Arduino Nano RP2040

Well I got a couple and did some tests. This is a relatively new microcontroller and some of the software support is still fairly immature. To recap it has 16M of flash and 264k of SRAM, which should be plenty, and will probably become an IoT standard expectation in terms of memory resources. It is also pretty fast, a high clock speed (133MHz) offsetting a relatively poorly endowed 32-bit processor (ARM M0+).

The standard Arduino support includes a basic MBED real-time operating system, so access is not directly to the bare metal. It would I guess be common for example to implement some kind of file system in that 16 Megs of flash, so some kind of lightweight OS is probably necessary.

But there is a problem. The 264k of SRAM is available to a C++ program as 32k of stack and the rest as a heap (less some OS requirements). Now normally in the Arduino world (and indeed the IoT world) the stack starts at the top of memory and works its way down. The heap starts at the bottom and works its way up. See for example the nice diagram in https://learn.adafruit.com/memories-of-an-arduino/optimizing-sram. This means that if I don’t use a heap, I get all of RAM as a stack.

But the MBED OS restricts the stack to just 32k.

What have I got against the heap?

First its a big source of hard to detect programming errors. Memory gets allocated, and the programmer forgets to deallocate it. Second memory allocation and deallocation leads to memory fragmentation, and handling this kind of memory is next to impossible in constant time. So it may leak a side-channel that may cause us a problem. Stack-only is safer.

Problem

The problem is that a stack-only program is therefore limited to 32k of memory – the other 232k is basically inaccessible. Fortunately there are a couple of workarounds.

First don’t use the Arduino MBED OS, as there is alternative firmware due to the wonderfully named Earle F. Philhower III. See https://github.com/earlephilhower/arduino-pico. This allows a full-sized stack, but unfortunately does not yet fully support our Arduino Nano RP2040 variant of the Raspberry Pico controller.

A second idea is to observe that a major memory requirement is for a large I/O buffer. By making it an option to allocate this from the heap rather than from the stack (and carefully deallocate it at the finish), then there is plenty of space on a 32k stack for our TLS1.3 client. Hopefully at some time in the future the stack/heap memory split in MBED OS will be made a programmers option.

Another problem

The Arduino Nano microcontroller also provides hardware support for some common crypto operations, using an ECC608a chip. Notably it provides random number generation, secure long-term key storage, symmetric crypto and comprehensive support for the secp256r1 elliptic curve.

Strangely the ECC608a symmetric crypto support is slower than a software implementation – see https://eprint.iacr.org/2021/058.pdf. Therefore we see no point in using it. However the random number generation and elliptic curve support is of much more value. For example the ECC608a can generate a private key and its associated X.509 public key certificate, either in self-signed form, or in a form ready to be signed by a certificate authority. The private key is kept in secure storage, and is inaccessible to anyone, even the individual who creates the private/public key pair. It is the boards own PKI private secret that uniquely identifies it. The ECC hardware can also quickly verify secp256r1 signatures on certificates in a certificate chain. And importantly the hardware is significantly faster than a software implementation (which anyway would not have access to the secure storage facility).

So we want to use this chip. However it is not yet officially supported for this board. We did get it to work, but it is unfortunately prone to rare random errors. We suspect that interrupts are getting missed, but anyhow hopefully its just another teething problem.

The good news

Is that our TLS1.3 client no longer requires modification to work on an IoT node. The full set of root certificates can be stored in flash memory, and compromises do not need to be made on array sizes to stay within the stack limit.