Pre Shared Keys and Session Resumption

I was expecting this part to be tricky, and it was.

As is well known one of the main achievements of TLS (other than encrypting the client-server traffic) is to authenticate the server to the client (and optionally the client to the server). The full TLS handshake achieves this by the server presenting a valid certificate chain, and the client checking it out. But that's slow and expensive. It would be better if authentication could be speeded up. One way to do this is by assuming that both parties already share a secret. But arranging in a secure way for some pre-shared secret between a client and server that have never met before is in general not easy – its the classic key distribution problem.

But in fact after a first full TLS handshake is completed, both parties do in fact share a stonking big secret, basically the key that they used to encrypt their communications. So if they ever want to connect again, they could use this secret (if they can remember it) to avoid the certificate processing on their next connection.

This is called session resumption, and its a big feature of TLS1.3. At the end of the full handshake the server passes a "ticket" to the client (based on their shared secrets), that the client can cash in on their next connection. The form of the ticket is up to the server and its format is not defined in TLS1.3. Suffice it to say that the server on receipt of a valid ticket can look up a shared secret on their side. Note that these tickets have a limited life-time.

Its important to note that a pre-shared secret does not have to arise in the context of session resumption. It could be derived out-of-band by any alternate mechanism we could dream up.

Getting it working

Not having a TLS server that outputs debug information makes development rather difficult. Its easy to configure a clientHello message to ask a server to accept a ticket, its not so easy to figure out why such a request might be ignored (with the server falling back to insist on the full handshake, which it will do if it senses that all is not well). So there was a lot of trial and error. Also the RFC is in places a little ambiguous. I had a look at some open source TLS implementations. Of the ones I looked at fizz appears to be the best (https://github.com/facebookincubator/fizz) but it was hard to get it to output the information I needed.

But the good news is that I did get it working. The latest version of client.cpp now connects to a server (like tls13.cloudfare.com), does a full TLS handshake, times out after 5 seconds and then uses a ticket to successfully resume the connection.

Whats next?

I have also implemented Key Update (see last blog), as it turned out to be very easy to do.

One advantage of a pre-shared key is that in theory it allows 0-RTT. What this means is that a client can send along with its initial clientHello some encrypted application data, without waiting for any response from the server. Conceptually its easy to see why this is possible – this "early data" can be encrypted with the pre-shared key. However there are serious security downsides to this as well. But since we have come this far, sure why not. After all what harm in sending an early HTML GET command to the server.

However first I need to do some code refactoring and tidying up.