NTNU
Det skapende universitet

Brukeraksessstyring i 802.11i RSN

Kenneth Helge Molnes

Master i kommunikasjonsteknologi
Oppgaven levert: Juni 2007
Hovedveileder: Stig Frode Mjølsnes, ITEM

Norges teknisk-naturvitenskapelige universitet
Institutt for telematikk

Oppgavetekst

Oppgaven går ut på å sette seg inn i nett-teknologien, tilegne seg forståelse og ferdigheter i verktøyet, og sette opp og beskrive et testoppsett av styring av brukeraksess i RSN basert på IEEE 802.1X modellen med fokus på gjensidig autentisering ved hjelp av digitale sertifikater og EAP-TLS over RADIUS. Basert på denne uttestingen skal kandidaten gjøre sin vurdering av hvor godt denne løsningen vil fungere teknisk og organisatorisk for brukere med små mobile terminaler i praksis.

Oppgaven gitt: 29. januar 2007
Hovedveileder: Stig Frode Mjølsnes, ITEM

SAMMENDRAG

I emnet TTM4137 Mobil sikkerhet er det utviklet en student lab [2] som muliggjør uttesting av sårbarhet i *IEEE 802.11* teknologi.
Denne oppgaven skal basere seg på dette og videreutvikle dette verktøyet slik at det egner seg for å gjøre undersøkelser av *Robust Security Network (RSN)* sikkerhetsarkitektur spesifisert i *IEEE 802.11i*. *RSN* innbefatter *AES-CCMP* krypto og et trenivå nøkkelhierarki.
Oppgaven går ut på å sette seg inn i nett-teknologien, tilegne seg forståelse og ferdigheter i verktøyet, og sette opp og beskrive et testoppsett av styring av brukeraksess i *RSN* basert på *IEEE 802.1X* modellen med fokus på gjensidig autentisering ved hjelp av digitale sertifikater og *EAP-TLS* over *RADIUS*. Basert på denne uttestingen skal kandidaten gjøre sin vurdering av hvor godt denne løsningen vil fungere teknisk og organisatorisk for brukere med små mobile terminaler i praksis.

Laben settes opp som vist i kapittel 3, og kjøres som vist i kapittel 4.

Problemet med web-innmelding er at en *Windows Mobile* enhet må anskaffe sertifikatene fra en webserver. Kun *Windows 2000/2003 Server* er støttet, og webserveren må være *Internet Information Services* og *CA*en må være *Microsoft Certificate Services.* Vil en da benytte seg av en *CA* eller *VPN*-server som ikke er basert på *Windows*, må en da benytte seg av egne installasjonsprogrammer eller finne et som passer sine behov. Et slikt program er *P12import* – et velrenommert *open source* program utviklet av Jacco de Leeuw. Det kan lastes ned fra [20].
Et alternativ til sertifikatinnmelding er import av sertifikat. PKCS#12 er standardformatet for lagring av private nøkler og sertifikat. Formatet er støttet av mange tilbydere – inkludert *Microsoft.* De fleste *VPN*-klientene støtter PKCS#12. Import av PKCS#12 filer støttes kun i *Microsofts Windows Mobile 6* og ikke på *Pocket PC 2003* og *Windows Mobile 5.0.* Dermed trenger en et program egnet for import av sertifikater.

Forsøk på å koble seg til laboppsettet i denne oppgaven med en Qtek9000 mislyktes grunnet enheten manglet støtte for *RSN* og *EAP-TLS*. Men siden enheten støtter *TKIP*, *PEAP* og *IEEE 802.1X,* kan laboppsettet modifiseres til å kjøre over *TKIP* og *PEAP*.

Dette gir ikke gjensidig autentisering – noe som er et krav i problemstillingen, og ble dermed ikke gjennomført i denne oppgaven.

For å koble til en handholdt enhet opp mot Trådløse Trondheim ved hjelp av *IEEE 802.1X* må en få utstedt et sertifikat fra Trådløse Trondheim. I oppgaveforfatterens forsøk med en Qtek9000 ble det utstedt et .der sertifikat som ble omgjort til et .cer sertifikat. Ved å installere dette og velge det fra profilen får en koblet til. Formatet på sertifikatet varierer fra enhet til enhet. Det er en forutsetning at brukeren befinner seg innenfor dekningsområdet til Trådløse Trondheim. En lignende gjennomgang er vist på [22] og i kapittel 4.2.

I denne oppgaven har det blitt tatt for seg teorien som trengs for forståelsen av denne laben. Det har også blitt beskrevet i detalj hvordan en skal gå frem for å sette opp et trådløst nettverk basert på *RSN*, *IEEE 802.1X* og *EAP-TLS* med digitale sertifikater. Med opplysningene gitt i denne masteroppgaven skal det ikke være noen problem å sette opp sitt eget nettverk, uavhengig av størrelse.
WPA er ikke perfekt – verden trenger ennå en metode som kan håndtere passordbeskyttet sertifikater uten å lagre passordet i klartekst. Men trådløse nettverksmetoder ser endelig ut til å bevege seg i en sikker retning.

Når det gjelder problemstillingen rundt handholdte enheter hadde ikke kandidaten riktig utstyr tilgjengelig. Qtek9000, som blir brukt her, støttet verken *RSN* eller *EAP-TLS*. Den støttet riktig nok *TKIP* og *PEAP*, så med litt modifikasjoner av laboppsettet kan den kobles til laben. Men siden laben da er basert på *PEAP* er ikke gjensidig autentisering mulig, og hele problemstillingen til denne oppgaven må da modifiseres. Enheten kan benyttes til å koble til Trådløse Trondheim siden de baserer seg på *PEAP* (per dags dato). Det er muligheter for at de skal bytte denne metoden mot en annen. En må da få utstedt et *IEEE 802.1X* sertifikat fra Trådløse Trondheim som benyttes til å koble seg opp mot *SSID*en ”ntnu1x”.

Den ledende leverandøren av slike handholdte enheter – HTC, tidligere Qtek, har foreløpig ikke enheter som støtter *RSN*. Dermed må en vente til dette er utført før hele denne laben er gjennomførbar på en handholdt enhet. Utførelse med *TKIP* og *EAP-TLS* er derimot gjennomførbart – det finnes enheter som støtter de metodene.

FORORD

Denne masteroppgaven skrives våren 2007 som den avsluttende delen av sivilingeniørutdanningen min ved Norges Teknisk- Naturvitenskapelige Universitet, *NTNU* – Institutt for Telematikk – Informasjonssikkerhet.
Oppgaven bygger videre på en laboppgave utviklet av L. Haukli, S. F. Mjølsnes og M. E. G. Moe [2]. Noe av bakgrunnsteorien er hentet fra min prosjektoppgave ”Aksessmetoder i trådløse bynett”[3] som jeg skrev høsten 2006 ved *NTNU*.

Jeg vil takke Stig Frode Mjølsnes for utformingen av denne oppgaven, veiledning underveis og utlån av Pocket PC. Jeg vil også takke Pål Sturla Sæther og Asbjørn Karstensen ved drift for hjelp med oppsettet av *Linux Ubuntu*.

Denne oppgaven har gitt meg en dypere og praktisk forståelse av teorien jeg lærte i faget TTM4137 Mobil sikkerhet og skrev om i prosjektoppgaven min [3]. Dette er lærdom jeg garantert får bruk for når jeg nå i høst tar fatt på min jobb som radioplanlegger hos Telenor Mobil AS.

Oppgaven har også introdusert meg for *Linux*. Før oppstarten av denne oppgaven hadde jeg minimal erfaring med dette operativsystemet. Mitt skeptiske syn til *Linux* har snudd i løpet av dette semesteret – og jeg kan nå se for meg å jobbe videre med dette operativsystemet.

INNHOLDSFORTEGNELSE

FIGURLISTE

TABELLISTE

FORKORTELSER

AES	Advanced Encryption Standard
AP	Aksesspunkt
AS	Authentication Server – typisk RADIUSserver
BSD	Berkeley Software Distribution
CA	Certificate Authority
CBC	Cipher Block Chaining
CCMP	Counter Mode with Cipher Block Chaining MAC
CHAP	Challenge Handshake Authentication Protocol
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Server
DoS	Denial of Service
EAP	Extensible Authentication Protocol
EAPOL	EAP Over LAN
ECB	Electronic Code Book
FMS	Fluhrer-Mantin-Shamir
GTK	Group Transient Key
GUI	Graphical User Interface
ICV	Integrity Check Value
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IV	Initialization Vector
KCK	Key Confirmation Key

KEK	Key Encryption Key
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MAC	Medium Access Control
MIC	Message Integrity Code – Integritetssjekk
MPDU	MAC Protocol Data Unit
MS-CHAP	Microsoft-CHAP
MSK	Master Session Key
NAS	Network Access Server – i RSN er dette snakk om AP
NTNU	Norges Teknisk- Naturvitenskapelige Universitet
PAP	Password Authentication Protocol
PKI	Public Key Infrastructure
PMK	Pairwise Master Key
PPP	Point-to-Point Protocol
PSK	Pre-shared Key
PTK	Pairwise Temporary Key
RADIUS	Remote Authentication Dial-In Service
RC4	Rivest Cipher 4
RSA	Rivest, Shamir og Adleman
RSN	Robust Security Network
RSNA	RSN Association
SIM	Subscriber Identity Module
SSID	Service Set Identifier
SSL	Secure Socket Layer
STA	Stasjon. Eksempelvis en bærbar pc med trådløskort
SQL	Structured Query Language
TCP	Transmission Control Protocol
TKIP	Temporal Key Integrity Protocol
TLS	Transport Layer Security
TSC	TKIP Sequence Counter
WEP	Wired Equivalent Privacy
WLAN	Wireless (Wi-Fi) Local Area Network
WPA	Wi-FI Protected Access

1. INNLEDNING

1.1 Bakgrunn

I emnet TTM4137 Mobil sikkerhet er det utviklet en student lab [2] som muliggjør uttesting av sårbarhet i *IEEE 802.11* teknologi.
Denne oppgaven skal basere seg på dette og videreutvikle dette verktøyet slik at det egner seg for å gjøre undersøkelser av *Robust Security Network (RSN)* sikkerhetsarkitektur spesifisert i *IEEE 802.11i* – med en separat, sentralisert autentiseringsserver. *RSN* innbefatter *AES-CCMP* krypto og et trenivå nøkkelhierarki.

Det er også interessant å finne ut om dette oppsettet egner seg i samspill med en handholdt enhet. I tilfelle også hvordan en kan koble seg opp mot Trådløse Trondheim, som en del av laben. Dette lar seg ikke gjennomføre på universitetsområdet før Trådløse Trondheim er tilstrekkelig utbygd til også å ombefatte dette. Før en får dekning på universitetsområdet må en ta med seg enheten innenfor dekningsområdet til Trådløse Trondheim – for eksempel inn til Trondheim sentrum, Midtbyen.

Oppgavene bygger også delvis videre på min prosjektoppgave ”*Aksesstyring for trådløse bynett*” [3]. Noe av bakgrunnsteorien er hentet derfra. Jeg har også besluttet å ta med teorien bak *WEP* og *TKIP* grunnet at jeg føler dette er nødvendig for forståelsen av *WPA* og *RSN*. *WEP* benyttes også i den originale laboppgaven [2].

1.2 Problemstilling

Oppgaven går ut på å sette seg inn i nett-teknologien, tilegne seg forståelse og ferdigheter i verktøyet, og sette opp og beskrive et testoppsett av styring av brukeraksess i *RSN* basert på *IEEE 802.1X* modellen med fokus på gjensidig autentisering ved hjelp av digitale sertifikater og *EAP-TLS* over *RADIUS*. Basert på denne uttestingen skal kandidaten gjøre sin vurdering av hvor godt denne løsningen vil fungere teknisk og organisatorisk for brukere med små mobile terminaler i praksis.

1.3 Avgrensninger

Oppgaven er avgrenset til å kun omhandle en supplikant og en autentikator. En mulig utvidelse av oppgaven kunne ha vært å legge til flere av de – dette vil ikke være et problem da fremgangsmåten er den samme.

Oppgaven er også begrenset til kun å omfatte *EAP-TLS,* og ikke resten av *EAP* metodene, siden de ikke er relevant for denne laboppgaven. En mulig utvidelse av oppgaven kunne vært å sette opp laben med andre *EAP* metoder. *EAP-SIM* er en spennende metode som innebefatter autentisering ved hjelp av *SIM*-kortet på mobiltelefoner. Viser til prosjektoppgaven min fra i høst [3] om opplysninger utover det som er skrevet i denne masteroppgaven.

Gjennomføring av laben på en handholdt enhet lot seg ikke gjennomføre på grunn av mangel på riktig utstyr. Jeg fikk utdelt den eneste enheten som var tilgjengelig, en Qtek9000. Under forsøket med å koble denne til laboppsettet, viste den seg å ikke støtte *RSN* og *EAP-TLS*. Enheten støttet *IEEE 802.1X, TKIP* og *PEAP* – så med diverse endringer av laben ville det være mulig å koble til enheten. Men siden laben da ikke vil basere seg på *RSN* og gjensidig autentisering, noe som er et krav i problemstillingen, ble ikke dette gjennomført.

Hvis ikke rett utstyr er tilgjengelig til høsten, når laben skal gjennomføres, anbefaler jeg å implementere løsningen med oppkobling mot Trådløse Trondheim på handholdte enheter. Denne løsningen baserer seg ikke på *RSN*, men vil likevel gi studentene verdifull erfaring og opplæring i bruken av handholdte enheter. Metoden for oppkobling av *RSN* på handholdte enheter vil sannsynligvis ikke variere alt for mye i forhold til oppsettet som vises her i kapittel 4.2. Dette kan oppgaven eventuelt utvides med.

1.4 Oppbygning

Oppgaven er delt opp i seks kapittel der kapittel 1 er innledning og selve oppgaven er i kapittel 2 til 4. Konklusjonen er i kapittel 6, mens det er avsatt plass i kapittel 5 for diskusjoner rundt oppgaven.

- Kapittel 2 er den teoretiske bakgrunnen der det blir tatt for seg teorien som er nødvendig for forståelsen av denne og mye av den originale laben.
- Kapittel 3 tar for seg forberedende oppstart av laben og hvilke endringer som må gjøres i konfigurasjonsfilene.
- Kapittel 4 tar for seg selve kjøringen av laben og en gjennomgang av hvordan dette oppsettet vil fungere på en handholdt enhet.

I Appendiks A til D er det vedlagt de fullstendige konfigurasjonsfilene og kjørelog, samt hele gjennomgangen av sertifikat genereringen.

2. TEORETISK BAKGRUNN

Dette kapitlet vil ta for seg en grundig bakgrunn i den teknologien som er nødvendig for forståelsen av denne masteroppgaven. For ytterlige opplysninger vises det til [1] og "*Aksessmetoder i trådløse bynett*" [3].

2.1 IEEE 802.11

IEEE 802.11 er den gjeldende *Wi-Fi* standarden i dag. Den består av en rekke standarder med *802.11g* som den mest kjente – det neste tilskuddet er *802.11n.* Den standarden vil tilby hastigheter opp til 100 Mbps. Der har det kommet to utkast – standarden ventes ferdigstilt i løpet av 2007.
IEEE 802.11 standarden definerer retningslinjene for *WLAN* som opererer i 2.4 og 5 GHz båndet:

- Produkter som benytter seg av *802.11b* standarden opererer i 2.4 Ghz båndet med en maksimal båndbredde på opptil 11 Mbps.
- Produkter som benytter seg av *802.11a* standarden opererer i 5 Ghz båndet med en maksimal båndbredde på opptil 54 Mbps.
- Produkter som benytter seg av *802.11g* standarden opererer i samme 2.4 Ghz bånd som *802.11b,* men med en maksimal båndbredde på opptil 54 Mbps.

Dette del-kapitlet vil ta for seg den grunnleggende teorien bak *WEP*.

2.1.1 Wired Equivalent Privacy – WEP

Den første krypteringsmetoden tatt i bruk i trådløse nettverk. Viste seg å være ekstremt enkel å dekryptere ved hjelp av å utnytte svakheter i algoritmen.

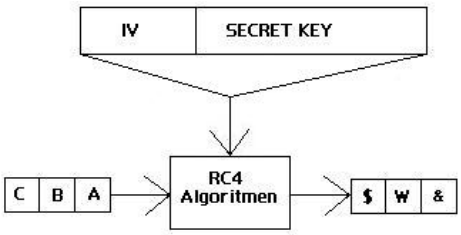
Kryptering i *WEP* kan bestå av to typer. Enten av en 40 bits nøkkel og en 24 bits *Initialization Vector, IV*, eller en 104 bits nøkkel og en 24 bits *IV* (*WEP2*).

En *Initialization Vector, IV,* er en blokk av typisk 24 bits (i *WEP*) som er forskjellig for hver datapakke en sender. En kan sende opptil ca 17 millioner (2^{24}) *IV*er før en begynner å bruke opp igjen tidligere sendte vektorer (i *WEP*). Etter noen timer har alle vektorene blitt brukt minst en gang. Hvis det benyttes samme *IV* sammen med samme krypteringsnøkkelen flere ganger, danner de basisen for et angrep.

Ved å legge en 24 bits *IV* på *WEP*-krypteringsnøkkelen forandrer en den 40 bits nøkkelen til en ny unik nøkkel på 64 bits (evt. 104 og 128 bits i *WEP2*). Dette gjøres uavhengig av om den samme krypteringsnøkkelen er blitt benyttet i en annen overføring av samme klarteksten med en annen *IV*.

*IV*er blir implementert forskjellig i blokk- og strømkryptering.

I blokkryptering vil krypteringen av den samme klarteksten med den samme nøkkelen gi den samme krypterte teksten hver gang. Ved å benytte XOR mellom en *IV* og blokkene av klarteksten unngår en dette sikkerhetsproblemet, som vist i figur 2.1.1.1.



Figur 2.1.1.1: *Bruk av IV* [1] side 75

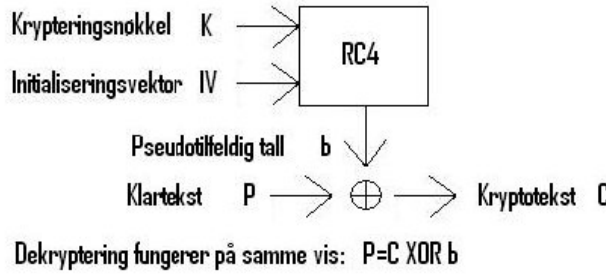
I strømkryptering blir det først produsert en rekke bits, typisk 256 bits, uten bruk av *IV*er. Disse blir ikke benyttet videre. Etter denne prosessen blir *IVer* benyttet.

Som nevnt i [3] kapittel 2.1.1, er det de svake nøklene som er den største svakheten til *RC4* algoritmen. Men hva om en legger til en *IV*-vektor til denne nøkkelstrømmen og ser bort fra de første bitene (anbefalt de første 256 bitene), og på denne måten unngå de svake nøklene? Dette kunne vært en løsning hadde det ikke vært for at *IV*en blir lagt til den hemmelige nøkkelen, og at *IV*en alltid forandrer seg. Så før eller senere vil *IV*en garantert generere en svak nøkkel. Dette er løst under *TKIP*.
Viser til [1], kapittel 6 "*How IEEE 802.11 WEP Works and Why It Doesn't*" for forklaring utover dette.

Siden *IV*-verdien må sendes ukryptert over nettet kan en angriper lett plukke opp disse pakkene for bruk i et angrep for å skaffe seg uautorisert tilgang til aksesspunktet.

Problemene med *IV*er illustreres ved at det er vanskelig å designe sikkerhetsprotokoller som baserer seg på strømkryptering. Mye på grunn av at den indre tilstanden til krypteringsprosessen ikke blir startet på nytt i løpet av en strøm. En løsning på dette blir tatt for seg under kapittel 2.2.2 – *TKIP***.**

WEP bruker *IV*er for å forsikre seg om at den pseudotilfeldig tallrekken fra *RC4*-algoritmen ikke blir gjenbrukt – noe som danner basis for angrep. Dette er vist i figur 2.1.1.2.



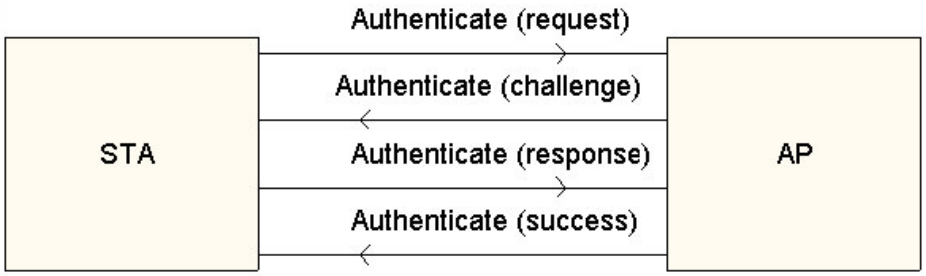
Figur 2.1.1.2: *RC4 brukt i WEP* [1] side 318

Uheldigvis for *WEP* lekker *RC4*-algoritmen ut informasjon om den hemmelige krypteringsnøkkelen. Forskere har funnet ut at når *RC4* blir brukt sammen med en *IV* og krypteringsnøkkelen vil enkelte *IV*-verdier produsere en svak nøkkel. En angriper som sniffer opp nok av disse svake nøklene kan regne ut den hemmelige krypteringsnøkkelen i lineær tid basert på nøkkelstørrelsen. Med andre ord vil dermed 104 bits krypteringsnøkkelen i *WEP2* bare ta litt lengre tid å knekke. Alle angrep mot

RC4-algoritmen baserer seg på de første bitene som genereres ut fra algoritmen.

I laben i emnet TTM4137 Mobil sikkerhet gjennomførte studentene en test [2] på hvor lang tid det tar å knekke en 64 bits *WEP* kryptering. Ved å påtrykke trafikk inn i nettet tok det dem fra en time til en dag å knekke denne krypteringen. Teoretisk nedre grense for å knekke denne krypteringen er cirka 3 minutter! Grunnen til at det tok så mye lengre tid i laben [2] var at det ble benyttet en *Cisco* router som var ekstremt avansert, og som unnlot å bruke de fleste svake *IV*ene.

Autentiseringssekvensen i *WEP* baserer seg på en challenge-response algoritme, som vist i figur 2.1.1.3. Problemet med denne algoritmen er at aksesspunktet (*AP*) sender klienten (*STA*) en tilfeldig ukryptert tekststreng på 128 bits som *STA* så skal kryptere (*challenge*). Etter krypteringen sender *STA* den krypterte teksten tilbake til *AP* (*response*).



Figur 2.1.1.3: *Challenge-response algoritmen i WEP* [1] side 71

Det er her den største svakheten i autentiseringsprosessen foregår. En hacker har nå kunnet snappe opp både klarteksten og den krypterte teksten, og kan dermed regne seg frem til hvilken som helst krypteringsnøkkel ved å bruke XOR på klarteksten og den krypterte teksten. Se eksempelet i [3], kapittel 2.1.1.

Autentiseringsprosessen er en stor svakhet fordi brukeren kan bli autentisert uten å vite nøkkelen. Med tiden trenger klienten uansett å vite nøkkelen for å skjønne de krypterte meldingene. Nøkkelen oppnås ved å utnytte de svake vektorene.

Tabell 2.1.1.1 viser en oppsummering av sikkerhetssvakhetene i *WEP*.

1	*IV*-verdien er for kort og ikke beskyttet mot gjenbruk
2	Måten nøkler konstrueres fra *IV*en gjør *WEP* mottakelig for angrep på de svake nøklene (*Fluhrer-Mantin-Shamir – FMS, attack*)
3	Finnes ingen effektiv måte å detektere en modifisert melding (meldingsintegritet)
4	*WEP* bruker hovednøkkelen direkte, og har ingen innebygd protokoll for oppdatering av nøkler
5	Ingen beskyttelse mot meldingsgjentagelse (*message replay*)

Tabell 2.1.1.1: *Svakheter i WEP* [1] side 234

En mulig forbedring av *WEP*-protokollen hadde vært innføring av *Public Key Infrastructure – PKI*. Dette ble ikke gjort.
Dermed klassifiserer *WEP* seg som en ikke-sikker krypteringsprotokoll, og jobbingen med en ny sikkerhetsmekanisme tok til. Denne blir kalt *Wi-Fi Protected Access – WPA*.
Viser til [1], kapittel 6 ”*How IEEE 802.11 WEP Works and Why It Doesn’t*” for opplysninger utover dette.

2.2 IEEE 802.11i – Robust Security Network, RSN

Tillegget til den standarden som spesifiserer den nye generasjonen innenfor sikkerhet blir kalt *IEEE 802.11i*. Standarden er utarbeidet av *Institute of Electrical and Electronics Engineers – IEEE*, der *802.11 Task Group* utarbeider standarder for trådløs teknologi. *IEEE 802.11i* definerer en ny type av trådløse nettverk kalt *Robust Security Networks – RSN*.

IEEE 802.11i er designet for å gi økt sikkerhet i *Medium Access Control, MAC*-laget for *IEEE 802.11* nettverk. Den definerer to sikkerhetsalgoritmer: *RSN* og pre-*RSN*. Pre-*RSN* består av *WEP* og *802.11* entitetsautentisering. *RSNA* tilbyr to datakonfidensialitetsprotokoller – *TKIP* og *CCMP*. Den inkluderer også *IEEE 802.1X* autentisering og nøkkelhåndteringsprotokoller.

Dette del-kapitlet skal ta for seg teorien bak den nye krypteringsalgortimen *AES*, samt *WPA* og *TKIP*.

2.2.1 Wi-Fi Protected Access – WPA

WPA var ment som en midlertidig løsning i påvente av *RSN*. *WPA* ble introdusert før *RSN* (også kalt *WPA2*) var ferdigstilt fordi *IEEE 802.11* gruppen mente at sikkerheten i *WEP* var langt fra tilfredsstillende, og at forbrukerne fortjente en sikkerhetsmetode som fungerte. Denne måtte lanseres så raskt som mulig.

WPA er en sikkerhetsløsning som er designet for å være kompatibel med alt av utstyr som *WEP* støttet. På denne måten skulle overgangen for forbrukerne være så smertefri som mulig. De skulle ikke ha behov for å kjøpe inn nytt utstyr, men kun å installere ny programvare. Dette førte til defineringen av *Temporal Key Integrity Protocol – TKIP*. Denne protokollen baserer seg på at det skal benyttes forskjellige engangskrypteringsnøkler for hver pakke eller sesjon som krypteres. *TKIP* protokollen er tillat som en valgfri modi under *RSN.* Mer om denne protokollen i kapittel 2.2.2.

Siden *WPA* baserer seg på den samme svake *RC4*-krypteringsalgoritmen som *WEP*, stiller sikkerhetseksperter seg skeptisk til denne protokollen. De velger heller å benytte seg av den nye standarden *RSN/WPA2* som baserer seg på den sikre *AES*-krypteringsalgoritmen. Mer om denne metoden i kapittel 2.2.3 og 2.2.5.

Viser til [1], kapittel 7 ”*WPA, RSN, and IEEE 802.11i*” for utdypende informasjon utover dette.

2.2.2 Temporal Key Integrity Protocol - TKIP

TKIP ble laget for å oppgradere *WEP* protokollen til å bli en sikker protokoll. Den har også blitt inkludert i *WPA/RSN* protokollen i *IEEE 802.11i*.
Det var behov for en løsning på problemene i *WEP* protokollen. Løsningen var å oppgradere eksisterende systemer med *TKIP* protokollen. På denne måte slapp forbrukerne å investere i nytt utstyr. En programvareoppdatering var alt som var nødvendig for å gjøre protokollen sikker.
TKIP kan ikke forandre på de store løsningene i *WEP* protokollen, som *RC4* algoritmen, men den kan legge til diverse feilkorrigerende verktøy rundt den eksisterende hardwaren. Hva som ble gjort er vist i tabell 2.2.2.1. Tallene i parentes viser til hvilke punkter i tabell 2.1.1.1: "*Svakheter i WEP*" den håndterer.

Hensikt	**Forandring**	**Svakhets punkt**
Meldingsintegritet	Tilføyer en meldingsintegritetsprotokoll for å forhindre tukling	(3)
IV-utvelgelse og -bruk	Forandrer reglene for hvordan *IV*-verdier blir utvalgt og bruker om igjen *IV*en som en gjenbruksteller (replay counter)	(1), (3)
Nøkkel for hver pakke	Forandrer krypteringsnøkkelen for hver pakke som sendes	(1), (2), (4)
IV-størrelsen	Øker størrelsen på *IV*en for å forhindre at samme *IV* brukes på nytt	(1), (4)
Nøkkelhåndtering	Tilføyer en mekanisme for å distribuere og forandre broadcastnøklene	(4)

Tabell 2.2.2.1: *Forrandringer fra WEP til TKIP* [1] side 235

Meldingsintegritet er en essensiell del av sikkerhet. Hvis en angriper klarer å modifisere en melding, så er det mange muligheter for hvordan systemet kan bli kompromittert. *WEP* benytter seg av en *Integrity Check Value – ICV* for å detektere modifiserte meldinger. Denne metoden tilbydde egentlig ingen sikkerhet. Derfor er ikke *ICV* betraktet som en del av *TKIP* – selv om verdien fremdeles sjekkes.

For å detektere tukling finnes det en metode som multipliserer sammen alle bytene i alle sendte og mottatte meldinger for å generere en sjekksumsverdi – for deretter å sende denne verdien sammen med selve meldingen. Mottageren kan utføre samme

beregning og verifiserer at den har samme verdi. Om bare én bit blir forandret blir summen betraktelig forandret.
En slik simpel metode er ikke brukbar innenfor sikkerhet fordi angriperen enkelt kan generer en ny gyldig sjekksum etter han har modifisert en melding.
I *TKIP* benyttes en lignende metode. Den kombinerer alle bit fra alle sendte og mottatte meldinger for å produsere en sjekkverdi som blir kalt *Message Integrity Code – MIC,* som sendes sammen med meldingen. Denne verdien blir generert ved hjelp av en spesiell ikke-reversibel prosess forbundet med en hemmelig nøkkel. Dermed kan ikke en angriper regenerere *MIC*-verdien uten at han vet den hemmelige nøkkelen. Bare den påtenkte mottakeren kan regenerere denne verdien og sjekke resultatet.

En mye brukt metode for å generere en slik *MIC*-verdi er *MIChael*. Det er en metode som krever få ressurser. Dermed er dette en populær metode grunnet de begrensede ressursene som finnes på trådløskort og i aksesspunkt.
MIChael er sårbar mot *brute-force* angrep. Den enkleste metoden for å demme opp mot denne trusselen er å slå av selve nettverket om et angrep detekteres. *MIChael* stenger ned kommunikasjonen i ett minutt. Noe som fører til at en angriper kun kan angripe nettet en gang hvert minutt. Dermed vil et angrep ta ekstremt lang tid. Men samtidig er ikke nettet tilgjengelig for autentiserte brukere. Dette kan dermed klassifiseres som et *Denial of Service – DoS,* angrep.
En kort oppsummering av *MIChael* viser at den genererer en *MIC*-verdi som blir addert sammen med meldingen før krypteringen finner sted, og sjekket av mottaker etter dekryptering. Denne verdien gir meldingsintegriteten som manglet i *WEP*.

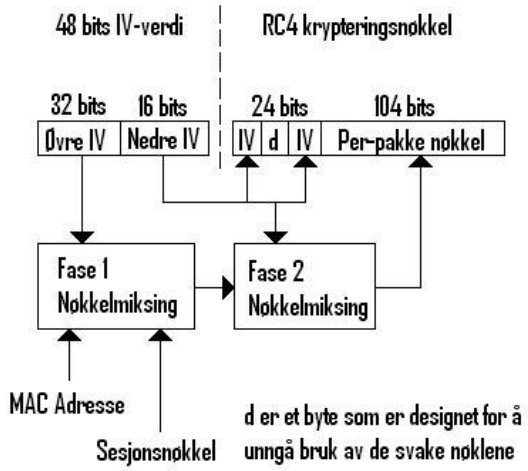
Bruken av *IV*er er også oppgradert i *TKIP*. I *WEP* hadde *IV*ene følgende svakheter:

- *IV*-verdien er for kort – slik at *IV*er blir gjenbrukt
- *IV*-verdien er ikke eksplisitt satt hos en stasjon – slik at flere stasjoner kan generere samme *IV*-verdi samtidig
- Måten *IV*en genereres sammen med nøkkelen gjør den mottagelig for *FMS* angrep

Det er følgende tre forskjeller mellom måten *IV*er blir brukt på i *TKIP* og *WEP*:

- *IV* størrelsen økes fra 24 til 48 bits
- *IV*-verdien har en birolle som sekvensteller for å unngå gjenbruksangrep, *replay attacks*.
- *IV*en er instruert til å unngå de svake nøklene.

En mye brukt metode for å unngå de svake *IV*-verdiene er å ikke benytte seg av disse. Men dette reduserer bare antall mulige *IV*er for bruk. En bedre metode for dette er å øke antall bits på *IV*en. I *TKIP* økes denne fra 24 til 48 bits. På denne måten økes gjenbrukshyppigheten til *IV*er fra ca én time i *WEP* til over 900 år i *TKIP*!
Å øke *IV* lengden virker enkelt nok, men det fører til problemer når en kommer til praktisk implementering. I *WEP* blir *IV*en lagt til i fronten av den hemmelige nøkkelen, og lager en 64 bits *RC4* krypteringsnøkkel. Hardwaren er dermed designet for akkurat denne typen krypteringsnøkkel og får dermed problemer med en større nøkkel. I *TKIP* er dette løst på en interessant måte. I stedet for å lage en ny *RC4* krypteringsnøkkel så blir *IV*en delt i to deler. De første 16 biten fra den nye *IV*en blir paddet ut til 24 bits for å passe inn i en måte som unngår de svake nøklene. Denne 24 bits verdien blir så brukt på samme måte som i *WEP*. Men i stedet for å legge denne verdien til en krypteringsnøkkel blir det laget en mikset nøkkel sammensatt av den hemmelige nøkkelen og de resterende 32 (48-16=32) biten fra *IV*en.
Denne metoden fører til at verdien av nøkkelen som blir brukt i *RC4* krypteringen er forskjellig for hver *IV*-verdi. Den fører også til at strukturen til *RC4* nøkkelen er en 24 bits *IV*-verdi sammensatt med et 104 bits hemmelig nøkkelfelt. Disse målene blir nådd ved hjelp av en 48 bits *IV*-verdi. Dette er vist i figur 2.2.2.1.



Figur 2.2.2.1: *Laging av RC4 krypteringsnøkkelen* [1] side 240

Ved å kombinere sesjonsnøkkelen, deler av *IV*en og *MAC*-adressen lages en mikset nøkkel. Ved å mikse inn *MAC*-adressen i krypteringsnøkkelen garanterer en at samme nøkkel ikke vil bli generert av to forskjellige brukere. *MAC*-adressen skal være unik for hver bruker. For å ikke belaste for mange ressurser er kombineringen delt inn i to faser. I fase én blir sesjonsnøkkelen og *MAC*-adressen mikset. I fase to, som utføres for hver pakke, blir *IV*en mikset sammen med resultatet fra fase én for å produsere krypteringsnøkkelen. Denne nøkkelen brukes så for å initialisere *RC4* krypteringshardwaren.
Fase to kan utføres på forhånd fordi *IV*-verdien økes inkrementelt. Dermed vet stasjonen hvilken *IV*-verdi som vil bli brukt neste gang.

For å demme opp for muligheten til et *replay* angrep innfører *TKIP* noe som heter *TKIP Sequence Counter – TSC*. I realiteten er *TSC* og en *IV*-verdi én og samme ting. Verdien starter på 0 og inkrementeres med 1 for hver pakke som sendes. I *TKIP* er *IV* verdien garantert mot gjenbruk. Om en bruker mottar en pakke som har en *TSC* verdi mindre enn, eller lik, sist mottatte verdi, kan han konkludere med at dette er en *replay* melding. Men dette vil dessverre ikke fungere i praksis. Hva om en pakke går tapt under overføringen? Da vil jo denne meldingen komme frem for sent i forhold til neste pakke, og dermed få en for lav *TSC*-verdi. En metode for å unngå dette er å kun tillate en mottatt kopi av hver pakke med lik *TSC*-verdi. Dersom det kommer en pakke med en *TSC*-verdi som er mottatt før, vil denne kastes og kategoriseres som en *replay*-melding.

For å unngå det kjente *FMS* angrepet i *WEP* har *TKIP* doblet lengden på *IV*en ved å koble den sammen med en *TSC*-verdi for å lage en hemmelig nøkkel som er mye mer kompleks enn den som benyttes i *WEP*. *TKIP* kutter også ut bruken av de svake nøklene. På denne måten kan *TKIP* benyttes med samme hardware som *WEP* og dermed slipper forbrukerne å investere i nytt utstyr. Disse forandringene til bruken av *IV*er fører til en betraktelig økning av sikkerheten i *WEP*.

Alle problemene med *WEP* er dermed løst ved implementeringen av *TKIP*. Løsningene tatt for seg her tillater bakoverkompabilitet med hardwaren brukt i *WEP*.

Selv om nøkkelmiksingsfunksjonen i *TKIP* har bedre sikkerhet enn *WEP*, er den ikke så sikker som antatt. Det er mulig å finne *MIC*-nøkkelen gitt en per-pakke-nøkkel; dessuten er hele sikkerheten brutt under varigheten av en *Temporal Key*, gitt to per-pakke-nøkler med samme *IV*. Denne sårbarheten betyr ikke at *TKIP* er en usikker metode, men den avslører at deler av *TKIP* er svak på egenhånd.

Viser til [1], kapittel 11 ”*TKIP*” for forklaring utover dette.

2.2.3 Advanced Encryption Standard – AES

Vinneren av den internasjonale konkurransen i kjølvannet av at *DES* algoritmen ble klassifisert som uholdbar. Utviklet av *Joan Daemen* og *Vincent Rijmen*.
AES er den mest populær blokkrypteringsalgoritmen av krypteringsalgoritmer som benytter seg av symmetriske nøkler.

Metoden kombinerer en nøkkel og en 128 bits ukryptert datablokk for å produsere en blokk av kryptert data. Det er ikke mulig å utføre denne overgangen uten kjennskap til den hemmelige nøkkelen. De krypterte og ukrypterte blokkene er samme størrelse. Konvertering av en 128 bits blokk er det eneste *AES* gjør – men den gjør det ekstremt effektivt og er veldig sikker. Det er svært tvilsomt at noen svakheter vil oppdages i denne algoritmen.

Denne standarden vil diskuteres nærmere i kapittel 2.2.5.

2.2.4 Wi-Fi Protected Access 2/Robust Security Network – WPA2/RSN

RSN er en protokoll som bygger på *Advanced Encryption Standard – AES* algoritmen og ikke den typiske *RC4* algoritmen som benyttes i *WPA/TKIP* og *WEP*. *AES* betegnes som en meget sterk krypteringsalgoritme – noe *RC4* ikke er.
Men med denne overgangen fra *RC4* til *AES* kommer det med diverse overgangsproblemer. Det samme utstyret som ble brukt i *WEP* og *WPA/TKIP* kan nødvendigvis ikke benyttes i *RSN*. Forbrukerne må da kjøpe seg nye nettverkskort og aksesspunkt som støtter denne løsningen.

WPA/TKIP er spesifisert slik at man *kan* benytte krypteringsalgoritmen *AES*, mens *WPA2/RSN* derimot *krever* at *AES* skal være støttet, noe som ofte medfører at et eldre trådløst nettverkskort og aksesspunkt ikke kan benyttes. *WPA2/RSN* kan oftest finnes under *Innstillinger* i aksesspunktet i to varianter – *Personal* og *Enterprise*. De skilles på hvordan de autentiserer brukere av det trådløse nettet. *Personal* fungerer på samme måte som man tidligere er vant til, hvor alle brukere må ha tilgang til et felles passord. Mens *Enterprise* benytter seg av protokollene *IEEE 802.1X* og *EAP*.

I *WPA2/RSN* benyttes nøklene annerledes enn i *WEP*. Mens *WEP* benyttet hovednøkkelen både til kryptering og dekryptering, brukes den kun til å generere nye nøkler i *WPA2/RSN*. En oversikt over nøkkelhierarkiet er vist i figur 2.2.4.1. Dermed er risikoen for eksponering av den parvise masternøkkelen svært liten. Basert på en delt hemmelighet genereres det midlertidige nøkler som kun benyttes en gang per sesjon eller pakke. På denne måten reduseres trusselnivået om en sesjonsnøkkel skulle eksponeres.

Figur 2.2.4.1: *Nøkkelhierarki i WPA2/RSN*

Sikkerhetslagene for *WPA2*/*RSN* sikkerhet kan deles inn i tre lag:

- *WLAN*:

  Dette laget har ansvaret for selve kommunikasjonen og krypteringen/dekrypteringen.

- Tilgangskontroll:

  Dette laget har ansvaret for å vedlikeholde sikkerhetsparametrene. Det vil si å kommunisere med autentiseringslaget for kun å slippe gjennom autoriserte brukere, samt å stoppe de som ikke er autentisert.

- Autentisering:

  Dette laget har ansvaret for selve autentiseringen av brukere og distribusjon av nøkler til de. Laget kan også kjøres i *pre-shared key modus – PSK* der alle brukere benytter seg av samme forhåndsutdelte passord for å bli autentisert.
  Autentiseringen kan være lokalisert i selve aksesspunktet i mindre systemer, men i større systemer er som regel dette laget lokalisert separat i en egen server. Mer om dette i kapittel 2.3.2.
  Med andre ord løser laget problemet med nøkkelhåndtering i *WEP* og gjør det enklere for bedrifter å implementere *WLAN*-sikkerhetssystemer.

Hvis pre-*RSN* og *RSNA* algoritmene tillates kjørt samtidig i et *WLAN*, bør brukere få lov til å velge hvilken av de to den vil kjøre på før den åpner en forbindelse. Aksesspunktene bør pålegge forskjellige rettigheter for de to. Om dette ikke blir gjort kan en angriper avsløre sikkerheten til hele systemet ved hjelp av et *Security Level Rollback Attack*.
For å unngå refleksjonsangrep bør ingen enhet kunne operere som autentikator og supplikant under den fire-veis *handshake*en med samme *PMK*.

Generelt vil en fullført autentisering si at supplikanten og autentikatoren verifiserer hverandres identitet og genererer en delt hemmelighet for videre generering av nøkler. Basert på denne delte hemmeligheten genererer og distribuerer nøkkelhåndteringsprotokollene nøkler som brukes i dataoverføringssesjoner.

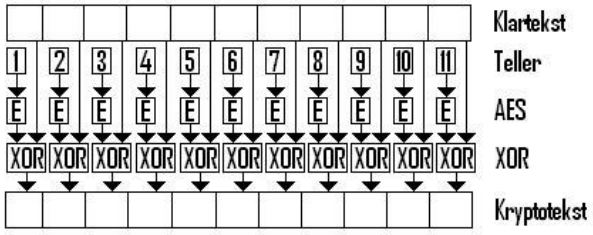
Viser til [1], kapittel 10 ”*WPA and RSN Key Hierarchy*” for utdypende informasjon utover dette.

2.2.5 AES Counter Mode with Cipher Block Chaining MAC Protocol – AES CCMP

AES-CCMP er den sikreste sikkerhetsprotokollen som er utviklet for *IEEE 802.11i.* I kapittel 2.2.2 ble *TKIP* gjennomgått. Selv om det er en meget sikker metode, er den ikke standard modi i *IEEE 802.11i* – det er blokkrypteringsalgoritmen *AES*.
AES-basert sikkerhet kan betraktes som sikrere enn *TKIP*-basert sikkerhet.

I *WPA2/RSN* er sikkerhetsprotokollen bygget rundt en løsning med *AES* som blir kalt *CCMP.* Den definerer et sett med regler som bruker *AES* som en blokkrypteringsalgoritme for å aktivere kryptering og beskyttelse for data. En grunn til at *CCMP* betraktes som sikrere enn *TKIP* er at den er bygget opp fra grunnen av – og ikke på en kompromissmetode slik som *TKIP*.
Valget av *AES* til *IEEE 802.11i* ble tatt før problemene med *WEP* ble kjent. Dermed var hensikten med protokollen at den på sikt skulle erstatte *WEP.* Det var ikke forventet at eksisterende hardware skulle være kompatibel den nye standarden. Dette fordi *AES* implementeres annerledes enn *RC4.* Men da feilene med *WEP* ble kjent var det behov for å bytte ut alt av hardware. Dette førte til utviklingen og lanseringen av *TKIP*, som skulle fungerer som en overgang mellom *WEP* og *WPA2/RSN* i en overgangsperiode. Som et resultat av dette har en nå tre potensielle løsninger: *WEP, TKIP(WPA)* og *CCMP(RSN).*
Det er store likheter mellom *WPA/TKIP* og *RSN/CCMP* baserte systemer. Blant annet nøkkelhåndteringen. Den største forskjellen skjer i de lave lagene der data blir kryptert og dekryptert.

AES kan operere i *Counter Mode* – tellemodi. Denne metoden benytter ikke blokkrypteringen direkte for å kryptere data. I stedet krypterer den verdien av en tilfeldig verdi kalt en *counter* – teller, for så å XOR resultatet med klarteksten for å produsere kryptoteksten. Telleren blir normalt inkrementert med 1 for hver produserte blokk – derav navnet. Dette er vist i figur 2.2.5.1.



Figur 2.2.5.1: *Eksempel på counter mode* [1] side 266

I dette eksemplet blir meldingen delt inn i blokker som hver blir XORet med de *AES*-krypterte telleverdiene. Denne telleverdien starter på 1 i eksemplet. Dette er ikke alltid realiteten – som regel starter denne på en tilfeldig verdi. En viktig ting er at mottakeren må vite hvilken verdi telleren starter på – samt hvordan den inkrementerer. Fordi telleren er forskjellig for hver blokk unngår en problemet med repeterende blokker under *Electronic Code Book – ECB* (ikke tatt for seg i denne oppgaven). Viser til [1], side 265 og 266 for forklaring utover dette.
Counter Mode har vært brukt i over tjue år, og er en kjent og tiltrodd metode. Dens enkelhet og modenhet gjør den attraktiv for bruk i *WPA2/RSN*. *Counter Mode* tilbyr ikke noe meldingsautentisering – kun kryptering. Derfor må dette legges til under *WPA2/RSN*.

Counter Mode + CBC MAC – CCM, ble laget spesielt for å passe inn i *IEEE 802.11i – WPA2/RSN*. Den bygges oppå *Counter Mode*. *CCM* bruker *Counter Mode* i forbindelse med en meldingsautentiseringsmetode kalt *Cipher Block Chaining – CBC*. Den brukes til å produsere en *MIC*.

En annen kjent metode er *CBC-MAC*. Den tar første blokken i meldingen og krypterer den med *AES*. Dette resultatet blir XORet med den andre blokken og deretter kryptert, og så videre. Resultatet blir en 128 bits blokk som kombinerer all data i meldingen.

Ulikt *RC4*-algoritmen brukt i *WEP* og *TKIP* bruker *CCMP CCM* modusen av *AES*-algoritmen med en 128 bits nøkkel og 128 bits blokkstørrelse. *CCMP* kombinerer *Counter Mode* for datakonfidensialitet og *CBC-MAC* for dataintegritet ved hjelp av en 8 oktett *MIC* og et 2 oktett *Length* felt.

CCMP tilbyr også en *MIC*-beskyttelse av både ramme*body*en og nesten hele *header*en i en *MAC*-ramme. Dette forhindrer en angriper fra å utnytte *MAC-header*en i et angrep. I tillegg benytter *CCMP* seg av en 48 bits pakkenummer for å unngå *replay*-angrep og for å konstruere et nytt *nounce* for hver pakke. Den store størrelsen på pakkenummeret eliminerer alle tvil om pakkenummergjenbruk i løpet av en assosiering.

CCM kombinerer de to metodene *Counter Mode* og *CBC-MAC*. I tillegg legger den til følgende tre punkter:

- Spesifiserer et tilfeldig tall (nounce) slik at påfølgende meldinger blir separert kryptografisk
- Smelter sammen krypteringen og autentiseringen til en enkel nøkkel
- Utvider autentiseringen til å dekke data som ikke skal krypteres

I de fleste andre krypteringsmetoder antas det at hele meldingen skal krypteres. Dette er ikke tilfelle for *IEEE 802.11* – her skal bare deler av meldingen krypteres. Hodet på pakken inneholder *MAC*-adressen brukt til levering av rammen og andre relevante felt til bruk i *Wi-Fi LAN*. Disse feltene må sendes i klartekst slik at trådløse enheter kan fungere som de skal – derfor blir bare datadelen kryptert. Men selv om hodet sendes ukryptert, ønsker fremdeles mottageren forsikring på at meldingen ikke har blitt forandret på veien frem. For å oppnå dette tillater *CCM* kryptering på deler av meldingen som er autentisert av *CBC-MAC*.

CCMP krypterer data på *MPDU* nivå. Det vil si pakkene som sendes over radiolinken. Det er én *MPDU* for hver ramme som overføres. Denne prosesseres av *CCMP*-algoritmen for å lage en ny kryptert *MPDU*. Bare datadelen krypteres, ikke hodet.

Viser til [1], kapittel 12 ”*AES-CCMP*” for informasjon utover dette.

2.3 IEEE 802.1X – Tilgangskontroll

Dette kapitlet er bygd opp rundt basisen for sikkerhet - tilgangskontroll.
Kombinasjonen av *IEEE 802.11, IEEE 802.1X, EAP og RADIUS* danner grunnlaget for en skalerbar løsning som passer til alle nettverk - fra hjemmenettverk til store bedriftsnettverk.
Hensikten med sikkerhet er å kunne separere de godkjente fra de ikke godkjente brukerne. Med andre ord kan en ikke oppnå sikkerhet uten tilgangskontroll.

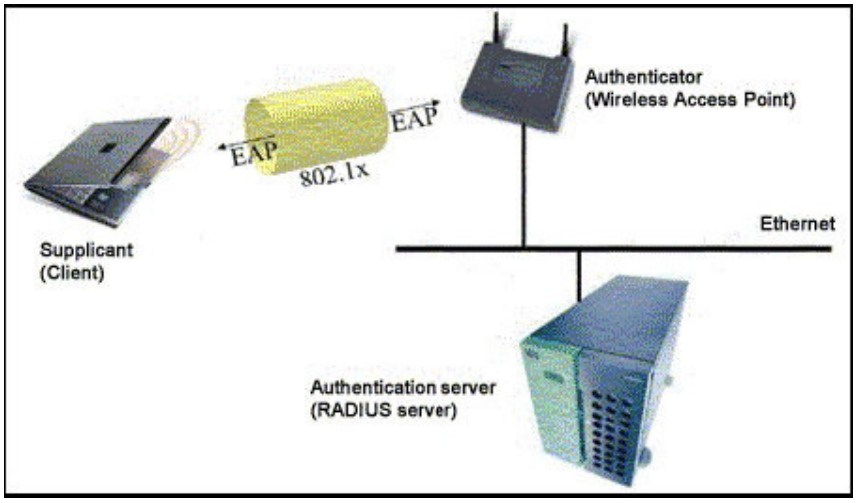
2.3.1 IEEE 802.1X

IEEE 802.1X er standarden for portbasert nettverkstilgangskontroll.
Hensikten med denne protokollen er å tilby tilgangskontroll på det tidspunktet en bruker ønsker å koble seg til nettverket. På grunn av protokollens skalerbarhet ble den raskt tilrettelagt for bruk i *Wireless LAN – WLAN, WPA* og *WPA2/RSN* løsninger.

Den deler nettverket inn i følgende tre deler, som vist i figur 2.3.1.1:

- Klient:
  Brukeren som vil koble seg til nettverket.
- Autentikator:
  Enheten som kontrollerer brukeren ved nettverksporten.
  I *WLAN* er dette aksesspunktet.
- Autentiseringsserver:
  Enheten som avgjør om brukeren får koble til eller ikke.
  Selve autentiseringsserveren er nødvendigvis ikke en utskilt enhet. Den kan også være en del av autentikatoren. Dette forekommer primært i mindre systemer.
  I *WLAN* er dette en *RADIUS*server.

Mellom autentikatoren og autentiseringsserveren benyttes det en protokoll som heter *RADIUS*. Mer om denne protokollen i kapittel 2.3.2.
Protokollen som brukes for kommunikasjon mellom klient og autentikator kalles *EAP*. Denne protokollen vil bli gjennomgått i kapittel 2.4.



Figur 2.3.1.1: *Oppsett av Extensible Authentication Protocol, EAP* [2]

IEEE 802.1X er designet for å kontrollere tilgangskontroll på individuelle *LAN*-porter. I *WLAN* gjøres dette på følgende måte:
Der blir hver mobile node betraktet som en klient som ønsker å koble seg til nettverket. For å oppnå dette må aksesspunktet lage en logisk port med en autentikator for hver bruker den kommer over. Hver autentikator er ansvarlig for tilgangskontroll for den brukeren den er tilegnet. Dette til motsetning til vanlige *LAN* der en autentikator kan holde styr på flere brukere samtidig.

I *WLAN*-systemer er det ekstremt viktig at autentiseringsprosessen kjøres flere ganger – ikke bare ved oppkobling som i vanlige *LAN*. Dette for å beskytte systemet mot identitetskapring. En angriper kan da bare vente til en klient er autentisert for deretter å stjele identiteten. Dette oppnås ved å introdusere meldingsintegritet som en del av autentiseringsprosessen. Ved hjelp av meldingsintegritet kan en sjekke at både klienten og aksesspunktet har de rette hemmelige nøklene på plass, og at kryptering er aktivert, før de gis tilgang til nettverket.

Når en ny klient vil koble seg til nettverket settes porten hos autentikatoren i en *ikke-autorisert* tilstand. I denne tilstanden kan bare *IEEE 802.1X* trafikk slippe gjennom. Trafikk av annen type vil bli blokkert. Autentikatoren sender så en *EAP-Request* melding til klienten for å kreve identiteten til klienten. Den sender så en *EAP-Response* melding til autentikatoren, som den sender videre til autentiseringsserveren. Den kan akseptere eller avslå *EAP-Request* meldingen. Om den aksepteres settes

porten til tilstanden *Autorisert* og klienten åpnes for nettverket. Når klienten logger av vil den sende en *EAP-Logoff* melding til autentikatoren, som da setter porten tilbake til en *ikke-autorisert* tilstand. Alt dette gjøres i programvare. Det vil si at det finnes ingen egen fysisk autentikator eller svitsj for hver klient, så antallet fysiske *IEEE 802.1X* entiteter i systemet forblir uforandret uansett antall klienter. Dette gjør metoden ekstremt skalerbart.

Viser til [1], kapittel 8 ”*Access Control: IEEE 802.1X, EAP, and RADIUS*” og [12] for forklaring utover dette.

2.3.2 Remote Authentication Dial-In User Service – RADIUS

RADIUS er en autentiseringsprotokoll som opererer på *TCP*-protokollen og som benyttes under *IEEE 802.1X*. Denne protokollen brukes til kommunikasjon mellom en autentikator og en autentiseringsserver – derav navnet *RADIUSserver*. Denne serveren kan lokaliseres og driftes av andre tilbydere i et annet nettverk enn dit eget. På denne måten slipper en kostnadene forbundet med innkjøp av programvare og selve serveren.

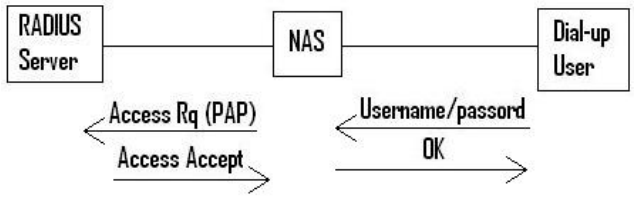
RADIUS definerer to ting. Den første er et sett av funksjonaliteter som bør være felles på tvers av autentiseringsservere. Dernest definerer den en protokoll som tillater andre enheter tilgang til disse funksjonalitetene.
Motivasjonen bak *RADIUS* var å ha en sentral autentiseringsserver som hadde kunnskap om alle brukerne og som tillot autentikatorene å sende brukerinformasjon til en sentral autentiseringsserveren for verifisering.

Kjerneprotokollen i *RADIUS* er relativt enkel. Det finnes kun fire relevante meldinger:

- Access-Request (*NAS*→*AS*)
- Access-Challenge (*NAS*←*AS*)
- Access-Accept (*NAS*←*AS*)
- Access-Reject (*NAS*←*AS*)

I *WPA2/RSN* nettverk tilsvarer aksesspunktet *NAS*, og *RADIUS*serveren *AS*. Disse fire meldingene reflekterer det faktum at *PPP*, modemoppringingsprotokollen, har to modi for autentisering: *PAP* og *CHAP*. *PAP* er en simpel brukernavn/passord metode. *CHAP* krever at serveren sender tilfeldig data, *challenge*, som klienten må kryptere og returnere for verifisering.

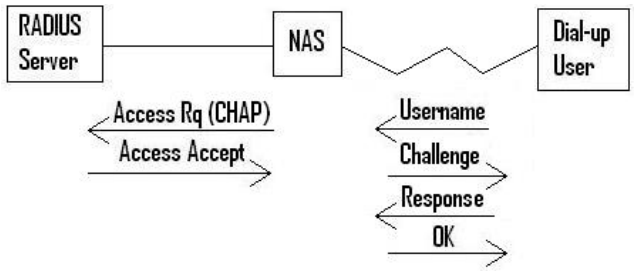
PAP operasjonen går som følger, vist i figur 2.3.2.1:
Klienten ringer opp – *NAS* svarer og indikerer at den bruker *PAP*-autentisering. Klienten logger så på med brukernavn og passord. *NAS* sender denne informasjonen i form av en *Access-Request* melding til *RADIUS*serveren. Denne serveren responderer enten med en *Access-Accept* eller *Access-Reject* melding. Dette er en veldig enkel metode – og er dermed utsatt for angrep. Det verste er faktisk at passordet sendes ukryptert, slik at en angriper lett kan plukke dette opp.



Figur 2.3.2.1: *PAP prosedyren* [1] side 140

CHAP er en noe bedre metode. Den forsøker å sikre autentiseringsprosessen – som vist i figur 2.2.2.2. I stedet for å sende passordet ukryptert over linken, sender den bare sitt brukernavn til *NAS* som så genererer en *challenge* tilbake. Klienten krypterer denne med sitt passord og sender en *response* melding tilbake. *NAS* verifiserer denne responsen med *RADIUS*serveren. På denne måten sendes ikke passordet ukryptert over linken. Men det er dermed ikke sakt at den er trygg for angrep. Selv om *challenge* forandres for hver gang er den utsatt for såkalt ordbokangrep – der en angriper prøver alle kjente kombinasjoner fra diverse språk og lignende. En angriper som har lyttet til overføringen har nå plukket opp både den krypterte og ukrypterte teksten, og kan dermed utføre XOR operasjoner for å komme frem til krypteringsnøkkelen. Dette er senere rettet på i *Microsofts MS-CHAP*.

Viser til [1], kapittel 8 ”*Access Control: IEEE 802.1X, EAP, and RADIUS*” for opplysninger utover dette.



Figur 2.3.2.2: *CHAP prosedyren* [1] side 141

I *WPA2/RSN* trenger en å benytte seg av *RADIUS* på en sikrere måte enn *PAP* og *CHAP*. For å støtte *EAP* brukes *challenge-response* meldingsutvekslingen til å overføre *EAP-request* og *-response* meldinger.

En av de viktigste sikkerhetsforskjellene mellom vanlig oppringingsnettverk og *WiFi*-nettverk er at autentisering kun skjer i initialiseringsprosessen under oppringingsnettverk. Mens i *WPA2/RSN* nettverk må det skje en kontinuerlig autentiseringsprosess for å forhindre identitetskapringsangrep. Dette kan skje ved stjeling av en autorisert *MAC*-adresse. For å forhindre slike angrep må autentiseringsserveren overføre en hemmelig masternøkkel til aksesspunktet før dataoverføring. Denne nøkkelen benyttes i krypteringsfasen.

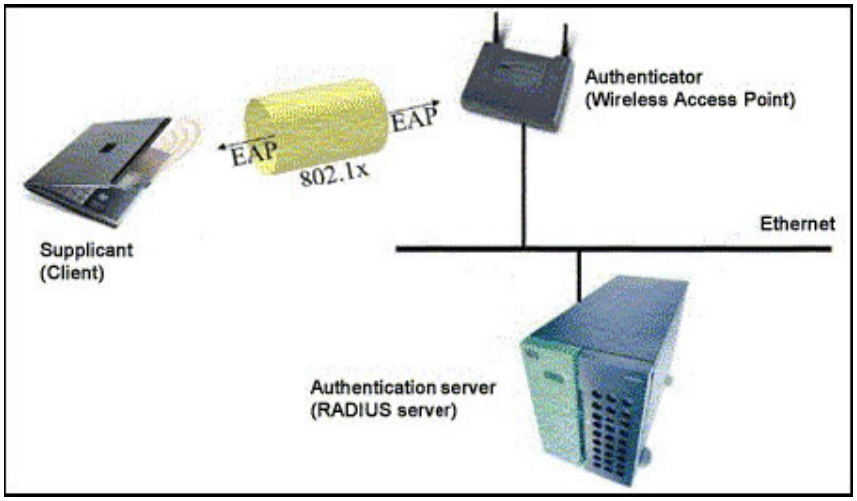
*RADIUS*serveren får også beskjed om når klienten logger seg på og av, for deretter å kunne regne ut kostnadene som klienten eventuelt skal belastes for. Denne funksjonen passer spesielt godt inn i *WLAN*-senarioer der en bruker må betale for å benytte seg av nettverket – som Trådløse Trondheim.
Serveren kan samtidig operere som en *DHCP*-server som deler ut *IP*-adresser til autoriserte brukere. I dette laboppsettet benyttes den ikke som en *DHCP*-server, da *NTNU*s server er mer enn tilfredsstillende til denne oppgaven.

Viser til [1], kapittel 8 ”*Access Control: IEEE 802.1X, EAP, and RADIUS*” samt [9], [10] og [12] for forklaring utover dette.

2.4 Extensible Authentication Protocol - EAP

EAP er en protokoll som benyttes i selve oppkobling og nedkoblingsfasen i *WLAN* og i punkt-til-punkt forbindelser. Oppsettet er vist i figur 2.4.1.
EAP var opprinnelig ikke tiltenkt *WLAN* – det var en oppringingsprotokoll. For å kapsle inne *EAP* meldinger til sending over *LAN*, ble protokollen *EAP Over LAN - EAPOL*, definert. Denne protokollen vil ikke bli diskutert nærmere i denne oppgaven. Viser til [1], kapittel 8 "*Access Control: IEEE 802.1X, EAP, and RADIUS*" for opplysninger utover dette.

Når *EAP* baserer seg på en *IEEE 802.1X* autentiseringsserver kan den tilby en sikker autentiseringsmekanisme og forhandle frem en sikker masternøkkel mellom klienten og serveren. Denne nøkkelen kan så brukes i en krypteringssesjon som benytter seg av *TKIP*- eller *AES*-kryptering.



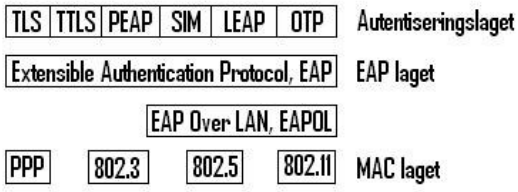
Figur 2.4.1: *Oppsett av Extensible Authentication Protocol, EAP* [2]

EAP alene er ikke tilstrekkelig for å tilby en sikker løsning. Den har følgende sikkerhetsproblemer:

- *EAP-Identity* meldingene er ikke beskyttet. Denne meldingen kan plukkes opp av en angriper – som dermed skaffer seg informasjon om identiteten til brukeren som forsøker å koble seg på

- *EAP-Success/Fail* meldingen er ikke beskyttet, og kan plukkes opp og utnyttes av en angriper

For å løse disse problemene må *EAP* kombineres med andre protokoller. En av disse vil bli tatt for seg i det påfølgende kapitlet. Det finnes mange andre *EAP* protokoller skreddersydd for ulike senarioer.



Figur 2.4.2: *IEEE 802.1X lagdeling*

Figur 2.4.2 viser lagdelingen i *IEEE 802.1X* satt i sammenheng med de forskjellige *EAP* protokollene.

Viser til [1], kapittel 8 ”*Access Control: IEEE 802.1X, EAP, and RADIUS*”, [3], kapittel 2.4 ”*Extensible Authentication Protocol - EAP*” og [11] for opplysninger om de øvrige *EAP* metodene.

2.4.1 EAP-Transport Layer Security – EAP-TLS

EAP-TLS er en populær autentiseringsprotokoll som tilbyr høy grad av sikkerhet. Den benytter seg av *TLS* – som blir omtalt som arvtageren til *SSL,* som basisen for autentisering. Dette er en sterk autentiseringsmetode som krever at både serveren og supplikanten benytter seg av digitale sertifikat.

Viser til [1], kapittel 9 "*Upper-Layer Authentication*" og [3], kapittel 2.3.3 " *Transport Layer Security – TLS*" for utdypende informasjon om *TLS*.

Protokollen baserer seg på *Public Key Infrastructure – PKI* for å sikre kommunikasjonen til autentiseringsserveren – vanligvis en *RADIUSserver*. Det største argumentet mot denne protokollen, men samtidig dens største styrke, er dens kompleksitet. Den krever bruk av et klientsertifikat i tillegg til et serversertifikat, samt oppsett av en *RADIUSserver.* Om nettverket består av mange brukere krever det en stor administrativ jobb å sette opp sertifikater til alle brukerne/gruppene i nettverket. Så det blir opp til hver enkel nettverksadministrator om det er verdt all jobbingen. Men om en velger å implementere denne protokollen har en et ekstremt sikkert system.

Et kompromittert passord er ikke alene nok for å bryte sikkerheten i denne protokollen. Angriperen må også skaffe seg tilgang til klientsertifikatet – uten å stjele selve mediet, for eksempel smartkortet, det ligger lagret på. Om et smartkort blir rapportert savnet er det en smal sak å revokere sertifikatet – og dermed utelukke det for videre misbruk. Et lite minus med denne protokollen er at etter autentiseringen blir brukernavn overført i klartekst. En liten, men bemerkelsesverdig, eksponering.

Det er egentlig ganske merkelig at denne protokollen ikke benyttes mer enn den gjør, siden den støttes av stort sett alle tilbydere av trådløst utstyr.
Selv om kravet til klientsertifikat er det som gjør *EAP-TLS* så sikkert som det er, illustrerer det problemstillingen mellom brukervennlighet og sikkerhet. Hvordan oppnå en fullgod sikkerhetsløsning uten for mye konfigurering av den enkelte klient.

Viser til [1], kapittel 8 "*Access Control: IEEE 802.1X, EAP, and RADIUS*" og [8] for utdypende informasjon.

2.5 Lab teori

Dette del-kapitlet vil ta for seg teorien for programmene som trengs for å kjøre laben.

2.5.1 FreeRadius

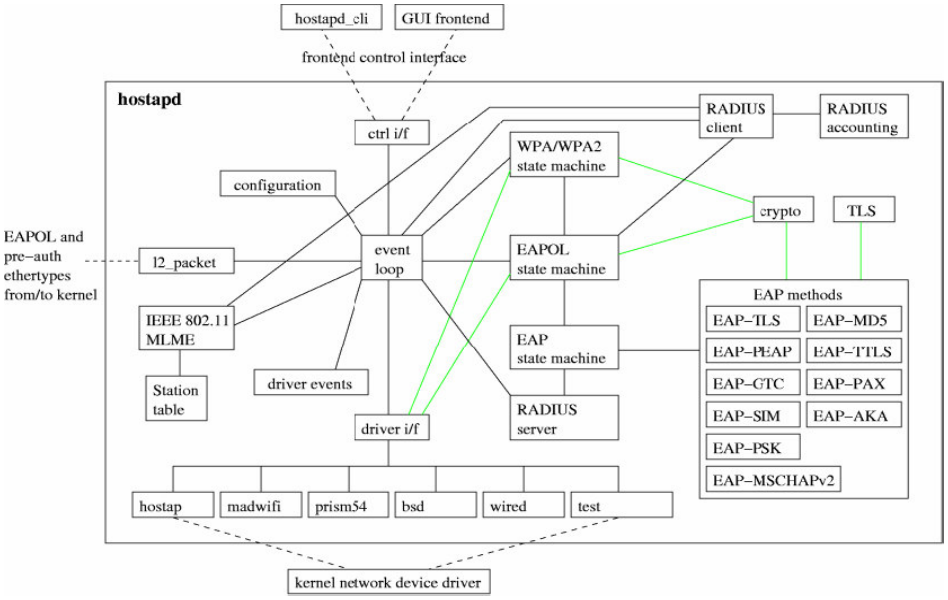
FreeRADIUS er en gratis ”open source” *RADIUS*server. Det vil si at brukere kan konfigurere programmet etter eget ønske.
Den er et alternativ til kommersielle *RADIUS*servere – siden den er gratis er den en av de mest populære. *FreeRADIUS* er inne på topp fem av de mest distrubierte *RADIUS* programmene og er ett av *RADIUS* programmene som autentiserer flest antall brukere hver dag.
FreeRADIUS skalerer veldig bra fra små til store systemer. Den er rask, fleksibel, konfigurerbar og støtter flere autentiseringsprotokoller enn mange kommersielle servere.

FreeRADIUS ble laget i 1999 av Alan DeKok og Miquel van Smoorenburg. Serveren ble raskt populær, mye på grunn av at den kunne integreres med moduler som *LDAP*, *SQL* og andre databaser. Støtte for *EAP* ble lagt til i 2001 – og komplettert med *PEAP* og *EAP-TTLS* i 2003. *FreeRADIUS* støtter nå alle kjente autentiseringsprotokoller og databaser.

For utdypende informasjon, se [13].

2.5.2 HostAPd

HostAPd er et serverprogram som implementerer *IEEE 802.11i* standarden. Det brukes i samarbeid med et trådløst nettverkskort med muligheten for å operere i mastermodus for å operere som et aksesspunkt med sikkert autentisering.
Programmet har også en innebygget *RADIUS*-autentiseringsserver funksjon. Denne funksjonen blir ikke benyttet i dette oppsettet. Her benyttes det en ekstern *RADIUS*server. Figur 2.5.2.1 viser arkitekturen til *HostAPd*.



Figur 2.5.2.1: *HostAPd arkitekturen* [2]

HostAPd kjøres, som *Wpa_supplicant*, som et *deamon* program. Det vil si et program som skal kjøres i bakgrunnen uten behov for mye bruker innblanding. Et grafisk grensesnitt kan også benyttes.

Den støtter følgende *WPA/IEEE 802.11i/EAP/IEEE 802.1X* egenskaper:

- *WPA-PSK* (*WPA*-Personal)
- *WPA* med EAP (*WPA*-Enterprise)
- Nøkkelhåndtering for *CCMP, TKIP, WEP104, WEP40*
- *WPA* og fullstendig *IEEE 802.11i/RSN/WPA2*
- *RSN*: *PMKSA* caching, preautentisering
- *RADIUS* accounting
- *RADIUS* autentiseringsserver med *EAP*

HostAPd støtter følgende *EAP* metoder:

- *EAP-TLS*
- *EAP-PEAP/MSCHAPv2*
- *EAP-PEAP/GTC*
- *EAP-PEAP/MD5-Challenge*
- *EAP-TTLS/EAP-MD5-Challenge*
- *EAP-TTLS/EAP-GTC*
- *EAP-TTLS/EAP-MSCHAPv2*
- *EAP-TTLS/MSCHAPv2*
- *EAP-TTLS/MSCHAP*
- *EAP-TTLS/PAP*
- *EAP-TTLS/CHAP*
- *EAP-SIM*
- *EAP-AKA*
- *EAP-PAX*
- *EAP-PSK*
- *EAP-SAKE*
- *EAP-GPSK*

Følgende metoder støttes også. Men siden de ikke genererer nøkkelmateriale kan de ikke brukes med *WPA* eller *IEEE 802.1X WEP*.

- *EAP-MD5-Challenge*
- *EAP-MSCHAPv2*
- *EAP-GTC*

HostAPd støtter følgende trådløskort/drivere:

- Host AP driver for Prism2/2.5/3
- MadWifi (Atheros ar521x)
- Prism54.org (Prism GT/Duette/Indigo)
- BSD net80211 layer (eks Atheros driver)(FreeBSD 6-CURRENT)

Se [14] for utdypende informasjon.

2.5.3 WPA Supplicant

WPA Supplicant er en supplikantprogramvare laget for *Linux, BSD* og *Microsoft Windows* som støtter *WPA* og *WPA2/RSN*. Den er kompatibel både med skrivebord/bærbare datamaskiner og embedded systemer.
WPA Supplicant komponenten installeres hos supplikanten i nettverket. Den iverksetter nøkkelforhandlinger med en autentikator samt kontrollerer roamingen og *IEEE 802.11* autentiseringen av *WLAN*-driveren.

WPA Supplicant er laget for å være et såkalt *"deamon"* program som kjøres i bakgrunnen og kontrollerer den trådløse forbindelsen uten brukeren skal merke så mye til den.

Den støtter følgende:

- *WPA-PSK*
- *WPA* med *EAP*
- Nøkkelhåndtering for *CCMP, TKIP, WEP104, WEP40*
- *WPA* og fullstendig *IEEE 802.11i/ WPA2/RSN*
- *WPA2/RSN: PMKSA* caching og preautentisering

Følgende *EAP* metoder støttes:

- *EAP-TLS*
- *EAP-PEAP/MSCHAPv2*
- *EAP-PEAP/TLS*
- *EAP-PEAP/GTC*
- *EAP-PEAP/OTP*
- *EAP-PEAP/MD5-Challenge*
- *EAP-TTLS/EAP-MD5-Challenge*
- *EAP-TTLS/EAP-GTC*
- *EAP-TTLS/EAP-OTP*
- *EAP-TTLS/EAP-MSCHAPv2*
- *EAP-TTLS/EAP-TLS*
- *EAP-TTLS/MSCHAPv2*
- *EAP-TTLS/MSCHAP*

- *EAP-TTLS/PAP*
- *EAP-TTLS/CHAP*
- *EAP-SIM*
- *EAP-AKA*
- *EAP-PSK*
- *EAP-FAST*
- *EAP-PAX*
- *EAP-SAKE*
- *EAP-GPSK*
- *LEAP*

Følgende metoder støttes også. Men siden de ikke genererer nøkkelmateriale kan de ikke brukes med *WPA* eller *IEEE 802.1X WEP*.

- *EAP-MD5-Challenge*
- *EAP-MSCHAPv2*
- *EAP-GTC*
- *EAP-OTP*

Se [15] for utdypende informasjon.

WPA_supplicant konfigureres via en tekstfil som inneholder opplysninger om hvilke(t) nettverk det skal assosieres med. Se kapittel 3.1.4.

3. FREMGANGSMÅTE

3.1 Lab fremgangsmåte

Dette kapitlet vil forklare hvordan en må gå frem for å få gjennomført den modifiserte versjonen av den aktuelle laben.

3.1.1 Generell informasjon

Første punkt på agendaen er å oppgradere *Linux Ubuntu* til den nyeste *Kernel* versjonen. Dette gjøres ved hjelp av de to påfølgende kommandoene.

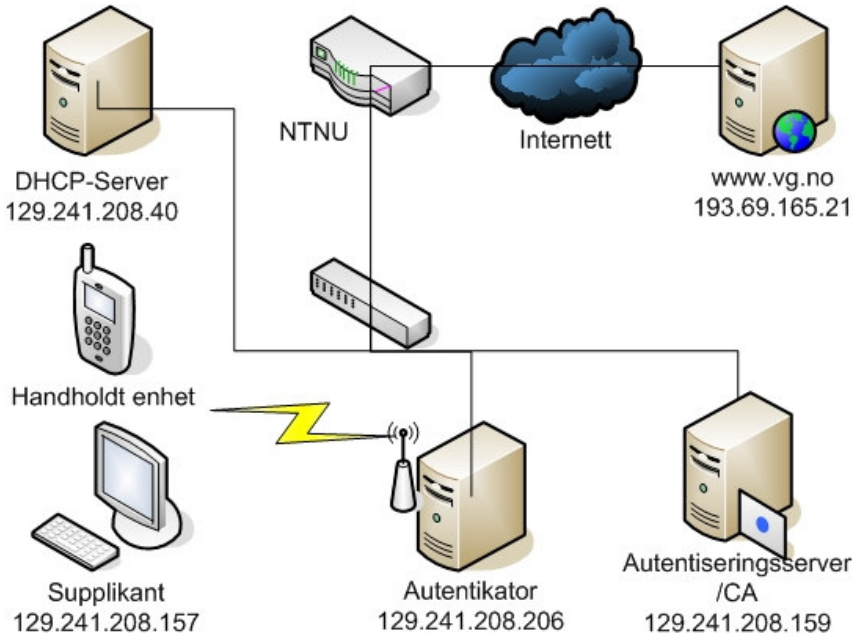
```
% apt-get install linux-generic
% apt-get install linux-headers-generic
```

Maskinen må nå startes på nytt. Pass på at rett versjon starter opp.

Deretter må nye drivere til nettverkskortet legges inn. Siden nettverkskortene som benyttes her er basert på *Atheros* chipset må det benyttes en driver med navn *MadWifi*. Driveren er tilgjengelig på [16].

Når dette er ferdig installert på alle tre maskinene er det på tide å spesifisere hvilke maskiner som skal være supplikant, autentikator og autentiseringsserver.

Oppsettet av laben er som vist i figur 3.1.1.1:



Figur 3.1.1.1: *Lab oppsettet*

Supplikant:

Supplikanten er en standard datamaskin, stasjonær eller bærbar, med et nettverkskort med trådløst grensesnitt. Maskinen kjører på *Linux Ubuntu* i dette oppsettet, men kan i grunn kjøre på hvilket som helst operativsystem. Denne masteroppgaven kan da i utgangspunktet ikke benyttes som oppskrift.

Supplikanten kan også være en handholdt enhet med *WLAN*-grensesnitt som støtter *IEEE 802.11X*, *IEEE 802.11i* og *EAP-TLS*.

Autentikator:

Autentikatoren er en standard datamaskin med to nettverkskort. Et med vanlig RJ45 plugg og et med trådløst grensesnitt. Maskinen kjører på *Linux Ubuntu* i dette oppsettet, men kan i grunn kjøre på hvilket som helst *Linux* operativsystem. Autentikatoren kan også være et vanlig aksesspunkt med *GUI* grensesnitt. Denne oppgaven kan da ikke brukes som utgangspunkt.

Fra det øyeblikket en supplikant initierer kontakt med aksesspunktet, tillater autentikatoren kun *EAP* trafikk. Kun etter autentiseringen er fullført gis supplikanten muligheten til å få en *IP*-adresse og får fullverdig tilgang til *WLAN*et.

Ved siden av å videreføre autentiseringen mellom supplikanten og serveren, har autentikatoren som oppgave å mate data fra autentiseringsserveren gjennom *TKIP* for å oppnå en *WEP* sesjonsnøkkel. Denne nøkkelen sendes så til supplikanten. Supplikanten blir med jevne mellomrom bedt om å re-autentisere seg. I denne prosessen blir *WEP*-nøkkelen erstattet av en ny en. Denne re-keying perioden konfigureres etter ønske i konfigurasjonsfilen hostapd.conf:

```
#Rekeying period in seconds. 0 = do not rekey (i.e., set keys only
once)
wep_rekey_period=300
```

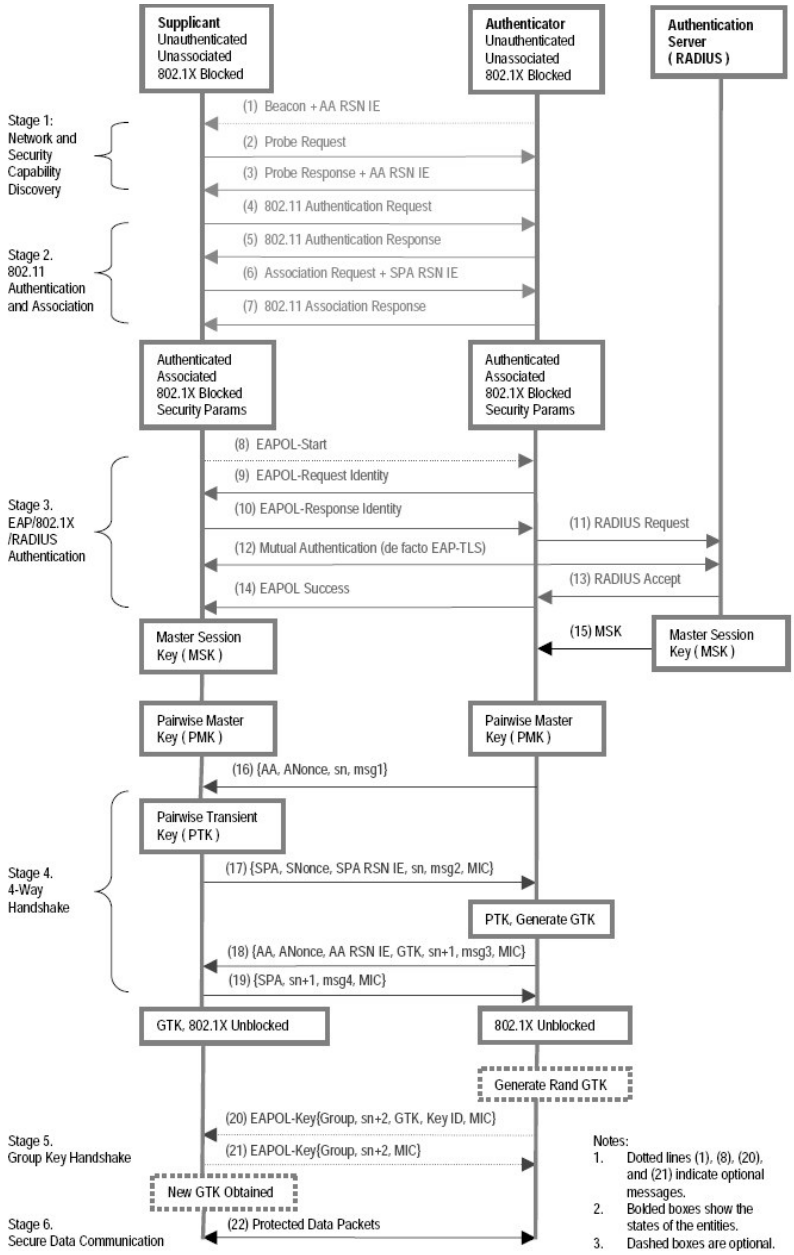
TKIP blir ikke benyttet i denne laben.

Autentiseringsserver/*CA*:
Autentiseringsserveren/*CA*en en standard datamaskin med ett nettverkskort med vanlig RJ45 plugg. Maskinen kjører på *Linux Ubuntu* i dette oppsettet, men kan i grunn kjøre på hvilket som helst *Linux* operativsystem.
Denne enheten kan byttes ut med et oppsett som kjører med en *Pre-shared key – PSK*-løsning (ikke gjort i denne oppgaven). I det oppsettet blir en delt nøkkel fordelt manuelt ut til alle supplikanter i nettverket. Denne løsningen er bedre enn en *WEP* løsning fordi denne delte nøkkelen ikke brukes som en krypteringsnøkkel, men mates inn i en *TKIP* prosess som igjen genererer dynamiske *WEP*-nøkler. Men dette en tungvind prosess.

DHCP-Server:
DHCP-Serveren er en server som benyttes av supplikanter i nettverket for å få tilegnet *IP*-adresser og andre parametere som default gateway, subnetmaske og *IP*-adresser til *DNS*-servere. Disse parametrene må settes manuelt om nettverket ikke innehar en slik tjeneste.

Det vil sendes en ping mot webserveren *www.vg.no* for å kontrollere at supplikanten har tilgang til Internett via nettverket til *NTNU*.

Gangen i *RSN* er delt inn i seks trinn, og skal være som vist i figur 3.1.1.2:



Figur 3.1.1.2: *RSN etableringsprosedyren* [21]

- Trinn 1 – Nettverks- og sikkerhetskapasitetsoppdagelse:
  Denne delen består av meldingene nummerert fra 1 til 3 på figur 3.1.1.2. Aksesspunktet vil enten broadcaste sine sikkerhetsspesifikasjoner gjennom *Beacon*rammen, eller svare på en stasjons *Probe Request*-ramme. En trådløs stasjon kan oppdage tilgjengelige aksesspunk gjennom å lytte på *Beacon*, eller ved å aktivt søke gjennom hver kanal.
- Trinn 2 – Autentisering og assosiering:
  Denne delen består av meldingene nummerert fra 4 til 7 på figur 3.1.1.2.

Stasjonen velger et aksesspunkt ut fra de tilgjengelige aksesspunktene den har søkt opp – og prøver å autentisere og assosiere seg til dette aksesspunktet. Stasjonen bør si fra om sin sikkerhetskapasitet under *Association Request* meldingen.

Etter dette trinnet er stasjonen og aksesspunktet i en autentisert tilstand. Men denne autentiseringen er svak, og vil bli supplemert videre av flere trinn. Ved slutten av dette trinnet er *802.1X*-porten fremdeles blokkert – ingen datapakker kan overføres.

- Trinn 3 – *EAP/802.1X/RADIUS* autentisering:
  Denne delen består av meldingene nummerert fra 8 til 14 på figur 3.1.1.2. Supplikanten og autentiseringsserveren kjører på en felles autentiseringsprotokoll (her *EAP-TLS*), med autentikatoren som et relé. Etter dette trinnet har supplikanten og autentiseringsserveren autentisert hverandre og generert en felles hemmelighet – *Master Session Key, MSK*. Supplikanten bruker denne *MSK*en til å generere en *Parwise Master Key, PMK*. Dette *AAA*-nøkkelmaterialet blir overført fra serveren til autentikatoren slik at den kan generer den samme *PMK* – vist av melding 15. Dette trinnet kan sløyfes dersom supplikanten og autentikatoren bruker en statisk *PSK* som *PMK*.
- Trinn 4 – Fire-veis handshake
  Denne delen består av meldingene nummerert fra 16 til 19 på figur 3.1.1.2. Uansett om *PMK*en er generert fra Trinn 3 – konfigurert ved hjelp av en *PSK*, eller brukt om igjen fra en lagret *PMK* – må den fire-veis handshaken gjennomføres for å oppnå en fullstendig *RSNA* etablering. Supplikanten og autentikatoren benytter seg av denne handshaken for å bekrefte tilstedeværelsen av *PMK*en, verifisere valget av krypteringsalgoritme, og genererer en ny *Pairwise Transient Key, PTK*, til de påfølgende datautvekslingene. Samtidig kan autentikatoren distribuere en *Group Transient Key, GTK*, i melding 18.
  Etter dette trinnet er en ny *PTK* (og *GTK*) delt mellom supplikanten og autentikatoren → *802.1X*-porten åpnes for datapakkeoverføring.
- Trinn 5 – *Group Key Handshake*
  Denne delen består av meldingene 20 og 21 på figur 3.1.1.2.
  I tilfelle multicast applikasjoner vil autentikatoren generere en ny *GTK*, og distribuere denne til supplikantene. Disse handshakene vil nødvendigvis ikke

bli utført om en ny *GTK* har blitt distribuert i Trinn 4; dette trinnet kan repeteres flere ganger med samme *PMK*.

- Trinn 6 – Sikker dataoverføring
  Denne delen består av melding 22 på figur 3.1.1.2.
  Ved hjelp av *PTK*en (eller *GTK*en) og den felles krypteringsalgoritmen fra tidligere, kan supplikanten og autentikatoren overføre datapakker.

Ved hjelp av disse handshakene har supplikanten og autentikatoren gjensidig autentisert hverandre – og etablert en sikker sesjon for dataoverføringer.

3.1.2 Sertifikathåndtering

Før en kan konfigurere *freeRADIUS* trengs det å genereres digitale sertifikater. Disse må signeres av en tiltrodd *CA*. Har en ikke en slik (som i dette laboppsettet) må en lage sin egen. En *CA* er et system som fungerer som en root i en *PKI* infrastruktur. Den er den sentraliserte overvåkeren som verifiserer, ved hjelp av en digital signatur, ektheten av alle sertifikater din organisasjon utsteder.

Før en kan lage en *CA* må en forsikre seg om at *OpenSSL* er installert. *OpenSSL* er en standardpakke som distribueres sammen med de fleste store *Linux* operativsystemene. En måte å sjekke om en har *OpenSSL* installert på, er å kjøre kommandoen:

```
% which openssl
```

/usr/bin/openssl skal være responsen her.
Eventuelt kan *OpenSSL* lastes ned fra [17].

Følgende felt må forandres på i openssl.conf. Lagringsplass av sertifikatene kan velges fritt. Filen åpnes med kommandoen

```
% sudo nano openssl.conf:
dir         = /etc/ssl/             # Where everything is kept
certs       = $dir/certs            # Where the issued certs are kept
crl_dir     = $dir/crl              # Where the issued crl are kept
database    = $dir/index.txt        # database index file.
new_certs_dir     = /etc/ssl        # default place for new certs.
certificate = /etc/ssl/cacert.pem   # The CA certificate
serial      = $dir/serial           # The current serial number
crlnumber   = $dir/crlnumber        # the current crl number
                                    # must be commented out to leave a
                                    # V1 CRL
crl         = $dir/crl.pem          # The current CRL
private_key = /etc/ssl/private/ca_private.pem # The private key
```

Før en kan begynne med sertifikat genereringen må en lage en database over de genererte sertifikatene kalt index.txt. Den filen lages ved hjelp av følgende kommando:

```
% echo '01' > serial
% touch index.txt
```

Innholdet i filen vises etter sertifikat genereringen.

Siden denne laben baserer seg på *EAP-TLS* er det behov for digitale sertifikater hos alle klienter, samt hos autentiseringsserveren. Disse genereres som vist i Appendiks A. Det første en må gjøre er å lage seg en *CA*. Dette gjøres ved å lage en privat nøkkel og et *self-signed* sertifikat ved hjelp av følgende kommandoer:

```
% openssl genrsa -des3 -out ./private/ca_private.pem 2048
% openssl req -new -x509 -key ./private/ca_private.pem -out
cacert.pem -days 1095
```

Nå er *CA*en laget og det er på tide å lage den private nøkkelen og sertifikatanmodningen til autentiseringsserveren. Det gjøres ved hjelp av følgende kommandoer:

```
% openssl genrsa -des3 -out ./private/as_private.pem 2048
% openssl req -new -key ./private/as_private.pem -out as.csr
```

Anmodningen as.csr sendes så til *CA* for signering. *CA* signerer sertifikatet ved hjelp av følgende kommando:

```
% openssl ca -out as.pem -infiles as.csr
```

CA har nå signert anmodningen as.csr og laget et digitalt sertifikat med navn as.pem.

Nå er tiden inne for å lage digitale sertifikat til supplikantene i nettverket. Først må hver supplikant generere sin egen private nøkkel ved hjelp av kommandoen:

```
% openssl genrsa -des3 -out ./private/supplicant_private.pem 2048
```

Deretter lages det en anmodning, som senere skal sendes til *CA,* ved hjelp av følgende kommando:

```
% openssl req -new -key ./private/supplicant_private.pem -out
supplicant.csr
```

Anmodningen supplicant.csr sendes så til *CA* for signering. *CA* signerer sertifikatet ved hjelp av følgende kommando:

```
% openssl ca -out supplicant.pem -infiles supplicant.csr
```

CA har nå signert anmodningen supplicant.csr og laget et digitalt sertifikat med navn supplicant.pem.

Nøkkelen supplicant_private.pem ble generert hos supplikanten og må ikke forlate denne maskinen. Den må bevares utilgjengelig for andre brukere.
Sertifikatene supplicant.pem og cacert.pem overføres på valgt måte fra *CA* til supplikanten. De kan også gjøres tilgjengelig fra utstederens *whitepage*. En *whitepage* er en lokal webside som supplikanten har tilgang til før autentiseringen har blitt gjennomført.
Hos autentiseringsserveren/*CA* skal ca_private.pem, cacert.pem, as_private.pem og as.pem ligge.
De private nøklene ca_private.pem og as_private.pem ble generert hos autentiseringsserveren/*CA* og må ikke forlate denne maskinen. Den må bevares utilgjengelig for andre brukere. Dette gjelder spesielt ca_private.pem. Blir denne kompromittert er ikke *CA*en lengre til å stole på, og alle signerte sertifikat bør revokeres.

Filen index.txt har nå følgende oversikt hvilke sertifikat *CA* har signert:

```
V      080229131520Z           01    unknown     /C=NO/ST=Sor-
       Trondelag/O=NTNU/OU=Diplom/CN=Supplicant/emailAddress=supplican
       t@diplom.no
V      080229134729Z           02    unknown     /C=NO/ST=Sor-
       Trondelag/O=NTNU/OU=Diplom/CN=AS/emailAddress=as@diplom.no
```

Skal nettverket bestå av supplikanter som kjører på *Microsoft Windows XP* må det gjøres diverse forrandringer ved laging av sertifikater. En må da lage en fil som heter xpextensions. Den lages ved hjelp av følgende kommando:

```
% sudo nano xpextensions
```

Denne filen bør ligge i samme directory som openssl.conf, og ha følgende innhold:

```
[ xpclient_ext] extendedKeyUsage = 1.3.6.1.5.5.7.3.2
[xpserver_ext] extendedKeyUsage = 1.3.6.1.5.5.7.3.1
```

Filen brukes som en del av en *OpenSSL* kommando for å generere sertifikater til *Microsoft Windows XP* supplikanter slik: `\ -extensions xpserver_ext -extfile ./xpextentions`
og tilsvarende for klienten: `\ -extensions xpclient_ext -extfile ./xpextentions.`

Sertifikatet må være i p12 format. Konverteringen gjøres på følgende måte:

```
% openssl pkcs12 -export -in supplicant.pem \ -inkey
./private/supplicant_private.pem -out supplicant.p12 -clcerts
```

En blir da bedt om å taste inn passordet for supplicant_private.pem, samt et nytt passord for det nye sertifikatet. Dette kan være det samme som det gamle. Filen overføres så til supplikanten.

For å installere sertifikatene må følgende punkter følges:

1. Kjør kommandoen *MMC* fra kjørlinjen.
2. I *Microsoft Management Console* velg *File → Add/Remove Snap-in*, legg til *Certificates snap-in* og sett den til å håndtere sertifikat for *My user account*, og på neste skjermbilde, kryss av bare for *Local Computer*.
3. Kopier *CA*-sertifikatet (cacert.pem) til en mappe på harddisken.
4. Fra *MMC* konsollen utvides *Consol Root* og *Certificates – Current User* og høyreklikk på *Trusted Root Certification Authorities*. I neste vindu velges så *All Tasks → Import.* Velg deretter cacert.pem for lagring i *Trusted Root Certification Authorities.*
5. Kopier ditt klientsertifikat og private nøkkel til harddisken.
6. Fra *MMC* konsollen velges *MMC→Console Root→Certificates.* Utvid *Personal* og høyreklikk på *Certificates*. I neste skjermbilde velges *All Tasks→Import*. Velg så det personlige sertifikatet og den private nøkkelen.
7. Denne *wizard*en ber deg deretter om å skrive inn passordet for sertifikatet. I samme vindu tilbyr den deg å krysse av for veldig sikker beskyttelse av den private nøkkelen. Dessverre bryter dette valget med *WPA*, så denne må stå ukrysset. La også valget om å gjøre nøkkelen eksportabel stå ukrysset. En er bedre tjent med å beskytte filen en nettopp importerte med passord enn å gjøre den importerte ubeskyttede versjonen eksportabel.

8. På neste skjermbilde krysses det av for at *wizard*en kan velge sertifikat lagring automatisk.

Nå er *Windows XP* systemet klart. Det eneste som gjenstår er å lage en trådløsnettverksprofil. Denne fremgangsmåten varierer fra trådløskort til trådløskort og til hvilken versjon av *XP* en kjører – så dette vil ikke gjennomgås her.

Etter denne konfigurasjonen vil *Windows* koble seg til nettverket og sette opp en *WPA* forbindelse automatisk.

Denne prosedyren har ikke blitt utført i denne laben. Men tas med som et alternativ dersom en vil kjøre laben med *Microsoft Windows XP* supplikanter. Viser til [5] og [6] for hele fremgangsmåten.

For utdypende opplysninger utover dette vises det til [7], kapittel 5 *"How to become a small-time CA"*.

3.1.3 Autentiseringsserver

Autentiseringsserveren må være koblet til nettverket ved en fast forbindelse med en twisted pair kabel.

I tillegg til den nevnte programvaren i kapittel 3.1.1 må autentiseringsserveren ha installert *freeRADIUS*. Fremgangsmåten følger her:

```
% tar zxvf freeradius-[version].tar.gz
% ./configure
% make
% su - root
% make install
```

Påfølgende må to konfigurasjonsfiler tilpasses. Disse er lokalisert i *freeRADIUS* folderen /usr/local/etc/raddb/.

Eap.conf:

Åpne filen med kommandoen

```
% sudo nano eap.conf
```

Modifiser påfølgende felt med de verdiene oppgitt her. Hele eap.conf filen er gjengitt i Appendiks D.

```
eap {
      default_eap_type = tls
      timer_expire = 600
      ignore_unknown_eap_types = yes
      md5 {
                }
      leap {
                }
      gtc {
          }
      tls {
                        private_key_password = serengeti
                        private_key_file =
                        /etc/ssl/private/as_private.pem
                        certificate_file = /etc/ssl/as.pem
                        CA_file = /etc/ssl/cacert.pem
```

```
                        dh_file = ${raddbdir}/certs/dh
                        random_file = ${raddbdir}/certs/random
                        fragment_size = 1024
                        include_length = yes
                }
        mschapv2 {
                }
          }
```

Clients.conf:

Åpne filen med kommandoen

```
% sudo nano clients.conf
```

Modifiser påfølgende felt med de verdiene oppgitt her. IP-adresserommet må selvsagt stemme overens med det subnettet klientene og autentikatoren befinner seg på. Her er det valgt ut adresserommet 129.241.208.0/8. Det vil si at alle klienter som har IP-adresse fra 129.241.208.1 til 129.241.208.254 kan koble seg til. Klienter vil si aksesspunkt. Supplikantene får sin IP-adresse fra *DHCP*-serveren.
Shortname og *secret* velges etter eget ønske. *Secret* spesifiserer en string som benyttes av autentikatoren som en krypteringsnøkkel for alle anmodningene den sender til autentiseringsserveren. *Shortname* er ganske enkelt et alias for autentikatoren brukt i logger og lignende.
Hele clients.conf filen er gjengitt i Appendiks D.

```
client 129.241.208.0/8
{
        secret = serengeti
        shortname = diplom
}
```

Ønskes strengere kontroll kan denne spesifiseres til kun å gjelde utvalgte klienter. Løkken må da settes til en enkelt IP-adresse. Ønskes flere utvalgte klienter tilgang må det lages flere tilsvarende løkker med rett IP-adresse.

Eksempel:

```
client 129.241.208.128
{
        secret = serengeti
```

```
        shortname = diplom
}
client 129.241.208.129
{
        secret = serengeti
        shortname = diplom
}
```

Radiusserveren er nå klar til å kjøres. Dette vises i kapittel 4.1.1.

3.1.4 Autentikator

Autentikatoren må være koblet til nettverket ved en fast forbindelse med en twisted pair kabel. Den må også ha et nettverkskort til med trådløst grensesnitt som kan settes i modus som aksesspunkt.
I tillegg til den nevnte programvaren i kapittel 3.1.1 må autentikatoren ha installert *HostAPd*. Fremgangsmåte følger her:

```
% apt-get install hostapd
```

Det trådløse nettverkskortet skal nå settes opp som et aksesspunkt. Det gjøres ved hjelp av følgende kommandoer:

```
% wlanconfig ath0 destroy
% wlanconfig ath0 create wlandev wifi0 wlanmode ap
% ifconfig ath0 up
```

Deretter settes opp *ESSID* og kanal for nettverket ved hjelp av følgende kommandoer:

```
% iwconfig ath0 essid Kenneth
% iwconfig ath0 channel 11
```

ESSID og kanal velges etter eget ønske. Kan lønne seg å kjøre en scan etter andre trådløsnett og velge kanal og *ESSID* som ikke er i bruk. Dette punktet kan ekskluderes grunnet at de samme innstillingene finnes i hostapd.conf filen. Kanal velges da tilfeldig.

For at autentikatoren skal fungere som et aksesspunkt er det nødvendig å rute trafikken fra det trådløse nettverkskortet over til nettverkskortet som er koblet til nettverket ved hjelp av en twisted pair kabel. Dette gjøres ved hjelp av en bro. Den settes opp ved hjelp av følgende kommandoer:

```
% brctl addbr br0
% brctl addif br0 eth1
% brctl addif br0 ath0
% ifconfig br0 up
```

Her er det viktig å sjekke på forhånd at grensesnittene virkelig heter eth1 og ath0.

For å sjekke statusen til broen, skriv inn følgende kommando:

```
%brctl show
```

Her skal du få opp informasjon om broen, samt hvilke grensesnitt som er koblet sammen i broen.

Til slutt må broen tilegnes en IP-adresse. Det gjøres ved følgende kommando:

```
% dhclient br0
```

Nå er aksesspunktet snart klart til oppstart. Men først må hostapd.conf konfigureres til å passe inn med resten av nettverket. Filen åpnes med kommandoen

```
% sudo nano hostapd.conf
```

Følgende felter må modifiseres. Pass også på at auth_server_port=1812 både i hostapd.conf og radiusd.conf som du finner på autentiseringsserveren i mappe /usr/local/etc/raddb/:

Hostapd.conf:

```
interface=ath0
bridge=br0
driver=madwifi
logger_syslog=-1
logger_syslog_level=0
logger_stdout=-1
logger_stdout_level=0
# Debugging: 0 = no, 1 = minimal, 2 = verbose, 3 = msg dumps, 4 =
excessive
debug=0
# Dump file for state information (on SIGUSR1)
dump_file=/tmp/hostapd.dump
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
ssid=Kenneth
macaddr_acl=2
auth_algs=1
# Require IEEE 802.1X authorization
ieee8021x=1
# The own IP address of the access point (used as NAS-IP-Address)
own_ip_addr=129.241.208.206
# RADIUS authentication server
auth_server_addr=129.241.208.159
auth_server_port=1812
auth_server_shared_secret=serengeti
# This field is a bit field that can be used to enable WPA (IEEE
802.11i/D3.0)
# and/or WPA2 (full IEEE 802.11i/RSN):
# bit0 = WPA
```

```
# bit1 = IEEE 802.11i/RSN (WPA2) (dot11RSNAEnabled)
wpa=2
# Set of accepted key management algorithms (WPA-PSK, WPA-EAP, or
both). The
# entries are separated with a space.
# (dot11RSNAConfigAuthenticationSuitesTable)
wpa_key_mgmt=WPA-EAP
wpa_pairwise=CCMP
```

Autentikatoren er nå klar til å kjøres. Dette vises i kapittel 4.1.2.

3.1.5 Supplikant

Supplikanten skal ikke være koblet til nettverket på noen måte, men må inneha et nettverkskort med trådløsgrensesnitt.
I tillegg til den nevnte programvaren i kapittel 3.1.1 må autentiseringsserveren ha installert *wpa supplicant*.
Den finnes på [18]. Denne pakkes ut og installeres på vanlig måte med

```
% make
% make install
```

Nå er supplikanten snart klart til oppstart. Men først må wpa_supplicant.conf konfigureres til å passe inn med resten av nettverket:

Wpa_supplicant.conf:
Åpne filen med kommandoen

```
% sudo nano wpa_supplicant.conf
```

Modifiser påfølgende felt med de verdiene oppgitt her. Hele wpa_supplicant.conf filen er gjengitt i Appendiks B. *SSID* kan så klart velges fritt etter ønske, men den må stemme overens med det som er spesifisert hos autentikatoren.

```
ctrl_interface=/var/run/wpa_supplicant
network=
{
      ssid="Kenneth"
      key_mgmt=WPA-EAP
      proto=WPA2
      pairwise=CCMP
      group=CCMP
      eap=TLS
      ca_cert="/etc/ssl/cacert.pem"
      client_cert="/etc/ssl/supplicant.pem"
      private_key="/etc/ssl/supplicant_private.pem"
      private_key_passwd="serengeti"
      identity="Supplicant"
}
```

Supplikanten er nå klar til å starte opp. Dette vises i kapittel 4.1.3.

4. RESULTATER

Dette kapitlet vil ta for seg hva som skjer når de forskjellige programmene kjører og kommuniserer seg imellom. Kapitlet vil også ta for seg en praktisk gjennomføring av løsningen på spørsmålet rundt handholdte enheter.

4.1 Kjøring av laben

I dette del-kapitlet vil det bli tatt for seg en grundig gjennomgang av hva som skjer når programmene *Wpa_supplicant*, *HostAPd* og *FreeRADIUS* kjøres.

4.1.1 Autentiseringsserver

FreeRadius startes med følgende kommando:

```
% radiusd -x
```

Det er viktig at serveren startes først – ellers får en ikke opp systemet. Etter startingen av serveren får en da frem følgende utskrift som sier hvilke moduler som lastes, hvilke porter den lytter til, samt at serveren er klar til å ta imot autentiseringsanmodninger:

```
Starting - reading configuration files ...
Using deprecated naslist file.  Support for this will go away soon.
Module: Loaded exec
rlm_exec: Wait=yes but no output defined. Did you mean output=none?
Module: Instantiated exec (exec)
Module: Loaded expr
Module: Instantiated expr (expr)
Module: Loaded PAP
Module: Instantiated pap (pap)
Module: Loaded CHAP
Module: Instantiated chap (chap)
Module: Loaded MS-CHAP
Module: Instantiated mschap (mschap)
Module: Loaded System
Module: Instantiated unix (unix)
Module: Loaded eap
rlm_eap: Loaded and initialized type md5
rlm_eap: Loaded and initialized type leap
```

```
rlm_eap: Loaded and initialized type gtc
rlm_eap_tls: Loading the certificate file as a chain
rlm_eap: Loaded and initialized type tls
rlm_eap: Loaded and initialized type mschapv2
Module: Instantiated eap (eap)
Module: Loaded preprocess
Module: Instantiated preprocess (preprocess)
Module: Loaded realm
Module: Instantiated realm (suffix)
Module: Loaded files
Module: Instantiated files (files)
Module: Loaded Acct-Unique-Session-Id
Module: Instantiated acct_unique (acct_unique)
Module: Loaded detail
Module: Instantiated detail (detail)
Module: Loaded radutmp
Module: Instantiated radutmp (radutmp)
Initializing the thread pool...
Listening on authentication *:1812
Listening on accounting *:1813
Ready to process requests.
```

Når supplikanten og autentikatoren har gjort sitt, kommer frem følgende skjermbilde hos autentiseringsserveren (utdrag):

Autentiseringsserveren mottar en ny anmodning fra en supplikant med id ”*Supplicant*”, via autentikatoren, for å bli autentisert i nettverket ”*Kenneth*”:

```
rad_recv: Access-Request packet from host 129.241.208.206:1148, id=0,
length=159
        User-Name = "Supplicant"
        NAS-IP-Address = 129.241.208.206
        NAS-Port = 0
        Called-Station-Id = "00-0F-CB-B5-0E-D2:Kenneth"
        Calling-Station-Id = "00-0F-CB-B3-67-11"
        Framed-MTU = 1400
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 0Mbps 802.11"
        EAP-Message = 0x0201000f01537570706c6963616e74
        Message-Authenticator = 0xd77384e38e24aefd9badd2516231f0d8
```

Autentiseringsserveren starter autentiseringen, via autentikatoren, ved hjelp av en rekke spørringer:

```
Sending Access-Challenge of id 0 to 129.241.208.206 port 1148
        EAP-Message = 0x010200060d20
        Message-Authenticator = 0x00000000000000000000000000000000
        State = 0x14b532961d329bd28a711c362ea8b3cc
```

Autentiseringsserveren og supplikanten utveksler sine digitale sertifikater:

```
--> User-Name = Supplicant
--> BUF-Name = CA
--> subject = /C=NO/ST=Sor-
Trondelag/L=Trondheim/O=NTNU/OU=Diplom/CN=CA/emailAddress=ca@diplom.n
o
--> issuer  = /C=NO/ST=Sor-
Trondelag/L=Trondheim/O=NTNU/OU=Diplom/CN=CA/emailAddress=ca@diplom.n
o
--> verify return:1
chain-depth=0,
error=0
--> User-Name = Supplicant
--> BUF-Name = Supplicant
--> subject = /C=NO/ST=Sor-
Trondelag/O=NTNU/OU=Diplom/CN=Supplicant/emailAddress=supplicant@dipl
om.no
--> issuer  = /C=NO/ST=Sor-
Trondelag/L=Trondheim/O=NTNU/OU=Diplom/CN=CA/emailAddress=ca@diplom.n
o
--> verify return:1
```

Autentiseringsserveren har nå autentisert supplikanten, og den kan nå utveksle datapakker på nettverket.

4.1.2 Autentikator

HostAPd må startes etter *FreeRADIUS*, og før *Wpa_supplicant*. Den startes ved hjelp av følgende kommando:

```
% hostapd -d /etc/hostapd/hostapd.conf
```

En får da frem følgende utskrift som sier at autentikatoren er klar til å ta imot autentiseringsanmodninger:

```
Configuration file: /etc/hostapd/hostapd.conf
ctrl_interface_group=0
Configure bridge br0 for EAPOL traffic.
Using interface ath0 with hwaddr 00:0f:cb:b5:0e:d2 and ssid 'Kenneth'
ath0: RADIUS Authentication server 129.241.208.159:1812
madwifi_configure_wpa: group key cipher=3
madwifi_configure_wpa: pairwise key ciphers=0x8
madwifi_configure_wpa: key management algorithms=0x1
madwifi_configure_wpa: rsn capabilities=0x0
madwifi_configure_wpa: enable WPA= 0x2
WPA: group state machine entering state GTK_INIT
GMK - hexdump(len=32): [REMOVED]
GTK - hexdump(len=16): [REMOVED]
WPA: group state machine entering state SETKEYSDONE
madwifi_set_key: alg=CCMP addr=00:00:00:00:00:00 key_idx=1
madwifi_set_privacy: enabled=1
SIOCGIWRANGE: WE(compiled)=20 WE(source)=13 enc_capa=0xf
ath0: IEEE 802.11 Fetching hardware channel/rate support not
supported.
Flushing old station entries
madwifi_sta_deauth: addr=ff:ff:ff:ff:ff:ff reason_code=3
Deauthenticate all stations
l2_packet_receive - recvfrom: Network is down
```

Nå har en fått vite diverse opplysninger om autentikatoren, blant annet *SSID* og hardwareadressen til nettverksinterfacet. Dette er utgangspunktet for autentikatoren – den er nå klar til å ta imot anmodninger fra supplikanter.

Videre følger et utdrag av kjørelogen når en supplikant ønsker å autentiseres med nettverket. Hele kjørelogen er gjengitt i Appendiks C.

En ny supplikant ønsker å autentiseres:

```
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.11: associated
  New STA
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: start authentication
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state INITIALIZE
```

Nettverket ønsker identiteten til supplikanten:

```
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: received EAP packet (code=2
id=1 len=15) from STA: EAP Response-Identity (1)
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: STA identity 'Supplicant'
```

Supplikanten har nå sendt over sin identitet – ”*Supplicant*” til autentikatoren.

Autentiseringsserveren sier fra til supplikanten hvilken *EAP* metode som benyttes (*EAP-TLS*):

```
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: decapsulated EAP packet
(code=1 id=2 len=6) from RADIUS server: EAP-Request-TLS (13)
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: decapsulated EAP packet
(code=3 id=7 len=4) from RADIUS server: EAP Success
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state SUCCESS
```

Her fullføres 4-way Handshaken i *RSN*:

```
ath0: STA 00:0f:cb:b3:67:11 WPA: sending 1/4 msg of 4-Way Handshake
ath0: STA 00:0f:cb:b3:67:11 WPA: received EAPOL-Key frame (2/4
Pairwise)
ath0: STA 00:0f:cb:b3:67:11 WPA: sending 3/4 msg of 4-Way Handshake
ath0: STA 00:0f:cb:b3:67:11 WPA: received EAPOL-Key frame (4/4
Pairwise)
ath0: STA 00:0f:cb:b3:67:11 WPA: pairwise key handshake completed
(RSN)
```

Avslutning av autentiseringen:

```
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state AUTHENTICATED
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: authorizing port
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: authenticated
```

Supplikanten er nå autentisert med nettverket og kan utveksle datapakker.

4.1.3 Supplikant

Wpa_supplicant må kjøres sist, og startes ved hjelp av én av følgende to kommandoer:

```
% wpa_supplicant -D madwifi -i ath1 -c /var/run/wpa_supplicant.conf
% wpa_supplicant -dd -K -t -i ath1 -c /var/run/wpa_supplicant.conf
```

I wpa_supplicant.conf filen står det, som nevnt tidligere, hvilke kriterier supplikanten skal velge nettverk etter. Et utdrag av kjørelogen vises her:

Supplikanten søker etter tilgjengelige aksesspunkt. Resultat = 24 stykker. Velger aksesspunkt etter prioriterte nettverk i wpa_supplicant.conf filen (”*Kenneth*”):

```
1176713230.885718: Received 4095 bytes of scan results (24 BSSes)
1176713230.885727: Scan results: 24
1176713230.885734: Selecting BSS from priority group 0
1176713230.885741: 0: 00:0f:cb:b5:0e:d2 ssid='Kenneth' wpa_ie_len=0
rsn_ie_len=22 caps=0x11
1176713230.885752:    selected based on RSN IE
```

Supplikanten mottar en anmodning om å sende sin identitet til autentiseringsservern:

```
1176713253.735206: EAP: Received EAP-Request id=1 method=1 vendor=0
vendorMethod=0
1176713253.735213: EAP: EAP entering state IDENTITY
1176713253.735222: CTRL-EVENT-EAP-STARTED EAP authentication started
1176713253.735229: EAP: EAP-Request Identity data -
hexdump_ascii(len=0):
1176713253.735235: EAP: using real identity - hexdump_ascii(len=10):
     53 75 70 70 6c 69 63 61 6e 74                     Supplicant
```

Supplikanten har nå sendt sin identitet ”*Supplicant*” til autentiseringsserveren via autentikatoren.

Supplikanten mottar en anmodning om hvilken *EAP* metode som skal benyttes (*TLS*). Som en konsekvens av dette må supplikanten laste sitt personlige sertifikat ved hjelp av sin private nøkkel:

```
1176713253.825558: EAP: EAP entering state GET_METHOD
1176713253.825565: EAP: Initialize selected EAP method: vendor 0
method 13 (TLS)
1176713253.959377: TLS: Trusted root certificate(s) loaded
1176713254.041856: OpenSSL: SSL_use_certificate_file (PEM) --> OK
```

```
1176713254.068300: OpenSSL: tls_connection_private_key -
SSL_use_PrivateKey_File (DER)
1176713254.099484: OpenSSL: SSL_use_PrivateKey_File (PEM) --> OK
1176713254.099516: SSL: Private key loaded successfully
1176713254.099531: CTRL-EVENT-EAP-METHOD EAP vendor 0 method 13 (TLS)
selected
```

Supplikanten laster inn *CA*en og autentiseringsserverens (samme maskin i dette oppsettet) digitale sertifikat:

```
1176713254.189822: TLS: tls_verify_cb - preverify_ok=1 err=0 (ok)
depth=1 buf='/C=NO/ST=Sor-
Trondelag/L=Trondheim/O=NTNU/OU=Diplom/CN=CA/emailAddress=ca@diplom.n
o'
1176713254.190368: TLS: tls_verify_cb - preverify_ok=1 err=0 (ok)
depth=0 buf='/C=NO/ST=Sor-
Trondelag/O=NTNU/OU=Diplom/CN=AS/emailAddress=as@diplom.no'
1176713254.190405: SSL: SSL_connect:SSLv3 read server certificate A
1176713254.190466: SSL: SSL_connect:SSLv3 read server certificate
request A
1176713254.190480: SSL: SSL_connect:SSLv3 read server done A
1176713254.190593: SSL: SSL_connect:SSLv3 write client certificate A
1176713254.190761: SSL: SSL_connect:SSLv3 write client key exchange A
1176713254.225361: SSL: SSL_connect:SSLv3 write certificate verify A
1176713254.275122: SSL: SSL_connect:SSLv3 write change cipher spec A
1176713254.275316: SSL: SSL_connect:SSLv3 write finished A
1176713254.275404: SSL: SSL_connect:SSLv3 flush data
1176713254.275485: SSL: SSL_connect:error in SSLv3 read finished A
```

Supplikanten og autentiseringsserveren gjennomfører den fire-veis handshaken i *RSN*, installerer *PTK* og setter porten i tilstand åpen, for deretter å gå inn i tilstanden *Autentisert.* Nøkkelhierakiet er vist i figur 2.2.4.1:

```
1176713254.357727: State: ASSOCIATED -> 4WAY_HANDSHAKE
1176713254.357734: WPA: RX message 1 of 4-Way Handshake from
00:0f:cb:b5:0e:d2 (ver=2)
1176713254.361924: WPA: RX message 3 of 4-Way Handshake from
00:0f:cb:b5:0e:d2 (ver=2)
1176713254.362132: WPA: Installing PTK to the driver.
1176713254.362186: wpa_driver_madwifi_set_key: alg=CCMP key_idx=0
set_tx=1 seq_len=6 key_len=16
1176713254.395179: EAPOL: External notification - portValid=1
```

```
1176713254.395269: EAPOL: SUPP_PAE entering state AUTHENTICATED
```

Supplikanten og autentiseringsserveren gjennomfører gruppe handshake og supplikanten installerer *GTK*. Nøkkelhierakiet er vist i figur 2.2.4.1:

```
.395320: State: 4WAY_HANDSHAKE -> GROUP_HANDSHAKE
1176713254.395360: RSN: received GTK in pairwise handshake -
hexdump(len=18): 01 00 c6 76 28 e9 fe 69 a5 89 99 35 f7 f8 d1 dc fa
a1
1176713254.395406: WPA: Group Key - hexdump(len=16): c6 76 28 e9 fe
69 a5 89 99 35 f7 f8 d1 dc fa a1
1176713254.395449: WPA: Installing GTK to the driver (keyidx=1 tx=0).
1176713254.395488: WPA: RSC - hexdump(len=6): 30 00 00 00 00 00
1176713254.395530: wpa_driver_madwifi_set_key: alg=CCMP key_idx=1
set_tx=0 seq_len=6 key_len=16
1176713254.395592: WPA: Key negotiation completed with
00:0f:cb:b5:0e:d2 [PTK=CCMP GTK=CCMP]
1176713254.395634: Cancelling authentication timeout
1176713254.395674: State: GROUP_HANDSHAKE -> COMPLETED
1176713254.395717: CTRL-EVENT-CONNECTED - Connection to
00:0f:cb:b5:0e:d2 completed (auth) [id=0 id_str=]
```

Supplikanten er nå autentisert med nettverket og kan gå inn i avslutningen av prosessen.

Neste steg på agendaen er å tilegne supplikanten en *IP*-adresse fra *DHCP*-serveren. Dette oppnås ved å kjøre følgende kommando:

```
% dhclient ath1

Internet Systems Consortium DHCP Client V3.0.4
Copyright 2004-2006 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/sw/dhcp/

wifi0: unknown hardware address type 801
wifi0: unknown hardware address type 801
Listening on LPF/ath1/00:0f:cb:b3:67:11
Sending on   LPF/ath1/00:0f:cb:b3:67:11
Sending on   Socket/fallback
DHCPDISCOVER on ath1 to 255.255.255.255 port 67 interval 7
```

```
DHCPDISCOVER on ath1 to 255.255.255.255 port 67 interval 7
DHCPOFFER from 129.241.208.40
DHCPREQUEST on ath1 to 255.255.255.255 port 67
DHCPACK from 129.241.208.40
bound to 129.241.208.157 -- renewal in 115823 seconds.
```

Supplikanten har nå mottatt en *IP*-adresse fra *DHCP*-serveren (129.241.208.40), som er gyldig i 115823 sekunder før adressen oppdateres. Dette blir som regel alltid samme adresse.

For å verifisere at supplikanten har kontakt med Internett kjøres det en ping mot en tilfeldig valgt nettadresse. Her www.vg.no.

```
% ping www.vg.no
PING www.vg.no (193.69.165.21) 56(84) bytes of data.
64 bytes from 193.69.165.21: icmp_seq=1 ttl=246 time=9.07 ms
64 bytes from 193.69.165.21: icmp_seq=2 ttl=246 time=9.71 ms
64 bytes from 193.69.165.21: icmp_seq=3 ttl=246 time=12.4 ms
64 bytes from 193.69.165.21: icmp_seq=4 ttl=246 time=9.29 ms
64 bytes from 193.69.165.21: icmp_seq=5 ttl=246 time=10.5 ms
64 bytes from 193.69.165.21: icmp_seq=6 ttl=246 time=9.83 ms
```

Supplikanten har kontakt med www.vg.no, og kan dermed utveksle datapakker.

For å oppsummere har supplikanten foretatt følgende operasjoner:

- sender en anmodning til kerneldriveren om å søke etter aksesspunkt i nærheten
- velger et aksesspunkt ut fra opplysningene lagt inn i konfigurasjonsfilen
- anmodner kerneldriveren om å assosiere med det valgte aksesspunket
- Fullfører *EAP* autentiseringen med autentiseringsserveren (via autentikatoren)
- Mottar masternøkkelen
- Fullfører *WPA* fire-veis-handshake og gruppenøkkel-handshake med autentikatoren. *WPA2* har integrert gruppenøkkel-handshaken i den fire-veis-handshaken
- Konfigurerer krypteringsnøkler for uni- og broadcast
- Vanlige datapakker kan sendes og mottas

4.2 Praktisk gjennomføring med handholdte enheter

Dette del-kapitlet vil ta for seg hvordan en kan importere digitale sertifikater over på en handholdt enhet. Dette har vist seg å være et problem for mange.

De fleste handholdte enhetene er som regel låst, slik at en kun kan installere sertifikater gjennom en installasjonsfil (.cab, .exe). Disse sertifikatene må være signert av en av de store *CA*ene.
Produsenten av enheten har forhåndsinstallert diverse sertifikater til bruk i de største nettverkene. Nettverkstilbydere har muligheten til å bli medlem i disse nettverkene. På den måten er sertifikatene allerede distribuert, og klienten kan koble seg på. Trådløse Trondheim driver og arbeider med en slik løsning. De har lansert *SSID*en ”eduroamtest” som brukes sammen med sertifikatet "GTE Cyber Trust Global Root". Dette sertifikatet er installert på de fleste Pocket PC-ene – blant annet Qtek9000, som ble brukt i denne oppgaven.

En mulighet for å omgå dette, og installere sertifikater som er signert av din egen, eller mindre, *CA*(er) er å lage eller installere et eget installasjonsprogram. Et slikt program er *P12import* – et høyt anerkjent *open source* program utviklet av Jacco de Leeuw. Det kan lastes ned fra [19].

Problemet med web-innmelding er at en *Windows Mobile* enhet må anskaffe sertifikatene fra en webserver. Kun *Windows 2000/2003 Server* er støttet, og webserveren må være *Internet Information Services* og *CA*en må være *Microsoft Certificate Services.* Vil en da benytte seg av en *CA* eller *VPN*-server som ikke er basert på *Windows*, må en da benytte seg av egne installasjonsprogrammer eller finne et som passer sine behov. *P12import* er et slikt program.

Et alternativ til sertifikatinnmelding er import av sertifikat. PKCS#12 er standardformatet for lagring av private nøkler og sertifikat. Formatet er støttet av mange tilbydere – inkludert *Microsoft.* De fleste *VPN*-klientene støtter PKCS#12. Import av PKCS#12 filer støttes kun i *Microsofts Windows Mobile 6* og ikke på *Pocket PC 2003* og *Windows Mobile 5.0.* Dermed trenger en et program egnet for import av sertifikater – som *P12Import*.

Videre følger en oppskrift på hvordan en installerer sertifikater på en handholdt enhet ved hjelp av *P12import*:

- Kopier p12imprt.exe til *Windows Mobile* enheten. En kan benytte seg av hvilken som helst metode for overføring: ActiveSync, et flash-minnekort, gjennom *LAN*, Blåtann, *WLAN*, infrarød og lignende. (Merk at p12imprt.exe kan kun åpnes på en *Windows Mobile* enhet og ikke på en vanlig *Microsoft Windows* maskin. Den er ikke en Win32 fil).
- Kopier sertifikatet, som må være i PKCS#12 format, til enheten.
- Kjør p12imprt.exe ved å dobbel-klikke på filen i *File Manager.*
- Skriv inn lokasjonen til sertifikatet, eller benytt deg av *Browse* knappen.
- Skriv inn passordet som ble brukt i krypteringen av sertifikatet.
- En skal nå se en rapport som viser hvilke sertifikater som er importert. Lukk P12import ved å trykke *Ok.*

For å se sertifikatene som har blitt importert til enheten, gjør som følger:

- Fra *Settings* menyen, klikk på *System→Certificates.*
- Kryss av for *Personal*. En vil nå se alle sertifikat som nylig er blitt importert.
- Ved å klikke på ønsket sertifikat får en opp detaljene til sertifikatet.
- Kryss av for *Root.* En vil nå se alle rot-sertifikat som er importert.
- Ved å klikke på ønsket sertifikat får en opp detaljene til sertifikatet.

For sletting av sertifikat, gjøres følgende:

- *Settings→System→Certificate applet.*
- Klikk og hold på ønsket sertifikat. En får da opp en meny, der ett av valgene er *Delete*. En må slette både root-sertifikatet og det personlige sertifikatet.

Fordeler med *P12imprt:*

- En kan bruke sertifikat fra hvilken som helst *CA,* ikke bare *Microsoft Windows CA Certificate Services.*
- Mange tredjeparts *CA*er, slik som *Thawte* og *Verisign,* støtter ikke web-innmeldingen under *Windows Mobile*. Der må en importere en PKCS#12 fil.
- Private nøkler kan genereres på hvilken som helst datamaskin, ikke bare på selve *Windows Mobile* enheten.
- Kan kjøres på alle *Pocket PC 2003* og *Windows Mobile* enheter (så lenge enheten ikke er låst).
- Det er gratis og tilgjengelig for nedlasting.

Ulemper med *P12imprt*:

- *GUI*en er simpel.
- Programmet er ganske stort (nesten 700 kB) sammenlignet med *Crtimprt, PFXimprt* og web-innmelding.
- På grunn av kodingen er det ikke mulig å importere et sertifikat til et smartkort inne i enheten. Det er heller ikke mulig å bruke andre sertifikat enn de som er basert på *RSA*. Støtte for dette kan komme senere (eksempelvis for *DSA/DSS*).

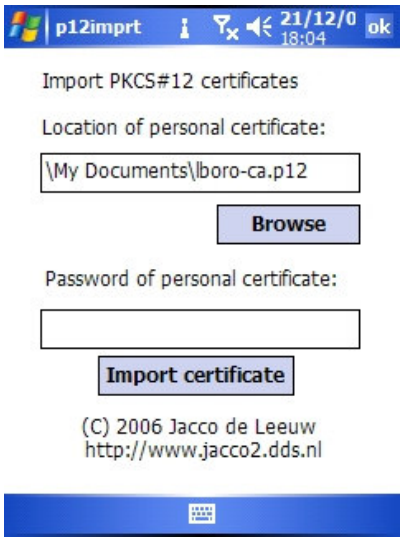
For utdypende informasjon, se [20].

Forsøk på å koble seg til laboppsettet i denne oppgaven med en Qtek9000 mislyktes grunnet enheten manglet støtte for *RSN* og *EAP-TLS*. Men siden enheten støtter *TKIP*, *PEAP* og *IEEE 802.1X,* kan laboppsettet modifiseres til å kjøre over *TKIP* og *PEAP*. Dette gir ikke gjensidig autentisering – noe som er et krav i problemstillingen, og ble dermed ikke gjennomført i denne oppgaven.

For å koble til en handholdt enhet opp mot Trådløse Trondheim ved hjelp av *IEEE 802.1X* må en få utstedt et sertifikat fra Trådløse Trondheim. I oppgaveforfatterens forsøk med en Qtek9000 ble det utstedt et .der sertifikat som ble omgjort til et .cer sertifikat. Ved å installere dette og velge det fra profilen får en koblet til. Formatet på sertifikatet varierer fra enhet til enhet. Det er en forutsetning at brukeren befinner seg innenfor dekningsområdet til Trådløse Trondheim. En lignende gjennomgang er hentet

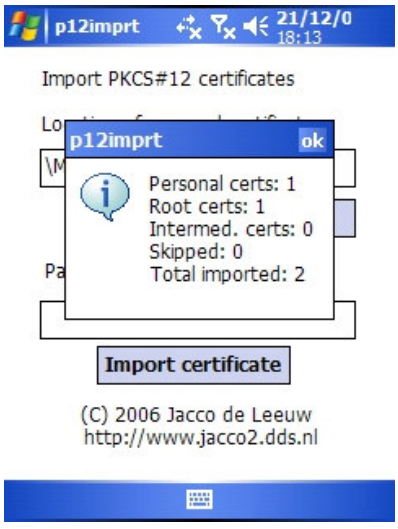
fra [22] og følger her (diverse navn må byttes ut med de korrekte fra Trådløse Trondheim):

1. For å bruke nettverket må det lastes ned (få distribuert) et digitalt sertifikat fra Trådløse Trondheim. Dette kopieres over til enheten.
2. P12import må lastes ned fra [20]. Dette programmet brukes bare i starten av installasjonen, og kan dermed slettes etterpå.
3. Kjør P12Import fra *Start*→File *Manager*. En vil nå få beskjed om å finne frem til sertifikatet – som vist i figur 4.2.1.



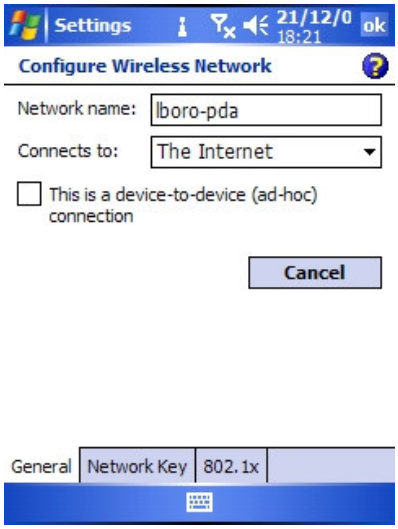
Figur 4.2.1: *P12Import 1 [22]*

4. Trykk så på *Import certificate*. Det skal nå komme frem et vindu, som vist i figur 4.2.2, som sier at sertifikatet er importert korrekt.



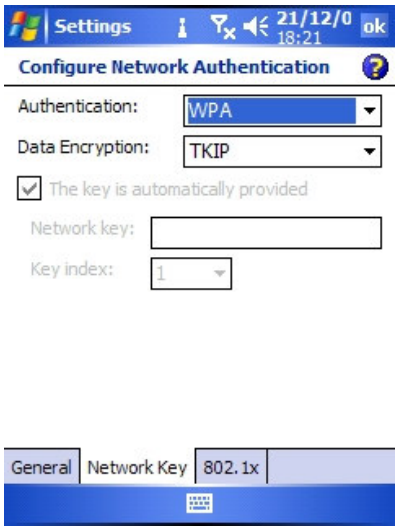
Figur 4.2.2: *P12Import 2 [22]*

5. Med WLAN påskrudd går en til *Start→Settings→Network cards*. En skal nå se en liste over tilgjengelige nettverk.
6. Trykk på *Add New* og konfigurer et nytt *WLAN* med navn ”*ntnu1x*”. Sett ”*connects to”* til Internet, som vist i figur 4.2.3.



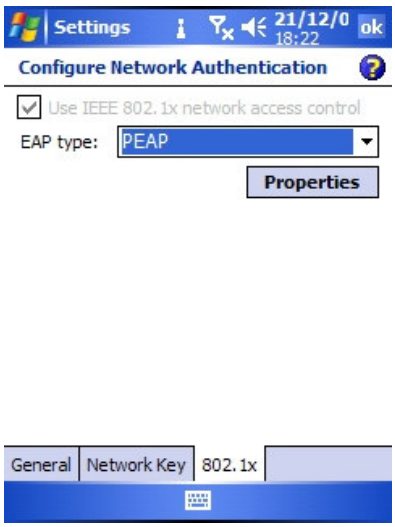
Figur 4.2.3: *P12Import 3 [22]*

7. Gå til fanen ”*Network Key*” og skift verdien på ”*Authentication*” til ”*WPA*” og ”*Data Encryption*” til ”*TKIP*”, som vist i figur 4.2.4.



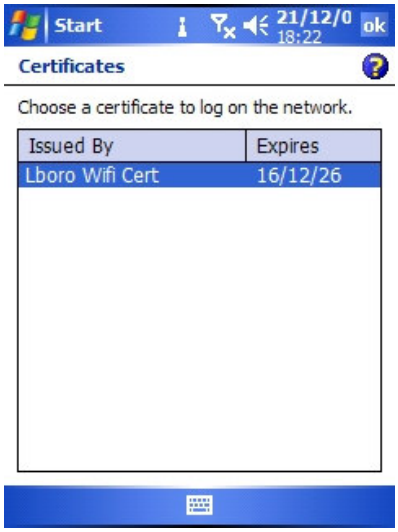
Figur 4.2.4: *P12Import 4 [22]*

8. Gå til fanen "*802.1X*" og skift "*EAP type*" til "*PEAP*", som vist i figur 4.2.5.



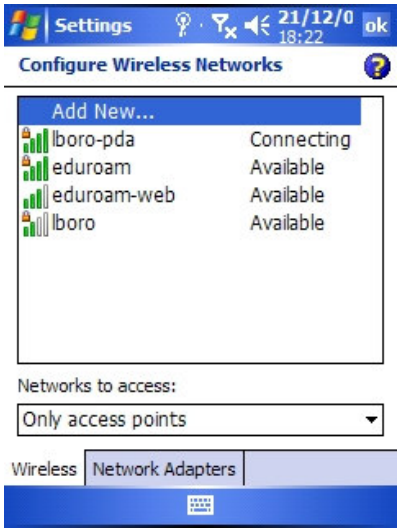
Figur 4.2.5: *P12Import 5 [22]*

9. Til slutt klikkes det på "*Properties*" knappen. En blir nå bedt om å velge hvilket sertifikat som skal brukes i påloggingssekvensen. Se etter sertifikatet som en importerte i punkt 3 og 4. Velg dette, som vist i figur 4.2.6, og trykk på "*OK*".



Figur 4.2.6: *P12Import 6 [22]*

10. Enheten er nå klar til bruk på nettverket til Trådløse Trondheim. En vil se nettverket til Trådløse Trondheim, som vist på figur 4.2.7, som "*ntnu1x*" når enheten forsøker å koble seg til nettverket.



Figur 4.2.7: *P12Import 7 [22]*

11. En vil bli bedt om å taste inn brukernavn og passord til NTNU nettverket når forbindelsen er oppkoblet. Ikke skriv inn noe under ”*Domain*”.

Som et alternativ kan en benytte seg av *SSID*en ”eduroamtest” og sertifikatet ” GTE Cyber Trust Global Root”. Dette er et forsøksnett som Trådløse Trondheim utforsker.

Denne oppskriften er av praktiske årsaker hentet fra [22]. Det ville blitt for omfattende å drive og ta skjermbilder fra Pocket PCen mens en sitter i Trondheim sentrum og arbeider.

5. DISKUSJON

5.1 Diskusjon av oppgaven

Denne oppgaven har tatt for seg teorien som er nødvendig for forståelsen av denne laben. *WEP* og *TKIP* brukes ikke i denne laben, men er tatt med for å vise historien frem til *RSN*. *WEP* blir også benyttet i den originale laboppgaven [2]. *TKIP* er også en valgfri modus i *RSN*.

Noen leverandører tilbyr en *MAC*-adressebasert tilgangskontroll-liste for autorisasjon til nettverket. Den spesifiserer hvilke stasjoner med tilhørende *MAC*-adresser som får tilgang til nettverket. Denne metoden betraktes som ubrukelig siden en angriper kan lett sniffe opp en gyldig *MAC*-adresse ved å lytte til den overførte trafikken mellom aksesspunktet og en gyldig bruker.
En annen tilgangskontrollmetode som betraktes som ubrukelig er *Closed System Authentication*. Den skrur av annonseringen av *SSID* i *Beacon-broadcasten*. *SSID*en betraktes som en hemmelighet – bare brukere som vet den får tilgang til nettverket. Denne metoden er ubrukelig fordi en angriper kan snappe opp *SSID*en fra *Probe Request/Response* rammen eller (Re)*Association Request* rammen til en gyldig bruker.

WPA/RSN er ikke en 100 % sikker metode. Hvis *FreeRADIUS* blir roothacket med rootprivilegier vil ikke ett passordbeskyttet sertifikat være beskyttet. Dette på grunn av at eap.conf lagres i klartekst. Passordbeskyttelsen er kun gjeldende ved, for eksempel, overføringen av sertifikatet fra *CA* til supplikant.
En mulig beskyttelsesmetode for eap.conf og lignende filer er å sette lese og skrivetilgangen til filene til kun å gjelde for rootbrukere. Som en konsekvens av dette må *FreeRADIUS* startes som root.
En annen metode for å unngå denne svakheten på er å generere alle private nøkler og sertifikat på *CA* maskinen, for deretter å overføre sertifikatet til supplikanten. På denne måten vil ikke sensitiv informasjon eksponeres i overføringen. Dette kan så klart kun gjøres på små, private nettverk grunnet den store administrative oppgaven med å generere private nøkler til alle supplikanter.

Når det gjelder problemstillingen angående handholdte enheter så var ikke den enheten som var tilgjengelig til meg, Qtek9000, avansert nok. Den støttet verken *RSN* eller *EAP-TLS.* Den støttet kun *TKIP* og *PEAP*. Dermed var gjensidig autentisering ikke mulig – noe som er et krav i problemstillingen til denne oppgaven.

For å koble til en handholdt enhet opp mot Trådløse Trondheim ved hjelp av *IEEE 802.1X* må en få utstedt et sertifikat fra Trådløse Trondheim. I mitt forsøk med en Qtek9000 enhet fikk jeg utdelt et .der sertifikat som jeg måtte omgjøre til et .cer sertifikat. Ved å installere dette ved hjelp av *P12Import* og velge det fra profilen fikk jeg koblet til. Formatet på sertifikatet varierer fra enhet til enhet. Det er en forutsetning at en er innenfor dekningsområdet for å få dette gjennomført. Oppsettet er vist i kapittel 4.2 – en lignende gjennomgang er vist på [22].

5.1.1 Problemområder

Jeg opplevde flere problemområder under utførelsen av denne oppgaven. Det første jeg problemet jeg støtte på var oppgraderingen av *Linux Ubuntu*. Her måtte jeg få hjelp av driftskontoret. Takk til dem. Etter jeg hadde kjørt disse oppdateringene måtte jeg legge inn diverse drivere på nytt – blant annet driverne til nettverkskortet – MadWifi. Siden jeg var nybegynner med Linux var dette en utfordring for meg. Etter noen ukers plundring med diverse drivere og den slags, var jeg oppe å gå. Dette var et problemområde jeg så for meg allerede før jeg tok fatt på oppgaven – og det viste seg å være et omfattende ett.

Ett problemområde jeg fremdeles ikke forstår er autentikatormaskinen sin trang til å bytte gateway-server. Etter en liten stund på rett gateway bytter den over til en annen gateway. Dette gjør den selv om jeg setter IP-adressen, subnet-masken og gatewayen statisk. Etter en liten stund bytter den igjen. Jeg har ikke dette problemet med de andre maskinene, så jeg har konkludert med at det må være en konfigurasjon på den maskinen som absolutt skal bruke den gatewayen. Problemet ligger ikke i mine konfigurasjonsfiler.
I slutten av mai måtte jeg koble ned laben på grunn av at *NTNU* skulle kjøre et kurs på Serengeti – det rommet jeg satt og jobbet på. Ved oppkobling av laben andre gangen, hadde jeg ikke lenger dette problemet. Maskinen holdt seg til den gatewayen den skulle.

5.2 Fremtidig arbeid

En fremtidig utvidelse av denne oppgaven kan innholde følgende temaer:

- Revokeringsliste. Lag en revokeringsliste over tidligere godkjente sertifikater, og sjekk om sertifikatene fremdeles får tilgang.
- Koble til flere supplikanter. Lag digitale sertifikat til flere supplikanter å la de koble seg opp til autentiseringsserveren for autentisering.
- Koble til flere autentikatorer. La flere autentikatorer koble seg til autentiseringsserveren. Som en konsekvens av dette må det også lages flere supplikanter – se punktet over.
- Forsøk på å bryte sikkerheten i autentiseringsserveren og *CA*en. Den delte hemmeligheten lagres i klartekst på denne maskinen. Dermed kan den være utsatt for angrep.
- Sett opp laben med andre *EAP* metoder. Blant annet er *EAP-SIM* en veldig interessant metode.
- Sette opp laben med en handholdt enhet som støtter både *RSN* og *EAP-TLS*. Siden rett utstyr ikke var tilgjengelig for meg, kan det være interessant for noen andre å koble til en handholdt enhet og se om de får det til å virke mot denne laben. Jeg vet lignende forsøk har blitt gjennomført med suksess før, så det skal la seg gjøre. Dog med *TKIP* og ikke med *RSN*. Se for øvrig [19] og [20] for mer informasjon om dette. Den aktuelle kandidaten som eventuelt skal gjøre dette må stille spørsmål ved sikkerheten til P12Import. Siden det er et open-source program, og ikke et kommersielt program, kan en ikke være helt sikker på om programmereren ikke har laget noen sikkerhetshull i programmet som kan utnyttes av en angriper. Jeg vil derfor anbefale kandidaten til å lete etter alternativer til P12import.
- Utføre flere omfattende tester med handholdte enheter som kobler opp mot Trådløse Trondheim.
- Finne frem til handholdte enheter som støtter *IEEE 802.1X, EAP-TLS* og *RSN*.

6. KONKLUSJON

Sikker trådløs kommunikasjon er endelig en realitet. Kommersielle standarder har endelig modnet nok til å tilby en omfattende løsning på sikkerhetsdilemmaet. Men som ved enhver form for sikkerhet må trådløs sikkerhet kontinuerlig utvikle seg for å være oppdatert mot de nyeste og mest sofistikerte trusselangrepene.

I denne oppgaven har det blitt tatt for seg teorien som trengs for forståelsen av denne laben. Det har også blitt beskrevet i detalj hvordan en skal gå frem for å sette opp et trådløst nettverk basert på *RSN*, *IEEE 802.1X* og *EAP-TLS* med digitale sertifikater. Med opplysningene gitt i denne masteroppgaven skal det ikke være noen problem å sette opp sitt eget nettverk, uavhengig av størrelse.
WPA er ikke perfekt – verden trenger ennå en metode som kan håndtere passordbeskyttet sertifikater uten å lagre passordet i klartekst. Men trådløse nettverksmetoder ser endelig ut til å bevege seg i en sikker retning.

Når det gjelder problemstillingen rundt handholdte enheter hadde ikke kandidaten riktig utstyr tilgjengelig. Qtek9000, som ble brukt her, støttet verken *RSN* eller *EAP-TLS*. Den støttet riktig nok *TKIP* og *PEAP*, så med litt modifikasjoner av laboppsettet kan den kobles til laben. Men siden laben da er basert på *PEAP* er ikke gjensidig autentisering mulig, og hele problemstillingen til denne oppgaven må da modifiseres.

Enheten kan benyttes til å koble til Trådløse Trondheim siden de baserer seg på *PEAP* (per dags dato). Det er muligheter for at de skal bytte denne metoden mot en annen. En må da få utstedt et *IEEE 802.1X* sertifikat fra Trådløse Trondheim som benyttes til å koble seg opp mot *SSID*en ”*ntnu1x*”. En lignende gjennomgang er vist på [22] og i kapittel 4.2.

Den ledende leverandøren av slike handholdte enheter – HTC, tidligere Qtek, har foreløpig ikke enheter som støtter *RSN*. Dermed må en vente til dette er utført før hele denne laben er gjennomførbar på en handholdt enhet. Utførelse med *TKIP* og *EAP-TLS* er derimot gjennomførbart – det finnes enheter som støtter de metodene.

REFERANSER OG KILDER

1 Real 802.11 Security – Jon Edney, William A. Arbaugh
ISBN-13: 978-0321136206

2 WLAN Security Analysis and Construction L. Haukli, S. F. Mjølsnes, M. G. Moe.
Tilgjengelig via institutt for telematikk.

3 Aksessmetoder i trådløse bynett – Kenneth Helge Molnes.
Tilgjengelig via institutt for telematikk.

4 Paranoid Penguin – Securing WLANs with WPA and FreeRADIUS, Part I av Mick Bauer
http://www.linuxjournal.com/article/8017
Publisert 1.3.2005. Hentet 28.3.2007

5 Paranoid Penguin – Securing WLANs with WPA and FreeRADIUS, Part II av Mick Bauer
http://www.linuxjournal.com/article/8095.
Publisert 6.4.2005. Hentet 28.3.2007

6 Paranoid Penguin – Securing WLANs with WPA and FreeRADIUS, Part III av Mick Bauer
http://www.linuxjournal.com/article/8151.
Publisert 28.4.2005. Hentet 30.4.2007

7 Linux Server Security – Mick Bauer
ISBN-13: 978-0596006709

8 Simon, D., og B. Aboba. 1999. PPP EAP TLS – RFC 2716
http://www.ietf.org/rfc/rfc2716.txt
Publisert oktober 1999. Hentet februar 2007.

9 C. Rigney. 2000. RADUIS – RFC 2865
http://www.ietf.org/rfc/rfc2865.txt
Publisert juni 2000. Hentet februar 2007.

10 C. Rigney. 2000. RADUIS Accounting – RFC 2866
http://www.ietf.org/rfc/rfc2866.txt
Publisert juni 2000. Hentet februar 2007.

11 Blunk, L. og J. Vollbrecht. 1998. PPP-EAP – RFC 2284
http://www.ietf.org/rfc/rfc2284.txt
Publisert mars 1998. Hentet februar 2007.

12 P. Congdon. 2003. 802.1X RADUIS – RFC 3580
http://www.ietf.org/rfc/rfc3580.txt
Publisert september 2003. Hentet februar 2007.

13 *www.freeradius.org*
Sist modifisert 1.5.2007. Hentet februar 2007.

14 *http://hostap.epitest.fi/hostapd/*
Sist modifisert 11.6.2007. Hentet februar 2007

15 *http://hostap.epitest.fi/wpa_supplicant/*
Sist modifisert 28.5.2007. Hentet februar 2007.

16 *http://sourceforge.net/projects/madwifi/*
Sist modifisert 20.3.2007. Hentet februar 2007.

17 *http://www.openssl.org/*
Hentet februar 2007.

18 *http://hostap.epitest.fi/releases/wpa_supplicant-0.5.7.tar.gz*
Publisert 11.2.2007. Hentet februar 2007. Ny versjon tilgjengelig 28.5.2007.

19 http://www.jacco2.dds.nl/networking/crtimprt.html
Sist modifisert 11.6.2007. Hentet juni 2007.

20 http://www.jacco2.dds.nl/networking/p12imprt.html
Sist modifisert 11.6.2007. Hentet juni 2007.

21 Security Analysis and Improvements for IEEE 802.11i – C. He og J. C. Mitchell
http://www.isoc.org/isoc/conferences/ndss/05/proceedings/papers/NDSS05-1107.pdf
Publisert 13.1.2005. Hentet februar 2007.

22 Loughborough University, UK
Configuring Wireless Networking on Windows Mobile 5 devices
http://www.lboro.ac.uk/computing/wireless/windows-mobile-5.html
Sist modifisert mars 2007. Hentet 14.6.2007

APPENDIKS A: Oppsett av lab – Sertifikat generering

CA privat nøkkel:

```
% openssl genrsa -des3 -out ./private/ca_private.pem 2048
Generating RSA private key, 2048 bit long modulus
.........................+++
............+++
e is 65537 (0x10001)
Enter pass phrase for ca_private.pem: serengeti
Verifying - Enter pass phrase for ca_private.pem: serengeti
```

CA sertifikat:

```
% openssl req -new -x509 -key ./private/ca_private.pem -out cacert.pem -days 1095
Enter pass phrase for ca_private.pem: serengeti
You are about to be asked to enter information that will be incorporated into your
certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:NO
State or Province Name (full name) [Some-State]:Sor-Trondelag
Locality Name (eg, city) []:Trondheim
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NTNU
Organizational Unit Name (eg, section) []:Diplom
Common Name (eg, YOUR name) []:CA
Email Address []:CA@diplom.no
```

Supplikant privat nøkkel:

```
% openssl genrsa -des3 -out ./private/supplicant_private.pem 2048
 Generating RSA private key, 2048 bit long modulus
..............................+++
........+++
e is 65537 (0x10001)
Enter pass phrase for ./private/supplicant_private.pem: serengeti
Verifying - Enter pass phrase for ./private/supplicant_private.pem: serengeti
```

Supplikant anmodning:

```
% openssl req -new -key ./private/supplicant_private.pem -out supplicant.csr
Enter pass phrase for ./private/supplicant_private.pem: serengeti
You are about to be asked to enter information that will be incorporated into your
certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:NO
State or Province Name (full name) [Some-State]:Sor-Trondelag
Locality Name (eg, city) []:Trondheim
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NTNU
Organizational Unit Name (eg, section) []:Diplom
Common Name (eg, YOUR name) []:Supplicant
Email Address []:supplicant@diplom.no

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:serengeti
An optional company name []:diplom
```

Supplikantsertifikat. Signering fra *CA*:

```
% openssl ca -out supplicant.pem -infiles supplicant.csr
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for /etc/ssl/private/ca_private.pem: serengeti
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4 (0x4)
        Validity
            Not Before: May  7 13:37:51 2007 GMT
            Not After : May  6 13:37:51 2008 GMT
        Subject:
            countryName               = NO
            stateOrProvinceName       = Sor-Trondelag
            organizationName          = NTNU
            organizationalUnitName    = Diplom
            commonName                = Supplicant
            emailAddress              = supplicant@diplom.no
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
```

```
            OpenSSL Generated Certificate
        X509v3 Subject Key Identifier:
            0F:0D:78:80:04:85:91:DF:CF:8A:B0:F0:67:E8:D5:45:B4:CB:60:1F
        X509v3 Authority Key Identifier:
            keyid:16:83:AE:0D:2F:0E:33:0C:FB:C6:BA:80:16:F7:C7:50:63:5C:28:DE

Certificate is to be certified until May  6 13:37:51 2008 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Autentiseringsserver privat nøkkel:

```
% openssl genrsa -des3 -out ./private/as_private.pem 2048
Generating RSA private key, 2048 bit long modulus
.+++
...+++
e is 65537 (0x10001)
Enter pass phrase for ./private/as_private.pem: serengeti
Verifying - Enter pass phrase for ./private/as_private.pem: serengeti
```

Autentiseringsserver anmodning:

```
% openssl req -new -key ./private/as_private.pem -out as.csr
Enter pass phrase for ./private/as_private.pem: serengeti
You are about to be asked to enter information that will be incorporated into your
certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:NO
State or Province Name (full name) [Some-State]:Sor-Trondelag
Locality Name (eg, city) []:Trondheim
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NTNU
Organizational Unit Name (eg, section) []:Diplom
Common Name (eg, YOUR name) []:AS
Email Address []:as@diplom.no

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:serengeti
An optional company name []:diplom
```

Autentiseringsserversertifikat. Signering fra *CA*:

```
% openssl ca -out as.pem -infiles as.csr
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for /etc/ssl/private/ca_private.pem: serengeti
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 5 (0x5)
        Validity
            Not Before: May  7 13:42:03 2007 GMT
            Not After : May  6 13:42:03 2008 GMT
        Subject:
```

```
            countryName               = NO
            stateOrProvinceName       = Sor-Trondelag
            organizationName          = NTNU
            organizationalUnitName    = Diplom
            commonName                = AS
            emailAddress              = as@diplom.no
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                52:FC:DD:88:52:D7:01:0A:92:5B:A9:32:DE:B3:8B:30:39:91:DC:BF
            X509v3 Authority Key Identifier:
                keyid:16:83:AE:0D:2F:0E:33:0C:FB:C6:BA:80:16:F7:C7:50:63:5C:28:DE

Certificate is to be certified until May  6 13:42:03 2008 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

APPENDIKS B: Oppsett av lab - Supplikant

Wpa_supplicant.conf:

```
##### Example wpa_supplicant configuration file
###############################
#
# This file describes configuration file format and lists all available
option.
# Please also take a look at simpler configuration examples in 'examples'
# subdirectory.
#
# Empty lines and lines starting with # are ignored

# NOTE! This file may contain password information and should probably be
made
# readable only by root user on multiuser systems.

# Note: All file paths in this configuration file should use full (absolute,
# not relative to working directory) path in order to allow working directory
# to be changed. This can happen if wpa_supplicant is run in the background.

# Whether to allow wpa_supplicant to update (overwrite) configuration
#
# This option can be used to allow wpa_supplicant to overwrite configuration
# file whenever configuration is changed (e.g., new network block is added
with
# wpa_cli or wpa_gui, or a password is changed). This is required for
# wpa_cli/wpa_gui to be able to store the configuration changes permanently.
# Please note that overwriting configuration file will remove the comments
from
# it.
#update_config=1

# global configuration (shared by all network blocks)
#
# Parameters for the control interface. If this is specified, wpa_supplicant
# will open a control interface that is available for external programs to
# manage wpa_supplicant. The meaning of this string depends on which control
# interface mechanism is used. For all cases, the existance of this parameter
# in configuration is used to determine whether the control interface is
# enabled.
#
# For UNIX domain sockets (default on Linux and BSD): This is a directory
that
# will be created for UNIX domain sockets for listening to requests from
```

```
# external programs (CLI/GUI, etc.) for status information and configuration.
# The socket file will be named based on the interface name, so multiple
# wpa_supplicant processes can be run at the same time if more than one
# interface is used.
# /var/run/wpa_supplicant is the recommended directory for sockets and by
# default, wpa_cli will use it when trying to connect with wpa_supplicant.
#
# Access control for the control interface can be configured by setting the
# directory to allow only members of a group to use sockets. This way, it is
# possible to run wpa_supplicant as root (since it needs to change network
# configuration and open raw sockets) and still allow GUI/CLI components to
be
# run as non-root users. However, since the control interface can be used to
# change the network configuration, this access needs to be protected in many
# cases. By default, wpa_supplicant is configured to use gid 0 (root). If you
# want to allow non-root users to use the control interface, add a new group
# and change this value to match with that group. Add users that should have
# control interface access to this group. If this variable is commented out
or
# not included in the configuration file, group will not be changed from the
# value it got by default when the directory or socket was created.
#
# When configuring both the directory and group, use following format:
# DIR=/var/run/wpa_supplicant GROUP=wheel
# DIR=/var/run/wpa_supplicant GROUP=0
# (group can be either group name or gid)
#
# For UDP connections (default on Windows): The value will be ignored. This
# variable is just used to select that the control interface is to be
created.
# The value can be set to, e.g., udp (ctrl_interface=udp)
#
# For Windows Named Pipe: This value can be used to set the security
descriptor
# for controlling access to the control interface. Security descriptor can be
# set using Security Descriptor String Format (see http://msdn.microsoft.com/
# library/default.asp?url=/library/en-us/secauthz/security/
# security_descriptor_string_format.asp). The descriptor string needs to be
# prefixed with SDDL=. For example, ctrl_interface=SDDL=D: would set an empty
# DACL (which will reject all connections). See README-Windows.txt for more
# information about SDDL string format.
#
ctrl_interface=/var/run/wpa_supplicant

# IEEE 802.1X/EAPOL version
# wpa_supplicant is implemented based on IEEE Std 802.1X-2004 which defines
```

```
# EAPOL version 2. However, there are many APs that do not handle the new
# version number correctly (they seem to drop the frames completely). In
order
# to make wpa_supplicant interoperate with these APs, the version number is
set
# to 1 by default. This configuration value can be used to set it to the new
# version (2).
eapol_version=1

# AP scanning/selection
# By default, wpa_supplicant requests driver to perform AP scanning and then
# uses the scan results to select a suitable AP. Another alternative is to
# allow the driver to take care of AP scanning and selection and use
# wpa_supplicant just to process EAPOL frames based on IEEE 802.11
association
# information from the driver.
# 1: wpa_supplicant initiates scanning and AP selection
# 0: driver takes care of scanning, AP selection, and IEEE 802.11 association
#    parameters (e.g., WPA IE generation); this mode can also be used with
#    non-WPA drivers when using IEEE 802.1X mode; do not try to associate
with
#    APs (i.e., external program needs to control association). This mode
must
#    also be used when using wired Ethernet drivers.
# 2: like 0, but associate with APs using security policy and SSID (but not
#    BSSID); this can be used, e.g., with ndiswrapper and NDIS drivers to
#    enable operation with hidden SSIDs and optimized roaming; in this mode,
#    the network blocks in the configuration file are tried one by one until
#    the driver reports successful association; each network block should
have
#    explicit security policy (i.e., only one option in the lists) for
#    key_mgmt, pairwise, group, proto variables
ap_scan=1

# EAP fast re-authentication
# By default, fast re-authentication is enabled for all EAP methods that
# support it. This variable can be used to disable fast re-authentication.
# Normally, there is no need to disable this.
fast_reauth=1

# OpenSSL Engine support
# These options can be used to load OpenSSL engines.
# The two engines that are supported currently are shown below:
# They are both from the opensc project (http://www.opensc.org/)
# By default no engines are loaded.
# make the opensc engine available
```

```
#opensc_engine_path=/usr/lib/opensc/engine_opensc.so
# make the pkcs11 engine available
#pkcs11_engine_path=/usr/lib/opensc/engine_pkcs11.so
# configure the path to the pkcs11 module required by the pkcs11 engine
#pkcs11_module_path=/usr/lib/pkcs11/opensc-pkcs11.so

# Dynamic EAP methods
# If EAP methods were built dynamically as shared object files, they need to
be
# loaded here before being used in the network blocks. By default, EAP
methods
# are included statically in the build, so these lines are not needed
#load_dynamic_eap=/usr/lib/wpa_supplicant/eap_tls.so
#load_dynamic_eap=/usr/lib/wpa_supplicant/eap_md5.so

# Driver interface parameters
# This field can be used to configure arbitrary driver interace parameters.
The
# format is specific to the selected driver interface. This field is not used
# in most cases.
#driver_param="field=value"

# Maximum lifetime for PMKSA in seconds; default 43200
#dot11RSNAConfigPMKLifetime=43200
# Threshold for reauthentication (percentage of PMK lifetime); default 70
#dot11RSNAConfigPMKReauthThreshold=70
# Timeout for security association negotiation in seconds; default 60
#dot11RSNAConfigSATimeout=60

# network block
#
# Each network (usually AP's sharing the same SSID) is configured as a
separate
# block in this configuration file. The network blocks are in preference
order
# (the first match is used).
#
# network block fields:
#
# disabled:
#      0 = this network can be used (default)
#      1 = this network block is disabled (can be enabled through ctrl_iface,
#          e.g., with wpa_cli or wpa_gui)
#
# id_str: Network identifier string for external scripts. This value is
passed
```

```
#      to external action script through wpa_cli as WPA_ID_STR environment
#      variable to make it easier to do network specific configuration.
#
# ssid: SSID (mandatory); either as an ASCII string with double quotation or
#      as hex string; network name
#
# scan_ssid:
#      0 = do not scan this SSID with specific Probe Request frames (default)
#      1 = scan with SSID-specific Probe Request frames (this can be used to
#          find APs that do not accept broadcast SSID or use multiple SSIDs;
#          this will add latency to scanning, so enable this only when
needed)
#
# bssid: BSSID (optional); if set, this network block is used only when
#      associating with the AP using the configured BSSID
#
# priority: priority group (integer)
# By default, all networks will get same priority group (0). If some of the
# networks are more desirable, this field can be used to change the order in
# which wpa_supplicant goes through the networks when selecting a BSS. The
# priority groups will be iterated in decreasing priority (i.e., the larger
the
# priority value, the sooner the network is matched against the scan
results).
# Within each priority group, networks will be selected based on security
# policy, signal strength, etc.
# Please note that AP scanning with scan_ssid=1 and ap_scan=2 mode are not
# using this priority to select the order for scanning. Instead, they try the
# networks in the order that used in the configuration file.
#
# mode: IEEE 802.11 operation mode
# 0 = infrastructure (Managed) mode, i.e., associate with an AP (default)
# 1 = IBSS (ad-hoc, peer-to-peer)
# Note: IBSS can only be used with key_mgmt NONE (plaintext and static WEP)
# and key_mgmt=WPA-NONE (fixed group key TKIP/CCMP). In addition, ap_scan has
# to be set to 2 for IBSS. WPA-None requires following network block options:
# proto=WPA, key_mgmt=WPA-NONE, pairwise=NONE, group=TKIP (or CCMP, but not
# both), and psk must also be set.
#
# proto: list of accepted protocols
# WPA = WPA/IEEE 802.11i/D3.0
# RSN = WPA2/IEEE 802.11i (also WPA2 can be used as an alias for RSN)
# If not set, this defaults to: WPA RSN
#
# key_mgmt: list of accepted authenticated key management protocols
# WPA-PSK = WPA pre-shared key (this requires 'psk' field)
```

```
# WPA-EAP = WPA using EAP authentication (this can use an external
#      program, e.g., Xsupplicant, for IEEE 802.1X EAP Authentication
# IEEE8021X = IEEE 802.1X using EAP authentication and (optionally)
dynamically
#      generated WEP keys
# NONE = WPA is not used; plaintext or static WEP could be used
# If not set, this defaults to: WPA-PSK WPA-EAP
#
# auth_alg: list of allowed IEEE 802.11 authentication algorithms
# OPEN = Open System authentication (required for WPA/WPA2)
# SHARED = Shared Key authentication (requires static WEP keys)
# LEAP = LEAP/Network EAP (only used with LEAP)
# If not set, automatic selection is used (Open System with LEAP enabled if
# LEAP is allowed as one of the EAP methods).
#
# pairwise: list of accepted pairwise (unicast) ciphers for WPA
# CCMP = AES in Counter mode with CBC-MAC [RFC 3610, IEEE 802.11i/D7.0]
# TKIP = Temporal Key Integrity Protocol [IEEE 802.11i/D7.0]
# NONE = Use only Group Keys (deprecated, should not be included if APs
support
#      pairwise keys)
# If not set, this defaults to: CCMP TKIP
#
# group: list of accepted group (broadcast/multicast) ciphers for WPA
# CCMP = AES in Counter mode with CBC-MAC [RFC 3610, IEEE 802.11i/D7.0]
# TKIP = Temporal Key Integrity Protocol [IEEE 802.11i/D7.0]
# WEP104 = WEP (Wired Equivalent Privacy) with 104-bit key
# WEP40 = WEP (Wired Equivalent Privacy) with 40-bit key [IEEE 802.11]
# If not set, this defaults to: CCMP TKIP WEP104 WEP40
#
# psk: WPA preshared key; 256-bit pre-shared key
# The key used in WPA-PSK mode can be entered either as 64 hex-digits, i.e.,
# 32 bytes or as an ASCII passphrase (in which case, the real PSK will be
# generated using the passphrase and SSID). ASCII passphrase must be between
# 8 and 63 characters (inclusive).
# This field is not needed, if WPA-EAP is used.
# Note: Separate tool, wpa_passphrase, can be used to generate 256-bit keys
# from ASCII passphrase. This process uses lot of CPU and wpa_supplicant
# startup and reconfiguration time can be optimized by generating the PSK
only
# only when the passphrase or SSID has actually changed.
#
# eapol_flags: IEEE 802.1X/EAPOL options (bit field)
# Dynamic WEP key required for non-WPA mode
# bit0 (1): require dynamically generated unicast WEP key
# bit1 (2): require dynamically generated broadcast WEP key
```

```
#      (3 = require both keys; default)
# Note: When using wired authentication, eapol_flags must be set to 0 for the
# authentication to be completed successfully.
#
# proactive_key_caching:
# Enable/disable opportunistic PMKSA caching for WPA2.
# 0 = disabled (default)
# 1 = enabled
#
# wep_key0..3: Static WEP key (ASCII in double quotation, e.g. "abcde" or
# hex without quotation, e.g., 0102030405)
# wep_tx_keyidx: Default WEP key index (TX) (0..3)
#
# stakey: Whether STAKey negotiation for direct links (IEEE 802.11e) is
# allowed. This is only used with RSN/WPA2.
# 0 = disabled (default)
# 1 = enabled
#stakey=1
#
# Following fields are only used with internal EAP implementation.
# eap: space-separated list of accepted EAP methods
#      MD5 = EAP-MD5 (unsecure and does not generate keying material ->
#                     cannot be used with WPA; to be used as a Phase 2 method
#                     with EAP-PEAP or EAP-TTLS)
#       MSCHAPV2 = EAP-MSCHAPv2 (cannot be used separately with WPA; to be
used
#             as a Phase 2 method with EAP-PEAP or EAP-TTLS)
#       OTP = EAP-OTP (cannot be used separately with WPA; to be used
#             as a Phase 2 method with EAP-PEAP or EAP-TTLS)
#       GTC = EAP-GTC (cannot be used separately with WPA; to be used
#             as a Phase 2 method with EAP-PEAP or EAP-TTLS)
#      TLS = EAP-TLS (client and server certificate)
#      PEAP = EAP-PEAP (with tunnelled EAP authentication)
#      TTLS = EAP-TTLS (with tunnelled EAP or PAP/CHAP/MSCHAP/MSCHAPV2
#                       authentication)
#      If not set, all compiled in methods are allowed.
#
# identity: Identity string for EAP
# anonymous_identity: Anonymous identity string for EAP (to be used as the
#      unencrypted identity with EAP types that support different tunnelled
#      identity, e.g., EAP-TTLS)
# password: Password string for EAP
# ca_cert: File path to CA certificate file (PEM/DER). This file can have one
#      or more trusted CA certificates. If ca_cert and ca_path are not
#      included, server certificate will not be verified. This is insecure
and
```

```
#      a trusted CA certificate should always be configured when using
#      EAP-TLS/TTLS/PEAP. Full path should be used since working directory
may
#      change when wpa_supplicant is run in the background.
#      On Windows, trusted CA certificates can be loaded from the system
#      certificate store by setting this to cert_store://<name>, e.g.,
#      ca_cert="cert_store://CA" or ca_cert="cert_store://ROOT".
#      Note that when running wpa_supplicant as an application, the user
#      certificate store (My user account) is used, whereas computer store
#      (Computer account) is used when running wpasvc as a service.
# ca_path: Directory path for CA certificate files (PEM). This path may
#      contain multiple CA certificates in OpenSSL format. Common use for
this
#      is to point to system trusted CA list which is often installed into
#      directory like /etc/ssl/certs. If configured, these certificates are
#      added to the list of trusted CAs. ca_cert may also be included in that
#      case, but it is not required.
# client_cert: File path to client certificate file (PEM/DER)
#      Full path should be used since working directory may change when
#      wpa_supplicant is run in the background.
#      Alternatively, a named configuration blob can be used by setting this
#      to blob://<blob name>.
# private_key: File path to client private key file (PEM/DER/PFX)
#      When PKCS#12/PFX file (.p12/.pfx) is used, client_cert should be
#      commented out. Both the private key and certificate will be read from
#      the PKCS#12 file in this case. Full path should be used since working
#      directory may change when wpa_supplicant is run in the background.
#      Windows certificate store can be used by leaving client_cert out and
#      configuring private_key in one of the following formats:
#      cert://substring_to_match
#      hash://certificate_thumbprint_in_hex
#      for example: private_key="hash://63093aa9c47f56ae88334c7b65a4"
#      Note that when running wpa_supplicant as an application, the user
#      certificate store (My user account) is used, whereas computer store
#      (Computer account) is used when running wpasvc as a service.
#      Alternatively, a named configuration blob can be used by setting this
#      to blob://<blob name>.
# private_key_passwd: Password for private key file (if left out, this will
be
#      asked through control interface)
# dh_file: File path to DH/DSA parameters file (in PEM format)
#      This is an optional configuration file for setting parameters for an
#      ephemeral DH key exchange. In most cases, the default RSA
#      authentication does not use this configuration. However, it is
possible
#      setup RSA to use ephemeral DH key exchange. In addition, ciphers with
```

```
#      DSA keys always use ephemeral DH keys. This can be used to achieve
#      forward secrecy. If the file is in DSA parameters format, it will be
#      automatically converted into DH params.
# subject_match: Substring to be matched against the subject of the
#      authentication server certificate. If this string is set, the server
#      sertificate is only accepted if it contains this string in the
subject.
#      The subject string is in following format:
#      /C=US/ST=CA/L=San Francisco/CN=Test AS/emailAddress=as@example.com
# altsubject_match: Substring to be matched against the alternative subject
#      name of the authentication server certificate. If this string is set,
#      the server sertificate is only accepted if it contains this string in
#      an alternative subject name extension.
#      altSubjectName string is in following format: TYPE:VALUE
#      Example: DNS:server.example.com
#      Following types are supported: EMAIL, DNS, URI
# phase1: Phase1 (outer authentication, i.e., TLS tunnel) parameters
#      (string with field-value pairs, e.g., "peapver=0" or
#      "peapver=1 peaplabel=1")
#      'peapver' can be used to force which PEAP version (0 or 1) is used.
#      'peaplabel=1' can be used to force new label, "client PEAP
encryption",
#      to be used during key derivation when PEAPv1 or newer. Most existing
#      PEAPv1 implementation seem to be using the old label, "client EAP
#      encryption", and wpa_supplicant is now using that as the default
value.
#      Some servers, e.g., Radiator, may require peaplabel=1 configuration to
#      interoperate with PEAPv1; see eap_testing.txt for more details.
#      'peap_outer_success=0' can be used to terminate PEAP authentication on
#      tunneled EAP-Success. This is required with some RADIUS servers that
#      implement draft-josefsson-pppext-eap-tls-eap-05.txt (e.g.,
#      Lucent NavisRadius v4.4.0 with PEAP in "IETF Draft 5" mode)
#      include_tls_length=1 can be used to force wpa_supplicant to include
#      TLS Message Length field in all TLS messages even if they are not
#      fragmented.
#      sim_min_num_chal=3 can be used to configure EAP-SIM to require three
#      challenges (by default, it accepts 2 or 3)
# phase2: Phase2 (inner authentication with TLS tunnel) parameters
#      (string with field-value pairs, e.g., "auth=MSCHAPV2" for EAP-PEAP or
#      "autheap=MSCHAPV2 autheap=MD5" for EAP-TTLS)
# Following certificate/private key fields are used in inner Phase2
# authentication when using EAP-TTLS or EAP-PEAP.
# ca_cert2: File path to CA certificate file. This file can have one or more
#      trusted CA certificates. If ca_cert2 and ca_path2 are not included,
#      server certificate will not be verified. This is insecure and a
trusted
```

```
#      CA certificate should always be configured.
# ca_path2: Directory path for CA certificate files (PEM)
# client_cert2: File path to client certificate file
# private_key2: File path to client private key file
# private_key2_passwd: Password for private key file
# dh_file2: File path to DH/DSA parameters file (in PEM format)
# subject_match2: Substring to be matched against the subject of the
#      authentication server certificate.
# altsubject_match2: Substring to be matched against the alternative subject
#      name of the authentication server certificate.
#
# fragment_size: Maximum EAP fragment size in bytes (default 1398).
#      This value limits the fragment size for EAP methods that support
#      fragmentation (e.g., EAP-TLS and EAP-PEAP). This value should be set
#      small enough to make the EAP messages fit in MTU of the network
#      interface used for EAPOL. The default value is suitable for most
#      cases.
#
# EAP-PSK variables:
# eappsk: 16-byte (128-bit, 32 hex digits) pre-shared key in hex format
# nai: user NAI
#
# EAP-PAX variables:
# eappsk: 16-byte (128-bit, 32 hex digits) pre-shared key in hex format
#
# EAP-SAKE variables:
# eappsk: 32-byte (256-bit, 64 hex digits) pre-shared key in hex format
#      (this is concatenation of Root-Secret-A and Root-Secret-B)
# nai: user NAI (PEERID)
#
# EAP-GPSK variables:
# eappsk: Pre-shared key in hex format (at least 128 bits, i.e., 32 hex
digits)
# nai: user NAI (ID_Client)
#
# EAP-FAST variables:
# pac_file: File path for the PAC entries. wpa_supplicant will need to be
able
#      to create this file and write updates to it when PAC is being
#      provisioned or refreshed. Full path to the file should be used since
#      working directory may change when wpa_supplicant is run in the
#      background. Alternatively, a named configuration blob can be used by
#      setting this to blob://<blob name>
# phase1: fast_provisioning=1 option enables in-line provisioning of EAP-FAST
#      credentials (PAC)
#
```

```
# wpa_supplicant supports number of "EAP workarounds" to work around
# interoperability issues with incorrectly behaving authentication servers.
# These are enabled by default because some of the issues are present in
large
# number of authentication servers. Strict EAP conformance mode can be
# configured by disabling workarounds with eap_workaround=0.

# Example blocks:

# Simple case: WPA-PSK, PSK as an ASCII passphrase, allow all valid ciphers
network={
       ssid="simple"
       psk="very secret passphrase"
       priority=5
}

# Same as previous, but request SSID-specific scanning (for APs that reject
# broadcast SSID)
network={
       ssid="second ssid"
       scan_ssid=1
       psk="very secret passphrase"
       priority=2
}

# Only WPA-PSK is used. Any valid cipher combination is accepted.
network={
       ssid="example"
       proto=WPA
       key_mgmt=WPA-PSK
       pairwise=CCMP TKIP
       group=CCMP TKIP WEP104 WEP40
       psk=06b4be19da289f475aa46a33cb793029d4ab3db7a23ee92382eb0106c72ac7bb
       priority=2
}

# Only WPA-EAP is used. Both CCMP and TKIP is accepted. An AP that used
WEP104
# or WEP40 as the group cipher will not be accepted.
network={
       ssid="example"
       proto=RSN
       key_mgmt=WPA-EAP
       pairwise=CCMP TKIP
       group=CCMP TKIP
       eap=TLS
```

```
        identity="user@example.com"
        ca_cert="/etc/cert/ca.pem"
        client_cert="/etc/cert/user.pem"
        private_key="/etc/cert/user.prv"
        private_key_passwd="password"
        priority=1
}

# EAP-PEAP/MSCHAPv2 configuration for RADIUS servers that use the new
peaplabel
# (e.g., Radiator)
network={
        ssid="example"
        key_mgmt=WPA-EAP
        eap=PEAP
        identity="user@example.com"
        password="foobar"
        ca_cert="/etc/cert/ca.pem"
        phase1="peaplabel=1"
        phase2="auth=MSCHAPV2"
        priority=10
}

# EAP-TTLS/EAP-MD5-Challenge configuration with anonymous identity for the
# unencrypted use. Real identity is sent only within an encrypted TLS tunnel.
network={
        ssid="example"
        key_mgmt=WPA-EAP
        eap=TTLS
        identity="user@example.com"
        anonymous_identity="anonymous@example.com"
        password="foobar"
        ca_cert="/etc/cert/ca.pem"
        priority=2
}

# EAP-TTLS/MSCHAPv2 configuration with anonymous identity for the unencrypted
# use. Real identity is sent only within an encrypted TLS tunnel.
network={
        ssid="example"
        key_mgmt=WPA-EAP
        eap=TTLS
        identity="user@example.com"
        anonymous_identity="anonymous@example.com"
        password="foobar"
        ca_cert="/etc/cert/ca.pem"
```

```
	phase2="auth=MSCHAPV2"
}

# WPA-EAP, EAP-TTLS with different CA certificate used for outer and inner
# authentication.
network={
	ssid="example"
	key_mgmt=WPA-EAP
	eap=TTLS
	# Phase1 / outer authentication
	anonymous_identity="anonymous@example.com"
	ca_cert="/etc/cert/ca.pem"
	# Phase 2 / inner authentication
	phase2="autheap=TLS"
	ca_cert2="/etc/cert/ca2.pem"
	client_cert2="/etc/cer/user.pem"
	private_key2="/etc/cer/user.prv"
	private_key2_passwd="password"
	priority=2
}

# Both WPA-PSK and WPA-EAP is accepted. Only CCMP is accepted as pairwise and
# group cipher.
network={
	ssid="example"
	bssid=00:11:22:33:44:55
	proto=WPA RSN
	key_mgmt=WPA-PSK WPA-EAP
	pairwise=CCMP
	group=CCMP
	psk=06b4be19da289f475aa46a33cb793029d4ab3db7a23ee92382eb0106c72ac7bb
}

# Special characters in SSID, so use hex string. Default to WPA-PSK, WPA-EAP
# and all valid ciphers.
network={
	ssid=00010203
	psk=000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
}


# EAP-SIM with a GSM SIM or USIM
network={
	ssid="eap-sim-test"
	key_mgmt=WPA-EAP
	eap=SIM
```

```
        pin="1234"
        pcsc=""
}


# EAP-PSK
network={
        ssid="eap-psk-test"
        key_mgmt=WPA-EAP
        eap=PSK
        identity="eap_psk_user"
        eappsk=06b4be19da289f475aa46a33cb793029
        nai="eap_psk_user@example.com"
}


# IEEE 802.1X/EAPOL with dynamically generated WEP keys (i.e., no WPA) using
# EAP-TLS for authentication and key generation; require both unicast and
# broadcast WEP keys.
network={
        ssid="1x-test"
        key_mgmt=IEEE8021X
        eap=TLS
        identity="user@example.com"
        ca_cert="/etc/cert/ca.pem"
        client_cert="/etc/cert/user.pem"
        private_key="/etc/cert/user.prv"
        private_key_passwd="password"
        eapol_flags=3
}


# LEAP with dynamic WEP keys
network={
        ssid="leap-example"
        key_mgmt=IEEE8021X
        eap=LEAP
        identity="user"
        password="foobar"
}

# EAP-FAST with WPA (WPA or WPA2)
network={
        ssid="eap-fast-test"
        key_mgmt=WPA-EAP
        eap=FAST
```

```
        anonymous_identity="FAST-000102030405"
        identity="username"
        password="password"
        phase1="fast_provisioning=1"
        pac_file="/etc/wpa_supplicant.eap-fast-pac"
}

network={
        ssid="eap-fast-test"
        key_mgmt=WPA-EAP
        eap=FAST
        anonymous_identity="FAST-000102030405"
        identity="username"
        password="password"
        phase1="fast_provisioning=1"
        pac_file="blob://eap-fast-pac"
}

# Plaintext connection (no WPA, no IEEE 802.1X)
network={
        ssid="plaintext-test"
        key_mgmt=NONE
}


# Shared WEP key connection (no WPA, no IEEE 802.1X)
network={
        ssid="static-wep-test"
        key_mgmt=NONE
        wep_key0="abcde"
        wep_key1=0102030405
        wep_key2="1234567890123"
        wep_tx_keyidx=0
        priority=5
}


# Shared WEP key connection (no WPA, no IEEE 802.1X) using Shared Key
# IEEE 802.11 authentication
network={
        ssid="static-wep-test2"
        key_mgmt=NONE
        wep_key0="abcde"
        wep_key1=0102030405
        wep_key2="1234567890123"
        wep_tx_keyidx=0
```

```
        priority=5
        auth_alg=SHARED
}


# IBSS/ad-hoc network with WPA-None/TKIP.
network={
        ssid="test adhoc"
        mode=1
        proto=WPA
        key_mgmt=WPA-NONE
        pairwise=NONE
        group=TKIP
        psk="secret passphrase"
}


# Catch all example that allows more or less all configuration modes
network={
        ssid="example"
        scan_ssid=1
        key_mgmt=WPA-EAP WPA-PSK IEEE8021X NONE
        pairwise=CCMP TKIP
        group=CCMP TKIP WEP104 WEP40
        psk="very secret passphrase"
        eap=TTLS PEAP TLS
        identity="user@example.com"
        password="foobar"
        ca_cert="/etc/cert/ca.pem"
        client_cert="/etc/cert/user.pem"
        private_key="/etc/cert/user.prv"
        private_key_passwd="password"
        phase1="peaplabel=0"
}

# Example of EAP-TLS with smartcard (openssl engine)
network={
        ssid="example"
        key_mgmt=WPA-EAP
        eap=TLS
        proto=RSN
        pairwise=CCMP TKIP
        group=CCMP TKIP
        identity="user@example.com"
        ca_cert="/etc/cert/ca.pem"
        client_cert="/etc/cert/user.pem"
```

```
        engine=1

        # The engine configured here must be available. Look at
        # OpenSSL engine support in the global section.
        # The key available through the engine must be the private key
        # matching the client certificate configured above.

        # use the opensc engine
        #engine_id="opensc"
        #key_id="45"

        # use the pkcs11 engine
        engine_id="pkcs11"
        key_id="id_45"

        # Optional PIN configuration; this can be left out and PIN will be
        # asked through the control interface
        pin="1234"
}

# Example configuration showing how to use an inlined blob as a CA
certificate
# data instead of using external file
network={
        ssid="example"
        key_mgmt=WPA-EAP
        eap=TTLS
        identity="user@example.com"
        anonymous_identity="anonymous@example.com"
        password="foobar"
        ca_cert="blob://exampleblob"
        priority=20
}

blob-base64-exampleblob={
SGVsbG8gV29ybGQhCg==
}


# Wildcard match for SSID (plaintext APs only). This example select any
# open AP regardless of its SSID.
network={
        key_mgmt=NONE
}
```

```
Kjørelog:
1176713223.740407: Cancelling scan request
1176713223.740415: RTM_NEWLINK: operstate=0 ifi_flags=0x11003
([UP][LOWER_UP])
1176713223.740422: RTM_NEWLINK, IFLA_IFNAME: Interface 'ath1' added
1176713223.740434: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713223.740441: RX EAPOL - hexdump(len=9): 02 00 00 05 01 00 00 05 01
1176713223.740450: Setting authentication timeout: 70 sec 0 usec
1176713223.740457: EAPOL: Received EAP-Packet frame
1176713223.740463: EAPOL: SUPP_PAE entering state RESTART
1176713223.740469: EAP: EAP entering state INITIALIZE
1176713223.740474: EAP: EAP entering state IDLE
1176713223.740479: EAPOL: SUPP_PAE entering state AUTHENTICATING
1176713223.740485: EAPOL: SUPP_BE entering state REQUEST
1176713223.740490: EAPOL: getSuppRsp
1176713223.740495: EAP: EAP entering state RECEIVED
1176713223.740530: EAP: Received EAP-Request id=0 method=1 vendor=0
vendorMethod=0
1176713223.740538: EAP: EAP entering state IDENTITY
1176713223.740544: CTRL-EVENT-EAP-STARTED EAP authentication started
1176713223.740551: EAP: EAP-Request Identity data - hexdump_ascii(len=0):
1176713223.740557: EAP: using real identity - hexdump_ascii(len=10):
     53 75 70 70 6c 69 63 61 6e 74                     Supplicant
1176713223.740572: EAP: EAP entering state SEND_RESPONSE
1176713223.740579: EAP: EAP entering state IDLE
1176713223.740585: EAPOL: SUPP_BE entering state RESPONSE
1176713223.740590: EAPOL: txSuppRsp
1176713223.740596: TX EAPOL - hexdump(len=19): 01 00 00 0f 02 00 00 0f 01 53
75 70 70 6c 69 63 61 6e 74
1176713223.740631: EAPOL: SUPP_BE entering state RECEIVE
1176713226.609723: EAPOL: startWhen --> 0
1176713230.885583: RTM_NEWLINK: operstate=0 ifi_flags=0x11003
([UP][LOWER_UP])
1176713230.885609: Wireless event: cmd=0x8b19 len=8
1176713230.885718: Received 4095 bytes of scan results (24 BSSes)
1176713230.885727: Scan results: 24
1176713230.885734: Selecting BSS from priority group 0
1176713230.885741: 0: 00:0f:cb:b5:0e:d2 ssid='Kenneth' wpa_ie_len=0
rsn_ie_len=22 caps=0x11
1176713230.885752:    selected based on RSN IE
1176713230.885758: Already associated with the selected AP.
1176713253.634850: EAPOL: authWhile --> 0
1176713253.634875: EAPOL: SUPP_BE entering state TIMEOUT
1176713253.634883: EAPOL: SUPP_PAE entering state CONNECTING
1176713253.634888: EAPOL: SUPP_BE entering state IDLE
1176713253.735119: RX EAPOL from 00:0f:cb:b5:0e:d2
```

```
1176713253.735142: RX EAPOL - hexdump(len=9): 02 00 00 05 01 01 00 05 01
1176713253.735154: EAPOL: Received EAP-Packet frame
1176713253.735162: EAPOL: SUPP_PAE entering state RESTART
1176713253.735169: EAP: EAP entering state INITIALIZE
1176713253.735175: EAP: EAP entering state IDLE
1176713253.735181: EAPOL: SUPP_PAE entering state AUTHENTICATING
1176713253.735187: EAPOL: SUPP_BE entering state REQUEST
1176713253.735192: EAPOL: getSuppRsp
1176713253.735197: EAP: EAP entering state RECEIVED
1176713253.735206: EAP: Received EAP-Request id=1 method=1 vendor=0
vendorMethod=0
1176713253.735213: EAP: EAP entering state IDENTITY
1176713253.735222: CTRL-EVENT-EAP-STARTED EAP authentication started
1176713253.735229: EAP: EAP-Request Identity data - hexdump_ascii(len=0):
1176713253.735235: EAP: using real identity - hexdump_ascii(len=10):
     53 75 70 70 6c 69 63 61 6e 74                     Supplicant
1176713253.735250: EAP: EAP entering state SEND_RESPONSE
1176713253.735256: EAP: EAP entering state IDLE
1176713253.735261: EAPOL: SUPP_BE entering state RESPONSE
1176713253.735267: EAPOL: txSuppRsp
1176713253.735273: TX EAPOL - hexdump(len=19): 01 00 00 0f 02 01 00 0f 01 53
75 70 70 6c 69 63 61 6e 74
1176713253.735300: EAPOL: SUPP_BE entering state RECEIVE
1176713253.825487: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713253.825509: RX EAPOL - hexdump(len=10): 02 00 00 06 01 02 00 06 0d 20
1176713253.825521: EAPOL: Received EAP-Packet frame
1176713253.825528: EAPOL: SUPP_BE entering state REQUEST
1176713253.825534: EAPOL: getSuppRsp
1176713253.825542: EAP: EAP entering state RECEIVED
1176713253.825550: EAP: Received EAP-Request id=2 method=13 vendor=0
vendorMethod=0
1176713253.825558: EAP: EAP entering state GET_METHOD
1176713253.825565: EAP: Initialize selected EAP method: vendor 0 method 13
(TLS)
1176713253.959377: TLS: Trusted root certificate(s) loaded
1176713253.998981: OpenSSL: tls_connection_client_cert -
SSL_use_certificate_file (DER) failed error:0D0680A8:asn1 encoding
routines:ASN1_CHECK_TLEN:wrong tag
1176713253.999014: OpenSSL: pending error: error:0D07803A:asn1 encoding
routines:ASN1_ITEM_EX_D2I:nested asn1 error
1176713254.027030: OpenSSL: pending error: error:140C800D:SSL
routines:SSL_use_certificate_file:ASN1 lib
1176713254.041856: OpenSSL: SSL_use_certificate_file (PEM) --> OK
1176713254.068300: OpenSSL: tls_connection_private_key -
SSL_use_PrivateKey_File (DER) failed error:0D094068:asn1 encoding
routines:d2i_ASN1_SET:bad tag
```

```
1176713254.068327: OpenSSL: pending error: error:0D0680A8:asn1 encoding
routines:ASN1_CHECK_TLEN:wrong tag
1176713254.068339: OpenSSL: pending error: error:0D07803A:asn1 encoding
routines:ASN1_ITEM_EX_D2I:nested asn1 error
1176713254.068351: OpenSSL: pending error: error:0D09A00D:asn1 encoding
routines:d2i_PrivateKey:ASN1 lib
1176713254.068363: OpenSSL: pending error: error:140CB00D:SSL
routines:SSL_use_PrivateKey_file:ASN1 lib
1176713254.099484: OpenSSL: SSL_use_PrivateKey_File (PEM) --> OK
1176713254.099516: SSL: Private key loaded successfully
1176713254.099531: CTRL-EVENT-EAP-METHOD EAP vendor 0 method 13 (TLS)
selected
1176713254.099539: EAP: EAP entering state METHOD
1176713254.099547: SSL: Received packet(len=6) - Flags 0x20
1176713254.099554: EAP-TLS: Start
1176713254.099604: SSL: (where=0x10 ret=0x1)
1176713254.099660: SSL: (where=0x1001 ret=0x1)
1176713254.099673: SSL: SSL_connect:before/connect initialization
1176713254.099927: SSL: (where=0x1001 ret=0x1)
1176713254.099936: SSL: SSL_connect:SSLv3 write client hello A
1176713254.099946: SSL: (where=0x1002 ret=0xffffffff)
1176713254.099952: SSL: SSL_connect:error in SSLv3 read server hello A
1176713254.099964: SSL: SSL_connect - want more data
1176713254.099971: SSL: 101 bytes pending from ssl_out
1176713254.099980: SSL: 101 bytes left to be sent out (of total 101 bytes)
1176713254.099989: EAP: method process -> ignore=FALSE methodState=MAY_CONT
decision=FAIL
1176713254.099996: EAP: EAP entering state SEND_RESPONSE
1176713254.100002: EAP: EAP entering state IDLE
1176713254.100009: EAPOL: SUPP_BE entering state RESPONSE
1176713254.100014: EAPOL: txSuppRsp
1176713254.100020: TX EAPOL - hexdump(len=111): 01 00 00 6b 02 02 00 6b 0d 00
16 03 01 00 60 01 00 00 5c 03 01 46 23 38 26 c1 8e 5b 8d e5 42 65 e8 15 e0 09
af 20 83 cf 31 d2 05 2c d6 83 2f f2 40 0e 1d 9a 4e 00 00 34 00 39 00 38 00 35
00 16 00 13 00 0a 00 33 00 32 00 2f 00 66 00 05 00 04 00 63 00 62 00 61 00 15
00 12 00 09 00 65 00 64 00 60 00 14 00 11 00 08 00 06 00 03 02 01 00
1176713254.100084: EAPOL: SUPP_BE entering state RECEIVE
1176713254.122343: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713254.122366: RX EAPOL - hexdump(len=1038): 02 00 04 0a 01 03 04 0a 0d
c0 00 00 08 9c 16 03 01 00 4a 02 00 00 46 03 01 46 23 38 28 8a 19 35 7c aa b6
de 5f fb 95 00 5b a2 eb e1 dd 1d df a3 9a da 0f 43 36 0b 59 07 9a 20 28 fd 42
9d d3 f2 5e dd 82 0f 25 5c 44 04 1e 38 8b 51 04 a7 ae 2d fa 41 75 35 d5 78 d3
61 90 ac 00 35 01 16 03 01 07 ad 0b 00 07 a9 00 07 a6 00 03 25 30 82 03 21 30
82 02 09 a0 03 02 01 02 02 01 03 30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00
30 81 83 31 0b 30 09 06 03 55 04 06 13 02 4e 4f 31 16 30 14 06 03 55 04 08 13
0d 53 6f 72 2d 54 72 6f 6e 64 65 6c 61 67 31 12 30 10 06 03 55 04 07 13 09 54
```

```
72 6f 6e 64 68 65 69 6d 31 0d 30 0b 06 03 55 04 0a 13 04 4e 54 4e 55 31 0f 30
0d 06 03 55 04 0b 13 06 44 69 70 6c 6f 6d 31 0b 30 09 06 03 55 04 03 13 02 43
41 31 1b 30 19 06 09 2a 86 48 86 f7 0d 01 09 01 16 0c 63 61 40 64 69 70 6c 6f
6d 2e 6e 6f 30 1e 17 0d 30 37 30 33 30 31 31 33 34 37 32 39 5a 17 0d 30 38 30
32 32 39 31 33 34 37 32 39 5a 30 6f 31 0b 30 09 06 03 55 04 06 13 02 4e 4f 31
16 30 14 06 03 55 04 08 13 0d 53 6f 72 2d 54 72 6f 6e 64 65 6c 61 67 31 0d 30
0b 06 03 55 04 0a 13 04 4e 54 4e 55 31 0f 30 0d 06 03 55 04 0b 13 06 44 69 70
6c 6f 6d 31 0b 30 09 06 03 55 04 03 13 02 41 53 31 1b 30 19 06 09 2a 86 48 86
f7 0d 01 09 01 16 0c 61 73 40 64 69 70 6c 6f 6d 2e 6e 6f 30 5c 30 0d 06 09 2a
86 48 86 f7 0d 01 01 01 05 00 03 4b 00 30 48 02 41 00 d6 86 b2 26 6f d5 1d 24
bf 52 66 03 29 c3 b7 3a ff 21 b4 c0 67 fc e7 97 b2 85 8b 6e 3a 62 68 fd 37 48
9f 63 b1 2e 88 29 72 62 25 d6 11 00 c9 d5 be 23 0e f2 f2 8c cc 4d 30 12 c6 63
46 be 76 4f 02 03 01 00 01 a3 7b 30 79 30 09 06 03 55 1d 13 04 02 30 00 30 2c
06 09 60 86 48 01 86 f8 42 01 0d 04 1f 16 1d 4f 70 65 6e 53 53 4c 20 47 65 6e
65 72 61 74 65 64 20 43 65 72 74 69 66 69 63 61 74 65 30 1d 06 03 55 1d 0e 04
16 04 14 2f 44 8c 6c 0e 85 41 8d b3 c7 7c 77 b2 52 8b bc 5e c1 88 0e 30 1f 06
03 55 1d 23 04 18 30 16 80 14 65 54 cd 60 84 47 27 81 69 4d 5b 83 49 1a 51 29
92 86 a0 7e 30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00 03 82 01 01 00 91 5e
2f d5 1b 62 d9 98 cb 24 0b 39 ab 04 1b 7e ce 5d 4d 9f 31 d9 a6 4f 89 6c ef f7
7b 1e ca de 81 d2 70 18 fd 37 94 23 0a 8a b6 e0 37 a3 5c 84 1e 75 dd fd f8 aa
cc b1 56 09 d2 51 fa 18 5a ca 78 e5 6e 90 da a4 69 04 ea 5b 1a 6f f9 ff 1b a9
db ef 51 15 59 1d 56 a2 13 c9 58 35 2a fe a9 26 de 2b ae ee a9 37 ea 8b 1b c7
82 69 27 64 71 12 b3 36 9c f5 b7 a4 fe 9d 6d c7 e8 46 5b 8c 48 9d ff 4e 77 54
1f 15 6e 1e 9b 0f 54 31 5a 2f 97 e5 17 f5 60 6f 59 80 04 28 ef 84 18 22 dc a8
52 b9 cf d1 52 f2 63 a7 a1 02 e2 e1 bc f7 05 b8 7a 1e 1c f7 1a e0 44 7e 8c 70
4c 6a e6 97 2e 7f c2 44 94 89 c7 88 23 c4 33 09 43 1a a3 fe b5 de be f3 3d 6b
5e 4e db f3 77 bf e9 5e 95 68 99 5a ee aa db e3 4e e6 72 18 d6 aa c8 a2 26 9f
65 24 e7 ae ca 22 60 be 06 30 45 89 96 03 b9 b5 b6 2f 87 4b 00 04 7b 30 82 04
77 30 82 03 5f a0 03 02 01 02 02 09 00 e3 b8 a4 1f 59 5c 46 a9 30 0d 06 09 2a
86 48 86 f7 0d 01 01 05 05 00 30 81 83 31 0b 30 09 06 03 55 04 06 13 02 4e 4f
31 16 30 14 06 03 55 04 08 13 0d 53 6f 72 2d 54 72 6f 6e 64 65 6c 61 67 31 12
30 10 06 03 55 04 07 13 09 54 72 6f 6e 64 68 65 69 6d 31 0d 30 0b 06 03 55 04
0a 13 04 4e 54 4e 55 31 0f 30 0d 06 03 55 04
1176713254.122643: EAPOL: Received EAP-Packet frame
1176713254.122651: EAPOL: SUPP_BE entering state REQUEST
1176713254.122657: EAPOL: getSuppRsp
1176713254.122663: EAP: EAP entering state RECEIVED
1176713254.122675: EAP: Received EAP-Request id=3 method=13 vendor=0
vendorMethod=0
1176713254.122683: EAP: EAP entering state METHOD
1176713254.122693: SSL: Received packet(len=1034) - Flags 0xc0
1176713254.122700: SSL: TLS Message Length: 2204
1176713254.122707: SSL: Need 1180 bytes more input data
1176713254.122714: SSL: Building ACK
1176713254.122719: EAP: method process -> ignore=FALSE methodState=MAY_CONT
decision=FAIL
```

```
1176713254.122726: EAP: EAP entering state SEND_RESPONSE
1176713254.122732: EAP: EAP entering state IDLE
1176713254.122738: EAPOL: SUPP_BE entering state RESPONSE
1176713254.122743: EAPOL: txSuppRsp
1176713254.122750: TX EAPOL - hexdump(len=10): 01 00 00 06 02 03 00 06 0d 00
1176713254.126911: EAPOL: SUPP_BE entering state RECEIVE
1176713254.144960: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713254.144986: RX EAPOL - hexdump(len=1038): 02 00 04 0a 01 04 04 0a 0d
c0 00 00 08 9c 0b 13 06 44 69 70 6c 6f 6d 31 0b 30 09 06 03 55 04 03 13 02 43
41 31 1b 30 19 06 09 2a 86 48 86 f7 0d 01 09 01 16 0c 63 61 40 64 69 70 6c 6f
6d 2e 6e 6f 30 1e 17 0d 30 37 30 33 30 31 31 33 30 37 34 38 5a 17 0d 31 30 30
32 32 38 31 33 30 37 34 38 5a 30 81 83 31 0b 30 09 06 03 55 04 06 13 02 4e 4f
31 16 30 14 06 03 55 04 08 13 0d 53 6f 72 2d 54 72 6f 6e 64 65 6c 61 67 31 12
30 10 06 03 55 04 07 13 09 54 72 6f 6e 64 68 65 69 6d 31 0d 30 0b 06 03 55 04
0a 13 04 4e 54 4e 55 31 0f 30 0d 06 03 55 04 0b 13 06 44 69 70 6c 6f 6d 31 0b
30 09 06 03 55 04 03 13 02 43 41 31 1b 30 19 06 09 2a 86 48 86 f7 0d 01 09 01
16 0c 63 61 40 64 69 70 6c 6f 6d 2e 6e 6f 30 82 01 22 30 0d 06 09 2a 86 48 86
f7 0d 01 01 01 05 00 03 82 01 0f 00 30 82 01 0a 02 82 01 01 00 bd f7 34 a0 72
95 43 d2 56 f5 ad f7 e5 e2 cb c4 6f 4f b9 53 37 a3 80 e8 d9 f0 8f e3 ab ce 89
e4 b1 39 77 5a 1a 14 53 94 5b fb c2 c9 84 a0 03 5b de 4d 24 54 da 5f ee 7d 50
53 c6 dd 2e 29 ff af ad d8 04 db 8d 18 f0 da 2d 57 ee 58 fc b6 12 dc 83 5d 8e
e8 43 33 44 b1 3a ae f2 5d d2 5a 2a ee b3 2a 90 ec a7 7d 85 a5 05 aa 91 c7 db
77 48 e6 9f 63 c9 fd a5 78 a8 5d 47 0d f4 01 d0 40 8c 7a ba ad 4b b4 dc 3e 12
87 a3 d9 17 56 82 56 26 ec 93 82 50 73 3b cc 97 4d f9 d7 8c 1b 74 4d 8d 4d 81
0c 33 16 02 82 79 7c 41 33 61 f7 18 65 3c b0 59 5c f8 22 ca d0 c3 ec fb f4 0b
f2 90 f4 df b2 ce 10 c7 e9 cf 77 f2 74 a7 2d 1c ed 36 73 fe 30 2d 33 c5 10 10
4d 41 e6 08 53 36 c1 04 e9 3c 49 17 c1 86 d2 bd ff 71 70 df 4f d4 b5 2e 35 d6
c2 4f e7 e9 5d 67 44 4d 4e e7 a1 e0 10 29 d5 21 6d 02 03 01 00 01 a3 81 eb 30
81 e8 30 1d 06 03 55 1d 0e 04 16 04 14 65 54 cd 60 84 47 27 81 69 4d 5b 83 49
1a 51 29 92 86 a0 7e 30 81 b8 06 03 55 1d 23 04 81 b0 30 81 ad 80 14 65 54 cd
60 84 47 27 81 69 4d 5b 83 49 1a 51 29 92 86 a0 7e a1 81 89 a4 81 86 30 81 83
31 0b 30 09 06 03 55 04 06 13 02 4e 4f 31 16 30 14 06 03 55 04 08 13 0d 53 6f
72 2d 54 72 6f 6e 64 65 6c 61 67 31 12 30 10 06 03 55 04 07 13 09 54 72 6f 6e
64 68 65 69 6d 31 0d 30 0b 06 03 55 04 0a 13 04 4e 54 4e 55 31 0f 30 0d 06 03
55 04 0b 13 06 44 69 70 6c 6f 6d 31 0b 30 09 06 03 55 04 03 13 02 43 41 31 1b
30 19 06 09 2a 86 48 86 f7 0d 01 09 01 16 0c 63 61 40 64 69 70 6c 6f 6d 2e 6e
6f 82 09 00 e3 b8 a4 1f 59 5c 46 a9 30 0c 06 03 55 1d 13 04 05 30 03 01 01 ff
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00 03 82 01 01 00 01 50 49 25 d5 11
09 f5 b6 5e ad 28 99 f1 5a d1 b0 f9 f3 a8 e1 cd 97 3a 7b 3a 87 da 43 1a d1 1c
a5 69 b1 22 00 5a e0 ab ed 60 b8 63 e9 7b c0 d1 53 c3 81 96 8a bd 5a 60 c4 22
ec 7a ce 0b d5 24 0c e2 ab b9 d5 b7 40 27 61 34 cf 95 6b 86 8b a4 65 61 43 00
96 87 75 be 10 b4 d7 2e 42 36 57 a9 91 ee bc c1 1e 6c d4 47 69 8d 94 4d fa 4c
fc 62 4c 38 d7 de fb a4 ec 61 0a 5e 38 7c cb bb 13 38 a2 ee d0 3a 2c 8c 07 0c
85 4c d8 13 35 45 d0 a7 a9 6e 8d b7 a3 2e dd 7a 1b 87 48 48 50 49 71 4e 27 b1
b1 a0 12 f7 b9 65 ad 4b 7b 0c b0 5f fb e9 f6 bb e3 05 17 d0 3e d5 a7 4e b3 00
ec 6f 68 88 23 ae 65 23 95 6f e0 b7 9e c9 9e eb 99 a2 90 0e 72 1c 88 70 94 34
```

```
0a 1b cd d0 19 4b 43 34 9c 4e 73 47 8d 2c eb a5 a5 b5 49 e7 32 30 e7 fc 6b dc
cc 03 c0 60 6b 8f b1 33 f3 20 2c 4f 87 0d 12
1176713254.145260: EAPOL: Received EAP-Packet frame
1176713254.145269: EAPOL: SUPP_BE entering state REQUEST
1176713254.145274: EAPOL: getSuppRsp
1176713254.145281: EAP: EAP entering state RECEIVED
1176713254.145293: EAP: Received EAP-Request id=4 method=13 vendor=0
vendorMethod=0
1176713254.145300: EAP: EAP entering state METHOD
1176713254.145311: SSL: Received packet(len=1034) - Flags 0xc0
1176713254.145317: SSL: TLS Message Length: 2204
1176713254.145326: SSL: Need 156 bytes more input data
1176713254.145349: SSL: Building ACK
1176713254.145354: EAP: method process -> ignore=FALSE methodState=MAY_CONT
decision=FAIL
1176713254.145361: EAP: EAP entering state SEND_RESPONSE
1176713254.145367: EAP: EAP entering state IDLE
1176713254.145373: EAPOL: SUPP_BE entering state RESPONSE
1176713254.145379: EAPOL: txSuppRsp
1176713254.145385: TX EAPOL - hexdump(len=10): 01 00 00 06 02 04 00 06 0d 00
1176713254.148493: EAPOL: SUPP_BE entering state RECEIVE
1176713254.179297: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713254.179323: RX EAPOL - hexdump(len=170): 02 00 00 a6 01 05 00 a6 0d 80
00 00 08 9c 8c 16 03 01 00 96 0d 00 00 8e 03 01 02 40 00 88 00 86 30 81 83 31
0b 30 09 06 03 55 04 06 13 02 4e 4f 31 16 30 14 06 03 55 04 08 13 0d 53 6f 72
2d 54 72 6f 6e 64 65 6c 61 67 31 12 30 10 06 03 55 04 07 13 09 54 72 6f 6e 64
68 65 69 6d 31 0d 30 0b 06 03 55 04 0a 13 04 4e 54 4e 55 31 0f 30 0d 06 03 55
04 0b 13 06 44 69 70 6c 6f 6d 31 0b 30 09 06 03 55 04 03 13 02 43 41 31 1b 30
19 06 09 2a 86 48 86 f7 0d 01 09 01 16 0c 63 61 40 64 69 70 6c 6f 6d 2e 6e 6f
0e 00 00 00
1176713254.179376: EAPOL: Received EAP-Packet frame
1176713254.179385: EAPOL: SUPP_BE entering state REQUEST
1176713254.179391: EAPOL: getSuppRsp
1176713254.179397: EAP: EAP entering state RECEIVED
1176713254.179406: EAP: Received EAP-Request id=5 method=13 vendor=0
vendorMethod=0
1176713254.179413: EAP: EAP entering state METHOD
1176713254.179424: SSL: Received packet(len=166) - Flags 0x80
1176713254.179430: SSL: TLS Message Length: 2204
1176713254.179513: SSL: (where=0x1001 ret=0x1)
1176713254.179521: SSL: SSL_connect:SSLv3 read server hello A
1176713254.189822: TLS: tls_verify_cb - preverify_ok=1 err=0 (ok) depth=1
buf='/C=NO/ST=Sor-
Trondelag/L=Trondheim/O=NTNU/OU=Diplom/CN=CA/emailAddress=ca@diplom.no'
1176713254.190368: TLS: tls_verify_cb - preverify_ok=1 err=0 (ok) depth=0
buf='/C=NO/ST=Sor-Trondelag/O=NTNU/OU=Diplom/CN=AS/emailAddress=as@diplom.no'
```

```
1176713254.190397: SSL: (where=0x1001 ret=0x1)
1176713254.190405: SSL: SSL_connect:SSLv3 read server certificate A
1176713254.190459: SSL: (where=0x1001 ret=0x1)
1176713254.190466: SSL: SSL_connect:SSLv3 read server certificate request A
1176713254.190474: SSL: (where=0x1001 ret=0x1)
1176713254.190480: SSL: SSL_connect:SSLv3 read server done A
1176713254.190586: SSL: (where=0x1001 ret=0x1)
1176713254.190593: SSL: SSL_connect:SSLv3 write client certificate A
1176713254.190754: SSL: (where=0x1001 ret=0x1)
1176713254.190761: SSL: SSL_connect:SSLv3 write client key exchange A
1176713254.225293: SSL: (where=0x1001 ret=0x1)
1176713254.225361: SSL: SSL_connect:SSLv3 write certificate verify A
1176713254.275022: SSL: (where=0x1001 ret=0x1)
1176713254.275122: SSL: SSL_connect:SSLv3 write change cipher spec A
1176713254.275274: SSL: (where=0x1001 ret=0x1)
1176713254.275316: SSL: SSL_connect:SSLv3 write finished A
1176713254.275364: SSL: (where=0x1001 ret=0x1)
1176713254.275404: SSL: SSL_connect:SSLv3 flush data
1176713254.275446: SSL: (where=0x1002 ret=0xffffffff)
1176713254.275485: SSL: SSL_connect:error in SSLv3 read finished A
1176713254.275528: SSL: SSL_connect - want more data
1176713254.275568: SSL: 2587 bytes pending from ssl_out
1176713254.275617: SSL: 2587 bytes left to be sent out (of total 2587 bytes)
1176713254.275656: SSL: sending 1398 bytes, more fragments will follow
1176713254.275697: EAP: method process -> ignore=FALSE methodState=MAY_CONT
decision=FAIL
1176713254.275740: EAP: EAP entering state SEND_RESPONSE
1176713254.275780: EAP: EAP entering state IDLE
1176713254.275821: EAPOL: SUPP_BE entering state RESPONSE
1176713254.275859: EAPOL: txSuppRsp
1176713254.275903: TX EAPOL - hexdump(len=1412): 01 00 05 80 02 05 05 80 0d
c0 00 00 0a 1b 16 03 01 08 85 0b 00 08 81 00 08 7e 00 03 fd 30 82 03 f9 30 82
02 e1 a0 03 02 01 02 02 01 01 30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00 30
81 83 31 0b 30 09 06 03 55 04 06 13 02 4e 4f 31 16 30 14 06 03 55 04 08 13 0d
53 6f 72 2d 54 72 6f 6e 64 65 6c 61 67 31 12 30 10 06 03 55 04 07 13 09 54 72
6f 6e 64 68 65 69 6d 31 0d 30 0b 06 03 55 04 0a 13 04 4e 54 4e 55 31 0f 30 0d
06 03 55 04 0b 13 06 44 69 70 6c 6f 6d 31 0b 30 09 06 03 55 04 03 13 02 43 41
31 1b 30 19 06 09 2a 86 48 86 f7 0d 01 09 01 16 0c 63 61 40 64 69 70 6c 6f 6d
2e 6e 6f 30 1e 17 0d 30 37 30 33 30 31 31 33 31 35 32 30 5a 17 0d 30 38 30 32
32 39 31 33 31 35 32 30 5a 30 7f 31 0b 30 09 06 03 55 04 06 13 02 4e 4f 31 16
30 14 06 03 55 04 08 13 0d 53 6f 72 2d 54 72 6f 6e 64 65 6c 61 67 31 0d 30 0b
06 03 55 04 0a 13 04 4e 54 4e 55 31 0f 30 0d 06 03 55 04 0b 13 06 44 69 70 6c
6f 6d 31 13 30 11 06 03 55 04 03 13 0a 53 75 70 70 6c 69 63 61 6e 74 31 23 30
21 06 09 2a 86 48 86 f7 0d 01 09 01 16 14 73 75 70 70 6c 69 63 61 6e 74 40 64
69 70 6c 6f 6d 2e 6e 6f 30 82 01 22 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05
00 03 82 01 0f 00 30 82 01 0a 02 82 01 01 00 a6 18 5a 7b f4 48 5c e7 58 ea e1
```

```
3c 5e 4d 1b d3 ea bb c8 d4 38 ee e4 fc b4 a2 23 79 46 2d 39 c4 9a 47 f5 f4 a6
21 dc 47 e9 2f 5f 40 b6 b8 32 3c 29 0e ed 03 a6 33 81 00 8f 6e 31 8d a2 24 34
3b d2 54 3c e5 62 4c 80 17 52 f4 c6 22 7d 01 99 c9 b1 18 08 24 8e 89 e2 82 77
d6 99 f2 b1 2f 0e cd 95 17 47 ad 11 cd 76 54 cf 82 90 57 4c 91 a3 13 23 b5 99
9c 80 0f 9b e6 69 ac cb d7 5d 12 3a 9b 2d fb 03 bc 98 fa c4 f8 49 a2 dc 92 b1
97 08 da 59 a8 a6 3d c8 f6 d5 1c 40 48 c4 5c f6 e3 23 1a 98 0f 6e 17 b5 43 5a
82 81 46 70 c6 18 be e2 88 04 32 72 1e 10 3b 06 f7 c3 47 fe ae 2a 3d bd 4b 37
9e f1 d0 7f 2d 02 86 fa dd 79 a9 d0 4b 7d 65 83 f8 1f 72 dc e4 ed 61 c8 b2 16
56 00 64 98 10 d4 a2 fa 03 14 1e 2f 2a 9c b2 7b 95 30 8e 3a a3 1d 77 99 3c 24
54 90 23 1d 6d ca a6 0e b3 01 09 02 03 01 00 01 a3 7b 30 79 30 09 06 03 55 1d
13 04 02 30 00 30 2c 06 09 60 86 48 01 86 f8 42 01 0d 04 1f 16 1d 4f 70 65 6e
53 53 4c 20 47 65 6e 65 72 61 74 65 64 20 43 65 72 74 69 66 69 63 61 74 65 30
1d 06 03 55 1d 0e 04 16 04 14 a2 fb 7f 6a b0 fa 2d 1a bc 4a e9 f8 db 4e be a0
7f 90 c0 8a 30 1f 06 03 55 1d 23 04 18 30 16 80 14 65 54 cd 60 84 47 27 81 69
4d 5b 83 49 1a 51 29 92 86 a0 7e 30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00
03 82 01 01 00 b9 36 63 a0 41 81 98 ea ec 0d 59 e4 7a ac f5 b9 e3 15 05 75 b4
43 68 53 52 c0 2e 0a 1f 36 18 7f 9d 31 3c 37 a1 6d d0 79 14 8b 33 c7 40 68 5d
43 01 ca 59 cf ba 57 9a 30 14 05 5e 5c 6d 82 b2 36 53 95 79 94 47 4e 29 51 e0
86 53 03 1a 20 b6 98 17 0c 93 9f 8d 09 b0 29 27 3a 99 1c 98 1d bd 9a 4d ca e7
40 2d 58 1b 94 39 66 0e eb 98 0d de cc 00 a4 41 5b ce 9a f6 bb 53 2e ab ce 93
81 e6 10 8e ee 50 ec cd f0 1c e2 8b 2a 2f cd 52 e1 58 83 8c 15 fd 37 74 2f d3
08 cd 8a 43 3e 34 75 34 d7 b7 c4 21 59 d6 5d ec ed 19 5f 9f ba 4f e3 7f ff c5
31 db 30 de b0 2b 86 11 8c a7 d9 10 03 5f a9 fc 91 f1 70 3f b3 8a 85 c5 da 48
a3 fe 37 cf e1 db 63 12 35 5f e9 54 5d c4 bd 26 0a 91 54 91 3e de f9 03 4a 25
2e 92 1b 03 20 ef ad a5 2a 52 06 10 72 69 9c 63 f6 fc 7d 88 b5 86 26 91 11 6e
3d 00 04 7b 30 82 04 77 30 82 03 5f a0 03 02 01 02 02 09 00 e3 b8 a4 1f 59 5c
46 a9 30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00 30 81 83 31 0b 30 09 06 03
55 04 06 13 02 4e 4f 31 16 30 14 06 03 55 04 08 13 0d 53 6f 72 2d 54 72 6f 6e
64 65 6c 61 67 31 12 30 10 06 03 55 04 07 13 09 54 72 6f 6e 64 68 65 69 6d 31
0d 30 0b 06 03 55 04 0a 13 04 4e 54 4e 55 31 0f 30 0d 06 03 55 04 0b 13 06 44
69 70 6c 6f 6d 31 0b 30 09 06 03 55 04 03 13 02 43 41 31 1b 30 19 06 09 2a 86
48 86 f7 0d 01 09 01 16 0c 63 61 40 64 69 70 6c 6f 6d 2e 6e 6f 30 1e 17 0d 30
37 30 33 30 31 31 33 30 37 34 38 5a 17 0d 31 30 30 32 32 38 31 33 30 37 34 38
5a 30 81 83 31 0b 30 09 06 03 55 04 06 13 02 4e 4f 31 16 30 14 06 03 55 04 08
13 0d 53 6f 72 2d 54 72 6f 6e 64 65 6c 61 67 31 12 30 10 06 03 55 04 07 13 09
54 72 6f 6e 64 68 65 69 6d 31 0d 30 0b 06 03 55 04 0a 13 04 4e 54 4e 55 31 0f
30 0d 06 03 55 04 0b 13 06 44 69 70 6c 6f 6d 31 0b 30 09 06 03 55 04 03 13 02
43 41 31 1b 30 19 06 09 2a 86 48 86 f7 0d 01 09 01 16 0c 63 61 40 64 69 70 6c
6f 6d 2e 6e 6f 30 82 01 22 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03
1176713254.276460: EAPOL: SUPP_BE entering state RECEIVE
1176713254.279252: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713254.279275: RX EAPOL - hexdump(len=10): 02 00 00 06 01 06 00 06 0d 00
1176713254.279288: EAPOL: Received EAP-Packet frame
1176713254.279295: EAPOL: SUPP_BE entering state REQUEST
1176713254.279301: EAPOL: getSuppRsp
1176713254.279308: EAP: EAP entering state RECEIVED
```

```
1176713254.279317: EAP: Received EAP-Request id=6 method=13 vendor=0
vendorMethod=0
1176713254.279325: EAP: EAP entering state METHOD
1176713254.279335: SSL: Received packet(len=6) - Flags 0x00
1176713254.279342: SSL: 1189 bytes left to be sent out (of total 2587 bytes)
1176713254.279351: EAP: method process -> ignore=FALSE methodState=MAY_CONT
decision=FAIL
1176713254.279358: EAP: EAP entering state SEND_RESPONSE
1176713254.279366: EAP: EAP entering state IDLE
1176713254.279371: EAPOL: SUPP_BE entering state RESPONSE
1176713254.279377: EAPOL: txSuppRsp
1176713254.279384: TX EAPOL - hexdump(len=1199): 01 00 04 ab 02 06 04 ab 0d
00 82 01 0f 00 30 82 01 0a 02 82 01 01 00 bd f7 34 a0 72 95 43 d2 56 f5 ad f7
e5 e2 cb c4 6f 4f b9 53 37 a3 80 e8 d9 f0 8f e3 ab ce 89 e4 b1 39 77 5a 1a 14
53 94 5b fb c2 c9 84 a0 03 5b de 4d 24 54 da 5f ee 7d 50 53 c6 dd 2e 29 ff af
ad d8 04 db 8d 18 f0 da 2d 57 ee 58 fc b6 12 dc 83 5d 8e e8 43 33 44 b1 3a ae
f2 5d d2 5a 2a ee b3 2a 90 ec a7 7d 85 a5 05 aa 91 c7 db 77 48 e6 9f 63 c9 fd
a5 78 a8 5d 47 0d f4 01 d0 40 8c 7a ba ad 4b b4 dc 3e 12 87 a3 d9 17 56 82 56
26 ec 93 82 50 73 3b cc 97 4d f9 d7 8c 1b 74 4d 8d 4d 81 0c 33 16 02 82 79 7c
41 33 61 f7 18 65 3c b0 59 5c f8 22 ca d0 c3 ec fb f4 0b f2 90 f4 df b2 ce 10
c7 e9 cf 77 f2 74 a7 2d 1c ed 36 73 fe 30 2d 33 c5 10 10 4d 41 e6 08 53 36 c1
04 e9 3c 49 17 c1 86 d2 bd ff 71 70 df 4f d4 b5 2e 35 d6 c2 4f e7 e9 5d 67 44
4d 4e e7 a1 e0 10 29 d5 21 6d 02 03 01 00 01 a3 81 eb 30 81 e8 30 1d 06 03 55
1d 0e 04 16 04 14 65 54 cd 60 84 47 27 81 69 4d 5b 83 49 1a 51 29 92 86 a0 7e
30 81 b8 06 03 55 1d 23 04 81 b0 30 81 ad 80 14 65 54 cd 60 84 47 27 81 69 4d
5b 83 49 1a 51 29 92 86 a0 7e a1 81 89 a4 81 86 30 81 83 31 0b 30 09 06 03 55
04 06 13 02 4e 4f 31 16 30 14 06 03 55 04 08 13 0d 53 6f 72 2d 54 72 6f 6e 64
65 6c 61 67 31 12 30 10 06 03 55 04 07 13 09 54 72 6f 6e 64 68 65 69 6d 31 0d
30 0b 06 03 55 04 0a 13 04 4e 54 4e 55 31 0f 30 0d 06 03 55 04 0b 13 06 44 69
70 6c 6f 6d 31 0b 30 09 06 03 55 04 03 13 02 43 41 31 1b 30 19 06 09 2a 86 48
86 f7 0d 01 09 01 16 0c 63 61 40 64 69 70 6c 6f 6d 2e 6e 6f 82 09 00 e3 b8 a4
1f 59 5c 46 a9 30 0c 06 03 55 1d 13 04 05 30 03 01 01 ff 30 0d 06 09 2a 86 48
86 f7 0d 01 01 05 05 00 03 82 01 01 00 01 50 49 25 d5 11 09 f5 b6 5e ad 28 99
f1 5a d1 b0 f9 f3 a8 e1 cd 97 3a 7b 3a 87 da 43 1a d1 1c a5 69 b1 22 00 5a e0
ab ed 60 b8 63 e9 7b c0 d1 53 c3 81 96 8a bd 5a 60 c4 22 ec 7a ce 0b d5 24 0c
e2 ab b9 d5 b7 40 27 61 34 cf 95 6b 86 8b a4 65 61 43 00 96 87 75 be 10 b4 d7
2e 42 36 57 a9 91 ee bc c1 1e 6c d4 47 69 8d 94 4d fa 4c fc 62 4c 38 d7 de fb
a4 ec 61 0a 5e 38 7c cb bb 13 38 a2 ee d0 3a 2c 8c 07 0c 85 4c d8 13 35 45 d0
a7 a9 6e 8d b7 a3 2e dd 7a 1b 87 48 48 50 49 71 4e 27 b1 b1 a0 12 f7 b9 65 ad
4b 7b 0c b0 5f fb e9 f6 bb e3 05 17 d0 3e d5 a7 4e b3 00 ec 6f 68 88 23 ae 65
23 95 6f e0 b7 9e c9 9e eb 99 a2 90 0e 72 1c 88 70 94 34 0a 1b cd d0 19 4b 43
34 9c 4e 73 47 8d 2c eb a5 a5 b5 49 e7 32 30 e7 fc 6b dc cc 03 c0 60 6b 8f b1
33 f3 20 2c 4f 87 0d 12 8c 16 03 01 00 46 10 00 00 42 00 40 6e e7 48 d7 1f 8a
ae 56 90 92 e0 31 d1 3a 9b c0 01 1b df 7b 01 ba 41 4b 31 f6 b3 b4 02 a3 a4 90
e3 cc 9a b8 99 89 39 05 28 de 8a 5e 55 76 10 f1 d3 3b f4 50 2b 84 f5 af 16 c1
7f 6a a8 65 6b ff 16 03 01 01 06 0f 00 01 02 01 00 4a 45 01 c9 0c 06 b5 de 58
```

```
c2 ee 96 93 7f 35 a0 5c ca 8d ca 4e 84 9b 02 1d 55 0c 95 dd 93 3f ca f2 97 e2
14 02 db 94 55 65 b5 e8 00 66 2c f9 ba d4 83 83 5e e6 94 98 84 a2 47 1f fe 9b
4d 72 20 0f 5e 78 10 c4 a0 23 b5 31 3a cf a4 c2 4f 3b e4 cf 8e 44 df 9d ec 8a
60 36 05 d4 a0 ba 60 2a 11 76 c5 07 2d 5f e5 7c 59 d2 3d be da 77 24 1f 8b 84
c7 1b 84 94 b6 76 89 3c ce ad cc 75 db 1e 21 2f ee e1 e4 3c 8b 75 3a 4c 79 07
e9 ad 42 13 9f d1 b4 12 2d e2 0a 3a 8d e2 cd f8 f0 94 eb 7a cc a6 00 e5 fd 0f
da 4c 1b 6f 3c 42 0e 51 a6 ba 0d a8 88 65 5a 22 90 79 06 a8 7f f1 59 6f d9 c9
50 4b 55 0a a5 1f 59 a0 f3 69 38 8d ac 45 0f 0c a0 dc 66 5c d3 09 d2 b7 24 07
82 d0 96 24 01 e6 cb b3 67 11 f0 0c 4a 75 43 af e4 a7 31 89 6e 4d e9 bd 45 4a
7f a4 3c 27 46 78 d4 52 b8 82 81 c0 40 14 03 01 00 01 01 16 03 01 00 30 47 f7
1b de 88 78 00 fb 0b 96 08 18 ce 57 02 48 86 00 1c 78 07 13 bb 77 02 8c ba b2
b0 cc 9c c1 fc d7 ec 9f d4 a9 0a 5c b4 78 49 be ca 93 d4 c0
1176713254.279835: EAPOL: SUPP_BE entering state RECEIVE
1176713254.353874: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713254.353897: RX EAPOL - hexdump(len=73): 02 00 00 45 01 07 00 45 0d 80
00 00 00 3b 14 03 01 00 01 01 16 03 01 00 30 27 12 fb 4c b2 72 b4 ec b7 59 f9
25 54 5d a2 8f 6e 96 a4 9f 25 0d 0b c0 ed e5 69 be ff c7 c2 a3 93 b4 d2 55 68
2c 68 27 b9 aa e4 10 dd b8 2c a6
1176713254.353926: EAPOL: Received EAP-Packet frame
1176713254.353934: EAPOL: SUPP_BE entering state REQUEST
1176713254.353940: EAPOL: getSuppRsp
1176713254.353946: EAP: EAP entering state RECEIVED
1176713254.353955: EAP: Received EAP-Request id=7 method=13 vendor=0
vendorMethod=0
1176713254.353962: EAP: EAP entering state METHOD
1176713254.353973: SSL: Received packet(len=69) - Flags 0x80
1176713254.353979: SSL: TLS Message Length: 59
1176713254.354200: SSL: (where=0x1001 ret=0x1)
1176713254.354212: SSL: SSL_connect:SSLv3 read finished A
1176713254.354243: SSL: (where=0x20 ret=0x1)
1176713254.354250: SSL: (where=0x1002 ret=0x1)
1176713254.354255: SSL: 0 bytes pending from ssl_out
1176713254.354278: OpenSSL: tls_connection_handshake - Failed to read
possible Application Data error:00000000:lib(0):func(0):reason(0)
1176713254.354287: SSL: No data to be sent out
1176713254.354293: EAP-TLS: Done
1176713254.354343: EAP-TLS: Derived key - hexdump(len=64): 24 b6 4f 31 63 86
d5 b9 81 7a 42 4a 66 71 25 fe 2d e0 ae 75 a1 f0 17 90 8a a5 1a 23 f0 92 df b3
0a c6 c5 99 f2 a2 3a 23 80 2b 88 68 65 fd f2 cf fc 50 c8 98 54 1a 4e 0d 33 d0
9a 2e 90 f7 ad b0
1176713254.354366: SSL: Building ACK
1176713254.354372: EAP: method process -> ignore=FALSE methodState=DONE
decision=UNCOND_SUCC
1176713254.354379: EAP: EAP entering state SEND_RESPONSE
1176713254.354386: EAP: EAP entering state IDLE
1176713254.354392: EAPOL: SUPP_BE entering state RESPONSE
```

```
1176713254.354397: EAPOL: txSuppRsp
1176713254.354403: TX EAPOL - hexdump(len=10): 01 00 00 06 02 07 00 06 0d 00
1176713254.354433: EAPOL: SUPP_BE entering state RECEIVE
1176713254.357362: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713254.357374: RX EAPOL - hexdump(len=8): 02 00 00 04 03 07 00 04
1176713254.357384: EAPOL: Received EAP-Packet frame
1176713254.357391: EAPOL: SUPP_BE entering state REQUEST
1176713254.357397: EAPOL: getSuppRsp
1176713254.357402: EAP: EAP entering state RECEIVED
1176713254.357409: EAP: Received EAP-Success
1176713254.357415: EAP: EAP entering state SUCCESS
1176713254.357424: CTRL-EVENT-EAP-SUCCESS EAP authentication completed
successfully
1176713254.357431: EAPOL: SUPP_BE entering state RECEIVE
1176713254.357437: EAPOL: SUPP_BE entering state SUCCESS
1176713254.357442: EAPOL: SUPP_BE entering state IDLE
1176713254.357534: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713254.357542: RX EAPOL - hexdump(len=121): 02 03 00 75 02 00 8a 00 10 00
00 00 00 00 00 00 01 de e0 a6 9a 3c 77 d6 44 e7 e6 89 1d b6 ce ad 72 7d 95 c9
b6 52 35 e7 69 0d 75 58 37 18 47 0d 3e 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 16 dd 14 00 0f ac 04 4e 22 fd 43 4c 32 ae 2b eb
be 33 df 2d 5b 14 68
1176713254.357580: EAPOL: Ignoring WPA EAPOL-Key frame in EAPOL state
machines
1176713254.357594: IEEE 802.1X RX: version=2 type=3 length=117
1176713254.357602:   EAPOL-Key type=2
1176713254.357608:   key_info 0x8a (ver=2 keyidx=0 rsvd=0 Pairwise Ack)
1176713254.357616:   key_length=16 key_data_length=22
1176713254.357621:   replay_counter - hexdump(len=8): 00 00 00 00 00 00 00 01
1176713254.357630:   key_nonce - hexdump(len=32): de e0 a6 9a 3c 77 d6 44 e7
e6 89 1d b6 ce ad 72 7d 95 c9 b6 52 35 e7 69 0d 75 58 37 18 47 0d 3e
1176713254.357643:   key_iv - hexdump(len=16): 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
1176713254.357653:   key_rsc - hexdump(len=8): 00 00 00 00 00 00 00 00
1176713254.357661:   key_id (reserved) - hexdump(len=8): 00 00 00 00 00 00 00
00
1176713254.357669:   key_mic - hexdump(len=16): 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
1176713254.357679: WPA: RX EAPOL-Key - hexdump(len=121): 02 03 00 75 02 00 8a
00 10 00 00 00 00 00 00 00 01 de e0 a6 9a 3c 77 d6 44 e7 e6 89 1d b6 ce ad 72
7d 95 c9 b6 52 35 e7 69 0d 75 58 37 18 47 0d 3e 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 16 dd 14 00 0f ac 04 4e 22 fd 43 4c 32
ae 2b eb be 33 df 2d 5b 14 68
1176713254.357727: State: ASSOCIATED -> 4WAY_HANDSHAKE
```

```
1176713254.357734: WPA: RX message 1 of 4-Way Handshake from
00:0f:cb:b5:0e:d2 (ver=2)
1176713254.357742: RSN: msg 1/4 key data - hexdump(len=22): dd 14 00 0f ac 04
4e 22 fd 43 4c 32 ae 2b eb be 33 df 2d 5b 14 68
1176713254.357754: RSN: PMKID from Authenticator - hexdump(len=16): 4e 22 fd
43 4c 32 ae 2b eb be 33 df 2d 5b 14 68
1176713254.357764: RSN: no matching PMKID found
1176713254.357770: WPA: PMK from EAPOL state machines - hexdump(len=32): 24
b6 4f 31 63 86 d5 b9 81 7a 42 4a 66 71 25 fe 2d e0 ae 75 a1 f0 17 90 8a a5 1a
23 f0 92 df b3
1176713254.357794: RSN: added PMKSA cache entry for 00:0f:cb:b5:0e:d2
1176713254.357802: RSN: the new PMK matches with the PMKID
1176713254.359290: WPA: Renewed SNonce - hexdump(len=32): be a7 92 ab a3 92
ec d4 b0 21 21 34 db 82 fc 59 84 62 59 10 33 31 dc 5e 78 a0 e9 f5 1e b6 4f a2
1176713254.359316: WPA: PMK - hexdump(len=32): 24 b6 4f 31 63 86 d5 b9 81 7a
42 4a 66 71 25 fe 2d e0 ae 75 a1 f0 17 90 8a a5 1a 23 f0 92 df b3
1176713254.359331: WPA: PTK - hexdump(len=64): d9 89 b8 4b bf bd 89 93 e9 ce
0b 43 20 b5 05 d6 cc 12 42 00 a5 16 62 fd f3 2b 67 59 d5 32 54 74 c7 1c 12 7d
a0 62 6c 5d 26 72 3b c8 8e 6c 1f bf 39 f0 7d 52 4b 21 6b d4 5d e1 35 9c 77 87
a1 2e
1176713254.359354: WPA: WPA IE for msg 2/4 - hexdump(len=22): 30 14 01 00 00
0f ac 04 01 00 00 0f ac 04 01 00 00 0f ac 01 00 00
1176713254.359368: WPA: Sending EAPOL-Key 2/4
1176713254.359377: WPA: TX EAPOL-Key - hexdump(len=121): 01 03 00 75 02 01 0a
00 00 00 00 00 00 00 00 00 01 be a7 92 ab a3 92 ec d4 b0 21 21 34 db 82 fc 59
84 62 59 10 33 31 dc 5e 78 a0 e9 f5 1e b6 4f a2 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 68 19 88 60
c6 b5 82 fb 69 63 4f df 1c ed d2 8b 00 16 30 14 01 00 00 0f ac 04 01 00 00 0f
ac 04 01 00 00 0f ac 01 00 00
1176713254.361103: RX EAPOL from 00:0f:cb:b5:0e:d2
1176713254.361146: RX EAPOL - hexdump(len=155): 02 03 00 97 02 13 ca 00 10 00
00 00 00 00 00 00 02 de e0 a6 9a 3c 77 d6 44 e7 e6 89 1d b6 ce ad 72 7d 95 c9
b6 52 35 e7 69 0d 75 58 37 18 47 0d 3e 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2d d1 b8 eb 64 41 6b
c1 ad 8c 2b 89 a5 ef 83 bd 00 38 de 31 22 62 6e de 0d cf 74 37 13 94 fd 8d 73
17 e6 5f d9 17 9e 2d 97 a2 7a e2 f4 3d aa 20 db f3 22 61 3b e1 d4 3d f9 5b b7
3e 9f cb 3b 25 30 0d fc 39 0c 48 7f 3a 19 5e
1176713254.361224: EAPOL: Ignoring WPA EAPOL-Key frame in EAPOL state
machines
1176713254.361263: IEEE 802.1X RX: version=2 type=3 length=151
1176713254.361302:   EAPOL-Key type=2
1176713254.361341:   key_info 0x13ca (ver=2 keyidx=0 rsvd=0 Pairwise Install
Ack MIC Secure Encr)
1176713254.361382:   key_length=16 key_data_length=56
1176713254.361421:   replay_counter - hexdump(len=8): 00 00 00 00 00 00 00 02
```

```
1176713254.361462:   key_nonce - hexdump(len=32): de e0 a6 9a 3c 77 d6 44 e7
e6 89 1d b6 ce ad 72 7d 95 c9 b6 52 35 e7 69 0d 75 58 37 18 47 0d 3e
1176713254.361508:   key_iv - hexdump(len=16): 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00
1176713254.361552:   key_rsc - hexdump(len=8): 30 00 00 00 00 00 00 00
1176713254.361593:   key_id (reserved) - hexdump(len=8): 00 00 00 00 00 00 00
00
1176713254.361634:   key_mic - hexdump(len=16): 2d d1 b8 eb 64 41 6b c1 ad 8c
2b 89 a5 ef 83 bd
1176713254.361678: WPA: RX EAPOL-Key - hexdump(len=155): 02 03 00 97 02 13 ca
00 10 00 00 00 00 00 00 00 02 de e0 a6 9a 3c 77 d6 44 e7 e6 89 1d b6 ce ad 72
7d 95 c9 b6 52 35 e7 69 0d 75 58 37 18 47 0d 3e 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2d d1 b8 eb
64 41 6b c1 ad 8c 2b 89 a5 ef 83 bd 00 38 de 31 22 62 6e de 0d cf 74 37 13 94
fd 8d 73 17 e6 5f d9 17 9e 2d 97 a2 7a e2 f4 3d aa 20 db f3 22 61 3b e1 d4 3d
f9 5b b7 3e 9f cb 3b 25 30 0d fc 39 0c 48 7f 3a 19 5e
1176713254.361761: RSN: encrypted key data - hexdump(len=56): de 31 22 62 6e
de 0d cf 74 37 13 94 fd 8d 73 17 e6 5f d9 17 9e 2d 97 a2 7a e2 f4 3d aa 20 db
f3 22 61 3b e1 d4 3d f9 5b b7 3e 9f cb 3b 25 30 0d fc 39 0c 48 7f 3a 19 5e
1176713254.361832: WPA: decrypted EAPOL-Key key data - hexdump(len=48): 30 14
01 00 00 0f ac 04 01 00 00 0f ac 04 01 00 00 0f ac 01 00 00 dd 16 00 0f ac 01
01 00 c6 76 28 e9 fe 69 a5 89 99 35 f7 f8 d1 dc fa a1 dd 00
1176713254.361885: State: 4WAY_HANDSHAKE -> 4WAY_HANDSHAKE
1176713254.361924: WPA: RX message 3 of 4-Way Handshake from
00:0f:cb:b5:0e:d2 (ver=2)
1176713254.361965: WPA: IE KeyData - hexdump(len=48): 30 14 01 00 00 0f ac 04
01 00 00 0f ac 04 01 00 00 0f ac 01 00 00 dd 16 00 0f ac 01 01 00 c6 76 28 e9
fe 69 a5 89 99 35 f7 f8 d1 dc fa a1 dd 00
1176713254.362018: WPA: Sending EAPOL-Key 4/4
1176713254.362059: WPA: TX EAPOL-Key - hexdump(len=99): 01 03 00 5f 02 03 0a
00 00 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 95 eb 55 42
da 42 47 38 ba 40 1e 3e 36 6b ad 32 00 00
1176713254.362132: WPA: Installing PTK to the driver.
1176713254.362186: wpa_driver_madwifi_set_key: alg=CCMP key_idx=0 set_tx=1
seq_len=6 key_len=16
1176713254.395179: EAPOL: External notification - portValid=1
1176713254.395269: EAPOL: SUPP_PAE entering state AUTHENTICATED
1176713254.395320: State: 4WAY_HANDSHAKE -> GROUP_HANDSHAKE
1176713254.395360: RSN: received GTK in pairwise handshake - hexdump(len=18):
01 00 c6 76 28 e9 fe 69 a5 89 99 35 f7 f8 d1 dc fa a1
1176713254.395406: WPA: Group Key - hexdump(len=16): c6 76 28 e9 fe 69 a5 89
99 35 f7 f8 d1 dc fa a1
1176713254.395449: WPA: Installing GTK to the driver (keyidx=1 tx=0).
1176713254.395488: WPA: RSC - hexdump(len=6): 30 00 00 00 00 00
```

```
1176713254.395530: wpa_driver_madwifi_set_key: alg=CCMP key_idx=1 set_tx=0
seq_len=6 key_len=16
1176713254.395592: WPA: Key negotiation completed with 00:0f:cb:b5:0e:d2
[PTK=CCMP GTK=CCMP]
1176713254.395634: Cancelling authentication timeout
1176713254.395674: State: GROUP_HANDSHAKE -> COMPLETED
1176713254.395717: CTRL-EVENT-CONNECTED - Connection to 00:0f:cb:b5:0e:d2
completed (auth) [id=0 id_str=]
1176713254.395756: wpa_driver_wext_set_operstate: operstate 0->1 (UP)
1176713254.395796: WEXT: Operstate: linkmode=-1, operstate=6
1176713254.395877: EAPOL: External notification - portValid=1
1176713254.395942: RTM_NEWLINK: operstate=1 ifi_flags=0x11043
([UP][RUNNING][LOWER_UP])
1176713254.395985: RTM_NEWLINK, IFLA_IFNAME: Interface 'ath1' added
1176713256.638946: EAPOL: startWhen --> 0
1176713283.697453: EAPOL: authWhile --> 0
```

APPENDIKS C: Oppsett av lab – Autentikator

Hostap.conf:

```
##### hostapd configuration file
##############################################
# Empty lines and lines starting with # are ignored

# AP netdevice name (without 'ap' prefix, i.e., wlan0 uses wlan0ap for
# management frames); ath0 for madwifi
interface=ath0
#interface=wifi0

# In case of madwifi driver, an additional configuration parameter, bridge,
# must be used to notify hostapd if the interface is included in a bridge.
This
# parameter is not used with Host AP driver.
bridge=br0

# Driver interface type (hostap/wired/madwifi/prism54; default: hostap)
driver=madwifi

# hostapd event logger configuration
#
# Two output method: syslog and stdout (only usable if not forking to
# background).
#
# Module bitfield (ORed bitfield of modules that will be logged; -1 = all
# modules):
# bit 0 (1) = IEEE 802.11
# bit 1 (2) = IEEE 802.1X
# bit 2 (4) = RADIUS
# bit 3 (8) = WPA
# bit 4 (16) = driver interface
# bit 5 (32) = IAPP
#
# Levels (minimum value for logged events):
#  0 = verbose debugging
#  1 = debugging
#  2 = informational messages
#  3 = notification
#  4 = warning
#
logger_syslog=-1
```

```
logger_syslog_level=0
logger_stdout=-1
logger_stdout_level=0

# Debugging: 0 = no, 1 = minimal, 2 = verbose, 3 = msg dumps, 4 = excessive
debug=0

# Dump file for state information (on SIGUSR1)
dump_file=/tmp/hostapd.dump

# Interface for separate control program. If this is specified, hostapd
# will create this directory and a UNIX domain socket for listening to
requests
# from external programs (CLI/GUI, etc.) for status information and
# configuration. The socket file will be named based on the interface name,
so
# multiple hostapd processes/interfaces can be run at the same time if more
# than one interface is used.
# /var/run/hostapd is the recommended directory for sockets and by default,
# hostapd_cli will use it when trying to connect with hostapd.
ctrl_interface=/var/run/hostapd

# Access control for the control interface can be configured by setting the
# directory to allow only members of a group to use sockets. This way, it is
# possible to run hostapd as root (since it needs to change network
# configuration and open raw sockets) and still allow GUI/CLI components to
be
# run as non-root users. However, since the control interface can be used to
# change the network configuration, this access needs to be protected in many
# cases. By default, hostapd is configured to use gid 0 (root). If you
# want to allow non-root users to use the contron interface, add a new group
# and change this value to match with that group. Add users that should have
# control interface access to this group.
#
# This variable can be a group name or gid.
#ctrl_interface_group=wheel
ctrl_interface_group=0


##### IEEE 802.11 related configuration
#######################################

# SSID to be used in IEEE 802.11 management frames
ssid=Kenneth

# Station MAC address -based authentication
```

```
# 0 = accept unless in deny list
# 1 = deny unless in accept list
# 2 = use external RADIUS server (accept/deny lists are searched first)
macaddr_acl=2

# Accept/deny lists are read from separate files (containing list of
# MAC addresses, one per line). Use absolute path name to make sure that the
# files can be read on SIGHUP configuration reloads.
#accept_mac_file=/etc/hostapd/accept
#deny_mac_file=/etc/hostapd/deny

# IEEE 802.11 specifies two authentication algorithms. hostapd can be
# configured to allow both of these or only one. Open system authentication
# should be used with IEEE 802.1X.
# Bit fields of allowed authentication algorithms:
# bit 0 = Open System Authentication
# bit 1 = Shared Key Authentication (requires WEP)
auth_algs=1

# Associate as a station to another AP while still acting as an AP on the
same
# channel.
#assoc_ap_addr=00:12:34:56:78:9a


##### IEEE 802.1X-2004 related configuration
##################################

# Require IEEE 802.1X authorization
ieee8021x=1

# Optional displayable message sent with EAP Request-Identity. The first \0
# in this string will be converted to ASCII-0 (nul). This can be used to
# separate network info (comma separated list of attribute=value pairs); see,
# e.g., draft-adrangi-eap-network-discovery-07.txt.
#eap_message=hello
#eap_message=hello\0networkid=netw,nasid=foo,portid=0,NAIRealms=example.com

# WEP rekeying (disabled if key lengths are not set or are set to 0)
# Key lengths for default/broadcast and individual/unicast keys:
# 5 = 40-bit WEP (also known as 64-bit WEP with 40 secret bits)
# 13 = 104-bit WEP (also known as 128-bit WEP with 104 secret bits)
#wep_key_len_broadcast=5
#wep_key_len_unicast=5
#Rekeying period in seconds. 0 = do not rekey (i.e., set keys only once)
#wep_rekey_period=300
```

```
# EAPOL-Key index workaround (set bit7) for WinXP Supplicant (needed only if
# only broadcast keys are used)
#eapol_key_index_workaround=0

# EAP reauthentication period in seconds (default: 3600 seconds; 0 = disable
# reauthentication).
#eap_reauth_period=3600

# Use PAE group address (01:80:c2:00:00:03) instead of individual target
# address when sending EAPOL frames with driver=wired. This is the most
common
# mechanism used in wired authentication, but it also requires that the port
# is only used by one station.
#use_pae_group_addr=1

##### Integrated EAP server
###################################################

# Optionally, hostapd can be configured to use an integrated EAP server
# to process EAP authentication locally without need for an external RADIUS
# server. This functionality can be used both as a local authentication
server
# for IEEE 802.1X/EAPOL and as a RADIUS server for other devices.

# Use integrated EAP server instead of external RADIUS authentication
# server. This is also needed if hostapd is configured to act as a RADIUS
# authentication server.
#eap_server=0

# Path for EAP server user database
#eap_user_file=/etc/hostapd/eap_user

# CA certificate (PEM or DER file) for EAP-TLS/PEAP/TTLS
#ca_cert=/etc/ssl/cacert.pem

# Server certificate (PEM or DER file) for EAP-TLS/PEAP/TTLS
#server_cert=/etc/hostapd/server.pem

# Private key matching with the server certificate for EAP-TLS/PEAP/TTLS
# This may point to the same file as server_cert if both certificate and key
# are included in a single file. PKCS#12 (PFX) file (.p12/.pfx) can also be
# used by commenting out server_cert and specifying the PFX file as the
# private_key.
#private_key=/etc/hostapd/server.prv
```

```
# Passphrase for private key
#private_key_passwd=secret passphrase

# Enable CRL verification.
# Note: hostapd does not yet support CRL downloading based on CDP. Thus, a
# valid CRL signed by the CA is required to be included in the ca_cert file.
# This can be done by using PEM format for CA certificate and CRL and
# concatenating these into one file. Whenever CRL changes, hostapd needs to
be
# restarted to take the new CRL into use.
# 0 = do not verify CRLs (default)
# 1 = check the CRL of the user certificate
# 2 = check all CRLs in the certificate path
#check_crl=1

# Configuration data for EAP-SIM database/authentication gateway interface.
# This is a text string in implementation specific format. The example
# implementation in eap_sim_db.c uses this as the file name for the GSM
# authentication triplets.
#eap_sim_db=/etc/hostapd/sim_db


##### IEEE 802.11f - Inter-Access Point Protocol (IAPP)
########################

# Interface to be used for IAPP broadcast packets
#iapp_interface=eth0


##### RADIUS client configuration
################################################
# for IEEE 802.1X with external Authentication Server, IEEE 802.11
# authentication with external ACL for MAC addresses, and accounting

# The own IP address of the access point (used as NAS-IP-Address)
own_ip_addr=129.241.208.206

# Optional NAS-Identifier string for RADIUS messages. When used, this should
be
# a unique to the NAS within the scope of the RADIUS server. For example, a
# fully qualified domain name can be used here.
#nas_identifier=ap.example.com

# RADIUS authentication server
auth_server_addr=129.241.208.159
auth_server_port=1812
```

```
auth_server_shared_secret=serengeti

# RADIUS accounting server
#acct_server_addr=129.241.208.159
#acct_server_port=1813
#acct_server_shared_secret=serengeti

# Secondary RADIUS servers; to be used if primary one does not reply to
# RADIUS packets. These are optional and there can be more than one secondary
# server listed.
#auth_server_addr=127.0.0.2
#auth_server_port=1812
#auth_server_shared_secret=secret2
#
#acct_server_addr=127.0.0.2
#acct_server_port=1813
#acct_server_shared_secret=secret2

# Retry interval for trying to return to the primary RADIUS server (in
# seconds). RADIUS client code will automatically try to use the next server
# when the current server is not replying to requests. If this interval is
set,
# primary server will be retried after configured amount of time even if the
# currently used secondary server is still working.
#radius_retry_primary_interval=600


# Interim accounting update interval
# If this is set (larger than 0) and acct_server is configured, hostapd will
# send interim accounting updates every N seconds. Note: if set, this
overrides
# possible Acct-Interim-Interval attribute in Access-Accept message. Thus,
this
# value should not be configured in hostapd.conf, if RADIUS server is used to
# control the interim interval.
# This value should not be less 600 (10 minutes) and must not be less than
# 60 (1 minute).
#radius_acct_interim_interval=600


##### RADIUS authentication server configuration
##############################

# hostapd can be used as a RADIUS authentication server for other hosts. This
# requires that the integrated EAP authenticator is also enabled and both
# authentication services are sharing the same configuration.
```

```
# File name of the RADIUS clients configuration for the RADIUS server. If
this
# commented out, RADIUS server is disabled.
#radius_server_clients=/etc/hostapd/radius_clients

# The UDP port number for the RADIUS authentication server
#radius_server_auth_port=1812

# Use IPv6 with RADIUS server (IPv4 will also be supported using IPv6 API)
#radius_server_ipv6=1


##### WPA/IEEE 802.11i configuration
##########################################

# Enable WPA. Setting this variable configures the AP to require WPA (either
# WPA-PSK or WPA-RADIUS/EAP based on other configuration). For WPA-PSK,
either
# wpa_psk or wpa_passphrase must be set and wpa_key_mgmt must include WPA-
PSK.
# For WPA-RADIUS/EAP, ieee8021x must be set (but without dynamic WEP keys),
# RADIUS authentication server must be configured, and WPA-EAP must be
included
# in wpa_key_mgmt.
# This field is a bit field that can be used to enable WPA (IEEE
802.11i/D3.0)
# and/or WPA2 (full IEEE 802.11i/RSN):
# bit0 = WPA
# bit1 = IEEE 802.11i/RSN (WPA2) (dot11RSNAEnabled)
wpa=2

# WPA pre-shared keys for WPA-PSK. This can be either entered as a 256-bit
# secret in hex format (64 hex digits), wpa_psk, or as an ASCII passphrase
# (8..63 characters) that will be converted to PSK. This conversion uses SSID
# so the PSK changes when ASCII passphrase is used and the SSID is changed.
# wpa_psk (dot11RSNAConfigPSKValue)
# wpa_passphrase (dot11RSNAConfigPSKPassPhrase)
#wpa_psk=0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef
#wpa_passphrase=insanepassphrase

# Optionally, WPA PSKs can be read from a separate text file (containing list
# of (PSK,MAC address) pairs. This allows more than one PSK to be configured.
# Use absolute path name to make sure that the files can be read on SIGHUP
# configuration reloads.
#wpa_psk_file=/etc/hostapd/wpa_psk
```

```
# Set of accepted key management algorithms (WPA-PSK, WPA-EAP, or both). The
# entries are separated with a space.
# (dot11RSNAConfigAuthenticationSuitesTable)
wpa_key_mgmt=WPA-EAP
#WPA-PSK
#WPA-EAP

# Set of accepted cipher suites (encryption algorithms) for pairwise keys
# (unicast packets). This is a space separated list of algorithms:
# CCMP = AES in Counter mode with CBC-MAC [RFC 3610, IEEE 802.11i/D7.0]
# TKIP = Temporal Key Integrity Protocol [IEEE 802.11i/D7.0]
# Group cipher suite (encryption algorithm for broadcast and multicast
frames)
# is automatically selected based on this configuration. If only CCMP is
# allowed as the pairwise cipher, group cipher will also be CCMP. Otherwise,
# TKIP will be used as the group cipher.
# (dot11RSNAConfigPairwiseCiphersTable)
wpa_pairwise=CCMP
#TKIP CCMP

# Time interval for rekeying GTK (broadcast/multicast encryption keys) in
# seconds. (dot11RSNAConfigGroupRekeyTime)
#wpa_group_rekey=600

# Rekey GTK when any STA that possesses the current GTK is leaving the BSS.
# (dot11RSNAConfigGroupRekeyStrict)
#wpa_strict_rekey=1

# Time interval for rekeying GMK (master key used internally to generate GTKs
# (in seconds).
#wpa_gmk_rekey=86400

# Enable IEEE 802.11i/RSN/WPA2 pre-authentication. This is used to speed up
# roaming be pre-authenticating IEEE 802.1X/EAP part of the full RSN
# authentication and key handshake before actually associating with a new AP.
# (dot11RSNAPreauthenticationEnabled)
#rsn_preauth=1
#
# Space separated list of interfaces from which pre-authentication frames are
# accepted (e.g., 'eth0' or 'eth0 wlan0wds0'. This list should include all
# interface that are used for connections to other APs. This could include
# wired interfaces and WDS links. The normal wireless data interface towards
# associated stations (e.g., wlan0) should not be added, since
# pre-authentication is only used with APs other than the currently
associated
```

```
        # one.
        #rsn_preauth_interfaces=eth1
        #
        # stakey: Whether STAKey negotiation for direct links (IEEE 802.11e) is
        # allowed. This is only used with RSN/WPA2.
        # 0 = disabled (default)
        # 1 = enabled
#stakey=1
```

Kjørelog:

```
        root@perulv-desktop:~# hostapd -d /etc/hostapd/hostapd.conf
        Configuration file: /etc/hostapd/hostapd.conf
        ctrl_interface_group=0
        Configure bridge br0 for EAPOL traffic.
        Using interface ath0 with hwaddr 00:0f:cb:b5:0e:d2 and ssid 'Kenneth'
        ath0: RADIUS Authentication server 129.241.208.159:1812
        madwifi_configure_wpa: group key cipher=3
        madwifi_configure_wpa: pairwise key ciphers=0x8
        madwifi_configure_wpa: key management algorithms=0x1
        madwifi_configure_wpa: rsn capabilities=0x0
        madwifi_configure_wpa: enable WPA= 0x2
        WPA: group state machine entering state GTK_INIT
        GMK - hexdump(len=32): [REMOVED]
        GTK - hexdump(len=16): [REMOVED]
        WPA: group state machine entering state SETKEYSDONE
        madwifi_set_key: alg=CCMP addr=00:00:00:00:00:00 key_idx=1
        madwifi_set_privacy: enabled=1
        SIOCGIWRANGE: WE(compiled)=20 WE(source)=13 enc_capa=0xf
        ath0: IEEE 802.11 Fetching hardware channel/rate support not supported.
        Flushing old station entries
        madwifi_sta_deauth: addr=ff:ff:ff:ff:ff:ff reason_code=3
        Deauthenticate all stations
        l2_packet_receive - recvfrom: Network is down
        ath0: STA 00:0f:cb:b3:67:11 IEEE 802.11: associated
          New STA
        ath0: STA 00:0f:cb:b3:67:11 WPA: event 1 notification
        madwifi_del_key: addr=00:0f:cb:b3:67:11 key_idx=0
        ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: start authentication
        IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state INITIALIZE
        IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state INITIALIZE
        IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
        IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_KEY_TX entering state NO_KEY_TRANSMIT
        IEEE 802.1X: 00:0f:cb:b3:67:11 KEY_RX entering state NO_KEY_RECEIVE
        IEEE 802.1X: 00:0f:cb:b3:67:11 CTRL_DIR entering state IN_OR_BOTH
        IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state INITIALIZE
```

```
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state IDLE
IEEE 802.1X: 00:0f:cb:b3:67:11 KEY_RX entering state NO_KEY_RECEIVE
IEEE 802.1X: 00:0f:cb:b3:67:11 CTRL_DIR entering state FORCE_BOTH
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 KEY_RX entering state NO_KEY_RECEIVE
ath0: STA 00:0f:cb:b3:67:11 WPA: start authentication
WPA: 00:0f:cb:b3:67:11 WPA_PTK entering state INITIALIZE
madwifi_del_key: addr=00:0f:cb:b3:67:11 key_idx=0
WPA: 00:0f:cb:b3:67:11 WPA_PTK_GROUP entering state IDLE
WPA: 00:0f:cb:b3:67:11 WPA_PTK entering state AUTHENTICATION
WPA: 00:0f:cb:b3:67:11 WPA_PTK entering state AUTHENTICATION2
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state DISCONNECTED
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: unauthorizing port
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state RESTART
IEEE 802.1X: station 00:0f:cb:b3:67:11 - new auth session, clearing State
IEEE 802.1X: Generated EAP Request-Identity for 00:0f:cb:b3:67:11 (identifier
0, timeout 30)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state CONNECTING
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state AUTHENTICATING
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:0f:cb:b3:67:11 (identifier 0)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
```

```
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: EAP timeout
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state TIMEOUT
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state ABORTING
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state INITIALIZE
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: aborting authentication
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state RESTART
IEEE 802.1X: station 00:0f:cb:b3:67:11 - new auth session, clearing State
IEEE 802.1X: Generated EAP Request-Identity for 00:0f:cb:b3:67:11 (identifier
1, timeout 30)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state IDLE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state CONNECTING
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state AUTHENTICATING
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:0f:cb:b3:67:11 (identifier 1)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 19 bytes from 00:0f:cb:b3:67:11
   IEEE 802.1X: version=1 type=0 length=15
   EAP: code=2 identifier=1 length=15 (response)
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: received EAP packet (code=2 id=1
len=15) from STA: EAP Response-Identity (1)
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: STA identity 'Supplicant'
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
ath0: RADIUS Sending RADIUS message to authentication server
ath0: RADIUS Next RADIUS client retransmit in 3 seconds

IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
```

```
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
ath0: RADIUS Received 64 bytes from RADIUS server
ath0: RADIUS Received RADIUS message
ath0: STA 00:0f:cb:b3:67:11 RADIUS: Received RADIUS packet matched with a
pending request, round trip time 0.08 sec
RADIUS packet matching with station 00:0f:cb:b3:67:11
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: using EAP timeout of 30 seconds
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: decapsulated EAP packet (code=1 id=2
len=6) from RADIUS server: EAP-Request-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:0f:cb:b3:67:11 (identifier 2)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 111 bytes from 00:0f:cb:b3:67:11
   IEEE 802.1X: version=1 type=0 length=107
   EAP: code=2 identifier=2 length=107 (response)
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: received EAP packet (code=2 id=2
len=107) from STA: EAP Response-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
  Copied RADIUS State Attribute
ath0: RADIUS Sending RADIUS message to authentication server
ath0: RADIUS Next RADIUS client retransmit in 3 seconds

IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
ath0: RADIUS Received 1100 bytes from RADIUS server
ath0: RADIUS Received RADIUS message
ath0: STA 00:0f:cb:b3:67:11 RADIUS: Received RADIUS packet matched with a
pending request, round trip time 0.02 sec
RADIUS packet matching with station 00:0f:cb:b3:67:11
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: using EAP timeout of 30 seconds
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: decapsulated EAP packet (code=1 id=3
len=1034) from RADIUS server: EAP-Request-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:0f:cb:b3:67:11 (identifier 3)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 10 bytes from 00:0f:cb:b3:67:11
   IEEE 802.1X: version=1 type=0 length=6
   EAP: code=2 identifier=3 length=6 (response)
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: received EAP packet (code=2 id=3
len=6) from STA: EAP Response-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
  Copied RADIUS State Attribute
```

```
ath0: RADIUS Sending RADIUS message to authentication server
ath0: RADIUS Next RADIUS client retransmit in 3 seconds

IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
ath0: RADIUS Received 1100 bytes from RADIUS server
ath0: RADIUS Received RADIUS message
ath0: STA 00:0f:cb:b3:67:11 RADIUS: Received RADIUS packet matched with a
pending request, round trip time 0.01 sec
RADIUS packet matching with station 00:0f:cb:b3:67:11
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: using EAP timeout of 30 seconds
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: decapsulated EAP packet (code=1 id=4
len=1034) from RADIUS server: EAP-Request-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:0f:cb:b3:67:11 (identifier 4)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 10 bytes from 00:0f:cb:b3:67:11
   IEEE 802.1X: version=1 type=0 length=6
   EAP: code=2 identifier=4 length=6 (response)
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: received EAP packet (code=2 id=4
len=6) from STA: EAP Response-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
  Copied RADIUS State Attribute
ath0: RADIUS Sending RADIUS message to authentication server
ath0: RADIUS Next RADIUS client retransmit in 3 seconds

IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
ath0: RADIUS Received 224 bytes from RADIUS server
ath0: RADIUS Received RADIUS message
ath0: STA 00:0f:cb:b3:67:11 RADIUS: Received RADIUS packet matched with a
pending request, round trip time 0.02 sec
RADIUS packet matching with station 00:0f:cb:b3:67:11
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: using EAP timeout of 30 seconds
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: decapsulated EAP packet (code=1 id=5
len=166) from RADIUS server: EAP-Request-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:0f:cb:b3:67:11 (identifier 5)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 1412 bytes from 00:0f:cb:b3:67:11
   IEEE 802.1X: version=1 type=0 length=1408
   EAP: code=2 identifier=5 length=1408 (response)
```

```
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: received EAP packet (code=2 id=5
len=1408) from STA: EAP Response-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
  Copied RADIUS State Attribute
ath0: RADIUS Sending RADIUS message to authentication server
ath0: RADIUS Next RADIUS client retransmit in 3 seconds

IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
ath0: RADIUS Received 64 bytes from RADIUS server
ath0: RADIUS Received RADIUS message
ath0: STA 00:0f:cb:b3:67:11 RADIUS: Received RADIUS packet matched with a
pending request, round trip time 0.00 sec
RADIUS packet matching with station 00:0f:cb:b3:67:11
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: using EAP timeout of 30 seconds
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: decapsulated EAP packet (code=1 id=6
len=6) from RADIUS server: EAP-Request-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:0f:cb:b3:67:11 (identifier 6)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 1199 bytes from 00:0f:cb:b3:67:11
   IEEE 802.1X: version=1 type=0 length=1195
   EAP: code=2 identifier=6 length=1195 (response)
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: received EAP packet (code=2 id=6
len=1195) from STA: EAP Response-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
  Copied RADIUS State Attribute
ath0: RADIUS Sending RADIUS message to authentication server
ath0: RADIUS Next RADIUS client retransmit in 3 seconds

IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
ath0: RADIUS Received 127 bytes from RADIUS server
ath0: RADIUS Received RADIUS message
ath0: STA 00:0f:cb:b3:67:11 RADIUS: Received RADIUS packet matched with a
pending request, round trip time 0.07 sec
RADIUS packet matching with station 00:0f:cb:b3:67:11
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: using EAP timeout of 30 seconds
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: decapsulated EAP packet (code=1 id=7
len=69) from RADIUS server: EAP-Request-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:0f:cb:b3:67:11 (identifier 7)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
```

```
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 10 bytes from 00:0f:cb:b3:67:11
   IEEE 802.1X: version=1 type=0 length=6
   EAP: code=2 identifier=7 length=6 (response)
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: received EAP packet (code=2 id=7
len=6) from STA: EAP Response-TLS (13)
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
  Copied RADIUS State Attribute
ath0: RADIUS Sending RADIUS message to authentication server
ath0: RADIUS Next RADIUS client retransmit in 3 seconds

IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
ath0: RADIUS Received 172 bytes from RADIUS server
ath0: RADIUS Received RADIUS message
ath0: STA 00:0f:cb:b3:67:11 RADIUS: Received RADIUS packet matched with a
pending request, round trip time 0.00 sec
RADIUS packet matching with station 00:0f:cb:b3:67:11
MS-MPPE-Send-Key - hexdump(len=32): [REMOVED]
MS-MPPE-Recv-Key - hexdump(len=32): [REMOVED]
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: old identity 'Supplicant' updated
with User-Name from Access-Accept 'Supplicant'
RSN: added PMKSA cache entry for 00:0f:cb:b3:67:11
RSN: added PMKID - hexdump(len=16): 4e 22 fd 43 4c 32 ae 2b eb be 33 df 2d 5b
14 68
ath0: STA 00:0f:cb:b3:67:11 WPA: Added PMKSA cache entry
ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: decapsulated EAP packet (code=3 id=7
len=4) from RADIUS server: EAP Success
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state SUCCESS
IEEE 802.1X: Sending EAP Packet to 00:0f:cb:b3:67:11 (identifier 7)
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 BE_AUTH entering state IDLE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:0f:cb:b3:67:11 REAUTH_TIMER entering state INITIALIZE
WPA: 00:0f:cb:b3:67:11 WPA_PTK entering state INITPMK
WPA: PMK from EAPOL state machine (len=32)
WPA: 00:0f:cb:b3:67:11 WPA_PTK entering state PTKSTART
ath0: STA 00:0f:cb:b3:67:11 WPA: sending 1/4 msg of 4-Way Handshake
WPA: Send EAPOL(secure=0 mic=0 ack=1 install=0 pairwise=8 ie_len=22 gtk_len=0
keyidx=0 encr=0)
IEEE 802.1X: 121 bytes from 00:0f:cb:b3:67:11
   IEEE 802.1X: version=1 type=3 length=117
ath0: STA 00:0f:cb:b3:67:11 WPA: received EAPOL-Key frame (2/4 Pairwise)
WPA: 00:0f:cb:b3:67:11 WPA_PTK entering state PTKCALCNEGOTIATING
PMK - hexdump(len=32): [REMOVED]
```

```
       PTK - hexdump(len=64): [REMOVED]
       WPA: 00:0f:cb:b3:67:11 WPA_PTK entering state PTKCALCNEGOTIATING2
       WPA: 00:0f:cb:b3:67:11 WPA_PTK entering state PTKINITNEGOTIATING
       madwifi_get_seqnum: addr=00:00:00:00:00:00 idx=1
       ath0: STA 00:0f:cb:b3:67:11 WPA: sending 3/4 msg of 4-Way Handshake
       WPA: Send EAPOL(secure=1 mic=1 ack=1 install=1 pairwise=8 ie_len=22
       gtk_len=16 keyidx=1 encr=1)
       Plaintext EAPOL-Key Key Data - hexdump(len=56): [REMOVED]
       IEEE 802.1X: 99 bytes from 00:0f:cb:b3:67:11
          IEEE 802.1X: version=1 type=3 length=95
       ath0: STA 00:0f:cb:b3:67:11 WPA: received EAPOL-Key frame (4/4 Pairwise)
       WPA: 00:0f:cb:b3:67:11 WPA_PTK entering state PTKINITDONE
       madwifi_set_key: alg=CCMP addr=00:0f:cb:b3:67:11 key_idx=0
       ath0: STA 00:0f:cb:b3:67:11 WPA: pairwise key handshake completed (RSN)
       IEEE 802.1X: 00:0f:cb:b3:67:11 AUTH_PAE entering state AUTHENTICATED
       ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: authorizing port
       madwifi_sta_clear_stats: addr=00:0f:cb:b3:67:11
       ath0: STA 00:0f:cb:b3:67:11 IEEE 802.1X: authenticated
       RSN: added PMKSA cache entry for 00:0f:cb:b3:67:11
       RSN: added PMKID - hexdump(len=16): 4e 22 fd 43 4c 32 ae 2b eb be 33 df 2d 5b
       14 68
       ath0: STA 00:0f:cb:b3:67:11 WPA: Added PMKSA cache entry (IEEE 802.1X)
IEEE 802.1X: 00:0f:cb:b3:67:11 - aWhile --> 0
```

APPENDIKS D: Oppsett av lab – Autentiseringsserver

Eap.conf:

```
# -*- text -*-
#
#  Whatever you do, do NOT set 'Auth-Type := EAP'.  The server
#  is smart enough to figure this out on its own.  The most
#  common side effect of setting 'Auth-Type := EAP' is that the
#  users then cannot use ANY other authentication method.
#
#       $Id: eap.conf,v 1.4.4.3 2006/04/28 18:25:03 aland Exp $
#
        eap {
                #  Invoke the default supported EAP type when
                #  EAP-Identity response is received.
                #
                #  The incoming EAP messages DO NOT specify which EAP
                #  type they will be using, so it MUST be set here.
                #
                #  For now, only one default EAP type may be used at a time.
                #
                #  If the EAP-Type attribute is set by another module,
                #  then that EAP type takes precedence over the
                #  default type configured here.
                #
                default_eap_type = tls

                #  A list is maintained to correlate EAP-Response
                #  packets with EAP-Request packets.  After a
                #  configurable length of time, entries in the list
                #  expire, and are deleted.
                #
                timer_expire     = 600

                #  There are many EAP types, but the server has support
                #  for only a limited subset.  If the server receives
                #  a request for an EAP type it does not support, then
                #  it normally rejects the request.  By setting this
                #  configuration to "yes", you can tell the server to
                #  instead keep processing the request.  Another module
                #  MUST then be configured to proxy the request to
                #  another RADIUS server which supports that EAP type.
                #
```

```
#  If another module is NOT configured to handle the
#  request, then the request will still end up being
#  rejected.
ignore_unknown_eap_types = yes

# Cisco AP1230B firmware 12.2(13)JA1 has a bug.  When given
# a User-Name attribute in an Access-Accept, it copies one
# more byte than it should.
#
# We can work around it by configurably adding an extra
# zero byte.
#cisco_accounting_username_bug = no

# Supported EAP-types

#
#  We do NOT recommend using EAP-MD5 authentication
#  for wireless connections.  It is insecure, and does
#  not provide for dynamic WEP keys.
#
md5 {
}

# Cisco LEAP
#
#  We do not recommend using LEAP in new deployments.  See:
#  http://www.securiteam.com/tools/5TP012ACKE.html
#
#  Cisco LEAP uses the MS-CHAP algorithm (but not
#  the MS-CHAP attributes) to perform it's authentication.
#
#  As a result, LEAP *requires* access to the plain-text
#  User-Password, or the NT-Password attributes.
#  'System' authentication is impossible with LEAP.
#
leap {
}

#  Generic Token Card.
#
#  Currently, this is only permitted inside of EAP-TTLS,
#  or EAP-PEAP.  The module "challenges" the user with
#  text, and the response from the user is taken to be
#  the User-Password.
#
#  Proxying the tunneled EAP-GTC session is a bad idea,
```

```
#  the users password will go over the wire in plain-text,
#  for anyone to see.
#
gtc {
        #  The default challenge, which many clients
        #  ignore..
        #challenge = "Password: "

        #  The plain-text response which comes back
        #  is put into a User-Password attribute,
        #  and passed to another module for
        #  authentication.  This allows the EAP-GTC
        #  response to be checked against plain-text,
        #  or crypt'd passwords.
        #
        #  If you say "Local" instead of "PAP", then
        #  the module will look for a User-Password
        #  configured for the request, and do the
        #  authentication itself.
        #
        #auth_type = PAP
}

## EAP-TLS
#
#  To generate ctest certificates, run the script
#
#       ../scripts/certs.sh
#
#  The documents on http://www.freeradius.org/doc
#  are old, but may be helpful.
#
#  See also:
#
#  http://www.dslreports.com/forum/remark,9286052~mode=flat
#
tls {
        # timer_expire = 60
        private_key_password = serengeti
        private_key_file = /etc/ssl/private/as_private.pem

        #  If Private key & Certificate are located in
        #  the same file, then private_key_file &
        #  certificate_file must contain the same file
        #  name.
        certificate_file = /etc/ssl/as.pem
```

```
#  Trusted Root CA list
CA_file = /etc/ssl/cacert.pem
dh_file = ${raddbdir}/certs/dh
random_file = ${raddbdir}/certs/random

#
#  This can never exceed the size of a RADIUS
#  packet (4096 bytes), and is preferably half
#  that, to accomodate other attributes in
#  RADIUS packet.  On most APs the MAX packet
#  length is configured between 1500 - 1600
#  In these cases, fragment size should be
#  1024 or less.
#
fragment_size = 1024

#  include_length is a flag which is
#  by default set to yes If set to
#  yes, Total Length of the message is
#  included in EVERY packet we send.
#  If set to no, Total Length of the
#  message is included ONLY in the
#  First packet of a fragment series.
#
include_length = yes

#  Check the Certificate Revocation List
#
#  1) Copy CA certificates and CRLs to same directory.
#  2) Execute 'c_rehash <CA certs&CRLs Directory>'.
#    'c_rehash' is OpenSSL's command.
#  3) Add 'CA_path=<CA certs&CRLs directory>'
#      to radiusd.conf's tls section.
#  4) uncomment the line below.
#  5) Restart radiusd
#check_crl = no

#
#  If check_cert_issuer is set, the value will
#  be checked against the DN of the issuer in
#  the client certificate.  If the values do not
#  match, the cerficate verification will fail,
#  rejecting the user.
#
# check_cert_issuer = "/C=GB/ST=Berkshire/L=Newbury/O=My $
```

```
        #
        #  If check_cert_cn is set, the value will
        #  be xlat'ed and checked against the CN
        #  in the client certificate.  If the values
        #  do not match, the certificate verification
        #  will fail rejecting the user.
        #
        #  This check is done only if the previous
        #  "check_cert_issuer" is not set, or if
        #  the check succeeds.
        #
        # check_cert_cn = %{User-Name}
        #
        # Set this option to specify the allowed
        # TLS cipher suites.  The format is listed
        # in "man 1 ciphers".
        # cipher_list = "DEFAULT"
}

#  The TTLS module implements the EAP-TTLS protocol,
#  which can be described as EAP inside of Diameter,
#  inside of TLS, inside of EAP, inside of RADIUS...
#
#  Surprisingly, it works quite well.
#
#  The TTLS module needs the TLS module to be installed
#  and configured, in order to use the TLS tunnel
#  inside of the EAP packet.  You will still need to
#  configure the TLS module, even if you do not want
#  to deploy EAP-TLS in your network.  Users will not
#  be able to request EAP-TLS, as it requires them to
#  have a client certificate.  EAP-TTLS does not
#  require a client certificate.
#
#ttls {
        #  The tunneled EAP session needs a default
        #  EAP type which is separate from the one for
        #  the non-tunneled EAP module.  Inside of the
        #  TTLS tunnel, we recommend using EAP-MD5.
        #  If the request does not contain an EAP
        #  conversation, then this configuration entry
        #  is ignored.
        # default_eap_type = md5

        #  The tunneled authentication request does
```

```
        #  not usually contain useful attributes
        #  like 'Calling-Station-Id', etc.  These
        #  attributes are outside of the tunnel,
        #  and normally unavailable to the tunneled
        #  authentication request.
        #
        #  By setting this configuration entry to
        #  'yes', any attribute which NOT in the
        #  tunneled authentication request, but
        #  which IS available outside of the tunnel,
        #  is copied to the tunneled request.
        #
        # allowed values: {no, yes}
        # copy_request_to_tunnel = no

        #  The reply attributes sent to the NAS are
        #  usually based on the name of the user
        #  'outside' of the tunnel (usually
        #  'anonymous').  If you want to send the
        #  reply attributes based on the user name
        #  inside of the tunnel, then set this
        #  configuration entry to 'yes', and the reply
        #  to the NAS will be taken from the reply to
        #  the tunneled request.
        #
        # allowed values: {no, yes}
        # use_tunneled_reply = no
#}

#
#  The tunneled EAP session needs a default EAP type
#  which is separate from the one for the non-tunneled
#  EAP module.  Inside of the TLS/PEAP tunnel, we
#  recommend using EAP-MS-CHAPv2.
#
#  The PEAP module needs the TLS module to be installed
#  and configured, in order to use the TLS tunnel
#  inside of the EAP packet.  You will still need to
#  configure the TLS module, even if you do not want
#  to deploy EAP-TLS in your network.  Users will not
#  be able to request EAP-TLS, as it requires them to
#  have a client certificate.  EAP-PEAP does not
#  require a client certificate.
#
# peap {
        #  The tunneled EAP session needs a default
```

```
                        #  EAP type which is separate from the one for
                        #  the non-tunneled EAP module.  Inside of the
                        #  PEAP tunnel, we recommend using MS-CHAPv2,
                        #  as that is the default type supported by
                        #  Windows clients.
                        # default_eap_type = mschapv2

                        #  the PEAP module also has these configuration
                        #  items, which are the same as for TTLS.
                        # copy_request_to_tunnel = no
                        # use_tunneled_reply = no

                        #  When the tunneled session is proxied, the
                        #  home server may not understand EAP-MSCHAP-V2.
                        #  Set this entry to "no" to proxy the tunneled
                        #  EAP-MSCHAP-V2 as normal MSCHAPv2.
                        # proxy_tunneled_request_as_eap = yes
                #}

                #
                #  This takes no configuration.
                #
                #  Note that it is the EAP MS-CHAPv2 sub-module, not
                #  the main 'mschap' module.
                #
                #  Note also that in order for this sub-module to work,
                #  the main 'mschap' module MUST ALSO be configured.
                #
                #  This module is the *Microsoft* implementation of MS-CHAPv2
                #  in EAP.  There is another (incompatible) implementation
                #  of MS-CHAPv2 in EAP by Cisco, which FreeRADIUS does not
                #  currently support.
                #
                mschapv2 {
                }
        }
```

Clients.conf:

```
#
# clients.conf - client configuration directives
#
#########################################################################

#########################################################################
#
```

```
#  Definition of a RADIUS client (usually a NAS).
#
#  The information given here over rides anything given in the
#  'clients' file, or in the 'naslist' file.  The configuration here
#  contains all of the information from those two files, and allows
#  for more configuration items.
#
#  The "shortname" is be used for logging.  The "nastype", "login" and
#  "password" fields are mainly used for checkrad and are optional.
#

#
#  Defines a RADIUS client.  The format is 'client [hostname|ip-address]'
#
#  '127.0.0.1' is another name for 'localhost'.  It is enabled by default,
#  to allow testing of the server after an initial installation.  If you
#  are not going to be permitting RADIUS queries from localhost, we suggest
#  that you delete, or comment out, this entry.
#
#client 127.0.0.1 {
        #
        #  The shared secret use to "encrypt" and "sign" packets between
        #  the NAS and FreeRADIUS.  You MUST change this secret from the
        #  default, otherwise it's not a secret any more!
        #
        #  The secret can be any string, up to 31 characters in length.
        #
        #secret          = serengeti

        #
        #  The short name is used as an alias for the fully qualified
        #  domain name, or the IP address.
        #
        #shortname       = localhost

        #
        # the following three fields are optional, but may be used by
        # checkrad.pl for simultaneous use checks
        #
        #

        #
        # The nastype tells 'checkrad.pl' which NAS-specific method to
        #  use to query the NAS for simultaneous use.
        #
        #  Permitted NAS types are:
```

```
        #
        #       cisco
        #       computone
        #       livingston
        #       max40xx
        #       multitech
        #       netserver
        #       pathras
        #       patton
        #       portslave
        #       tc
        #       usrhiper
        #       other           # for all other types

        #
        #nastype     = other    # localhost isn't usually a NAS...

        #
        #  The following two configurations are for future use.
        #  The 'naspasswd' file is currently used to store the NAS
        #  login name and password, which is used by checkrad.pl
        #  when querying the NAS for simultaneous use.
        #
        #       login           = !root
        #       password  = someadminpas
#}

#client some.host.org {
#       secret            = testing123
#       shortname         = localhost
#}

#
#  You can now specify one secret for a network of clients.
#  When a client request comes in, the BEST match is chosen.
#  i.e. The entry from the smallest possible network.
#
client 129.241.208.0/8 {
        secret             = serengeti
        shortname   = diplom
}
```

Kjørelog:

```
Starting - reading configuration files ...
Using deprecated naslist file.  Support for this will go away soon.
```

```
Module: Loaded exec
rlm_exec: Wait=yes but no output defined. Did you mean output=none?
Module: Instantiated exec (exec)
Module: Loaded expr
Module: Instantiated expr (expr)
Module: Loaded PAP
Module: Instantiated pap (pap)
Module: Loaded CHAP
Module: Instantiated chap (chap)
Module: Loaded MS-CHAP
Module: Instantiated mschap (mschap)
Module: Loaded System
Module: Instantiated unix (unix)
Module: Loaded eap
rlm_eap: Loaded and initialized type md5
rlm_eap: Loaded and initialized type leap
rlm_eap: Loaded and initialized type gtc
rlm_eap_tls: Loading the certificate file as a chain
rlm_eap: Loaded and initialized type tls
rlm_eap: Loaded and initialized type mschapv2
Module: Instantiated eap (eap)
Module: Loaded preprocess
Module: Instantiated preprocess (preprocess)
Module: Loaded realm
Module: Instantiated realm (suffix)
Module: Loaded files
Module: Instantiated files (files)
Module: Loaded Acct-Unique-Session-Id
Module: Instantiated acct_unique (acct_unique)
Module: Loaded detail
Module: Instantiated detail (detail)
Module: Loaded radutmp
Module: Instantiated radutmp (radutmp)
Initializing the thread pool...
Listening on authentication *:1812
Listening on accounting *:1813
Ready to process requests.

rad_recv: Access-Request packet from host 129.241.208.206:1148, id=0,
length=159
        User-Name = "Supplicant"
        NAS-IP-Address = 129.241.208.206
        NAS-Port = 0
        Called-Station-Id = "00-0F-CB-B5-0E-D2:Kenneth"
        Calling-Station-Id = "00-0F-CB-B3-67-11"
        Framed-MTU = 1400
```

```
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 0Mbps 802.11"
        EAP-Message = 0x0201000f01537570706c6963616e74
        Message-Authenticator = 0xd77384e38e24aefd9badd2516231f0d8
Sending Access-Challenge of id 0 to 129.241.208.206 port 1148
        EAP-Message = 0x010200060d20
        Message-Authenticator = 0x00000000000000000000000000000000
        State = 0x14b532961d329bd28a711c362ea8b3cc
rad_recv: Access-Request packet from host 129.241.208.206:1148, id=1,
length=269
        User-Name = "Supplicant"
        NAS-IP-Address = 129.241.208.206
        NAS-Port = 0
        Called-Station-Id = "00-0F-CB-B5-0E-D2:Kenneth"
        Calling-Station-Id = "00-0F-CB-B3-67-11"
        Framed-MTU = 1400
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 0Mbps 802.11"
        EAP-Message =
0x0202006b0d00160301006001000005c030146233826c18e5b8de54265e815e009af2083cf31d
2052cd6832ff2400e1d9a4e000034003900380035001600130000a00330032002f006600050004
00630062006100150012000900650064006000140011000800060003020100
        State = 0x14b532961d329bd28a711c362ea8b3cc
        Message-Authenticator = 0x59ddbb99172f11c97591d3d250649346
    TLS_accept:error in SSLv3 read client certificate A
rlm_eap: SSL error error:00000000:lib(0):func(0):reason(0)
Sending Access-Challenge of id 1 to 129.241.208.206 port 1148
        EAP-Message =
0x0103040a0dc00000089c160301004a020000460301462338288a19357caab6de5ffb95005ba
2ebe1dd1ddfa39ada0f43360b59079a2028fd429dd3f25edd820f255c44041e388b5104a7ae2d
fa417535d578d36190ac00350116030107ad0b0007a90007a60003253082032130820209a0030
20102020103300d06092a864886f70d01010505003081833310b3009060355040613024e4f3116
30140603550408130d536f722d54726f6e64656c6167311230100603550407130954726f6e646
865696d310d300b060355040a13044e544e55310f300d060355040b13064469706c6f6d310b30
090603550403130243413 11b301906092a864886f70d01
        EAP-Message =
0x0901160c6361406469706c6f6d2e6e6f301e170d3037303330313133343732395a170d30383
032323931333343732395a306f310b3009060355040613024e4f3116301406035504081 30d536f
722d54726f6e64656c6167310d300b060355040a13044e544e55310f300d060355040b1306446
9706c6f6d310b3009060355040313024153311b301906092a864886f70d010901160c61734064
69706c6f6d2e6e6f305c300d06092a864886f70d0101010500034b003048024100d686b2266fd
51d24bf52660329c3b73aff21b4c067fce797b2858b6e3a6268fd37489f63b12e8829726225d6
1100c9d5be230ef2f28ccc4d3012c66346be764f020301
        EAP-Message =
0x0001a37b307930090603551d1304023000302c06096086480186f842010d041f161d4f70656
e53534c2047656e6572617465642043657274696669636174653 01d0603551d0e041604142f44
```

```
8c6c0e85418db3c77c77b2528bbc5ec1880e301f0603551d230418301680146554cd608447278
1694d5b83491a51299286a07e300d06092a864886f70d010105050003820101009l5e2fd51b62
d998cb240b39ab041b7ece5d4d9f31d9a64f896ceff77b1ecade81d27018fd3794230a8ab6e03
7a35c841e75ddfdf8aaccb15609d251fa185aca78e56e90daa46904ea5b1a6ff9ff1ba9dbef51
15591d56a213c958352afea926de2baeeea937ea8b1bc7
        EAP-Message =
0x826927647112b3369cf5b7a4fe9d6dc7e8465b8c489dff4e77541f156e1e9b0f54315a2f97e
517f5606f59800428ef841822dca852b9cfd152f263a7a102e2e1bcf705b87a1e1cf71ae0447e
8c704c6ae6972e7fc2449489c78823c43309431aa3feb5debef33d6b5e4edbf377bfe95e95689
95aeeaadbe34ee67218d6aac8a2269f6524e7aeca2260be063045899603b9b5b62f874b00047b
308204773082035fa003020102020900e3b8a41f595c46a9300d06092a864886f70d010105050
0308183310b300906035504061302e4f311630140603550408130d536f722d54726f6e64656c
6167311230100603550407130954726f6e646865696d31
        EAP-Message = 0x0d300b060355040a13044e544e55310f300d06035504
        Message-Authenticator = 0x00000000000000000000000000000000
        State = 0x5ac8722ff26a10c27c7438e37ad1d6fa
rad_recv: Access-Request packet from host 129.241.208.206:1148, id=2,
length=168
        User-Name = "Supplicant"
        NAS-IP-Address = 129.241.208.206
        NAS-Port = 0
        Called-Station-Id = "00-0F-CB-B5-0E-D2:Kenneth"
        Calling-Station-Id = "00-0F-CB-B3-67-11"
        Framed-MTU = 1400
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 0Mbps 802.11"
        EAP-Message = 0x020300060d00
        State = 0x5ac8722ff26a10c27c7438e37ad1d6fa
        Message-Authenticator = 0xc60a2aceed1c2b521b2cacd23b8adab9
Sending Access-Challenge of id 2 to 129.241.208.206 port 1148
        EAP-Message =
0x0104040a0dc00000089c0b13064469706c6f6d310b300906035504031302434131lb3019060
92a864886f70d010901160c6361406469706c6f6d2e6e6f301e170d303730333031333303734
385a170d313030323238313303734385a308183310b300906035504061302e4f31163014060
3550408130d536f722d54726f6e64656c6167311230100603550407130954726f6e646865696d
310d300b060355040a13044e544e55310f300d060355040b13064469706c6f6d310b300906035
504031302434131lb301906092a864886f70d010901160c6361406469706c6f6d2e6e6f308201
22300d06092a864886f70d01010105000382010f003082
        EAP-Message =
0x010a0282010100bdf734a0729543d256f5adf7e5e2cbc46f4fb95337a380e8d9f08fe3abce8
9e4b139775a1a1453945bfbc2c984a0035bde4d2454da5fee7d5053c6dd2e29ffafadd804db8d
18f0da2d57ee58fcb612dc835d8ee8433344b13aaef25dd25a2aeeb32a90eca77d85a505aa91c
7db7748e69f63c9fda578a85d470df401d0408c7abaad4bb4dc3e1287a3d91756825626ec9382
50733bcc974df9d78c1b744d8d4d810c33160282797c413361f718653cb0595cf822cad0c3ecf
bf40bf290f4dfb2ce10c7e9cf77f274a72d1ced3673fe302d33c510104d41e6085336c104e93c
4917c186d2bdff7170df4fd4b52e35d6c24fe7e95d6744
```

```
        EAP-Message =
0x4d4ee7a1e01029d5216d0203010001a381eb3081e8301d0603551d0e041604146554cd60844
72781694d5b83491a51299286a07e3081b80603551d230481b03081ad80146554cd6084472781
694d5b83491a51299286a07ea18189a48186308183310b300906035504061302e4f311630140
603550408130d536f722d54726f6e64656c6167311230100603550407130954726f6e64686569
6d310d300b060355040a13044e544e55310f300d060355040b13064469706c6f6d310b3009060
355040313024341311b301906092a864886f70d010901160c6361406469706c6f6d2e6e6f8209
00e3b8a41f595c46a9300c0603551d13040530030101ff
        EAP-Message =
0x300d06092a864886f70d01010505000382010100015049 25d51109f5b65ead2899f15ad1b0f
9f3a8e1cd973a7b3a87da431ad11ca569b122005ae0abed60b863e97bc0d153c381968abd5a60
c422ec7ace0bd5240ce2abb9d5b740276134cf956b868ba465614300968775be10b4d72e42365
7a991eebcc11e6cd447698d944dfa4cfc624c38d7defba4ec610a5e387ccbbb1338a2eed03a2c
8c070c854cd8133545d0a7a96e8db7a32edd7a1b8748485049714e27b1b1a012f7b965ad4b7b0
cb05ffbe9f6bbe30517d03ed5a74eb300ec6f688823ae6523956fe0b79ec99eeb99a2900e721c
887094340a1bcdd0194b43349c4e73478d2ceba5a5b549
        EAP-Message = 0xe73230e7fc6bdccc03c0606b8fb133f3202c4f870d12
        Message-Authenticator = 0x00000000000000000000000000000000
        State = 0xc2a70fbd90f42d5e444f8456418c516a
rad_recv: Access-Request packet from host 129.241.208.206:1148, id=3,
length=168
        User-Name = "Supplicant"
        NAS-IP-Address = 129.241.208.206
        NAS-Port = 0
        Called-Station-Id = "00-0F-CB-B5-0E-D2:Kenneth"
        Calling-Station-Id = "00-0F-CB-B3-67-11"
        Framed-MTU = 1400
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 0Mbps 802.11"
        EAP-Message = 0x020400060d00
        State = 0xc2a70fbd90f42d5e444f8456418c516a
        Message-Authenticator = 0xde72eaf2b7b5ab1d9c3afcab88632f70
Sending Access-Challenge of id 3 to 129.241.208.206 port 1148
        EAP-Message =
0x010500a60d800000089c8c16030100960d00008e030102400088008630818331 0b300906035
5040613024e4f311630140603550408130d536f722d54726f6e64656c6167311230100603550 4
07130954726f6e646865696d310d300b060355040a13044e544e55310f300d060355040b13064
469706c6f6d310b3009060355040313024341311b301906092a864886f70d010901160c636140
6469706c6f6d2e6e6f0e000000
        Message-Authenticator = 0x00000000000000000000000000000000
        State = 0x3f7bd742bba3aaa475d55c8949c32f46
rad_recv: Access-Request packet from host 129.241.208.206:1148, id=4,
length=1580
        User-Name = "Supplicant"
        NAS-IP-Address = 129.241.208.206
        NAS-Port = 0
```

```
        Called-Station-Id = "00-0F-CB-B5-0E-D2:Kenneth"
        Calling-Station-Id = "00-0F-CB-B3-67-11"
        Framed-MTU = 1400
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 0Mbps 802.11"
        EAP-Message =
0x020505800dc000000a1b16030108850b00088100087e0003fd308203f9308202e1a00302010
2020101300d06092a864886f70d0101050500308183310b300906035504061302e4f31163014
0603550408130d536f722d54726f6e64656c6167311230100603550407130954726f6e6468656
96d310d300b060355040a13044e544e55310f300d060355040b13064469706c6f6d310b300906
035504031302434131b301906092a864886f70d010901160c6361406469706c6f6d2e6e6f301
e170d3037303330313133313532305a170d303830323239313313532305a307f310b30090603
55040613024e4f3116301406035504081 30d536f722d54
        EAP-Message =
0x726f6e64656c6167310d300b060355040a13044e544e55310f300d060355040b13064469706
c6f6d311330110603550403130a537570706c6963616e743123021060 92a864886f70d010901
1614737570706c6963616e74406469706c6f6d2e6e6f30820122300d06092a864886f70d01010
105000382010f003082010a0282010100a6185a7bf4485ce758eae13c5e4d1bd3eabbc8d438ee
e4fcb4a22379462d39c49a47f5f4a621dc47e92f5f40b6b8323c290eed03a63381008f6e318da
224343bd2543ce5624c801752f4c6227d0199c9b11808248e89e28277d699f2b12f0ecd951747
ad11cd7654cf8290574c91a31323b5999c800f9be669ac
        EAP-Message =
0xcbd75d123a9b2dfb03bc98fac4f849a2dc92b19708da59a8a63dc8f6d51c4048c45cf6e3231
a980f6e17b5435a82814670c618bee2880432721e103b06f7c347feae2a3dbd4b379ef1d07f2d
0286fadd79a9d04b7d6583f81f72dce4ed61c8b2165600649810d4a2fa03141e2f2a9cb27b953
08e3aa31d77993c245490231d6dcaa60eb301090203010001a37b307930090603551d13040230
00302c06096086480186f842010d041f161d4f70656e53534c2047656e6572617465642043657
2746966696361746 5301d0603551d0e04160414a2fb7f6ab0fa2d1abc4ae9f8db4ebea07f90c0
8a301f0603551d230418301680146554cd608447278169
        EAP-Message =
0x4d5b83491a51299286a07e300d06092a864886f70d0101050500038201010 0b93663a041819
8eaec0d59e47aacf5b9e3150575b443685352c02e0a1f36187f9d313c37a16dd079148b33c740
685d4301ca59cfba579a3014055e5c6d82b23653957994474e2951e08653031a20b698170c939
f8d09b029273a991c981dbd9a4dcae7402d581b9439660eeb980ddecc00a4415bce9af6bb532e
abce9381e6108eee50eccdf01ce28b2a2fcd52e158838c15fd37742fd308cd8a433e347534d7b
7c42159d65deced195f9fba4fe37fffc531db30deb02b86118ca7d910035fa9fc91f1703fb38a
85c5da48a3fe37cfe1db6312355fe9545dc4bd260a9154
        EAP-Message =
0x913edef9034a252e921b0320efada52a52061072699c63f6fc7d88b5862691116e3d00047b3
08204773082035fa003020102020900e3b8a41f595c46a9300d06092a864886f70d0101050500
308183310b300906035504061302e4f311630140603550408130d536f722d54726f6e64656c6
167311230100603550407130954726f6e646865696d310d300b060355040a13044e544e55310f
300d060355040b13064469706c6f6d310b300906035504031302434131b301906092a864886f
70d010901160c6361406469706c6f6d2e6e6f301e170d303730333031313330373438 5a170d31
303032323831333037343 85a308183310b300906035504
```

```
        EAP-Message =
0x0613024e4f3116301406035504081 30d536f722d54726f6e64656c616731123010060355040
7130954726f6e646865696d310d300b060355040a13044e544e55310f300d060355040b130644
69706c6f6d310b300906035504031 3024341311b301906092a864886f70d010901160c6361406
469706c6f6d2e6e6f30820122300d06092a864886f70d010101050003
        State = 0x3f7bd742bba3aaa475d55c8949c32f46
        Message-Authenticator = 0xfa7169a1111927ba3f4fcee91c00b5dd
Sending Access-Challenge of id 4 to 129.241.208.206 port 1148
        EAP-Message = 0x010600060d00
        Message-Authenticator = 0x00000000000000000000000000000000
        State = 0xef6264734d4b59257df40968f3174f66
rad_recv: Access-Request packet from host 129.241.208.206:1148, id=5,
length=1365
        User-Name = "Supplicant"
        NAS-IP-Address = 129.241.208.206
        NAS-Port = 0
        Called-Station-Id = "00-0F-CB-B5-0E-D2:Kenneth"
        Calling-Station-Id = "00-0F-CB-B3-67-11"
        Framed-MTU = 1400
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 0Mbps 802.11"
        EAP-Message =
0x020604ab0d0082010f003082010a0282010100bdf734a0729543d256f5adf7e5e2cbc46f4fb
95337a380e8d9f08fe3abce89e4b139775a1a1453945bfbc2c984a0035bde4d2454da5fee7d50
53c6dd2e29ffafadd804db8d18f0da2d57ee58fcb612dc835d8ee8433344b13aaef25dd25a2ae
eb32a90eca77d85a505aa91c7db7748e69f63c9fda578a85d470df401d0408c7abaad4bb4dc3e
1287a3d91756825626ec938250733bcc974df9d78c1b744d8d4d810c33160282797c413361f71
8653cb0595cf822cad0c3ecfbf40bf290f4dfb2ce10c7e9cf77f274a72d1ced3673fe302d33c5
10104d41e6085336c104e93c4917c186d2bdff7170df4f
        EAP-Message =
0xd4b52e35d6c24fe7e95d67444d4ee7a1e01029d5216d0203010001a381eb3081e8301d06035
51d0e041604146554cd6084472781694d5b83491a51299286a07e3081b80603551d230481b030
81ad80146554cd6084472781694d5b83491a51299286a07ea18189a48186308183310b3009060
355040613024e4f3116301406035504081 30d536f722d54726f6e64656c616731123010060355
0407130954726f6e646865696d310d300b060355040a13044e544e55310f300d060355040b130
64469706c6f6d310b300906035504031 3024341311b301906092a864886f70d010901160c6361
406469706c6f6d2e6e6f820900e3b8a41f595c46a9300c
        EAP-Message =
0x0603551d13040530030101ff300d06092a864886f70d01010505000382010100015049 25d51
109f5b65ead2899f15ad1b0f9f3a8e1cd973a7b3a87da431ad11ca569b122005ae0abed60b863
e97bc0d153c381968abd5a60c422ec7ace0bd5240ce2abb9d5b740276134cf956b868ba465614
300968775be10b4d72e423657a991eebcc11e6cd447698d944dfa4cfc624c38d7defba4ec610a
5e387ccbbb1338a2eed03a2c8c070c854cd8133545d0a7a96e8db7a32edd7a1b8748485049714
e27b1b1a012f7b965ad4b7b0cb05ffbe9f6bbe30517d03ed5a74eb300ec6f688823ae6523956f
e0b79ec99eeb99a2900e721c887094340a1bcdd0194b43
```

```
        EAP-Message =
0x349c4e73478d2ceba5a5b549e73230e7fc6bdccc03c0606b8fb133f3202c4f870d128c16030
10046100004200406ee748d71f8aae569092e031d13a9bc0011bdf7b01ba414b31f6b3b402a3
a490e3cc9ab89989390528de8a5e557610f1d33bf4502b84f5af16c17f6aa8656bff160301010
60f00010201004a4501c90c06b5de58c2ee96937f35a05cca8dca4e849b021d550c95dd933fca
f297e21402db945565b5e800662cf9bad483835ee6949884a2471ffe9b4d72200f5e7810c4a02
3b5313acfa4c24f3be4cf8e44df9dec8a603605d4a0ba602a1176c5072d5fe57c59d23dbeda77
241f8b84c71b8494b676893cceadcc75db1e212feee1e4
        EAP-Message =
0x3c8b753a4c7907e9ad42139fd1b4122de20a3a8de2cdf8f094eb7acca600e5fd0fda4c1b6f3
c420e51a6ba0da888655a22907906a87ff1596fd9c9504b550aa51f59a0f369388dac450f0ca0
dc665cd309d2b7240782d0962401e6cbb36711f00c4a7543afe4a731896e4de9bd454a7fa43c2
74678d452b88281c040140301000101160301003047f71bde887800fb0b960818ce5702488600
1c780713bb77028cbab2b0cc9cc1fcd7ec9fd4a90a5cb47849beca93d4c0
        State = 0xef6264734d4b59257df40968f3174f66
        Message-Authenticator = 0x9198a3f4b889c22cf86dc51021c94c75
chain-depth=1,
error=0
--> User-Name = Supplicant
--> BUF-Name = CA
--> subject = /C=NO/ST=Sor-
Trondelag/L=Trondheim/O=NTNU/OU=Diplom/CN=CA/emailAddress=ca@diplom.no
--> issuer  = /C=NO/ST=Sor-
Trondelag/L=Trondheim/O=NTNU/OU=Diplom/CN=CA/emailAddress=ca@diplom.no
--> verify return:1
chain-depth=0,
error=0
--> User-Name = Supplicant
--> BUF-Name = Supplicant
--> subject = /C=NO/ST=Sor-
Trondelag/O=NTNU/OU=Diplom/CN=Supplicant/emailAddress=supplicant@diplom.no
--> issuer  = /C=NO/ST=Sor-
Trondelag/L=Trondheim/O=NTNU/OU=Diplom/CN=CA/emailAddress=ca@diplom.no
--> verify return:1
rlm_eap: SSL error error:00000000:lib(0):func(0):reason(0)
Sending Access-Challenge of id 5 to 129.241.208.206 port 1148
        EAP-Message =
0x010700450d800000003b1403010001011603010030271 2fb4cb272b4ecb759f925545da28f6
e96a49f250d0bc0ede569beffc7c2a393b4d255682c6827b9aae410ddb82ca6
        Message-Authenticator = 0x00000000000000000000000000000000
        State = 0xa72044ec99d7b8bddaab5c819a276f3b
rad_recv: Access-Request packet from host 129.241.208.206:1148, id=6,
length=168
        User-Name = "Supplicant"
        NAS-IP-Address = 129.241.208.206
        NAS-Port = 0
```

```
        Called-Station-Id = "00-0F-CB-B5-0E-D2:Kenneth"
        Calling-Station-Id = "00-0F-CB-B3-67-11"
        Framed-MTU = 1400
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 0Mbps 802.11"
        EAP-Message = 0x020700060d00
        State = 0xa72044ec99d7b8bddaab5c819a276f3b
        Message-Authenticator = 0xb5eeb72302717662379e161e178e4a0f
Sending Access-Accept of id 6 to 129.241.208.206 port 1148
        MS-MPPE-Recv-Key =
0x24b64f316386d5b9817a424a667125fe2de0ae75a1f017908aa51a23f092dfb3
        MS-MPPE-Send-Key =
0x0ac6c599f2a23a23802b886865fdf2cffc50c898541a4e0d33d09a2e90f7adb0
        EAP-Message = 0x03070004
        Message-Authenticator = 0x00000000000000000000000000000000
 User-Name = "Supplicant"
```