Engineer

TLS

protokoll för autentiserade, krypterade anslutningar

Vad det är

TLS, utskrivet Transport Layer Security, är det protokoll som en klient och en server använder för att skydda sina applikationsdata mot avlyssning och obemärkta ändringar och för att autentisera den andra änden. Vanligtvis autentiserar klienten servern; klientautentisering är valfri. Produktmenyer och certifikatbutiker kallar det fortfarande SSL, efter Secure Sockets Layer, protokollet som TLS ersatte. RFC 7568 förklarade den sista SSL-versionen föråldrad i juni 2015, så allt som heter SSL i dag är TLS under ytan, eller borde vara det.

TLS ligger mellan ett applikationsprotokoll och transportlagret under det. HTTPS är HTTP som färdas inuti en TLS-anslutning, och samma lager skyddar inlämning av e-post, IMAP, databasanslutningar och protokollet Gemini. HTTP/3 går längre: det körs över QUIC, som bygger in handskakningen från TLS 1.3 i själva transporten i stället för att lägga den ovanpå TCP, så som RFC 9001 beskriver.

Den aktuella versionen är TLS 1.3. RFC 9846, publicerad i juli 2026, uppdaterar specifikationen utan att ändra versionsnumret; den ersätter RFC 8446 från augusti 2018. En vanlig certifikatbaserad anslutning börjar med en handskakning. Klienten skickar ett ClientHello med de versioner, chiffersviter och nyckelandelar den stöder, vanligen tillsammans med Server Name Indication (SNI), det värdnamn den vill nå, så att en adress kan betjäna många webbplatser. Servern väljer och skickar sitt ServerHello. De efterföljande handskakningsmeddelandena är krypterade, även certifikatet den visar upp. Certifikatkedjan knyter serverns nyckel till en certifikatutfärdare som klienten redan litar på. Klienten kontrollerar kedjan, namnet och datumen, och servern bevisar att den har den matchande privata nyckeln genom att signera handskakningen. En återupptagen anslutning kan i stället autentiseras med en i förväg delad nyckel som etablerats tidigare.

Tre saker skiljer 1.3 från 1.2. En vanlig fullständig handskakning tar en tur och retur över nätet i stället för två, men ett omförsök kan lägga till ytterligare en. Nyckeletablering med publika nycklar ger framåtsekretess: en inspelning av dagens trafik förblir förseglad även om serverns långsiktiga privata nyckel läcker nästa år. Läget med enbart en i förväg delad nyckel är ett undantag och ger inte framåtsekretess. Och en klient som återvänder får skicka ”tidiga data” redan i sitt första meddelande, så kallad 0-RTT, innan handskakningen är klar, till ett pris som RFC:n skriver ut: tidiga data har varken framåtsekretess eller skydd mot att spelas upp igen på en annan anslutning.

Använd begreppet i ett beslut

Använd ”Vad är TLS?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Konton, lösenord och behörigheter ordnade så att alla har exakt den åtkomst deras arbete kräver, och de som slutar förlorar den samma dag.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Skrev 1 000 RBAC-regler som säkrar Linux-servrar — 1,000 åtkomstregler, skrivna för att täppa till en säkerhetslucka

Skrev 1 000 RBAC-regler som säkrar Linux-servrar Hör av er →

Varför det spelar roll

Att använda TLS är redan avgjort. Kvar finns två beslut: vilka versioner du godtar och var den krypterade anslutningen tar slut.

Först versionerna. RFC 8996, publicerad i mars 2021, säger att TLS 1.0 och 1.1 inte får användas. BCP 195 omfattar RFC 9325 och dess uppdateringar från juli 2026. RFC 9852 kräver att nya protokoll som använder TLS stöder 1.3 och använder den som standard; 1.2 tillåts som ett ytterligare alternativ, som inte är standard, för interoperabilitet. Det är inget generellt krav på att bara driftsätta 1.3. RFC 10015 förbjuder också RSA-nyckelutbyte och Diffie–Hellman-nyckelutbyte över ändliga kroppar i TLS 1.2, även den efemära varianten; detta förbjuder inte RSA-signaturer på certifikat. För en befintlig tjänst som behöver äldre klienter bör bara säkert konfigurerad 1.2 behållas bredvid 1.3, med ECDHE och autentiserad kryptering som AES-GCM. En tjänst med bara 1.3 har en mindre attackyta och en kortare konfiguration, och den stänger ute varje klient som högst talar 1.2: en äldre telefon, en kortterminal, en partners integration på en gammal Java-körmiljö. Logga den förhandlade versionen i några veckor och bestäm sedan utifrån siffrorna, inte utifrån vad som låter modernt.

Sedan termineringen. Där TLS slutar finns den privata nyckeln, och där blir trafiken klartext igen. Det finns tre vanliga platser för det:

- **Ett CDN eller en lastbalanserare i molnet.** Leverantören har nyckeln, eller ett certifikat den själv har utfärdat, tar handskakningslasten och ser varje förfrågan i klartext. Steget därifrån till ursprungsservern behöver ett eget TLS, annars går det okrypterat över nätet.
- **En omvänd proxy på värddatorn,** t.ex. nginx, Apache, HAProxy eller Caddy. En enda process har nycklarna för alla webbplatser på maskinen och skickar okrypterade förfrågningar vidare till applikationerna bakom, över loopback eller en lokal socket.
- **Applikationen själv.** Varje tjänst har sin egen nyckel. Det passar trafik mellan tjänster och ömsesidig TLS (mutual TLS), där också klienten visar upp ett certifikat, och det mångdubblar de ställen där ett certifikat måste förnyas.

De flesta miljöer blandar alla tre. Det som räknas är att någon för varje steg kan säga var klartexten finns och vem som skulle kunna läsa den.

Så ser det ut i praktiken

Företagets implementation av publika tjänster från september 2026 visar hur lite TLS-terminering kräver. Den använde en molninstans med en kärna, 512 MB minne, varav ungefär 464 MB gick att använda, och 10 GB disk. På den terminerade webbservern TLS för sju domäner tillsammans med git-forgen, en Tor-onion-tjänst, en Gemini-server, en Gopher-server och spärrning av intrångsförsök. Automatisering av säkerhetskopiering hade implementerats separat; det fastställde inte att krypterade säkerhetskopior var aktiva. Det var terminering på värddatorn i sin enklaste form: en enda webbserver svarade för alla sju webbplatserna, och deras kryptering kostade varken en extra maskin eller en extra tjänst. Designen behandlade minnet som den bindande begränsningen, och den dokumenterade driftsättningen rymdes inom de 464 MB.

Regeln som företaget arbetar efter står på sidan om hur vi arbetar säkert: kryptering under överföring överallt, med certifikat som utfärdas och förnyas automatiskt och en förnyelse som övervakas. Skälet som anges där är att TLS oftast fallerar genom ett certifikat som ingen höll ett öga på. Felen nedan är de som återstår när förnyelsen sköter sig själv.

Var det går fel

Vissa klienter faller bort med ett handskakningsfel medan webbläsare ansluter som vanligt. De två sidorna delar ingen version eller chiffersvit. Någon har smalnat av servern till bara 1.3, eller till en kort lista sviter, och en äldre klient erbjuder inget som finns på den. Eller så möter en server som fortfarande bara erbjuder CBC-sviter en klient som har slutat med dem. Felmeddelandet säger ”handshake failure” eller ”protocol version” och inte mycket mer. Testa varje version utifrån:

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

Logga den förhandlade versionen och sviten innan du smalnar av något. På en befintlig tjänst vars klienter fortfarande behöver 1.2, behåll säkert konfigurerad 1.2 med ECDHE och AES-GCM bredvid 1.3 medan övergången planeras. För ett nytt protokoll som använder TLS, stöd 1.3 och använd den som standard; aktivera 1.2 bara när interoperabiliteten kräver det.

Webbläsare litar på certifikatet, medan skript, curl och mobilappar avvisar det. Servern skickar sitt eget certifikat utan det mellanliggande certifikat som länkar det till roten. Webbläsare fyller ofta luckan med mellanliggande certifikat som de har sett förut, eller hämtar det som saknas själva. De flesta TLS-bibliotek gör inget av det och rapporterar en okänd utfärdare. Konfigurera hela kedjefilen, serverns certifikat först och sedan de mellanliggande, och testa med en klient som inte sparar något i cache: raden med `openssl s_client` ovan skriver ut den kedja som servern faktiskt skickade.

Certifikatkontrollen stängs av ”tills vidare”, och inställningen följer med ut i produktion. En testserver med ett självsignerat certifikat, eller en företagsproxy som signerar om trafiken, och någon lägger till `verify=False`, `curl -k`, `InsecureSkipVerify` eller `NODE_TLS_REJECT_UNAUTHORIZED=0`. Anslutningen förblir krypterad, men till den som svarar: vem som helst däremellan kan visa upp ett eget certifikat och läsa allt. Lägg rätt CA-certifikat i klientens förtroendelager, eller peka klienten uttryckligen mot det, och låt CI stoppa bygget när de flaggorna dyker upp.

Vissa besökare får ett certifikat för en annan webbplats. Servern visar sitt standardcertifikat när ett ClientHello nämner ett värdnamn som den saknar konfiguration för, eller inget namn alls. Det händer när en ny domän pekas mot adressen innan dess serverblock finns, och när en övervakningssond eller en gammal klient skickar sin förfrågan utan SNI. Webbläsaren visar att namnet inte stämmer, och på en delad värd avslöjar den dessutom för besökaren vilka andra webbplatser som bor där. Ge varje publikt namn ett eget block och ett certifikat som täcker det, och låt standardblocket vägra handskakningen i stället för att visa upp en främlings certifikat.

En betalning, eller vilken annan förfrågan som helst, utförs två gånger. Termineringspunkten godtar tidiga data via 0-RTT, en POST kommer in bland dem, och en angripare eller ett omförsök i nätet skickar samma byte igen på en ny anslutning. TLS 1.3 tillåter tidiga data men kan inte garantera skydd mot uppspelning mellan anslutningar. Stäng antingen av tidiga data, vänta med att agera på dem tills handskakningen är klar eller avvisa förfrågningar som inte säkert kan spelas upp igen med 425 Too Early, enligt RFC 8470. En gateway måste också skicka vidare markeringen Early-Data och veta att ursprungsservern förstår den: väntan på ett senare steg tar inte bort risken för uppspelning från ett tidigare.

Läs mer

- RFC 9846: The Transport Layer Security (TLS) Protocol Version 1.3 — specifikationen från juli 2026, som ersätter RFC 8446 utan att ändra protokollets versionsnummer 1.3; handskakning, poster och begränsningar för tidiga data.
- RFC 9325: Recommendations for Secure Use of TLS and DTLS — rekommendationerna för driftsättning i BCP 195, lästa tillsammans med följande uppdateringar.
- RFC 9852: New Protocols Using TLS Must Require TLS 1.3 — obligatoriskt stöd för 1.3 och användning som standard i nya protokoll, med valfri interoperabilitet med 1.2; uppdateringen gäller inte DTLS.
- RFC 10015: Deprecating Obsolete Key Exchange Methods in TLS 1.2 and DTLS 1.2 — förbjudna nyckelutbyten med RSA och Diffie–Hellman över ändliga kroppar, skilda från certifikatsignaturer och från nyckeletablering i TLS 1.3.
- RFC 8996: Deprecating TLS 1.0 and TLS 1.1 — det formella slutet för de två äldre versionerna och skälen bakom det.
- RFC 8470: Using Early Data in HTTP — headern Early-Data, samordning med gateways och statusen 425.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är TLS?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/tls/