Going Hybrid

How is the transition from pre-quantum to post quantum cryptography going to pan out? In truth (a) its not going to happen for a while, and (b) its not going to happen overnight, some night in the future. The smart way to proceed is to first go hybrid. Basically this means introducing a post-quantum component into our protocols.

In the case of TLS1.3 this will impact both the key exchange and the digital signatures. The two cases can be considered separately.

Key Exchange

The concern here would be for an opponent who records our TLS handshakes, and stores those recordings away until sometime in the future when a quantum computer allows the crypto to be broken and the exchanged data to be decrypted. So something needs to be done about this well before an actual quantum computer exists.

Most TLS1.3 key exchanges use the popular X25519 Montgomery elliptic curve. This has the advantage of being very fast, and very small – public/private keys and shared secrets are all just 32 bytes in length. However this is broken in a post-quantum world, so we need to back it up with a post quantum KEM. Considering the NIST recommendations for standardisation, the obvious choice now would be Kyber. This still leaves one question – at what security level should we implement Kyber to match X25519? Mapping lattice schemes to well understood levels of security (which might be loosely understood as 128/192/256 bit levels of security), is notoriously difficult. Three levels are proposed, but the inventors themselves recommend not using the lowest security scheme as improvements in attacks on lattices might be expected in the near future. So they recommend the Kyber768 variant, which in theory maps to 192-bit security, but which provides a good margin of safety for current use at the 128-bit level, which matches the security offered by X25519.

We use the simple “concatenation” approach. The public keys for both X25519 and Kyber768 are concatenated into a single bloated key, which is then handled by TLS in the standard way. The final session key is found by applying a standard KDF (Key Derivation Function) to the concatenation of the pair of shared secrets generated by each scheme. An attacker needs to break both schemes to break the encryption.

It all works fine, but note that the public keys now swell from 32-bytes to 1184+32 bytes!

Authentication

This is probably less urgent, but important nonetheless. Its less urgent as only when a quantum computer actually exists can authentication be broken, allowing Man-In-The-Middle attacks to be launched against TLS. The idea here is again conceptually quite simple: Sign certificates and transcripts twice, and verify both signatures.

Clearly this will require non-standard certificates, but here the great work of the Open Quantum Safe (OQS) project can be exploited. They provide a forked version of OpenSSL which generates hybrid certificate chains. A variety of hybrid options are supported, but the choice is now much more constrained by the NIST recommendations. Here we will adopt a hybrid approach using a SECP256r1 standard elliptic curve signature, alongside a Dilithium2 signature. Again there is a question of the security level to adopt for Dilithium to match the elliptic curve. But in this case the decision is already made for us, as OQS themselves choose Dilithium2 to match SECP256r1. The 2 refers to the security level, and again it represents a conservative choice in terms of matching the elliptic curve. A minor glitch arises as OQS decided to use SHA384 instead of SHA256 as the hash function to use with the elliptic curve. We are unsure of the reason for this (the TLS standard only supports SHA256 with SECP256r1), and queried it with the OQS team (who themselves seem to be somewhat confused about it). But in the end it was simpler to support the non-standard scheme.

Public keys and signatures are again simply concatenated, and processed as opaque data by TLS, so minimal changes to the protocol were required.

But again observe in particular that certificates (which embed a public key and have an appended signature) blow up in size significantly.

Performance impact

For Kyber and Dilithium we use the new C++ and Rust implementations from MIRACL in the SAL, as a smaller footprint alternative to the Rust versions of the reference code provided by OQS, although the latter can also be used with minimal changes to the SAL.

We have it working on a PC, but next we will implement on our IoT node. The sizes of public keys and signatures will not be much greater than for our purely PQ experiment reported on in our last blog, so no surprises expected there. However compute time will clearly increase, as there will be twice as many public key operations required by both parties. But I am still confident that the timings will be acceptable.