Engineer

TLS

protokoll for autentiserte, krypterte forbindelser

Hva det er

TLS, kort for Transport Layer Security, er protokollen en klient og en server bruker for å beskytte applikasjonsdataene mot avlytting og ubemerkede endringer og for å autentisere den andre enden. Vanligvis autentiserer klienten serveren; autentisering av klienten er valgfri. Produktmenyer og sertifikatbutikker kaller det fortsatt SSL, etter Secure Sockets Layer, protokollen TLS avløste. Den siste SSL-versjonen ble erklært foreldet i RFC 7568 i juni 2015, så alt som heter SSL i dag, er TLS under panseret, eller burde være det.

TLS ligger mellom en applikasjonsprotokoll og transportlaget under den. HTTPS er HTTP fraktet inne i en TLS-forbindelse, og det samme laget beskytter innsending av e-post, IMAP, databaseforbindelser og Gemini-protokollen. HTTP/3 går et steg lenger: det kjører over QUIC, som bygger håndtrykket fra TLS 1.3 inn i selve transporten i stedet for å legge det oppå TCP, slik RFC 9001 beskriver.

Gjeldende versjon er TLS 1.3. RFC 9846, publisert i juli 2026, oppdaterer spesifikasjonen uten å endre versjonsnummeret; den erstatter RFC 8446 fra august 2018. En vanlig sertifikatbasert forbindelse åpner med et håndtrykk. Klienten sender en ClientHello med versjonene, chifferpakkene og nøkkelandelene den støtter, vanligvis sammen med Server Name Indication (SNI), vertsnavnet den vil nå, slik at én adresse kan betjene mange nettsteder. Serveren velger og sender sin ServerHello. Håndtrykksmeldingene som følger, er kryptert, også sertifikatet den legger fram. Sertifikatkjeden knytter serverens nøkkel til en sertifiseringsinstans klienten allerede stoler på. Klienten sjekker kjeden, navnet og datoene, og serveren beviser at den har den tilhørende private nøkkelen ved å signere håndtrykket. En gjenopptatt forbindelse kan i stedet autentiseres med en forhåndsdelt nøkkel som ble etablert tidligere.

Tre ting skiller 1.3 fra 1.2. Et vanlig fullt håndtrykk tar én rundtur i stedet for to, men et nytt forsøk kan legge til enda en. Nøkkeletablering med offentlige nøkler gir forward secrecy: et opptak av dagens trafikk forblir forseglet selv om serverens langsiktige private nøkkel lekker neste år. Modusen med bare en forhåndsdelt nøkkel er et unntak og gir ikke forward secrecy. Og en klient som kommer tilbake, kan sende «early data» allerede i sin aller første melding, kjent som 0-RTT, før håndtrykket er ferdig, til en pris RFC-en gjør rede for: tidlige data har verken forward secrecy eller vern mot å bli spilt av på nytt over en annen forbindelse.

Bruk begrepet i en beslutning

Bruk «Hva er TLS?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Kontoer, passord og tilganger ordnet slik at folk har nøyaktig den tilgangen jobben krever, og de som slutter, mister den samme dag.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Skrev 1 000 RBAC-regler for å sikre Linux-servere — 1,000 tilgangsregler skrevet for å tette et sikkerhetshull

Skrev 1 000 RBAC-regler for å sikre Linux-servere Ta kontakt →

Hvorfor det er viktig

At TLS skal brukes, er avgjort for lengst. Det som gjenstår, er hvilke versjoner du godtar, og hvor den krypterte forbindelsen slutter.

Først versjonene. RFC 8996, publisert i mars 2021, sier at TLS 1.0 og 1.1 ikke skal brukes. BCP 195 omfatter RFC 9325 og oppdateringene fra juli 2026. RFC 9852 krever at nye protokoller som bruker TLS, støtter 1.3 og bruker den som standard; 1.2 tillates som et ekstra alternativ som ikke er standard, av hensyn til interoperabilitet. Det er ikke et generelt krav om å bruke bare 1.3 i alle utrullinger. RFC 10015 forbyr også RSA-nøkkelutveksling og Diffie–Hellman-nøkkelutveksling over endelige kropper i TLS 1.2, også den flyktige varianten; dette forbyr ikke RSA-signaturer på sertifikater. For en eksisterende tjeneste som trenger eldre klienter, bør bare sikkert konfigurert 1.2 beholdes ved siden av 1.3, med ECDHE og autentisert kryptering som AES-GCM. En tjeneste med bare 1.3 har mindre angrepsflate og kortere konfigurasjon, og den stenger ute alle klienter som høyst snakker 1.2: en eldre telefon, en kortterminal, en partners integrasjon på en gammel Java-runtime. Logg den forhandlede versjonen i noen uker, og bestem deg ut fra tallene heller enn ut fra hva som høres moderne ut.

Deretter termineringen. Der TLS slutter, ligger den private nøkkelen, og trafikken blir klartekst igjen. Tre steder er vanlige:

- **Et CDN eller en lastbalanserer i skyen.** Leverandøren har nøkkelen, eller et sertifikat den har utstedt selv, tar belastningen fra håndtrykkene og ser hver forespørsel i klartekst. Strekningen derfra til opprinnelsesserveren trenger sin egen TLS, ellers går den ukryptert over nettet.
- **En omvendt proxy på verten,** f.eks. nginx, Apache, HAProxy eller Caddy. Én prosess har nøklene for alle nettstedene på maskinen og sender forespørslene videre i klartekst til applikasjonene bak, over loopback eller en lokal socket.
- **Selve applikasjonen.** Hver tjeneste har sin egen nøkkel. Det passer for trafikk mellom tjenester og for gjensidig TLS (*mutual TLS*), der også klienten legger fram et sertifikat, og det mangedobler antallet steder der et sertifikat må fornyes.

De fleste driftsmiljøer blander alle tre. Det som teller, er at noen for hvert ledd kan si hvor klarteksten finnes, og hvem som kan lese den.

Slik ser det ut i praksis

Selskapets implementering av offentlig tilgjengelige tjenester fra september 2026 viser hvor lite TLS-terminering krever. Den brukte en skyinstans med én kjerne, 512 MB minne, hvorav omtrent 464 MB var brukbart, og 10 GB disk. På den terminerte webserveren TLS for sju domener ved siden av git-forgen, en Tor-onion-tjeneste, en Gemini-server, en Gopher-server og utestenging av inntrengere. Automatisering av sikkerhetskopiering var implementert separat; det fastslo ikke at krypterte sikkerhetskopier var aktive. Dette var terminering på verten i sin enkleste form: én webserver svarte for alle sju nettstedene, og krypteringen deres kostet verken en ekstra maskin eller en ekstra tjeneste. Utformingen behandlet minnet som den bindende begrensningen, og den dokumenterte utrullingen kjørte innenfor disse 464 MB.

Regelen selskapet arbeider etter, står på sikkerhetssiden: kryptering under overføring overalt, med sertifikater som utstedes og fornyes automatisk, og en fornyelse som overvåkes. Begrunnelsen der er at TLS oftest svikter på et sertifikat ingen fulgte med på. Feilene nedenfor er de som er igjen når fornyelsen går av seg selv.

Hvor det går galt

Noen klienter faller av med et mislykket håndtrykk, mens nettleserne kobler til uten problemer. Endene har ingen felles versjon eller chifferpakke. Noen har snevret serveren inn til bare 1.3, eller til en kort liste med chifferpakker, og en eldre klient tilbyr bare det serveren har strøket. Eller en server som fortsatt bare tilbyr CBC-pakker, møter en klient som har sluttet å støtte dem. Varselet sier «handshake failure» eller «protocol version» og lite mer. Test hver versjon utenfra:

```
openssl s_client -connect example.com:443 \
    -servername example.com -tls1_2 </dev/null
```

Logg den forhandlede versjonen og chifferpakken før du snevrer inn noe. På en eksisterende tjeneste med klienter som fortsatt trenger 1.2, behold sikkert konfigurert 1.2 med ECDHE og AES-GCM ved siden av 1.3 mens overgangen planlegges. For en ny protokoll som bruker TLS, støtt 1.3 og bruk den som standard; aktiver 1.2 bare når interoperabilitet krever det.

Nettleseren stoler på sertifikatet, mens skript, curl og mobilapper avviser det. Serveren sender sitt eget sertifikat uten mellomsertifikatet som knytter det til roten. Nettlesere fyller ofte hullet med mellomsertifikater de har møtt før, eller henter det manglende selv. De fleste TLS-biblioteker gjør ingen av delene og melder om en ukjent utsteder. Sett opp hele kjedefilen, serverens sertifikat først og deretter mellomsertifikatene, og test med en klient som ikke har noen cache: linjen med `openssl s_client` ovenfor skriver ut kjeden serveren faktisk sendte.

Sertifikatsjekken slås av «inntil videre», og innstillingen havner i produksjon. Det begynner med en testserver med et selvsignert sertifikat, eller en bedriftsproxy som signerer trafikken på nytt, og så legger noen til `verify=False`, `curl -k`, `InsecureSkipVerify` eller `NODE_TLS_REJECT_UNAUTHORIZED=0`. Forbindelsen er fortsatt kryptert, men til den som svarer: hvem som helst i mellom kan legge fram sitt eget sertifikat og lese alt. Legg det riktige CA-sertifikatet i klientens tillitslager, eller pek klienten eksplisitt på det, og la CI feile på disse flaggene.

Noen besøkende får et sertifikat for et annet nettsted. Serveren legger fram standardsertifikatet sitt når en ClientHello navngir en vert den mangler konfigurasjon for, eller når den ikke navngir noen vert i det hele tatt. Det skjer

når et nytt domene pekes mot adressen før serverblokken finnes, og når en overvåkingsprobe eller en gammel klient sender uten SNI. Nettleseren melder at navnet avviker, og på en delt vert avslører feilen også hvilke andre nettsteder som bor der. Gi hvert offentlige navn sin egen blokk og et sertifikat som dekker det, og la standardblokken avvise håndtrykket i stedet for å vise fram en fremmeds sertifikat.

En betaling, eller en hvilken som helst annen forespørsel, blir utført to ganger. Termineringspunktet godtar tidlige data over 0-RTT, en POST kommer inne i dem, og en angriper eller et nytt forsøk i nettverket sender de samme bytene igjen over en ny forbindelse. TLS 1.3 tillater tidlige data, men kan ikke garantere vern mot gjentakelse mellom forbindelser. Slå enten av tidlige data, vent med å handle på dem til håndtrykket er fullført, eller avvis forespørsler som ikke trygt kan gjentas, med 425 Too Early, slik RFC 8470 angir. En gateway må også sende Early-Data-markeringen videre og vite at opprinnelsesserveren forstår den: å vente på et senere ledd fjerner ikke gjentakelsesrisikoen fra et tidligere ledd.

Les mer

- RFC 9846: The Transport Layer Security (TLS) Protocol Version 1.3 — spesifikasjonen fra juli 2026, som erstatter RFC 8446 uten å endre protokollens versjonsnummer 1.3; håndtrykk, poster og begrensninger ved tidlige data.
- RFC 9325: Recommendations for Secure Use of TLS and DTLS — anbefalingene om utrulling i BCP 195, lest sammen med oppdateringene som følger.
- RFC 9852: New Protocols Using TLS Must Require TLS 1.3 — obligatorisk støtte for 1.3 og bruk som standard i nye protokoller, med valgfri interoperabilitet med 1.2; oppdateringen gjelder ikke DTLS.
- RFC 10015: Deprecating Obsolete Key Exchange Methods in TLS 1.2 and DTLS 1.2 — forbudte nøkkelutvekslinger med RSA og Diffie–Hellman over endelige kropper, skilt fra sertifikatsignaturer og fra nøkkeletablering i TLS 1.3.
- RFC 8996: Deprecating TLS 1.0 and TLS 1.1 — den formelle slutten for de to eldre versjonene og begrunnelsen for den.
- RFC 8470: Using Early Data in HTTP — Early-Data-headeren, koordinering med gateways og statuskoden 425.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er TLS?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/tls/