# Dataintegritet i WiFi

Vi har nu gått igenom ett antal grunddefinitioner av trådlösa nät, tittat på olika standarder för trådlös kommunikation (kring t.ex. HiPerLAN, LTE, WiMax, WiFi etc.) och sedan gjort djupdykningar i de olika tekniker som idag kallas WiFi.

Vi har ju definierat två grundproblem med radiobaserade nätverk (oavsett om det är WiFi eller något annat vi pratar om) – nämligen dels hur man autenticerar och auktoriserar användaren i nätverket – för att trafiken inte ska kunna avlyssnas, men också ett problem med integriteten – att ett meddelande (möjligen) kan förändras mellan sändare och mottagare. I första lektionen pratade vi om att i radionät (andra än WiFi) har man löst de problemen med t.ex. frekvensbandhopp.

Förra kapitlet pratade vi om hur vi på olika sätt kan säkra själva accessen till nätet, och säkra accessen till datat – med de olika säkerhetsstandarderna WEP, WPA (WPA2 och WPA3) och 802.1x – något som vi kommit fram till är ett av ”grundproblemen med trådlösa nätverk”

Nu ska vi titta lite mer på själva integriteten i datat på det som skickas i vårt trådlösa nät och vad man kan göra åt det.

När man pratar ”dataintegritet” menas rent definitionsmässigt att A) datat kan kontrolleras från sändare och mottagare – och att de kan se att datat kommer från rätt källa samt B) att datat inte förändrats ”under färden” mellan sändare och mottagare. Datakom-historiskt görs detta genom att meddelandet på olika sätt ”taggas” med ett kontrollmeddelande. Kontrollmeddelandet har på olika sätt information om vem som skickat meddelandet, någonting som kan kontrollera ursprungsmeddelandet och det inte har ändrats, samt i vissa fall information om eventuell felhantering (att meddelandet t.ex. är en omsändning av ett tidigare förlorat meddelande). För att skapa ett sådant kontrollmeddelande används normalt sett olika matematiska formler för ”message digest” – att komprimera text genom att använda checksummering.

Då matematiken som används vid message digest (och andra checksummeringsfunktioner) gravt påminner om kryptologi brukar viss begreppsförvirring uppstå – men ett klargörande här är att kryptologi eller kryptering används för att vanställa texten så att den inte är igenkännbar för utomstående medan message digest egentligen bara är en kontroll att texten A) kommer från rätt avsändare och B) inte har förändrats på vägen.

Vissa kryptostandarder har stöd för message digest, medan andra inte har det. Message digest (och liknande kontrollfunktioner) kan emellertid också användas då texten sänds i klartext.

***I MÅN AV TID: Här en teoretisk genomgång matematiskt av kryptering, och meddelandekontroll, rent matematiskt. Den är mycket träig och jag rekommenderar den egentligen inte till de som inte är utpräglade nördar, men den kan vara bra som referens till diskussionen ovan:***

<https://www.youtube.com/watch?v=xKtmNb6dU4k>

## Dataintegritet i WEP

När vi härom dagen pratade om WEP pratade vi egentligen bara om krypteringsfunktionerna i WEP och nämde då att man använde ett krypto för strömmande datablock med en fast – ”pre shared” key – att nyckeln är fast och inte ändrar sig. Kryptot heter RC4 och implementerat med de nyckellängder som användes i WEP blev det ett väldigt ”svagt” krypto. WEP-standarden hade emellertid även ett system för ”message digest” och kontroll av att datat inte förändrat sig – en standard som heter ICV – Integrity Check Value.

Då hårdvaran som fanns för tiden då WEP blev en standard (och då standarden skulle ”motsvara” de krav som fanns i kabelbundet Ethernet) ska emellertid ICV ses ungefär som ”frame checksum” i Ethernet. Problemet är att checksumman egentligen erbjuder väldigt få – så som i ”i princip inga” skydd mot att förändras under gång. Det finns t.ex. inga möjligheter att kontrollera huruvida checksumman själv har förändrats – vilket har setts som ett av de stora problemen med just meddelandeintegriteten i WEP.

Checksumman räknas ut \_i klartext\_ på ramen \_i klartext\_ före den krypteras – och påförs ramen som den väldigt lilla meddelandesträngen ICV. Algoritmen som används för att räkna ut ICV heter CRC32 (där CRC står för Cyclic Redundancy Check och 32 är antalet bitar som används i meddelandet. 32 bitar betyder i praktiken 4 bytes, eller tecken, för checksumman) – en algoritm som från början är utvecklat för felhantering av t.ex. ”oljud” eller ”brus” på ackustiska lagringsfunktioner (som magnetband). För att uttrycka sig milt kan man säga att just CRC-algoritmen är bättre till kontroll av oljud (och att ta bort det) än det är för ”message digest”, då antalet bitar i meddelandet helt enkelt är för få för att genomföra någon djuplodande kontroll av meddelandet. Själva kontrollmeddelandet tillfördes slutet på ett pakets payload – varför det var enkelt för en attackerare att helt enkelt antingen bortse från det eller påföra nya checksummor (och i klartext) på ett paket för att låta påskina för sändare och mottagare att paketet var oändrat.

## Dataintegritet i WPA – MIC/MIChael

När WPA kom var ju, som vi pratade om i förra lektionen, ett av grundkraven att ingen ny hårdvara skulle behöva användas för att använda standarden. Man hade emellertid lärt sig (den hårda vägen) att WEPs implementationer av säkerhetsfunktioner som kryptering (i form av RC4) och dataintegritet (med ICVs CRC32-algoritm) inte fungerade tillfredställande varför man gjorde implementationen med TKIP – ett system som å ena sidan består av \_samma\_ kryptoalgoritm som i WEP (med RC4), men med en längre nyckel och nya funktioner för att dynamiskt räkna fram själva nyckeln beroende på bl.a. sändarens och mottagarens MAC-adresser. I TKIP-systemet används därför systemet MIC (Message Integrity Check) i stället för ICV för att kontrollera integriteten på meddelanden som skickas i nätet.

I WEP med ICV finns en möjlighet för man-in-the-middle-attacks genom vad man kallar ”the bit flip attack”. Rent matematiskt ändras den sista biten i kontrollmeddelandet och kontrollen utförs, fastän den borde falera, och ett meddelande kan framstå som om det kommer från en annan avsändare än det ”borde” eller så kan meddelandet framstå som oförändrat ”fastän det är det”. (Matematiken bakom själva attacken är rätt bökig, så jag kommer inte gå igenom det på djupet i den här kursen).

I och med WPA och TKIP kom möjligheten att använda längre checksummor och med MIC har man funktionaliteten att både A) data ”payload” – alltså meddelandet inom varje paket och B) data ”header” – alltså sändare och mottagare och olika kontrollfunktioner i paketet.

I designen av MIC har man tagit hänsyn till hårdvarans begränsningar. Problemen löstes genom att man uppdaterade ”firmware” i samma hårdvara man tidigare kört WEP i. Det använder en algoritm som ger 64 bitars checksummar och använder sändarens och mottagarens MAC-adresser samt meddelandets text (Data Payload) som in-värden i beräkningarna. Checksumman (Message Digest) anses då inte obemärkt kunna manipuleras då det bygger på både delar av huvudet och på datats payload.

Den stora fördelen med MIC är att själva nyckeln till Message Digest (som ju är framräknat på både MAC-adresserna och Data Payload) är att det är krypterat – även detta – med de nycklar som räknats fram av TKIP.

MIC innehåller en mängd ”motmedel” mot att meddelandet har förändrats – som att kopplet kopplas ned, nycklarna räknas om och stationerna som är inblandade i kommunikationen blir ”blockade” med ett temporärt MAC-adress-filter.

Medialänk: <https://www.youtube.com/watch?v=pTeDxEiYtyw>

## Dataintegritet i WPA2 – CCMP

Vi har nu tittat på att WEP och WPA faktiskt hade funktioner för dataintegritet (i och med ICU och MIC) – även om vi pratade om dem när vi pratade WEP och WPA första gången. Vi berörde emellertid att i WPA2 används andra kryptosystem – som ställer högre krav på routerns eller AP:s CPU – nämligen AES. Till AES finns en mängd olika funktioner för att se till att dataintegriteten säkerställs, och WPA2 kom CCMP.

CCMP kan på olika sätt – i ännu högre grad än TKIP med MIC – säkerställa framförallt avsändare och mottagare av ett meddelande är korrekta. CCMP används dessutom med längre nyckellängder (128 bitar i stället för de 64 som används inom TKIP) och krypteras med AES i stället för RC4 – vilket ytterligare höjt säkerheten.

Medialänk: <https://www.youtube.com/watch?v=8hTGoSiRcy0>

## Dataintegritet i 802.1x – EAP (EAP-TLS, EAP-PEAP och EAP-FAST)

Som vi pratade om när vi pratade säkerhetsstandarder i förra kapitlet/kurstillfället så är modernare standarder mer inriktade på att skydda och/eller kontrollera själva användaren – eller ”användandet” om man så vill – av nätverket, snarare än tekniken själv.

I och med introducerandet av EAP i 802.1x har man på olika sätt försökt att höja användarens dataintegritet genom att faktiskt kontrollera sändare och mottagare i högre grad. Detta görs genom att man lägger ut tjänsten att autenticera användaren ur själva säkerhetsstandarden som körs (t.ex. WPA-2) och ut till en extern server (som kör t.ex. RADIUS). Det gör att användaren i högre grad kan kontrolleras i autenticeringsservern – och med mer komplixa autenticerings (användarkontroll) och auktorisations (tjänstekontroll/”skiktning”)-kontroller än vad standarden i sig själv, så som den är implementerad på routern eller accesspunkten medger.

Det gör att man får ett routern/accesspunkten blir ett mellansteg mellan användarens klient och autenticeringstjänsten. Klienten kan verifieras innan den skickas vidare i nätet. I EAP-sammanhang brukar klienten kallas för ”supplicant”. Accesspunkten eller routern blir då ”authenticatior” och servern som kör t.ex. radius kallas ”Authentication server”.

### EAP-TLS

I EAP-TLS använder både Supplicant och Authentication Server ett certifikatbaserat autenticeringssystem. När autenticeringen börjar skickar AS sitt certifikat till Supplicant. Supplicant kommer att använda certifikatet från AS som kryptonyckel, för att kryptera sina svar tillbaka till servern. Genom det här certifikatet – som ju kan kontrolleras – kan klienten se att AS är det system det utger sig för att vara och som *klient autenticera AS*.

Detta sker även åt andra hållet – där klienten skickar sitt certifikat mot AS och AS kan använda detta dels som en kryptonyckel att kryptera trafik tillbaka till klienten, men kan också verifiera att klienten är den den utger sig för att vara.

För att kontrollera serverns certifikat/nyckel hos klienten och vice versa behövs ytterligare en tjänst introduceras – en tjänst som både sändare och mottagare litar på – som kan säkerställa att klientens och serverns certifikat faktiskt stämmer. En Certification Authority, eller CA. En CA kan sedan ”signera” respektive certifikat – genom att lägga på en del i certifikatet/nyckeln av sitt eget certifikat/nyckel – kallad en ”signatur”. Själva signaturen av respektive certifikat är alltså en checksumma/message digest av respektive certifikat/nyckels innehåll, samt en bit av CA nyckel.

Detta kan således sägas vara en standard för att kontrollera och autenticera användaren (och hens tillgång till nätet) men också vara en standard för att höja dataintegriteten – och se till att datat inte påverkats under tiden – eftersom datat både är krypterat (från en motpart som du lämnat ut nyckeln till – och kunnat verifiera genom en oberoende tredje part som ni bägge litar på) OCH varje nyckel är verifierad genom den tredje partens signatur.

Medialänk: <https://www.youtube.com/watch?v=pPfwemHBblk>

### EAP-PEAP

Då EAP-TLS kräver att alla klienter i nätverket innehar ett personligt – unikt – certifikat och att detta är signerat av en CA som både klient och server litar på blir hanterandet av just klientcertifikat tämligen snabbt – för en nätverksadministratör – betungande att ansöka om, certifiera, signera och – då ”bäst före-datumet” går ut – förnya. EAP-PEAP är utvecklat för att förenkla användandet av EAP i storskaliga nätverk med många klienter.

Slarvigt uttryckt kan sägas att EAP-PEAP ”fejkar” eller ”emulerar” delar av EAP-TLS-processen.

Processen kan sägas vara ”förenklad” på det sätt att de certifikat som behövs för handskakningen EAP-TLS här bytts ut mot kryptonycklar – men som inte behöver vara signerade av en oberoende CA. De här nycklarna kommer uppfattas av både klient och server som om de vore ”riktiga certifikat” men den används i EAP-processen endast som kryptonycklar, för att kryptera trafiken som skall sändas antingen från servern till klienten (som då är krypterat med klientens nyckel) eller från klienten till servern (och är då krypterat med serverns nyckel).

I EAP-PEAP har man i stället gjort så att man efter handskakningen (och utbyte av ”certifikat” – eller i det här fallet rättare uttryckt ”nycklar” – har skett) har man en krypterad tunnel (åt bägge håll) mellan klient och server. I denna tunnel sker sedan en till EAP-autenticering – alltså en autenticering till inom tunneln. EAP är skyddat med EAP igen – alltså EAP Protected EAP.

I den andra (redan krypterade) EAP-fasen skickar klienten inte ett ”fejkat certifikat” igen, utan i stället skickas här andra ”cridentials”. Dessa kan vara användarnamn och lösenord, något slags engångslösenord eller liknande. Då det redan finns en etablerad kryptotunnel mellan klient och server kommer datat i den andra autenticeringen vara krypterat och därför ses som ”säker” för utväxling av nya kryptonycklar för vidare kommunikation.

Den här tekniken har blivit mycket populär då den är \_nästan\_ lika säker som EAP-TLS (för alla utomstående ser det här ut som en EAP-TLS-session), men den kräver mycket mindre bandbredd, kontroll av certifikat och administration, då det enda som behövs är ett certifikat hos AS, att skicka ut som nyckel till klienten/supplicant.

### EAP-FAST

Det har funnits andra implementationer av olika EAP-protokollstandarder. En av dessa är EAP-FAST (EAP Flexible Authentication over Secure Tunnels). EAP-FAST kom som ett svar på tidigare standarder inom EAP-familjen. I WEP hade man kunnat använda LEAP (Lightweight EAP) och när standarderna ritades om i och med WPA-standarderna introducerade en nätverkshårdvaruleverantör – Cisco – en egen EAP-standard – EAP-FAST.

Denna standard ska ses som en ”mellanstandard” mellan LEAP och EAP-TLS både funktionellt och tekniskt. EAP-FAST är vad förkortningen säger – snabbt och ”lättviktigt” (och utan tyngre kontrollfunktioner som inblandning av CA) och kräver ingen speciellt snabb hårdvara – samtidigt som det faktiskt sköter själva EAP-processerna inom en krypterad tunnel.

EAP-FAST har blivit allt mindre populärt genom åren – dels pga att EAP-TLS idag med lätthet körs i modern hårdvara, men också att skälen för att inte använda CA och ändå ha en krypterad tunnel lösts av det modernare EAP-PEAP.

Medialänk: <https://www.youtube.com/watch?v=N5A1NsE2nhM>