Trust Management

At the highest level TLS is a protocol in which one party sends a credential to the other which proves their identity and allows the derivation of trusted cryptographic keys.

At a lower level a sender who possesses a private key sends a blob of data to a receiver which provably binds the senders identity to the associated public key.

Going lower again, that blob of data is probably a “certificate chain”, and the identity and public key are embedded in the certificate at the front of the chain.

Ideally both server and client should be both senders and receivers of certificate chains, to allow for mutual authentication. Commonly however only the server authenticates to the client in this way, and the client either does not authenticate at all, or does so via some non-TLS mechanism, maybe just username/password once the link is established. Note that a “live” client may authenticate by various elaborate means, including biometrics. An IoT thing on the other hand may have to resort to using a certificate chain, or some other blob of data which fulfils the same purpose.

The blob of data is fixed (read-only), and could be stored in a file. It is not a secret, and it contains no secrets. It could be “cached” by the recipient so that it doesn’t have to be sent again.

For the implementer there is a need to have a function which inputs that blob of data and the claimed identity, and reports back either Valid or Invalid. If Valid it also extracts the public key.

The private key may be kept in a file (insecure!), it may be embedded in the application (dodgy!) or it may be kept in some secure hardware vault (probably the best option, if available).

A Trust Abstraction Layer?

Now in the future the PKI/X509/ASN.1 certificate chain approach to the provision of that blob of data may change. So maybe it makes sense to try and “abstract out” the whole trust resolution issue. This may not be so easy – PKI and certificates have been part of TLS/SSL since the very start. They are part of its culture.

One thing that should relatively easily be abstracted out is certificate chain checking. There may be 3rd party tools which already do a good job of this. In our current client we have implemented a rather basic chain-checker, but it could certainly be improved upon, and it would be nice to be able to optionally use external resources. Maybe even resources that check for revoked certificates by looking up CRLs (Certificate Revocation Lists).

Also we may want to provide a range of plug-in certificate chain resolvers ourselves, from the extremely simple (for a closed IoT system for example) to the very elaborate.

Second thoughts

As a way of preparing better for a post quantum future, I decided to implement the Dilithium proposal for lattice-based signature. It is one of the three finalists in the NIST competition, and from the buzz around it, a very likely winner.

What an eye opener! All is not at all well in the PQ signature world. Dilithium is inelegant, shallow, quite fast and extremely clever. It typically takes multiple (~5) attempts until it produces a safe signature. For standard levels of security it generates public keys of 15316 bits and signatures of 26344 bits. Even RSA on steroids doesn't produce keys and signatures anywhere near that size. Dilithium is also quite greedy of RAM, at least 32K of memory seems to be required to store its arrays of polynomials. On IoT nodes its timings are competitive with elliptic curves, although signature times obviously can vary dramatically depending on how many attempts are required to achieve success.

So simply using Dilithium as a drop-in replacement will result in massively bulked up certificates, and require IoT nodes with more RAM. Now I see the sense in going with the KEM based alternative (as discussed in my last blog). Lattice-based KEMS are much smaller. But even then its not a straightforward choice – live with massive certificates, or introduce another flow into TLS. Either way we get a slower, kludgier TLS.

What the world needs is an isogeny based signature scheme, to restore keys to sensible sizes, and reintroduce some mathematical elegance (and depth) into our cryptography.