Exploiting hardware

In the world of IoT, crypto co-processors are having a moment, and they are creeping into new IoT node designs. A good example would be the Microchip ATECCX08 family. These provide (by design) a lot of the crypto capability needed by a TLS implementation.

Crypto co-processors introduce the promise of secure key storage, true random number generation, fast implementation, and a reduced software (Read-Only Memory) footprint. In particular they claim to be able to generate and keep safe an unclonable TLS private key, from which an associated public key certificate can be issued and used to authenticate each particular device, a standard method for TLS client authentication. They cannot be ignored.

Should we trust secure hardware?

Well, no, not really. I take claims of secure storage with a pinch of salt, but whatever protection they provide is going to be better than anything we can do in pure software, even if there is a back-door. Same goes for their random number generators, although we will only use them to provide seeding for our own PRNG, which will mix in any other sources of unpredictable entropy we can lay our hands on.

What is provided, and how can we make use of it?

Surprisingly exploiting the crypto co-processor is nowhere near as easy as it should be. I bought an Arduino Nano 33 IoT board complete with the ATECC608A chip. Now the Arduino ecosystem is one of the most extensive out there. It does provide an ECCX08 library – but it only supports a fraction of the functionality of the crypto co-processor. Furthermore the full data-sheet is apparently only officially available under NDA from Microchip, although it does seem to have leaked out onto the Internet. So the support provided is kind of half-hearted, like they really don't want you to know how to fully exploit the chip. Strange.

But not a show-stopper. One of first things I noticed was that while library support was provided for generating private/public key pairs on the NIST standard secp256r1 elliptic curve, there was no support for implementing a simple ECDH key exchange. Despite the fact that the chip supported this function. However it didn't take long to clone the library and add in the missing function, using the information provided in the data-sheet.

In fact everything needed for the secp256r1 curve is supported by the hardware, and therefore it is now possible to drop the software support for this curve, with a nice reduction in code size. The speed-up is also very noticeable, and very welcome. I just hope the hardware doesn't leak any side-channels, as obviously I can't fix that. One downside of exploiting a crypto co-processor is that it represents an expansion of the trust surface.

The original ATECC508A chip supported ECDA/ECDH on the NIST standard secp256r1 curve, SHA256 and HMAC, plus random numbers. The follow-up ATECC608A added AES-GCM support, plus HKDF, both very useful for TLS 1.3.

Worryingly new designs are advised to move up to the ATECC608B chip which adds no extra functionality, but claims to be "security enhanced".

For a possible explanation see https://i.blackhat.com/USA-20/Thursday/us-20-Heriveaux-Black-Box-Laser-Fault-Injection-On-A-Secure-Memory.pdf

Which rather nicely sums up the reason that I don’t trust secure hardware.

Another issue – what about the cryptographic primitives that are not supported? A big omission is support for the X25519 curve for key exchange, although its not strictly needed. A bigger issue might be the lack of support for RSA, for verifying signatures on certificates. However RSA verification is already very fast in software, so there is less justification for hardware support.

So overall I think that the ATECC608 makes good choices for efficient TLS1.3 support on IoT devices, and we plan to provision a SAL (Security Abstraction Layer) that makes full use of it.