The TII TLS 1.3 Project – Vision and objectives

This project will deliver a full clean-room implementation of the TLS1.3 standard as described in RFC8446. It will be delivered as a client written in both C++ and Rust, and a Rust Server. It will have general application, but will be particularly well suited for Internet of Things (IoT) applications. It will be a solid and reliable implementation of the current standard, written and documented to professional standards, and directly competitive with commercial products like WolfSSL. The use of Rust as an implementation language will be seen as providing confidence in the correctness of the implementation, as it eliminates the possibility of certain type of programmer error that have proven problematic in the past when implementing TLS and its SSL predecessor.

What is TLS?

TLS 1.3 is a cryptographic protocol which protects internet communications. It depends on several important cryptographic primitives in order to succeed.

1. A Hash function (on which is also built a HMAC and Key Derivation Functions)
2. A symmetric cipher used in an AEAD mode of operation
3. A KEM, or key exchange algorithm.
4. A digital signature algorithm

In each case a limited range of choice is allowed. For example for key exchange and digital signature one of a number of standard elliptic curves are permitted. Some of these choices are of necessity legacy, as due to its use of the Public Key Infrastructure, certificates are used that may have been issued decades ago. Certificates are used to verify the public keys used to check the validity of digital signatures.

TLS is of course most commonly used in the context of website access, creating a secure channel between a browser and a server. But this is far from being its only application. In an IoT setting TLS may be responsible for connecting an IoT node to its controlling server, for the exchange of application specific data. One advantage of using TLS is that it has been widely studied (and attacked) by the international community. It is uber mature. In its current manifestation as version 1.3 there are no known weaknesses. One plausible concern however would be that, to maintain backward compatibility, it has become bloated and weighed down with legacy code, which can easily result in programmer error. By writing a clean-room implementation of only the latest version TLS 1.3, we will side-step this problem.

How will our implementation differ?

It may be the case that in a particular context a deployer may want to use non-standard primitives. For example for reasons of national security it may be desirable to use nationally approved and non-publicly disclosed primitives. An example would be the use of non-standard elliptic curves chosen using state-of-the-art criteria, rather than older legacy standards which in some cases exhibit known vulnerabilities.

Another consideration may be the need in the near future for wholesale replacement of some or all of the existing primitives with post-quantum alternatives. In the event of the invention of a quantum computer entirely new methods for key exchange and digital signature must be used, and even choices of hash functions and AEAD ciphers may be impacted. New post-quantum certificates will be issued, and will need to be processed.

Given the above there is a clear need for a new implementation of TLS to be super agile and flexible, and to adopt a pluggable architecture, so that cryptographic primitives can be plugged in and out with ease. Furthermore these primitives may need to be chosen from a wide variety of sources. Cryptographic primitives are notoriously difficult to write efficiently and securely. In particular recall that in an IoT setting side-channel attacks are a very real threat. Therefore there is unlikely to exist a single resource from which all of the best-of-class primitives can be found. For performance reasons many are written in assembly language, and of course this requires good quality implementations for a whole range of processors, as are often found in the IoT setting. Furthermore some primitives in an IoT node may be available via fast secure hardware, and a TLS implementation should be able to easily exploit such a resource, if it were available.

Therefore the TII implementation will deploy a flexible and agile Security Abstraction Layer (SAL) to permit a pluggable architecture where cryptographic primitives from a variety of sources (both software and hardware) can be used.

In the IoT context an important issue will be the potentially severe constraints on memory (SRAM and Flash), and on CPU processing capability. It is vital therefore that care be taken in the course of development of the TLS client that it not be profligate with resources. Simply put, an implementation that doesn’t fit snugly inside of an IoT node, which for example exhausts its available SRAM, is clearly useless.

The current state of affairs as regards the best choice of post-quantum primitives, is an area of ongoing and active research. At the time of writing the NIST competition is coming to an end, and it is expected that one or more new primitives will be standardised in the various categories. However that is unlikely to be the end of the story, as it will probably be many years until the threat of a Cryptographically Relevant Quantum Computer (CRQC) becomes real, and in the meantime post-quantum cryptography is likely to undergo further refinement and development. This will, in our opinion, likely lead to a new NIST-like competition in the medium-term future, and the emergence of new and improved standards.

Given the above it is an important design objective of the TII-TLS implementation that it provide an ideal test-bed for observing and measuring the impact of various choices of post-quantum primitives on the performance of the TLS protocol, in particular in the IoT setting. It is to be hoped that this will result in our making a meaningful contribution to ongoing international research.

The future

It would also be our ambition over time to abstract out as much as possible the trust management aspect of TLS, so that trust establishment alternatives other than PKI can be considered (such as the use of pre-shared keys).

Finally we are aware that TLS 1.3 is unlikely to be the end of the story. There will be a TLS 1.4 or a TLS 2.0. These will involve changes to the core protocol orthogonal to the choices of cryptographic protocols. We hope to be in a good position to rapidly track (and indeed get ahead of) any such developments.