## Starting

I can't emphasis enough that my own starting knowledge of TLS is embarrassingly limited. I know what SSL technology does for us – it authenticates a Server to a Client (but not the other way around – the client authenticates via Username/Password if there is a need to do so, or with credit card details, it depends on the Website).

To this end the Server has possession of a PKI certificate containing their public key. This X.509 cert is digitally signed by a Certificate Authority (CA). The client accesses the certificate (it is a public document), checks the signature with the CA public key (which is conveniently embedded into their browser), generates a random AES session key, encrypts that with the server public key extracted from the cert, and sends it to the server. The server decrypts the session key with their private key, and now client and server share a session key and the talking can begin.

That's actually quite complicated, and involves a lot of encryption keys. There is the final AES session key, the Servers long term public/private key pair, and the CA's public key. In fact as we shall see, there are even more than that.

This was my starting point level of knowledge. But now I need to drill down a little deeper.

The only way I reckon to really understand something, is to make one yourself. The starting point should be RFC8446, but standards documents are deeply unfriendly, and anyway others before me have already gone to the trouble of digesting it.. So my actual starting point for implementing TLS1.3 was this website

https://tls13.ulfheim.net/

which helpfully does a byte-by-byte walk-through of the most common sequence of events that take place during the set up of a typical TLS session. Another useful site is

https://commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art080

and there are also some nice videos out there like

https://www.youtube.com/watch?v=yPdJVvSyMqk

I recommend that if you want to follow this blog, first watch this short video.

Obviously some crypto capability will be required, and I am not intending on writing any brand new crypto implementations myself at this stage. Been there, done that. My current offering and the one I will be using here initially can be found at

https://github.com/miracl/core

which provides all of the crypto tools I will need for now.

## The Initial Handshake

The whole protocol kicks off with an exchange of Hello handshake messages where server and client introduce themselves to each-other, and more importantly make clear to one another their individual capabilities and preferences, particularly in terms of which crypto methods to use.

Basically a short initial negotiation takes place. A major contribution of TLS1.3 is to simplify and shorten this negotiation, into a simple over-and-back.

I was surprised to see that this initial exchange of Hellos is also used to negotiate an unauthenticated Diffie-Hellman key exchange. The mutual key that results from this exchange is then used to generate more keys to encrypt everything else that follows. From one key of sufficient entropy many more can be spawned, and here the HKDF HMACing method is used, based on standard hash functions. So to even get to this stage the crypto tools I needed were X25519 key exchange, HKDF, and standard hash functions like SHA256

Following on from the negotiation of capabilities/preferences, a modern AEAD symmetric suite is chosen for this encryption – usually an AES-GCM mode. So old fashioned Cipher-Block chaining is now a thing of the past...

Not so fast. TLS1.3 tries to modernise the crypto, but the requirement for backward compatibility is a curse. The main problem is that certificates can live for 10 years or more, and so public keys, and keys used to sign certs, may be from a bygone age. So there is still a need to support legacy methods like RSA, or else we couldn't process the X.509 certificates.

That's my next step… I can use the openssl command line tool to generate some sample certs…

*Peeking ahead*

*I was vaguely aware of the PSK – preshared keys – capability of TLS. Basically it is possible to dispense with PKI and Certs and use instead keys that were negotiated via some other out-of-band method. This opens up some intriguing possibilities. Could those keys be established using Identity based encryption? Could a PAKE (password authenticated key exchange) method be used? PAKEs are in the process of being standardised right now. Could some post-quantum methods be used here to get post quantum into TLS without changing the standard?*