On using IBE for TLS – a research proposal.

This has been suggested before, and mentioned before in these blogs – see the paper by Banerjee and Chandrakasan (B&C). https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9148829&tag=1

We will also refer to KEMTLS – see https://eprint.iacr.org/2020/534 by Schwabe, Stebila and Wiggers. The idea behind KEMTLS is to use a KEM rather than a digital signature to establish authenticity of an entity. The downside is that, using standard public key cryptography, the first party needs to know the second party's public key before it can send it a KEM-encapsulated to-be-agreed session key.

However using an IBE-KEM this situation changes. Now the client can immediately encapsulate a session key to the server, knowing only its identity, which it already knows (typically the URL being visited). This changes the dynamic and solves a lot of problems.

There is a problem with IBE that is rarely fully acknowledged. It is not in fact sufficient to know "only" the identity of the other party. It is also a requirement to know the public value associated with the entity which issued private keys – the IBE Trusted Authority (TA). However using standard PKI the client currently needs to know the Certificate Authority (CA) root certificate public key. The requirement for the client to know either a PKI CA public key or an IBE TA public key amounts to basically the same thing. It would be built into the application. So we can assume that every client knows the public value associated with the TA that issued the Server's private IBE key.

So how exactly to integrate IBE into TLS1.3 without turning it into a different protocol? Ideally we would like to bootstrap into all of the confidence that using TLS gives us. So we would not like to have to make changes to the state machine.

B&C suggest creating IBETLS by bundling an encapsulated key share using an IBE-KEM, with the key exchange encapsulated key KEX-KEM, and then dropping the Certificate and Certificate Verify messages from the full handshake.

Here we propose a different approach. We suggest that more a natural way to integrate IBE into TLS1.3 is to modify the TLS1.3 *resumption* handshake rather than the *full* handshake. The intuition is that **knowing the identity of the Server, and its TA public value, is equivalent to already possessing a pre-shared key (PSK) which authenticates the server**.

Therefore to use IBE the client simply performs a resumption hand-shake, sending an IBE-KEM encapsulation of a shared secret as the PSK identity in the standard pre-shared key extension. The server decapsulates the implicitly pre-shared secret, and the handshake proceeds as normal.

Authenticating the client is a little more complicated. However is no compelling reason to use IBE for this, so either client authentication is carried using the post-handshake certificate-based method supported by TLS (which can exploit the hardware support client side IoT devices commonly have for PKI secrets), or some method external to TLS can be used.

A feature of the resumption/PSK handshake is the ability to send encrypted "early data" as part of the client hello. So we can also do this here.

After a successful connection, a standard resumption ticket can be issued and the classic TLS1.3 resumption protocol used for subsequent connections.

Implementation

To implement an IBE-KEM we can use either a pairing-based protocol or the post-quantum lattice based method of Ducas et al. as described in B&C. Fortunately we already have an implementation of the latter, and the MIRACL library has extensive support for pairing-based crypto.

Using pairings we could combine Boneh & Franklin IBE for the IBE-KEM with X25519 for the KEX-KEM. Using lattices we could combine Ducas et al IBE with the soon-to-be standardised Kyber for the KEX-KEM.

Using pairings the implementation will be slow, but the bandwidth small. Using lattices the implementation will be fast, but the bandwidth large. An obvious extension is to go “hybrid” and combine both methods.