TLS for App Developers

This thread has been locked by a moderator.

5.0k

Transport Layer Security (TLS) is the most important security protocol on the Internet today. Most notably, TLS puts the *S* into *HTTPS*, adding security to the otherwise insecure HTTP protocol.

IMPORTANT TLS is the successor to the Secure Sockets Layer (SSL) protocol. SSL is no longer considered secure and it's now rarely used in practice, although many folks still say SSL when they mean TLS.

TLS is a complex protocol. Much of that complexity is hidden from app developers but there are places where it's important to understand specific details of the protocol in order to meet your requirements. This post explains the fundamentals of TLS, concentrating on the issues that most often confuse app developers.

Note If you're working on TLS in the local environment, for example, to talk to a Wi-Fi based accessory, see TLS For Accessory Developers.

Server Certificates

For standard TLS to work the server must have a **digital identity**, that is, the combination of a certificate and the private key matching the public key embedded in that certificate. TLS Crypto Magic™ ensures that:

- The client gets a copy of the server's certificate.
- The client knows that the server holds the private key matching the public key in that certificate.

In a typical TLS handshake the server passes the client a list of certificates, where item 0 is the server's certificate (the **leaf certificate**), item N is (optionally) the certificate of the certificate authority that ultimately issued that certificate (the **root certificate**), and items 1…N-1 are any **intermediate certificates** required to build a **cryptographic chain of trust** from 0 to N.

Note The cryptographic chain of trust is established by means of digital signatures. Certificate X in the chain is **issued** by certificate X+1. The owner of certificate X+1 uses their private key to digitally sign certificate X. The client can verify this signature using the public key embedded in certificate X+1. Eventually this chain terminates in a **trusted anchor**, that is, a certificate that the client trusts by default. Typically this anchor is a self-signed root certificate from a certificate authority.

Note Item N is optional for reasons I'll explain below. Also, the list of intermediate certificates may be empty (in the case where the root certificate directly issued the leaf certificate) but that's uncommon for servers in the real world.

Once the client gets the server's certificate, it must evaluate trust on that certificate to confirm that it's talking to the right server. There's three levels of trust evaluation here:

Basic X.509 trust evaluation checks that there's a cryptographic chain of trust from the leaf through the intermediates to a trusted root certificate. The client has a set of trusted root certificates built in (these are from well-known **certificate authorities**, or CAs), and a site admin can add more via a configuration profile.

This step also checks that none of the certificates have expired, and various other more technical criteria (like the Basic Constraints extension).

Note This explains why the server does not have to include the root certificate in the list of certificates it passes to the client; the client has to have the root certificate installed if trust evaluation is to succeed.

In addition, **TLS trust evaluation** (per RFC 2818) checks that the DNS name that you connected to matches the DNS name in the certificate. Specifically, the DNS name must be listed in the Subject Alternative Name extension.

Note The Subject Alternative Name extension can also contain IP addresses, although that's a *much* less well-trodden path. Also, historically it was common to accept DNS names in the Common Name element of the Subject but that is no longer the case on Apple platforms.

App Transport Security (ATS) adds its own security checks.

Basic X.509 and TLS trust evaluation are done for all TLS connections. ATS is only done on TLS connections made by `URLSession` and things layered on top `URLSession` (like `WKWebView`). In many situations you can override trust evaluation; for details, see Technote 2232 HTTPS Server Trust Evaluation). Such overrides can either tighten or loosen security. For example:

- You might tighten security by checking that the server certificate was issued by a specific CA. That way, if someone manages to convince a poorly-managed CA to issue them a certificate for *your* server, you can detect that and fail.
- You might loosen security by adding your own CA's root certificate as a trusted anchor.

IMPORTANT If you rely on loosened security you have to disable ATS. If you leave ATS enabled, it will require that default server trust evaluation succeed regardless of any customisations you do.

Client Certificates

The previous section discusses server trust evaluation, which is required for all standard TLS connections. That process describes how the client decides whether to trust the server. **Client certificate authentication**, sometimes known as mutual TLS (mTLS), is the opposite of that, that is, it's the process by which the server decides whether to trust the client.

Client certificate authentication is optional. The server must request a certificate from the client and the client may choose to supply one or not (although if the server requests a certificate and the client does not supply one it's likely that the server will then fail the connection).

At the TLS protocol level this works much like it does with the server certificate. For the client to provide this certificate it must apply a digital identity to the connection, and TLS Crypto Magic™ assures the server that, if it gets a certificate from the client, the client holds the private key associated with that certificate.

Where things diverge is in trust evaluation. Trust evaluation of the client certificate is done on the server, and the server uses its own rules to decided whether to trust a specific client certificate. For example:

- Some servers do basic X.509 trust evaluation and then check that the chain of trust leads to one specific root certificate; that is, a client is trusted if it holds a digital identity whose certificate was issued by a specific CA.
- Some servers just check the certificate against a list of known trusted client certificates.

When the client sends its certificate to the server it actually sends a list of certificates, much as I've described above for the server's certificates. In *many* cases the client only needs to send item 0, that is, its leaf certificate. That's because:

- The server already has the intermediate certificates required to build a chain of trust from that leaf to its root.
- There's no point sending the root, as I discussed above in the context of the server trust evaluation.

However, there are no hard and fast rules here; the server does its client trust evaluation using its own internal logic, and it's possible that this logic might require the client to present intermediates, or indeed present the root certificate even though it's typically redundant. If you have problems with this, you'll have to ask the folks running the server to explain these requirements.

Note If you need to send additional certificates to the server, you can do this by passing them to the `certificates` parameter of the method you use to create your `URLCredential` (typically `init(identity:certificates:persistence:)`).

One thing that bears repeating is that trust evaluation of the client certificate is done on the server, not the client. The client does not care whether the client certificate is trusted or not. Rather, it simply passes that certificate the server and it's up to the server to make that decision.

When a server requests a certificate from the client, it may supply a list of acceptable certificate authorities [1]. Safari uses this to filter the list of client identities it presents to the user. If you are building an HTTPS server and find that Safari doesn't show the expected client identity, make sure you have this configured correctly. If you're building an iOS app and want to implement a filter like Safari's, get this list using:

- The `distinguishedNames` property, if you're using `URLSession`
- The `sec_protocol_metadata_access_distinguished_names` routine, if you're using Network framework

[1] See the `certificate_authorities` field in Section 7.4.4 of RFC 5246, and equivalent features in other TLS versions.

TLS Standards

- RFC 6101 The Secure Sockets Layer (SSL) Protocol Version 3.0 (historic)
- RFC 2246 The TLS Protocol Version 1.0
- RFC 4346 The Transport Layer Security (TLS) Protocol Version 1.1
- RFC 5246 The Transport Layer Security (TLS) Protocol Version 1.2
- RFC 8446 The Transport Layer Security (TLS) Protocol Version 1.3
- RFC 4347 Datagram Transport Layer Security
- RFC 6347 Datagram Transport Layer Security Version 1.2
- RFC 9147 The Datagram Transport Layer Security (DTLS) Protocol Version 1.3

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

Revision History:

- **2023-02-28** Added an explanation mTLS acceptable certificate authorities.
- **2022-12-02** Added links to the DTLS RFCs.
- **2022-08-24** Added links to the TLS RFCs. Made other minor editorial changes.
- **2022-06-03** Added a link to TLS For Accessory Developers.
- **2021-02-26** Fixed the formatting. Clarified that ATS only applies to `URLSession`. Minor editorial changes.
- **2020-04-17** Updated the discussion of Subject Alternative Name to account for changes in the 2019 OS releases. Minor editorial updates.
- **2018-10-29** Minor editorial updates.
- **2016-11-11** First posted.

Network Security CFNetwork

Reply

Posted 6 years ago by eskimo

Add a Comment