Having fun with TLS

TLS is likely to be around in one form or another for a very long time. But it will evolve over time. There will for example be a TLS 2.0 which integrates fully post-quantum primitives. In the meantime there may be a TLS 1.4 which updates some of the existing primitives while integrating hybrid pre/post quantum methods.

First we should isolate out the various cryptographic components. There is the hashing and symmetric encryption. There may be developments here, more modern sponge based hashing may be used (the SHA2 family is already looking old), and there may be light-weight encryption to facilitate low-powered devices, which may displace the AES in some applications. Better AEAD modes than (the rather fragile) GCM may become popular.

But the most interesting developments will be in the public-key sphere. Recall that TLS requires

(a) An unauthenticated key exchange, and
(b) A digital signature on the protocol transcript by the server (and optionally the client), to authenticate the link in one (or both) directions.

Going Post-Quantum

The simplest way to make TLS secure in a post quantum setting is to replace the key exchange and digital signature algorithms. There are numerous ways of doing this, and the industry will be guided by the outcome of the NIST competition. Currently I like the idea of an isogeny based key exchange like SIKE, and a lattice based digital signature scheme like Dilithium.

Client authentication

While awaiting a quantum computer, it might be interesting to look again at client-side authentication.

The standard TLS client side authentication requires the client to have an X.509 certificate of their own, containing a private signing key. As is well established this is a deeply unattractive solution, and it is rarely used. So alternative methods can be considered.

The simplest idea is to do client authentication as part of the application. Typically the first HTML page thrown up by a server that requires client authentication may be one requesting Username and Password. And any other client authentication scheme (maybe a two-factor method?) can be implemented in a similar way, after the TLS handshake completes.

But it would be cool if the client-side authentication could be more tightly integrated into the TLS protocol flow, so that the amount of toing and froing could be kept to a minimum. Recall that standard TLS client side authentication involves the application of a digital signature to a hash of the protocol transcript. In theory any alternative method of digital signature could be substituted here. So maybe identity based signature, or so-called certificate-less signature might be considered? Or a designated verifier signature scheme? Signatures come in all sort of flavours (blind, ring, aggregate, proxy, group, redactable, linkable etc), many of which may be relevant, and interesting to research. I have a designated verifier signature scheme of my own, using pairing-based cryptography, which allows multi-factor authentication of the client.

For another example, an asymmetric Password Authenticated Key Exchange can be considered, where the client authenticates with a simple PIN. This draft draft-sullivan-tls-opaque-01 (ietf.org) already considers methods of integration of a PAKE with TLS. It suggests that we might be able to repurpose/reuse some of the existing TLS data exchanges to facilitate integration. It would be fun to experiment with this.

Another Way

A completely different approach would be to implement a novel authenticated key exchange protocol between client and server, and to drop the negotiated key into TLS as a pre-shared key, and let TLS take it from there.