

For å få hentet ut riktig data må informasjonen settes sammen fra tilfeldige byte til logiske setninger som står for seg selv. Hver setning starter med ASCII "$" og slutter med <CR><LF> (Carriage Return og Line Feed). Innlest data blir behandlet byte for byte, slik at $ = 0x24, CR=0x0D og LF=0x0A.

Hvis et array ikke fylles helt opp, vil ubrukte plasser ha verdien null. Derfor blir kun 1 byte avlest for hver gang det leses. Da unngås at null dukker opp i her og der. Følgende **figur 40** viser hvordan koden fungerer i prinsippet:



Figur 40: Hvordan GPS.java fungerer

All kode fra Sender.java er plassert inn i GPS på grunn av problematikk rundt ikke-statiske metodekall inn i et statisk miljø. GPS har dermed tatt over alle rammeverkfunksjoner.

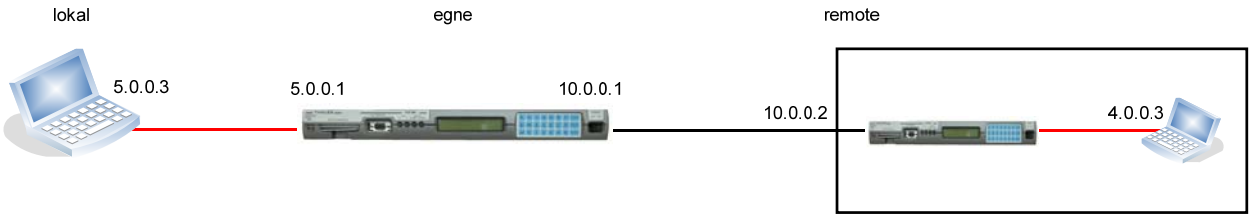
[27] Java SDK – Hello World – Comm API: http://tip.no/tini/1_java_SDK_commAPI.asp av Ola Lie, 2007

3. Testing

Testing er utført på koden i stadiet før GPS klassen ble utviklet, og bruker derfor Sender.java som rammeverk. Posisjon og tid er derfor konstanter under testing.

3.1.1 Oppsett på laben

Den konfidensielle Pc-en som er benyttet er en bærbar Pentium 4 Mobile på 1,7Ghz. Den har 512 MB systemminne. Denne ble så koblet til TCE621 AES på sort side, og en ugradert vilkårlig maskin til den røde siden. TCE621 ble tilbakestilt og restartet. Oppsettet er vist i **figur 41**.



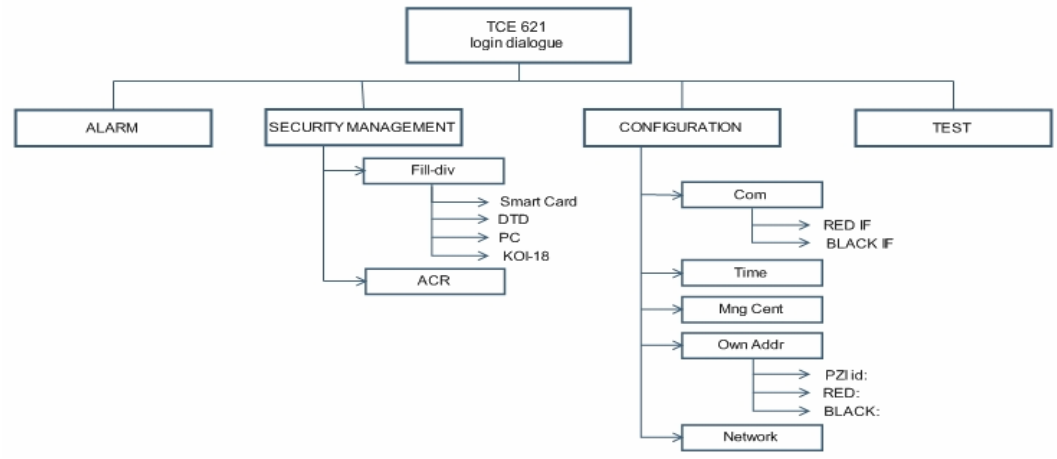
Figur 41: Oppsett på lab

3.1.1.1 Initiell konfigurasjon

Den første menyen heter *initial configuration* og lar brukeren sette grunnleggende innstillinger.

1. **Operatør og sikkerhetspassord**: Her settes egne passord for tilgang til TCE621
2. **Operasjonsmodus**: For å velge om TCE621 står foran et management center eller ikke. Her gjør den ikke det, og settes derfor til *user-loc*.
3. **(Nøkkel) distribusjonen**: Manuell eller automatisk, og ble satt til manuell.
4. **Nøkkeltype**: Den manuelle nøkkelen benyttet er en *KVT* nøkkel, og tilsvarer en transportnøkkel.
5. **IP adresser**: Her settes sort og rød IP-adresse med tilhørende subnettmaske. Disse adressene kan ikke endres i etterkant, da kommende innlastede nøkler låses til IP-adressene.
6. **Valg av IP versjon:** IPv4.
7. **Dato og tid:** Først og fremst for systemloggen.

Neste meny etter å ha logget inn med valgt passord kommer *hovedmenyen*. Med bistand fra labansatte ble det henvist til hvilke innstillinger som måtte settes for å få opp linken mellom sort og rød side. **Figur 42** viser de essensielle undermenyene som er av interesse i denne sammenheng.



Figur 42: TCE621 – Undermenyer

3.1.1.2 Nøkkelinnlasting

Thales produserte en papirnøkkel av den gode gamle sorten, som mates inn og digitaliseres ved bruk av en KOI-18 (se **figur 43**). Dette gjøres fra *security management* → *Fill-Div* → *KOI-18* slik vist av **figur 42**. TCE621 signaliserer at den er klar ved at det står *reading* på displayet. Måten nøkkelen lastes på er ved å ta nøkkelstrimmelen, som har masse små hull stiplet i seg, og dra den *omsorgsfullt* gjennom KOI-18. Kombinasjonen av hull på strimmelen representerer ulike heksadesimale verdier som blir lest av inne i KOI-18 ved hjelp av små fjærer og videreformidlet til TCE621. Papirstrimlene kommer i ulike lengder på bakgrunn av bitstørrelsen på nøkkelen.

Dersom innlesingen av nøkkelen er vellykket kommer det opp en melding om *0/0 error* på skjermen før TCE621 går videre i menyen hvor parametrene for den enkelte nøkkelen settes.

1. **Nøkkeltype**: KVT.
2. **Remote sort adresse**: IP til *remote* enhet som innlastet nøkkel gjelder for. Den ble satt til 10.0.0.2 i henhold til **figur 41**.
3. **Nøkkelnummer**: Dette tilsvarer en av perimeterne i ESP formatet og inngår i *security association*[28]. Dette nummeret står på papirnøkkelen.
4. **Nøkkelperiode og oppdateringsperiode**: Varighet for nøkkelen i sin helhet, og hvor lang tid hver delnøkkel skal vare. Denne funksjonaliteten ble ikke benyttet og derfor satt til null.

Dette innebærer at det benyttes en nøkkel for hver kommunikasjonskanal mellom TCE621 maskiner. Den vanligste måten nøkkeldistribusjon skjer på, er via at et eller flere management senter sender ut krypterte nøkler på forespørsel fra enhetene, altså sentralisert styring.

Figur 43: KOI-18

3.1.1.3 Aksesslister - ACR

Nå som nøkkelen er lastet inn må det settes opp regler som tillater kommunikasjon mellom enheter *internt* og *eksternt*. I dette tilfellet senderen og mottakeren slik **figur 41** viser.
Dette valget er å finne under *security management* og heter *ACR* (Access Control Records).
Her velges *new*, og nødvendige parameter er IP-adressene for gjeldene testnett.

Lokal rød IP-adresse:	5.0.0.3
Egen svart IP-adresse:	10.0.0.1
Remote rød IP-adresse:	4.0.0.3
Remote svart IP-adresse:	10.0.0.2

Ingen kommunikasjon er tillatt om det ikke er satt opp eksplisitt her. Reglene kan gjelde mellom enkeltadresser, subnett eller kombinasjoner av disse.

---

[28] Punkt 2.2.3.1

Nå er alt konfigurert, og WireShark ble kjørt opp på begge sider av TCE621. *Mottakerprogrammet* ble startet opp fra PC-en på innsiden og *senderprogrammet* fra den konfidensielle PC-en på utsiden. Her ble det bråstopp for pakkene mot TCE621, noe som var erfaringsmessig forventet.

For å sjekke om enhetene har kontakt seg imellom ble *test* → *ping* benyttet.
Ping mot lokale røde adresse (5.0.0.3) og ekstern sort adresse (10.0.0.2) fungerte.
Et nytt testforsøk var å sende en pakke fra innsiden (5.0.0.3) og ut. Dette gikk uten problem, og det gav en pekepinn på at konfigurasjonen til TCE621 var korrekt. Av det ble konklusjonen at det var *senderkoden* som ikke fungerte slik den måtte.

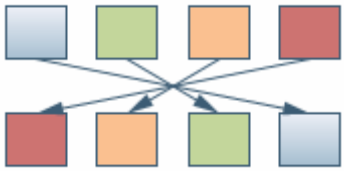
3.1.2 Feilsøking

Senderprogrammet virket som beskrevet i følge standarden, men siden Thales har forandret på en del punkter gikk videre feilsøking ut på å om det var udokumenterte faktorer eller misforståelser som kunne være årsaken.

3.1.2.1 Little Big Endian

Første feilmeldingen TCE621 gav var meldingen ”38 AccessControllError” i audit loggen.
Det betyr at pakken enda ikke har nådd frem til selve dekrypteringsprosessen, men er stoppet ved det eksterne grensesnittet. Dette betyr at nøkkelen ikke ble godtatt, og kan skyldes at nøkkelnummer ikke stemmer overens. En av de ansatte viste et pakkesniff av en fungerende TCE621 tilkobling med tilsvarende nøkkelnummer/SPI nummer. I displayet stod det 12345678 mens i WireShark lå dette som 78563412.
Bakgrunnen for denne representasjonen kommer av måten prosessoren håndterer store tall satt sammen av mange bytes. Det som skiller håndteringen er rekkefølgen på bytene. Dette kan sammenliknes med at vi i vesten leser fra venstre til høyre, mens det finnes andre kulturer som leser fra høyre til venstre. Måten benyttet i senderprogrammet blir kalt Big Endian, og er det formatet som brukes i all nettverkstrafikk. Little endian, altså fra høyre mot venstre, brukes blant annet i Intel- og AMD-prosessorer, samt trolig TCE621.

Nøkkelnummeret, variabelen **spiKey**, ble endret fra *00 00 00 01* til *01 00 00 00* slik figur 43 viser.



Figur 44: Omstokking av byte fra Big Endian til Little Endian

3.1.2.2 InitialVektor telling

Dette er flyttet til vedlegg B

3.1.2.3 Selve krypteringsnøkkelen

Dette er flyttet til vedlegg B

3.1.2.4 Little Big Endian enda en gang

Dette er flyttet til vedlegg B

3.1.3 Resultater og målinger

Resultatet av feilsøkingen er at den produserte koden nå fungerer som den skal og blir levert dekryptert til mottakeren på rød side av TCE621.

3.1.3.1 Hurtigheten til koden

Hastigheten til koden på testmaskinen er 19 pakker i sekunder med en payload på 100byte. Prosessorstatistikk viser at prosessoren utnyttes 100% ved denne lasten. Målinger utført ved å skru av enkeltfunksjoner viser hvor flaskehalsene ligger:

Fullstendig utregning med sending:
1000 pakker på 52,7 sek = 19 pakker/sek
Full beregning, men uten å pakke inn i IP og sende (IPsende.java):
30000 pakker på 36,7 sek = 817 pakker/sek
*Kun ESP beregning, uten å først pakke nyttedata inn i IP/UDP. **100** byte nyttedata:*
300000 pakker på 66,1 sek = 4540 pakker/sek
*Kun ESP beregning, uten å først pakke nyttedata inn i IP/UDP. **1000** byte nyttedata:*
110000 pakker på 54,5 sek = 2010 pakker/sek

Dette kan forklares ved at Ipsend.java klassen kaller på api funksjoner på nytt for hver gang en pakke skal sendes. Om fart er ønskelig vil det hjelpe veldig å omorganisere måten IPsende.java fungerer på ved å legge mye av dens metodekall inn i Sender.java sin main() metode slik at dette bare kjører én gang.

3.1.3.2 Minnebruk

Under langvarig testing viste det seg at minnebruken til java-programmet økte konstant. Før eller siden vil derfor programmet krasje ved at maskinen går tom for ledig minne. Dette kommer trolig av at en variabel ikke blir tømt underveis.

3.1.3.3 Tilkobling av GPS

GPS er koblet til via COM port mot PC, informasjon hentet inn og automatisk plassert i nyttedatafeltet. Denne utvidelsen har ført til en del endringer i etterkant vedrørende måten hele programmet er bygget opp på, og er nærmere forklart under implementasjonsdelen av oppgaven.

3.1.3.4 Lagring av tilstand

Det er ikke implementert noen funksjon for å lagre tilstanden til programmet ved avslutning. Alle tellere starter derfor på nytt for hver gang programmet starter opp. Dette kan lett fikses ved å skrive verdier i aktuelle variabler til en fil på harddisken for så å laste fra den hver gang programmet starter.

3.1.3.5 ARP

MAC (nettverkskortadressen) til TCE621 må tastes inn manuelt i koden for at OSI nivå 2 ruting skal nå frem. Dette fordi det ikke er implementert noen ARP[29] protokoll.

[29] Address Resolution Protocol

4. Diskusjon

Dette er en oppgave som gir muligheten til flere løsningsmetoder, noe som gruppen selv har merket gjennom de valg og avgjørelser som er blitt tatt underveis.

4.1 Arbeidsmetoder

Hvilken arbeidsmetode bør benyttes for at oppgaven skal løses optimalt?

Alternativene som har vært oppe til diskusjon er hvorvidt prosessen bør være basert på en typisk ”prøve og feile” metode der kjernen av metoden går ut på å lete seg frem til løsninger underveis til målet, eller om en strukturert og detaljert fremdriftsplan bør utvikles på forhånd før realiseringen starter.

Med en ”prøve og feile” prosess menes her en metode der målet er gitt, mens veien frem er svært uklar. I denne sammenheng vil kreativ tenking, samt en god del prøving være nødvendig for å finne veien underveis. En slik prosess krever også at tiden til disposisjon strekker til, og ikke stresser i stor grad, da stress vil hemme kreativiteten. Sett opp mot denne settingen er målet definert med en stram tidsfrist. I tillegg er det krav om at en fremdriftsplan skal opprettes og følges som en del av et kontrolltiltak fra skolens side. I og med at denne metoden ikke sier stort om tidsmessige perspektiver eller hvor mye arbeid som gjenstår, blir det vaskelig å styre prosjektet inn mot tidsfristene. Dette svekker dette alternativet.

En strukturert og detaljert fremdriftsplan innebærer at veien frem mot mål blir nøye planlagt i forkant av selve implementasjonen. Fordelen ved dette blir at strukturen på arbeidet blir veldig forutsigbart og oversiktlig. Det er lett å lede fremdriften når konkrete delmål med tidsfrister er satt. Det er også lettere å fordele oppgaver som nå fremstår som klare og uavhengige. Metoden krever at det blir avsatt tid i forkant av implementeringen for planlegging. For dette prosjektet har det vært vanskelig å forutsi hvordan enkelte deler av prosjektet kan løses, og hvor lang tid det vil ta og løse. Dette, samt at tiden er knapp, svekker en slik fremgangsmåte.

Da begge alternativene har både ønskede og uønskede konsekvenser, konkludere vi med at metoden må bli en mellomting tilpasset delopppgavene. Oppgaven med å beskrive sikkerhetsprotokollen var egnet for en strukturert fremdriftsplan fordi den var oversiktlig og konkret. Design og implementasjonsfasen var vanskelig å forutse, og krevde større vekting av ”prøve og feile” metoden. Likevel var det viktig å ha kontroll på fremdriften. Derfor ble det laget grove skisse av delmålene[30] og deres samspill. Denne ble naturligvis modifisert i løpet av perioden.

[30] Se figur 31 og 32

Hvilke konsekvenser innebærer det å jobbe med konfidensielt materiale?

Gradert informasjon er beskyttet av Sikkerhetsloven. I loven er det en forskrift om informasjonssikkerhet § 6-11, og den sier; ”*Informasjon gradert KONFIDENSIELT skal oppbevares i sperret område eller i godkjent oppbevaringsenhet plassert innenfor beskyttet område.*” Dette medfører at gruppen er avhengig av en safe eller en tilsvarende enhet innenfor et avgrenset område, hvor gradert materiell kan oppbevares når det ikke er i bruk. Da informasjonen er på digitalt format, er det godkjent å håndtere denne på en laptop med sikkerhetsgodkjent harddisk. Maskinen må ikke kobles til nettverk som informasjonen ikke er klarert for.

Loven begrenser handlingsfriheten i forbindelse med arbeidsmetode, prosedyrer og lokasjon. Prosedyrene fører til at mye tid går på administrering, dette viser seg i at materiale må låses inn og ut, ofte av eksternt personell. Dette igjen medfører at muligheten for å arbeide med prosjektet blir begrenset av andres arbeidstid. En vesentlig hindring nettverkspolicyen medfører, er at Internett ikke blir tilgjenglig i samme miljø som programmeringsmiljøet. Dette gjør enkle oppgaver mer kompliserte ved at tilgang til nødvendige hjelpemidler nå ikke er like tilgjenglige. Å behandle gradert informasjon krever en ny form for oppmerksomhet rundt behandlingen av informasjonen. Dette bidrar likevel positivt til at forståelsen for og bevisstgjøringen rundt behandlingen av gradert informasjon øker.

Periodene med mest bruk av konfidensielt materiale blir derfor tilbrakt i Thales lokaler. Dette gir muligheten til å både jobbe og lagre det sensitive materialet i samme lokale og samtidig gi tilgang til det utover normal arbeidstid. Samtidig skaper dette en større arbeidsro og fleksibilitet da alle distraksjoner fra skolen side ikke får samme påvirkning som den ellers hadde fått.

4.2 Plattformvalg

Hvilket operativsystem og programmeringsspråk vil være hensiktsmessig?

De mest utbredte operativsystemene er Linux og Windows. Nettopp av den grunn er det store utviklingsmiljø rundt dem, noe som igjen fører til at det er lett å skaffe nødvendig informasjon som dokumentasjon, rammeverk og kodesnutter.

Linux er et veldig åpent operativsystem, der brukeren har store muligheter til å tilpasse systemet etter eget behov. Det har en såkalt ”raw socket” som vil si at vilkårlige pakker kan sendes og hentes ut fra nettverket. Ulempen med Linux i denne sammenheng er at gruppen ikke er detaljkjent med Linux, og mangler ferdigheter i å få ting til å virke der.

Windows er et mer lukket system der brukerens modifiseringsmuligheter i utgangspunktet er mer begrenset. For å kunne modifisere systemet, må tredjeparts programmer lastes inn. Dette forenkles ved at et godt utviklet brukergrensesnitt automatiserer denne prosessen. Dette forsterkes ved at gruppemedlemmene har god kjennskap og mye erfaring med Windows.

Helst bør programmet utviklet i dette prosjektet kunne kjøre på begge plattformer. Hvis Windows skal kunne brukes, må nødvendig mellomvare være tilgjenglig.

Når det gjelder programmeringsspråk er det to språk gruppen kjenner til: Java og C.
C programmer er generelt hurtige da de benytter prosessoren effektivt. Språket ligger nært maskinvaren, og gir derfor utvikleren mer kontroll. En vesentlig ulempe i gruppens situasjon er at ingen kan C i noen nevneverdig grad, og det vil ta mye tid å bli tilstrekkelig god nok.
Java er et mer abstrakt programmeringsspråk. Det kan enkelt overføres fra et system til et annet da det er maskinvareuavhengig. Dette gjør samtidig programmer skrevet i Java tregere. Gruppen har også mye erfaring og kunnskap rundt Java.

Konklusjonen så langt blir derfor at Windows med Java er den beste utviklingsplattformen for dette prosjektet, samt at den lett kan porteres til Linux.

Hvilke eksisterende moduler kan benyttes inn i dette prosjektet og hvilke må utvikles selv?

Det er ønskelig med en modulær kode, det vil si at den er delt inn i selvstendige komponenter med klare grensesnitt seg imellom. Komponenter må håndteres av et rammeverk. For å konkretisere disse begrepene, vil en modul typisk være beregning av AES, MAC og tilsvarende. Rammeverket blir så et overordnet nivå som setter modulene i gang og håndterer informasjonsflyten mellom dem. Et slikt rammeverk kan enten oppdrives som ferdigskrevet kode, eller produseres på egenhånd.

Å benytte kode som er ferdigskrevne vil være meget tidsbesparende gitt at kodene er skrevet på en slik måte at den kan benyttes til den aktuelle oppgaven, ved kun å måtte forandre enkle trivielle ting i koden. Mulighetene for å finne nettopp så treffende koder vil samtidig være lite sannsynlig da oppgaven er meget særegen. Dersom gruppen velger å skrive alle kodene selv vil det medføre mye tidkrevende arbeid i å lage det, mot kun å forstå hvordan det fungerer. Derimot vil det å skrive alle kodene selv medføre at gruppen har full oversikt over hva som skjer, og lettere kunne forandre på enkelt elementer i koden og feilsøke den dersom det skulle være nødvendig.

Delkonklusjonen her blir at det er mest fornuftig å lage rammeverket selv. For å kunne lage et eget rammeverk i Windows kreves utvidede rettigheter mot nettverkskortet på maskinen. Her er det to hovedretninger som skiller seg ut: Pcap og TUN/TAP som begge er kerneldrivere.

For Pcap fant gruppen et bibliotek til Java som heter Jpcap. Dette tillater Java-programmer å nå Pcap for kapring og sending av pakker. TUN/TAP er drivere som oppretter virtuelle nettverksgresesnitt. Ved å bruke en slik TUN driver, kan programmet bli enda mer modulbasert enn hva Pcap tillater. Ulempene er at gruppen ikke fant noen måte å bruke dette med Java, samt at det krever en tung installasjonsprosess.

I og med at Pcap finnes både til Linux og Windows valgte gruppen å bruke Pcap med Jpcap.

Konklusjonen blir derfor at utviklingsplattformen blir Windows med Java, der rammeverket produseres av gruppen ved bruk av Jpcap og Pcap.

5. Konklusjon

Prosjektet gikk ut på å sette seg inn i Thales sikkerhetsprotokoll rundt TCE621 AES, samt å utvikle programvare som i et labmiljø kan sende meldinger til en TCE621 AES og få disse forstått.

TCE621 AES er et krypteringsapparat utviklet for å kryptere nettverkstrafikk mellom datanettverk over usikre nett som for eksempel Internett.

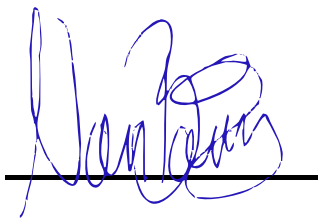
Sikkerhetsprotokollen benyttet er basert på IPsec og er en variant av ESP i tunnelmodus. Dette innebærer at IP pakker fra det interne sikre nettet blir kryptert som de er og innkapslet i en ekstern IP-pakke for ruting over det eksterne nettverket. Krypteringen skjer ved bruk av AES, også kjent som Rijndael algoritmen. AES er benyttet i counter mode. I tillegg til kryptering må den interne IP-pakken autentiseres, noe som er løst ved å bruke CMAC algoritmen.

Hovedsakelig fungerer sikkerhetsprotokollen til Thales som de offentlige standardene tilsier. Unntakene fra dette er at ESP formatet er noe omdefinert, telleren i counter mode benytter et tilbakekoblet skiftregister med et udokumentert polynom, samt at TCE621 er little endian i motsetning til big endian som er standarden for kommunikasjon mellom nettverksenheter. Detaljer rundt dette er gradert og blir derfor ikke nevnt her. Se vedlegg B.

Programvaren som ble utviklet skulle simulere en mobil kryptoenhet som er interoperabel med TCE621 AES. Denne mobile enheten er tiltenkt å overføre posisjonsdata som den innhenter via en GPS mottaker.

Programvaren ble utvikler på en Windows XP plattform skrevet i Java med noen eksterne tilleggsmoduler. Posisjoner ble overført ved å koble GPS til COM1 porten på testmaskinen. Mellom XP plattformen og TCE621 ble det benyttet et trådbundet TCP/IP nettverk. Prosjektet tar ikke stilling til hva slags trådløs informasjonsbærer det bør benyttes i en mobil utgave. Programvaren er i stand til å kommunisere enveis med TCE621 AES, og dette viser at en mobil enhet kan realiseres med denne eller tilsvarende programvare.

Jørstadmoen 19. november 2007



Nan Barås

André Nordbø

6. Litteraturhenvisninger

Alle litteraturhenvisninger er også plassert som fotnoter nederst på den aktuelle siden der referansen blir gjort. Dette for at leseren skal ha lett tilgang på henvisningene.

[1] DISKO: hjelpekorps.org/disko/ av Norges Røde Kors
[2] SitaWare – The C4I framework: systematic.dk/UK/Products/SitaWare/ av Systematic
[3] Bigger Role for Blue Force Tracking: military-information-technology.com/article.cfm?DocID=504 av Mickey McCarter
[8] AES flash illustrasjon: www.cs.bc.edu/~straubin/cs381-05/blockciphers/rijndael_ingles2004.swf av Enrique Zabala
[9] Block cipher modes of operation: en.wikipedia.org/wiki/Modes_of_operation av 13.Oktober 2007
[12] An Illustratated Guide to IPsec: www.unixwiz.net/techtips/iguide-ipsec.html av Steve Friedl
[13] RFC 2406
[14] Network Security Essentials, Third Edition, William Stallings, Pearson Prentice Hall, 2007: Kapittel 3
[16] RFC 2409
[18] Global Positioning System: en.wikipedia.org/wiki/Global_Positioning_System av 18.November 2007
[19] The NMEA 0183 Protocol: fort21.ru/download/NMEAdescription.pdf
[20] NMEA 0183: en.wikipedia.org/wiki/NMEA_0183 av 8.November 2007
[21] Universal TUN/TAP driver: vtun.sourceforge.net/tun/faq.html av Maxim Krasnyansky
[22] Pcap: http://en.wikipedia.org/wiki/Pcap av 7.November 2007
[22] The Windows Packet Capture Library: www.winpcap.org/
[23] Jpcap – a Java library for capturing and sending network packets: netresearch.ics.uci.edu/kfujii/jpcap
[24] RFC 1071
[25] Dokumentasjon for Jpcap: http://netreserch.ics.uci.edu/kfujii/jpcap/doc/javadoc/index.html
[26] Mappen der Rijndael.java er å laste ned: http://www.cs.ucdavis.edu/~rogaway/ocb/ocb-java/ av *Paulo Barreto*
[27] Java SDK – Hello World – Comm API: http://tip.no/tini/1_java_SDK_commAPI.asp av Ola Lie, 2007

7. Vedlegg

Vedlegg A: Implementasjonskode
Vedlegg B: Graderte underpunkt
Vedlegg C: ASCII tabellen