Engineer

# TLS

protokol til autentificerede, krypterede forbindelser

## Hvad det er

TLS, Transport Layer Security, er den protokol, en klient og en server bruger til at beskytte deres applikationsdata mod aflytning og ubemærkede ændringer og til at autentificere den anden ende. Normalt autentificerer klienten serveren; autentificering af klienten er valgfri. I produktmenuer og hos certifikatforhandlere hedder det stadig ofte SSL efter Secure Sockets Layer, den protokol, TLS afløste. RFC 7568 erklærede den sidste SSL-version for forældet i juni 2015, så alt, der i dag kaldes SSL, er TLS nedenunder, eller burde være det.

TLS ligger mellem en applikationsprotokol og transporten under den. HTTPS er HTTP inde i en TLS-forbindelse, og samme lag beskytter afsendelse af mail, IMAP, databaseforbindelser og Gemini-protokollen. HTTP/3 går et skridt videre og kører over QUIC, der bygger håndtrykket fra TLS 1.3 direkte ind i transporten i stedet for at lægge det oven på TCP, som RFC 9001 beskriver.

Den gældende version er TLS 1.3. RFC 9846, udgivet i juli 2026, opdaterer specifikationen uden at ændre versionsnummeret; den erstatter RFC 8446 fra august 2018. En almindelig certifikatbaseret forbindelse begynder med et håndtryk. Klienten sender en ClientHello med de versioner, cipher suites og nøgleandele, den understøtter, som regel med Server Name Indication (SNI), det værtsnavn, den vil have fat i, så én adresse kan betjene mange sites. Serveren vælger og sender sin ServerHello. De efterfølgende håndtryksbeskeder er krypteret, også certifikatet, serveren viser frem. Certifikatkæden binder serverens nøgle til en certifikatudsteder, klienten allerede stoler på. Klienten kontrollerer kæden, navnet og datoerne, og serveren beviser, at den har den tilhørende private nøgle, ved at signere håndtrykket. En genoptaget forbindelse kan i stedet autentificeres med en på forhånd delt nøgle, som er etableret tidligere.

Tre ting adskiller 1.3 fra 1.2. Et almindeligt fuldt håndtryk kræver én tur frem og tilbage i stedet for to, men et genforsøg kan tilføje endnu en. Nøgleetablering med offentlige nøgler giver forward secrecy: en optagelse af dagens trafik forbliver lukket, selv hvis serverens langsigtede private nøgle slipper ud næste år. Tilstanden med kun en på forhånd delt nøgle er en undtagelse og giver ikke forward secrecy. Og en klient, der vender tilbage, kan sende »early data« allerede i sin første besked, kendt som 0-RTT, før håndtrykket er færdigt, til en pris, som RFC'en skriver ligeud: de data har hverken forward secrecy eller beskyttelse mod at blive afspillet igen på en anden forbindelse.

Brug begrebet i en beslutning

Brug »Hvad er TLS?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Konti, adgangskoder og rettigheder sat op, så folk har præcis den adgang, jobbet kræver, og fratrådte mister den samme dag.

**Se vores ydelser** Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har skrevet 1.000 RBAC-regler til Linux-servere — 1,000 adgangsregler skrevet for at lukke et sikkerhedshul

**Har skrevet 1.000 RBAC-regler til Linux-servere** Kontakt os →

## Hvorfor det betyder noget

At bruge TLS er afgjort for længst. Tilbage står to beslutninger: hvilke versioner man accepterer, og hvor den krypterede forbindelse ender.

Først versionerne. RFC 8996 fra marts 2021 siger, at TLS 1.0 og 1.1 ikke må bruges. BCP 195 omfatter RFC 9325 og dens opdateringer fra juli 2026. RFC 9852 kræver, at nye protokoller, der bruger TLS, understøtter 1.3 og bruger den som standard; den tillader 1.2 som en ekstra mulighed, der ikke er standard, af hensyn til interoperabilitet. Det er ikke et generelt krav om kun at udrulle 1.3. RFC 10015 forbyder også RSA-nøgleudveksling og Diffie–Hellman-nøgleudveksling over endelige legemer i TLS 1.2, også den flygtige variant; det forbyder ikke RSA-signaturer på certifikater. Til en eksisterende tjeneste, der skal betjene ældre klienter, bør kun sikkert konfigureret 1.2 beholdes ved siden af 1.3, med ECDHE og autentificeret kryptering som AES-GCM. En tjeneste med kun 1.3 har en mindre angrebsflade og en kortere konfiguration, og den lukker alle klienter ude, der højst taler 1.2: en ældre telefon, en betalingsterminal, en samarbejdspartners integration på en gammel Java-runtime. Log den forhandlede version i et par uger, og beslut ud fra tallene frem for ud fra, hvad der lyder moderne.

Dernæst termineringen. Der, hvor TLS ender, ligger den private nøgle, og der bliver trafikken til klartekst igen. Der er tre almindelige steder:

- **Et CDN eller en load balancer i skyen.** Udbyderen har nøglen eller et certifikat, den selv har udstedt, tager belastningen fra håndtrykkene og ser hver forespørgsel i klartekst. Strækningen derfra til origin-serveren skal have sin egen TLS, ellers krydser den netværket ukrypteret.
- **En reverse proxy på værten,** for eksempel Nginx, Apache, HAProxy eller Caddy. Én proces har nøglerne til alle sites på maskinen og sender forespørgslerne ukrypteret videre til programmerne bag sig over loopback eller en lokal socket.
- **Selve applikationen.** Hver tjeneste har sin egen nøgle. Det egner sig til trafik mellem tjenester og til mutual TLS, hvor klienten også viser et certifikat, og det mangedobler antallet af steder, hvor et certifikat skal fornys.

De fleste miljøer blander alle tre. Det afgørende er, at nogen for hver strækning kan sige, hvor klarteksten findes, og hvem der kan læse den.

## Sådan ser det ud i praksis

Virksomhedens implementering af offentligt tilgængelige tjenester fra september 2026 viser, hvor lidt TLS-terminering kræver. Den brugte en cloudinstans med én kerne, 512 MB hukommelse, hvoraf omkring 464 MB kunne bruges, og 10 GB disk. På den terminerede webserveren TLS for syv domæner side om side med git-forgen, en Tor-onion-tjeneste, en Gemini-server, en Gopher-server og blokering af indtrængningsforsøg. Automatisering af sikkerhedskopiering var implementeret separat; det dokumenterede ikke, at krypterede sikkerhedskopier var aktive. Det var terminering på værten i sin enkleste form: én webserver svarede for alle syv sites, og krypteringen af dem kostede hverken en ekstra maskine eller en ekstra tjeneste. Designet behandlede hukommelsen som den bindende begrænsning, og den dokumenterede udrulning kørte inden for de 464 MB.

Reglen, virksomheden arbejder efter, står på dens sikkerhedsside: kryptering under overførsel overalt, med certifikater, der udstedes og fornys automatisk, og en fornyelse, der overvåges. Begrundelsen er, at TLS oftest svigter på grund af et certifikat, som ingen overvågede. Fejlene nedenfor er dem, der er tilbage, når fornyelsen kører af sig selv.

# Hvor det går galt

**Nogle klienter bliver afvist med en håndtryksfejl, mens browsere forbinder fint.** De to ender har ingen version eller cipher suite til fælles. Nogen har strammet serveren til kun 1.3 eller til en kort liste af suiter, og en ældre klient har intet af det at byde på; eller en server, der stadig kun tilbyder CBC-suiter, møder en klient, der har droppet dem. Fejlbeskeden lyder »handshake failure« eller »protocol version« og siger ikke meget mere. Test hver version udefra:

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

Log den forhandlede version og suite, før noget strammes. På en eksisterende tjeneste, hvis klienter stadig behøver 1.2, beholdes sikkert konfigureret 1.2 med ECDHE og AES-GCM ved siden af 1.3, mens overgangen planlægges. En ny protokol, der bruger TLS, skal understøtte 1.3 og bruge den som standard; aktivér kun 1.2, når interoperabilitet kræver det.

**Browsere stoler på certifikatet, mens scripts, curl og mobilapps afviser det.** Serveren sender sit eget certifikat uden det mellemcertifikat, der forbinder det med roden. Browsere udfylder ofte hullet med mellemcertifikater, de har mødt før, eller henter selv det manglende; de fleste TLS-biblioteker gør ingen af delene og melder en ukendt udsteder. Konfigurér hele kædefilen, serverens eget certifikat først og derefter mellemcertifikaterne, og test med en klient uden cache: `openssl s_client` -linjen ovenfor viser den kæde, serveren faktisk sendte.

**Certifikatkontrollen slås fra »indtil videre«, og indstillingen ender i produktion.** En testserver med et selvsigneret certifikat eller en virksomhedsproxy, der gensignerer trafikken, og nogen tilføjer `verify=False`, `curl -k`, `InsecureSkipVerify` eller `NODE_TLS_REJECT_UNAUTHORIZED=0`. Forbindelsen er stadig krypteret, bare til hvem der end svarer: enhver undervejs kan vise sit eget certifikat og læse det hele. Læg det rigtige CA-certifikat i klientens tillidslager, eller peg klienten direkte på det, og lad CI fejle på de flag.

**Nogle besøgende får et certifikat til et helt andet site.** Serveren viser sit standardcertifikat, når en ClientHello nævner en vært, som den mangler konfiguration til, eller slet ikke nævner en vært. Det sker, når et nyt domæne peges på adressen, før dets serverblok findes, og når en overvågningsprobe eller en gammel klient udelader SNI. Browseren melder en navnefejl, og på en delt

vært afslører den samtidig, hvilke andre sites der bor der. Giv hvert offentligt navn sin egen blok og et certifikat, der dækker det, og lad standardblokken afvise håndtrykket frem for at vise et fremmed certifikat.

**En betaling eller en anden forespørgsel bliver udført to gange.** Termineringen tager imod 0-RTT early data, en POST kommer med i dem, og en angriber eller et genforsøg i netværket sender de samme bytes igen på en ny forbindelse. TLS 1.3 tillader early data, men kan ikke garantere beskyttelse mod genafspilning mellem forbindelser. Slå enten early data fra, vent med at handle på dem, til håndtrykket er færdigt, eller afvis forespørgsler, der ikke tåler genafspilning, med 425 Too Early, som RFC 8470 angiver. En gateway skal også sende Early-Data-markeringen videre og vide, at dens origin-server forstår den: at vente på et senere hop fjerner ikke risikoen for genafspilning fra et tidligere hop.

## Læs mere

- RFC 9846: The Transport Layer Security (TLS) Protocol Version 1.3 — specifikationen fra juli 2026, der erstatter RFC 8446 uden at ændre protokollens versionsnummer 1.3; håndtryk, records og begrænsninger ved early data.
- RFC 9325: Recommendations for Secure Use of TLS and DTLS — anbefalingerne til udrulning i BCP 195, læst sammen med de følgende opdateringer.
- RFC 9852: New Protocols Using TLS Must Require TLS 1.3 — obligatorisk understøttelse af 1.3 og brug som standard i nye protokoller, med valgfri interoperabilitet med 1.2; opdateringen gælder ikke DTLS.
- RFC 10015: Deprecating Obsolete Key Exchange Methods in TLS 1.2 and DTLS 1.2 — forbudte nøgleudvekslinger med RSA og Diffie–Hellman over endelige legemer, skilt fra certifikatsignaturer og fra nøgleetablering i TLS 1.3.
- RFC 8996: Deprecating TLS 1.0 and TLS 1.1 — den formelle afslutning på de to ældre versioner og begrundelsen for den.
- RFC 8470: Using Early Data in HTTP — Early-Data-headeren, koordinering med gateways og statuskoden 425.



Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

**Se, hvad det koster** Kontakt os →

# Gør begreberne praktiske

Spørg, hvordan »Hvad er TLS?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

**Se vores ydelser**

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

Tag denne side med dig

Visitkort

**Lavet med kærlighed af D.**

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

**Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.**

**Udskrevet fra https://engineer.company/da/glossary/tls/**