Using other libraries

The TLS implementation is currently built on the MIRACL Core library, which provides much if not all of the cryptography necessary for a TLS 1.3 implementation.

But there are gaps in its coverage. For example it does not implement ChaCha20/Poly1305 which is a nice AEAD algorithm based on a stream cipher. It is faster than the AES-GCM alternative, and appears to have better immunity to side-channel attacks.

A very good implementation is available in the popular Libsodium library. This is a C library, which also supports a very fast X25519 implementation, plus fast hashing and other stuff. It does not cover as much of the TLS requirements as MIRACL Core, but what it does do, it does well.

So there is a case to be made for allowing the TLS client to optionally be built using libsodium functions where they are available, and falling back on the MIRACL core functionality for things that libsodium does not implement. It is also likely that different libraries may have different strengths, so a client may want to change some of its preferences based on those strengths.

Making a start

To make a start we first included support for ChaCha20/Poly1305 using the libsodium implementation. And if libsodium is available we also make ChaCha20/Poly1305 the client's number one choice of cipher suite.

Recall that the client is written in C++, while libsodium is a C library. This is easily handled in a standard way by wrapping the `#include <sodium.h>` header in an `extern "C"` clause. This resolves the name-mangling issue that is the only important incompatibility between the two languages.

Next we observed that libsodium has a nice random number generator, that uses OS specific resources to generate random numbers. So we can make use of that.

The X25519 implementation in libsodium is state-of-the-art, so we now make use of that as well. All of the code changes are in the single `tls_crypto_api.cpp` module.

To exploit these changes, install libsodium, edit `tls1_3.h` and `#define USE_LIB_SODIUM`, and when compiling the client simply include `-lsodium` in the command line.

Now the client is using state-of-the-art implementations for AEAD encryption and X25519 key exchange! These are probably the most compute intensive components of TLS1.3

Downsides

Besides the lack of complete coverage of TLS requirements, libsodium can sometimes be an awkward fit. It is not available in other languages, although wrappers are available. A pure rust implementation would certainly be nice. It also uses some assembly language, and architecture/OS specific features, so it is not trivial to port. Its not at all clear how it could be used on a bare-metal IoT node.

There is support for AES-GCM, but only with AES-256 and not AES-128. It does not directly support HKDF key derivation functions, although it does support HMAC. Again there is support for SHA256 and SHA512, but not for SHA384. In general it likes to do things its own way.

But we can pick out the “good bits” of libsodium, and if the architecture/OS/programming language is compatible, we can make good use of them.