Engineer

TLS

🤝 Protokoll für authentifizierte, verschlüsselte Verbindungen

Was es ist

Transport Layer Security, kurz TLS, ist das Protokoll, mit dem ein Client und ein Server ihre Anwendungsdaten vor Mitlesen und unbemerkten Veränderungen schützen und die Gegenseite authentifizieren. Meist authentifiziert der Client den Server; die Authentifizierung des Clients ist optional. In Produktmenüs und bei Zertifikathändlern heißt es noch immer SSL, nach dem abgelösten Protokoll Secure Sockets Layer. RFC 7568 erklärte die letzte SSL-Version im Juni 2015 für veraltet. Was heute SSL genannt wird, ist technisch also TLS, oder sollte es sein.

TLS liegt zwischen einem Anwendungsprotokoll und dem darunterliegenden Transport. HTTPS ist HTTP innerhalb einer TLS-Verbindung. Dieselbe Schicht schützt die Einlieferung von E-Mails, IMAP, Datenbankverbindungen und das Gemini-Protokoll. HTTP/3 geht weiter: Es läuft über QUIC, das laut RFC 9001 den TLS-1.3-Handshake direkt in den Transport einbaut, statt ihn über TCP zu legen.

Die aktuelle Version ist TLS 1.3. RFC 9846 vom Juli 2026 aktualisiert ihre Spezifikation ohne Änderung der Versionsnummer und ersetzt RFC 8446 vom August 2018. Eine gewöhnliche zertifikatsbasierte Verbindung beginnt mit einem Handshake. Der Client sendet ein ClientHello mit den unterstützten Versionen, Cipher Suites und Schlüsselanteilen, gewöhnlich mit Server Name Indication (SNI), dem gewünschten Hostnamen, damit eine Adresse viele Websites bedienen kann. Der Server wählt aus und sendet sein ServerHello. Die folgenden Handshake-Nachrichten sind verschlüsselt, auch das vorgelegte Zertifikat. Dessen Kette bindet den Serverschlüssel an eine Zertifizierungsstelle, der der Client bereits vertraut. Der Client prüft Kette, Namen und Gültigkeitsdaten. Der Server beweist durch Signieren des Handshakes, dass er den passenden privaten Schlüssel besitzt. Eine wiederaufgenommene Verbindung kann sich stattdessen über einen zuvor vereinbarten gemeinsamen Schlüssel authentifizieren.

Drei Dinge unterscheiden 1.3 von 1.2. Ein gewöhnlicher vollständiger Handshake benötigt einen Netzwerk-Roundtrip statt zwei; ein erneuter Versuch kann allerdings einen weiteren hinzufügen. Eine auf öffentlichen Schlüsseln beruhende Schlüsselvereinbarung bietet Forward Secrecy: Eine Aufzeichnung des heutigen Verkehrs bleibt verschlossen, selbst wenn der langfristige private Serverschlüssel nächstes Jahr offengelegt wird. Der ausschließlich auf einem vorab vereinbarten Schlüssel beruhende Modus ist eine Ausnahme und bietet keine Forward Secrecy. Außerdem kann ein wiederkehrender Client schon mit der ersten Nachricht „Early Data“ senden, bekannt als 0-RTT, bevor der Handshake endet. Den Preis benennt der RFC ausdrücklich: Early Data haben weder Forward Secrecy noch Schutz gegen das erneute Einspielen auf einer anderen Verbindung.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist TLS?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Konten, Passwörter und Berechtigungen werden so geordnet, dass alle genau den beruflich erforderlichen Zugriff haben und ihn am Tag ihres Ausscheidens verlieren.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 1.000 RBAC-Regeln für sichere Linux-Server — 1,000 Zugriffsregeln zum Schließen einer Sicherheitslücke

1.000 RBAC-Regeln für sichere Linux-Server Kontakt aufnehmen →

Warum es wichtig ist

Ob TLS verwendet wird, ist entschieden. Offen bleibt, welche Versionen akzeptiert werden und wo die verschlüsselte Verbindung endet.

Zuerst die Versionen. RFC 8996 vom März 2021 untersagt TLS 1.0 und 1.1. BCP 195 umfasst RFC 9325 und dessen Aktualisierungen vom Juli 2026. RFC 9852 verlangt für neue Protokolle, die TLS nutzen, Unterstützung für 1.3 und dessen Auswahl als Standard. Für die Interoperabilität erlaubt er 1.2 als zusätzliche Option, die nicht standardmäßig gewählt wird. Das ist keine allgemeine Pflicht zu einer reinen 1.3-Bereitstellung. RFC 10015 verbietet außerdem den RSA-Schlüsselaustausch und den Diffie-Hellman-Schlüsselaustausch über endlichen Körpern in TLS 1.2, einschließlich der ephemeren Variante. RSA-Signaturen auf Zertifikaten verbietet das nicht. Für einen bestehenden Dienst mit älteren Clients nur sicher konfiguriertes 1.2 neben 1.3 beibehalten, mit ECDHE und authentifizierter Verschlüsselung wie AES-GCM. Ein reiner 1.3-Dienst hat eine kleinere Angriffsfläche und eine kürzere Konfiguration. Er schließt aber jeden Client aus, der höchstens 1.2 spricht: ein älteres Telefon, ein Kartenterminal oder

eine Partnerintegration auf einer alten Java-Laufzeitumgebung. Die ausgehandelte Version einige Wochen protokollieren und dann nach Zahlen entscheiden, nicht danach, was modern klingt.

Danach die Terminierung. Wo TLS endet, liegt der private Schlüssel, und der Verkehr wird wieder zu Klartext. Drei Orte sind üblich:

- **Ein CDN oder ein Cloud-Loadbalancer.** Der Anbieter hält den Schlüssel oder ein selbst ausgestelltes Zertifikat, übernimmt die Handshake-Last und sieht jede Anfrage im Klartext. Der Weg von dort zum Ursprungsserver braucht eigenes TLS, sonst verläuft er unverschlüsselt über das Netzwerk.
- **Ein Reverse Proxy auf dem Host,** etwa Nginx, Apache, HAProxy oder Caddy. Ein Prozess hält die Schlüssel aller Websites der Maschine und reicht unverschlüsselte Anfragen über Loopback oder einen lokalen Socket an die dahinterliegenden Anwendungen weiter.
- **Die Anwendung selbst.** Jeder Dienst hält seinen eigenen Schlüssel. Das passt zu Verkehr zwischen Diensten und zu gegenseitigem TLS, bei dem auch der Client ein Zertifikat vorlegt. Es vervielfacht die Orte, an denen ein Zertifikat erneuert werden muss.

Die meisten Infrastrukturen mischen alle drei. Entscheidend ist, dass jemand für jeden Abschnitt sagen kann, wo Klartext vorliegt und wer ihn lesen könnte.

Wie es in der Praxis aussieht

Die Umsetzung der öffentlichen Unternehmensdienste vom September 2026 zeigt, wie wenig TLS-Terminierung braucht. Sie verwendete eine einkernige Cloud-Instanz mit 512 MB Arbeitsspeicher, davon ungefähr 464 MB nutzbar, und 10 GB Festplattenspeicher. Darauf terminierte der Webserver TLS für sieben Domains, neben Git-Forge, Tor-Onion-Dienst, Gemini-Server, Gopher-Server und Sperren zur Abwehr von Einbruchsversuchen. Backup-Automatisierung war separat implementiert; das belegte keine aktiven verschlüsselten Sicherungen. Das war Terminierung auf dem Host in ihrer einfachsten Form: Ein Webserver bediente alle sieben Websites, und ihre Verschlüsselung brauchte weder eine weitere Maschine noch einen weiteren Dienst. Der Entwurf behandelte den Arbeitsspeicher als maßgebliche Grenze, und die dokumentierte Bereitstellung lief innerhalb dieser 464 MB.

Die Regel, nach der das Unternehmen arbeitet, steht auf seiner Sicherheitsseite: Verschlüsselung bei jeder Übertragung, automatisch ausgestellte und erneuerte Zertifikate sowie überwachte Erneuerung. Als Grund nennt sie, dass TLS am häufigsten an einem unbeobachteten Zertifikat scheitert. Die folgenden Fehler bleiben übrig, wenn die Erneuerung von selbst läuft.

Wo es schiefgeht

Manche Clients scheitern am Handshake, während Browser problemlos verbinden. Beide Seiten haben keine gemeinsame Version oder Cipher Suite. Jemand hat den Server auf 1.3 oder eine kurze Liste von Suites beschränkt, und ein älterer Client bietet nichts Passendes an. Oder ein Server mit ausschließlich CBC-Suites trifft auf einen Client, der sie nicht mehr unterstützt. Die Meldung lautet „handshake failure“ oder „protocol version“ und sagt kaum mehr. Jede Version von außen testen:

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

Vor jeder Einschränkung die ausgehandelte Version und Suite protokollieren. Bei einem bestehenden Dienst, dessen Clients noch 1.2 brauchen, sicher konfiguriertes 1.2 mit ECDHE und AES-GCM neben 1.3 behalten und die Migration planen. Ein neues Protokoll mit TLS muss 1.3 unterstützen und standardmäßig wählen. 1.2 nur aktivieren, wenn die Interoperabilität es erfordert.

Browser vertrauen dem Zertifikat; Skripte, curl und mobile Apps lehnen es ab. Der Server sendet sein eigenes Zertifikat ohne das Zwischenzertifikat, das es mit der Wurzel verbindet. Browser schließen die Lücke oft mit bereits bekannten Zwischenzertifikaten oder laden das fehlende selbst nach. Die meisten TLS-Bibliotheken tun beides nicht und melden einen unbekannten Aussteller. Die vollständige Kettendatei konfigurieren, zuerst das Serverzertifikat und dann die Zwischenzertifikate, und mit einem Client ohne Cache testen. Der obige Aufruf von `openssl s_client` gibt die tatsächlich vom Server gesendete Kette aus.

Die Zertifikatsprüfung wird „vorerst“ abgeschaltet, und die Einstellung geht in Produktion. Bei einem Testserver mit selbst signiertem Zertifikat oder einem Unternehmensproxy, der den Verkehr neu signiert, ergänzt jemand `verify=False`, `curl -k`, `InsecureSkipVerify` oder `NODE_TLS_REJECT_UNAUTHORIZED=0`.

Die Verbindung bleibt verschlüsselt, unabhängig davon, wer auf der Gegenseite antwortet. Jeder dazwischen kann ein eigenes Zertifikat vorlegen und alles lesen. Das richtige CA-Zertifikat in den Vertrauensspeicher des Clients aufnehmen oder den Client ausdrücklich darauf verweisen und CI bei diesen Flags scheitern lassen.

Manche Besucher erhalten ein Zertifikat für eine andere Website. Der Server zeigt sein Standardzertifikat, wenn ein ClientHello einen nicht konfigurierten Hostnamen oder gar keinen nennt. Das passiert, wenn eine neue Domain auf die Adresse zeigt, bevor ihr Serverblock existiert, oder wenn eine Überwachungsprobe oder ein alter Client kein SNI sendet. Der Browser meldet einen unpassenden Namen. Auf einem gemeinsam genutzten Host erfährt der Besucher außerdem, welche anderen Websites dort liegen. Jedem öffentlichen Namen einen eigenen Block mit passendem Zertifikat geben und den Standardblock den Handshake ablehnen lassen, statt ein fremdes Zertifikat anzubieten.

Eine Zahlung oder eine andere Anfrage wird zweimal ausgeführt. Die terminierende Stelle akzeptiert 0-RTT-Early-Data, darin kommt ein POST an, und ein Angreifer oder ein erneuter Übertragungsversuch im Netzwerk sendet dieselben Bytes auf einer neuen Verbindung noch einmal. TLS 1.3 erlaubt Early Data, kann aber keinen Schutz gegen verbindungsübergreifendes Replay garantieren. Early Data entweder abschalten, mit den darin angeforderten Aktionen bis zum Ende des Handshakes warten oder Anfragen, die sich nicht gefahrlos wiederholen lassen, wie in RFC 8470 vorgesehen mit 425 Too Early ablehnen. Ein Gateway muss außerdem die Early-Data-Kennzeichnung weiterreichen und wissen, dass der Ursprungsserver sie versteht: Auf einem späteren Abschnitt zu warten beseitigt nicht das Replay-Risiko eines früheren.

Weiterführende Lektüre

- RFC 9846: The Transport Layer Security (TLS) Protocol Version 1.3 — die Spezifikation vom Juli 2026, die RFC 8446 ohne Änderung der Protokollversion 1.3 ersetzt; Handshake, Records und die Grenzen von Early Data.
- RFC 9325: Recommendations for Secure Use of TLS and DTLS — die Bereitstellungsempfehlungen in BCP 195, zusammen mit den folgenden Aktualisierungen zu lesen.
- RFC 9852: New Protocols Using TLS Must Require TLS 1.3 — verpflichtende Unterstützung und Standardauswahl von 1.3 für neue Protokolle mit optionalem 1.2 für die Interoperabilität; das Update gilt nicht für DTLS.

- RFC 10015: Deprecating Obsolete Key Exchange Methods in TLS 1.2 and DTLS 1.2 — verbotener RSA-Schlüsselaustausch und Diffie-Hellman-Schlüsselaustausch über endlichen Körpern, abgegrenzt von Zertifikatssignaturen und der Schlüsselvereinbarung in TLS 1.3.
- RFC 8996: Deprecating TLS 1.0 and TLS 1.1 — das formale Ende der beiden älteren Versionen und seine Gründe.
- RFC 8470: Using Early Data in HTTP — der Early-Data-Header, die Abstimmung mit Gateways und der Status 425.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist TLS?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/tls/