Engineer

TLS

protocol for authenticated, encrypted connections

What it is

Transport Layer Security, TLS for short, is the protocol a client and a server use to protect their application data from eavesdropping and unnoticed changes, and to authenticate the other end. Usually the client authenticates the server; client authentication is optional. Product menus and certificate shops still call it SSL, after the Secure Sockets Layer protocol it replaced. RFC 7568 deprecated the last SSL version in June 2015, so anything labelled SSL today is TLS underneath, or ought to be.

TLS sits between an application protocol and the transport below it. HTTPS is HTTP carried inside a TLS connection, and the same layer protects mail submission, IMAP, database connections and the Gemini protocol. HTTP/3 goes further: it runs over QUIC, which builds the TLS 1.3 handshake into the transport itself instead of layering it on TCP, as RFC 9001 describes.

The current version is TLS 1.3. RFC 9846, published in July 2026, updates its specification without changing the version number; it replaces RFC 8446 from August 2018. A normal certificate-based connection opens with a handshake. The client sends a ClientHello listing the versions, cipher suites and key shares it supports, usually with Server Name Indication (SNI), the host name it wants, so that one address can serve many sites. The server chooses and sends its ServerHello. The handshake messages that follow are encrypted, including the certificate it presents. That certificate chain ties the server’s key to a certificate authority the client already trusts. The client checks the chain, the name and the dates, and the server proves it holds the matching private key by signing the handshake. A resumed connection can instead authenticate through a pre-shared key established earlier.

Three things set 1.3 apart from 1.2. An ordinary full handshake takes one round trip instead of two, although a retry can add another. Public-key-based key establishment provides forward secrecy: a recording of today's traffic stays sealed even if the server's long-term private key leaks next year. The pre-shared-key-only mode is an exception and does not provide forward secrecy. And a returning client may send "early data" with its very first message, known as 0-RTT, before the handshake finishes, at a price the RFC spells out: early data is neither forward secret nor protected against being replayed on another connection.

Use the concept in a decision

Put "What is TLS?" to work in a real decision. We can help relate the concept to the system you are building or buying. Accounts, passwords and permissions arranged so people hold exactly the access their job needs, and leavers lose it that day.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

Wrote 1,000 RBAC rules to secure Linux servers Get in touch →

Why it matters

Using TLS is settled. The decisions that remain are which versions to accept and where the encrypted connection ends.

Versions first. RFC 8996, published in March 2021, says TLS 1.0 and 1.1 must not be used. BCP 195 includes RFC 9325 and its July 2026 updates. RFC 9852 requires new protocols using TLS to support and default to 1.3; it allows 1.2 as an additional, non-default option for interoperability. That is not a universal requirement for 1.3-only deployment. RFC 10015 also prohibits RSA key exchange and finite-field Diffie–Hellman key exchange in TLS 1.2, including ephemeral finite-field DH; this does not prohibit RSA signatures on certificates. For an existing service that needs older clients, retain only securely configured 1.2 alongside 1.3, with ECDHE and authenticated encryption such as AES-GCM. A 1.3-only service has a smaller attack surface and a shorter configuration, and it shuts out every client that speaks 1.2 at most: an older phone, a card terminal, a partner's integration on an old Java runtime. Log the negotiated version for a few weeks, then decide on the numbers rather than on what sounds modern.

Termination second. Wherever TLS ends, the private key lives there and the traffic turns back into plain text. There are three common places for it:

- **A CDN or a cloud load balancer.** The provider holds the key, or a certificate it issued itself, absorbs the handshake load, and sees every request in the clear. The hop from there to the origin needs TLS of its own, or it crosses the network unencrypted.
- **A reverse proxy on the host,** such as Nginx, Apache, HAProxy or Caddy. One process holds the keys for every site on the machine and passes plain requests to the applications behind it over loopback or a local socket.
- **The application itself.** Each service holds its own key. That suits traffic between services and mutual TLS, where the client presents a certificate as well, and it multiplies the places where a certificate has to be renewed.

Most estates mix all three. What counts is that someone can say, for each hop, where the plain text is and who could read it.

What it looks like in practice

The company's September 2026 public-services implementation shows how little TLS termination needs. It used a single-core cloud instance with 512 MB of memory, about 464 MB of it usable, and 10 GB of disk. On it the web server terminated TLS for seven domains alongside the git forge, a Tor onion service, a Gemini server, a Gopher server and intrusion banning. Backup automation was implemented separately; that did not establish that encrypted backups were active. That was termination on the host at its plainest: one web server answered for all seven sites, and their encryption cost no extra machine and no extra service. The design treated memory as the binding constraint, and the recorded deployment ran inside those 464 MB.

The rule the company works to is written on its security page: encryption in transit everywhere, with certificates issued and renewed automatically and the renewal monitored. The reason given there is that TLS most often fails through a certificate nobody was watching. The failures below are the ones left once renewal runs by itself.

Where it goes wrong

Some clients drop with a handshake failure while browsers connect fine. The two ends share no version or cipher suite. Someone narrowed the server to 1.3 only, or to a short list of suites, and an older client offers nothing on it; or a server still offering only CBC suites meets a client that dropped them. The alert says "handshake failure" or "protocol version" and little else. Test each version from outside:

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

Log the negotiated version and suite before narrowing anything. On an existing service whose clients still need 1.2, keep securely configured 1.2 with ECDHE and AES-GCM next to 1.3 while planning the migration. For a new protocol using TLS, support and default to 1.3; enable 1.2 only when interoperability requires it.

Browsers trust the certificate, and scripts, curl and mobile apps reject it. The server sends its own certificate without the intermediate that links it to the root. Browsers often fill the gap from intermediates they have met before or fetch the missing one themselves; most TLS libraries do neither and report an unknown issuer. Configure the full chain file, the server's certificate first and then the intermediates, and test with a client that keeps no cache: the `openssl s_client` line above prints the chain the server actually sent.

Certificate checking is switched off "for now", and the setting ships. A test server with a self-signed certificate, or a corporate proxy that re-signs traffic, and someone adds `verify=False`, `curl -k`, `InsecureSkipVerify` or `NODE_TLS_REJECT_UNAUTHORIZED=0`. The connection stays encrypted, to whoever answers it: anyone in between can present a certificate of their own and read everything. Put the right CA certificate in the client's trust store, or point the client at it explicitly, and make CI fail on those flags.

Some visitors get a certificate for a different site. The server presents its default certificate whenever a ClientHello names a host it has no configuration for, or names none at all. That happens when a new domain is pointed at the address before its server block exists, and when a monitoring probe or an old client sends no SNI. The browser shows a name mismatch, and on a shared host it also tells the visitor which other sites live there. Give every public name its own block and a certificate that covers it, and make the default block refuse the handshake rather than offer a stranger's certificate.

A payment, or any other request, is carried out twice. The terminator accepts 0-RTT early data, a POST arrives inside it, and an attacker or a network retry sends the same bytes again on a new connection. TLS 1.3 permits early data but cannot guarantee protection against replay between connections. Either disable early data, defer acting on it until the handshake completes, or reject requests that cannot safely be replayed with 425 Too Early, as RFC 8470 specifies. A

gateway must also pass on the Early-Data indication and know that its origin understands it: waiting on a later hop does not erase the replay risk from an earlier one.

Further reading

- RFC 9846: The Transport Layer Security (TLS) Protocol Version 1.3 — the July 2026 specification, replacing RFC 8446 without changing the protocol's 1.3 version number; handshake, records and early-data limitations.
- RFC 9325: Recommendations for Secure Use of TLS and DTLS — the deployment recommendations in BCP 195, read with the following updates.
- RFC 9852: New Protocols Using TLS Must Require TLS 1.3 — mandatory 1.3 support and default for new protocols, with optional 1.2 interoperability; the update does not apply to DTLS.
- RFC 10015: Deprecating Obsolete Key Exchange Methods in TLS 1.2 and DTLS 1.2 — prohibited RSA and finite-field DH key exchanges, distinguished from certificate signatures and from TLS 1.3 key establishment.
- RFC 8996: Deprecating TLS 1.0 and TLS 1.1 — the formal end of the two older versions and the reasons behind it.
- RFC 8470: Using Early Data in HTTP — the Early-Data header, gateway coordination and the 425 status.



Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is TLS?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

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

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/tls/