Engineer

HTTPS

🔑 the same requests, over an encrypted connection

What it is

HTTPS is HTTP sent through a TLS connection. The requests and responses are the same messages as before, with the same methods, status codes and headers; what changes is the connection under them, which is now encrypted and authenticated. The `https` URI scheme is defined in RFC 9110, the HTTP semantics specification from June 2022, which took over from RFC 2818, the short May 2000 document that first wrote down HTTP over TLS. Its default port is 443, where plain HTTP uses 80.

RFC 9110 attaches two rules to the scheme. A client MUST confirm that the identity the server presents acceptably matches the host in the address, which in practice means the certificate has to cover that name; a connection to the wrong name has failed, however strong its encryption. And `http://example.com/` and `https://example.com/` are distinct origins with separate namespaces. They share a host name and nothing else, as far as the protocol is concerned.

HTTPS is not tied to one version of HTTP. HTTP/1.1 and HTTP/2 run over TLS on TCP. HTTP/3 runs over QUIC, which carries the TLS 1.3 handshake inside the transport, and RFC 9114 rules out using it to reach a plain `http://` server directly, so the newest version arrives with HTTPS or not at all.

What the encryption covers is easiest to see with an address. For `https://shop.example/orders/1234?ref=mail`, the path, the query string, every header including cookies and `Authorization`, and both message bodies are encrypted. The network between visitor and server still sees the server's IP address and port, the timing and size of the traffic, and in most connections the host name `shop.example` as well: the browser looked it up in DNS, and the client names it in the opening TLS message so that one address can serve many sites. HTTPS hides what was asked for, not whom it was asked of.

Use the concept in a decision

Put “What is HTTPS?” 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

The live question for a public site is whether anything should still answer on `http://`, and browsers now make most of that call themselves.

The first lever is the secure context, a W3C term for a page delivered over HTTPS or from the machine itself. Browsers keep a growing set of features for secure contexts only. Service workers are one: `navigator.serviceWorker` is declared for secure contexts, so on a page loaded over plain HTTP the property is simply absent, and so is every feature built on it, offline caching first among them. The Secure Contexts specification also recommends that authors of new specifications guard sensitive features the same way, so the restriction is the default for new work rather than a case-by-case exception.

The second lever is mixed content. Once a page is itself HTTPS, the browser sorts each of its fetches under the W3C Mixed Content rules. Images, audio and video requested over `http://` are upgraded to `https://` automatically, and they fail if that address does not answer. Scripts, stylesheets and data requests are blocked outright with a network error. One leftover plain address in a template can therefore take out a whole feature, not just a picture.

That leaves very little for port 80. A listener that answers every request with a permanent redirect to the same path on `https://` still earns its place, because people type bare names and old links still say `http://`. The certificate authority’s HTTP challenge also arrives there when a certificate is issued or renewed. HSTS then lets a returning browser skip even the redirect. Anything beyond those jobs on port 80 is a page served without the guarantees above.

Loopback is the one standing exception for ordinary sites. The specification counts `localhost`, `127.0.0.1` and `::1` as potentially trustworthy, so a page served from the developer’s own machine gets secure-context features without

a certificate. That convenience for developers is the source of one of the faults below.

What it looks like in practice

The company's own site follows that rule. On the production host, Caddy answers every web request and renews the HTTPS certificate itself, from Let's Encrypt, so keeping the certificate current is the web server's job rather than a person's. The same pages are published as a Tor onion service, and that address begins with `http://` on purpose. An onion address encodes the service's identity public key. Tor authenticates the service and encrypts the connection before HTTP is exchanged, which is why this mirror uses HTTP over Tor. The credits page also links Qualys SSL Labs, which grades the live HTTPS configuration for anyone who asks.

Client work shows the secure-context rule from the other side. For a platform whose users are often seafarers and field staff on phones, frequently offline, the company delivered full Progressive Web App support, with a service worker caching fonts, images, audio, video and CDN assets cache-first. Because service workers only really behave over HTTPS, the offline and install behaviour was tested over HTTPS on real devices instead of being assumed. The same need shows up in the tooling of a polyglot monorepo with 578 task targets, where the go-task layer the company wrote puts an HTTPS variant for testing PWA and mobile next to the native and Docker development modes.

Where it goes wrong

Scripts stop running after the switch to HTTPS, and some images vanish. The pages still contain `http://` addresses, hard-coded in a template, stored in CMS content or pasted in with a third-party snippet. The browser blocks the scripts and stylesheets and upgrades the images, which then break wherever the other host has no HTTPS. The browser console names each blocked address. Search the templates and the stored content for the old scheme, and rewrite each hit as `https://` or as a path relative to the site:

```
grep -rnE '(src|href)="http://' templates/ content/
```

A `Content-Security-Policy: upgrade-insecure-requests` header carries the pages through the clean-up, but it leaves the stored addresses as they were.

The site redirects to itself until the browser gives up. A CDN or load balancer terminates TLS and forwards the request to the application over plain HTTP. The application sees `http`, redirects to `https`, the proxy forwards the retry as `http` again, and the loop ends in a "too many redirects" error. The same blindness makes the application build absolute links that start with `http://`. The proxy has to pass on the original scheme, in the standard `Forwarded: proto=https` header from RFC 7239 or the older `X-Forwarded-Proto`, and the application should believe that header only when it comes from the proxy's own address, since any client can send it. Then let one layer own the redirect, normally the one that holds the certificate.

A session cookie set over HTTPS travels in the clear. RFC 9110 warns that cookies do not respect the line it draws between the two origins. Under the cookie rules of RFC 6265, a cookie without the `Secure` attribute is sent on plain `http://` requests to the same host as well. A visitor typing the bare name is enough to send it, and so is an attacker on the network who answers any unrelated plain request with a redirect to the site's `http://` address. Set `Secure` on every cookie the site issues, and add HSTS so the browser never makes the plain request in the first place.

The service worker runs on the laptop and is missing on the phone. The developer opens `http://localhost:3000`, a secure context, and everything works. The phone on the same network opens `http://192.168.1.20:3000`, which is not one, so `navigator.serviceWorker` is undefined, a library that checks for it skips registration, and offline mode is quietly absent. Serve the development build over HTTPS as well. Caddy, for one, issues certificates for local names and IP addresses from its own local certificate authority and tries to add that root to the machine's trust store. Its documentation warns that the step is not guaranteed and that getting the root into every trust store is the administrator's job, and a test phone has to be given that root by hand.

Further reading

- RFC 9110, section 4.2.2: https URI Scheme — the scheme, its default port, and the rule that a client must check the server's identity.
- Secure Contexts — which pages count as secure, why loopback qualifies, and the advice to restrict new features.
- Mixed Content — which requests from an HTTPS page are upgraded and which are blocked.
- RFC 7239: Forwarded HTTP Extension — the standard header a proxy uses to pass on the scheme the visitor used.

- Automatic HTTPS — Caddy's own account of its certificates, its redirect from HTTP and its local certificate authority.

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 HTTPS?" 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/https/