Time to consolidate and plan a way forward

This is going to be a big project, no doubt about it! The trick is of course to break it down into smaller pieces. One advantage of implementing a client-server protocol like this, it that one can work independently on the client and on the server. So it seems like a good idea to concentrate first on creating a TLS1.3 client. This can then be tested against the many TLS1.3 servers already out there. And it is the client side which will have to be integrated into the challenging IoT environment – the server can be assumed to be far less constrained in terms of computing resources.

A lot of unexciting work has gone into simplifying the code and extracting out stuff into separate modules, which can be independently tested.

Choices

One scary aspect of TLS1.3 that I mentioned in my last blog is the potential number of choices available. Turns out not to be such a big issue.

Now a TLS1.3 client is unlikely to be expected to have the intelligence to make decisions at run time as to which encryption algorithms to use. So the basic idea is that the client suggests what it is capable of, and the server responds with its preference from among the choices presented by the client. By offering a very limited number of choices, the client can make life much easier for itself, and keep itself small

Section 9.1 of https://tools.ietf.org/html/rfc8446#page-102 is an eye-opener in this regard. For example a compliant client need only support the cipher suite TLS_AES_128_GCM_SHA256. The number of choices is in all cases tightly constrained. This eliminates the possibility of "fall-back" attacks where the client and server are tricked into falling back on an old broken legacy cipher.

Therefore a client need only suggest the cipher suite TLS_AES_128_GCM_SHA256 and a compliant server must accept it. However this method is quite flexible, for example our client may suggest TLS_AES_128_GCM_SHA256 and a Post-Quantum cipher suite. If the client is connecting to one of our servers, the post-quantum suite can be chosen, or maybe a hybrid suite.

It appears that only the SHA-2 family of hash functions are allowed, so newer SHA-3 based methods are not required, and SHA-1 methods are deprecated (and we will not support). Also AEAD encryption uses only AES_GCM, or possibly CHACHA20_POLY1305. Since we do not have an implementation of the latter to hand (and it is only something which SHOULD rather than MUST be implemented, we will simply omit it for now.

Processing Certificates

As I pointed out last time this is a big part of the client's task. However we do not need to support any old certificate that is out there. For example if a certificate is signed using the legacy MD-5 hash, TLS1.3 drops the link with a "bad certificate" error. So although lots of diverse certificates must be accepted, there is a limit.

The code to process Certificate chains is not really TLS1.3 specific, and I would imagine it hasn't changed much from earlier versions of TLS. Nonetheless we need to write the code, and that will be my next task.

Peeking ahead

I am looking forward to getting into the Pre-Shared Keys feature of TLS1.3. It will be great to dispense with certificates and maybe allow support for non-PKI alternative key exchange algorithms.