Dear all,

an updated implementation of Falcon is available on the Falcon Web site (downloadable archive and browsable source code): https://falcon-sign.info/
There is a note that describes how it works, and contains some benchmarks: https://falcon-sign.info/falcon-impl-20190802.pdf

Highlights:
- It's fully constant-time (keygen, signature generation).
- Optionally, it's even constant-time for signature verification, in the unusual case the message, signature and public key are all secret and the message has low entropy.
- It can run on hardware without a FPU.
- It can leverage a hardware FPU, and (on x86) it can optionally use AVX2 and FMA opcodes.
- It is faster and more RAM-efficient than the previous reference platform. It uses less than 3 kB of stack space, allowing use on embedded microcontrollers where stacks are usually short.

A new private key storage format has been designed; private keys now fit in 1281 bytes (Falcon-512) or 2305 bytes (Falcon-1024). Public key encodings and sizes are unchanged. The per-signature random nonce (40 bytes) is now integrated in the signature format, to ease interoperability and usage; this brings average signature size to about 651.59 bytes (std.dev: 2.55) for Falcon-512, 1261.06 bytes (std.dev: 3.57) for Falcon-1024. An additional fixed-size signature format is added (809 and 1577 bytes, respectively) to support the optional constant-time encoding and decoding of signature values.

On a Skylake core in 64-bit mode, per-signature generation cost is down to about **389000 cycles** (82000 cycles for verification) for Falcon-512, i.e. the number of signatures per second, on a single core of my MacBook Pro (i7-6567U, 3.3 GHz turboboosted to 3.6 GHz) is over 9000. For Falcon-1024, signature generation is 790000 cycles (158000 cycles for verification), i.e. more than 4500 sign/s, at the highest long-term security level.

On an ARM Cortex M4, a Falcon-512 signature can be obtained in **19.6 million cycles** (41.4 million cycles for the RAM-efficient version, which uses less than 40 kB of RAM), and verification is 511000 cycles.

Internally, the source code combines four distinct engines that share the same structure. It can be viewed as the fusion of four implementations:
- "fpu": uses a hardware FPU. Tested on x86 (32-bit and 64-bit, with GCC, Clang and MSVC), PowerPC (ppc, ppc64le and ppc64be, with GCC, Clang and XLC), and ARM (aarch64 and armhf, with GCC and Clang).
- "avx2": hardware FPU + AVX2 intrinsics (and optionally FMA intrinsics). Tested on x86 32-bit and 64-bit, with GCC, Clang and MSVC.
- "int": uses integer code only, portable everywhere.
- "cxm4": like "int", but with inline assembly for some operation (ARMv7-M, works on the ARM Cortex M3 and M4, but since the M3 has non-constant-time multiplications, the code is constant-time only on the M4).

The "int" code has been integrated into PQClean under the name "clean": https://github.com/PQClean/PQClean
The "cxm4" code has been integrated into pqm4: https://github.com/mupq/pqm4
The "int", "fpu" and "avx2" have been submitted to SUPERCOP ("int" is called "ref" in this one).
All four implementations have been sent to the NIST under the NIST API for their own benchmarks.

--Thomas Pornin, for the Falcon team.
Hi Facon team;

I have two remarks:

1- In the document -> algorithm 6 -> line 11 you wrote:
   F<-- F'(x^2)g'(x^2)/g(x), I ask if it is exact or it should be F<-- F'(x^2)f'(x^2)/f(x).

2- I ask if it is possible to sample F,g and f and compute G=(q-gF)/f ?

I think it is more easy and faster!

Best regards