From: 'Thomas Pornin' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Saturday, August 3, 2019 8:47 AM
To: pqc-forum
Subject: [pqc-forum] OFFICIAL COMMENT: Updated Falcon implementation

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.

From: EL HASSANE LAAJI <e.laaji@ump.ac.ma>
Sent: Sunday, September 15, 2019 10:23 AM
To: pqc-forum; pqc2019; pqc-comments
Subject: Officiel comment: FALCON

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

From: Thomas Prest <thomas.prest@pqshield.com>
Sent: Monday, September 16, 2019 1:32 PM
To: EL HASSANE LAAJI; pqc-forum; pqc2019; pqc-comments
Subject: Re: [pqc-forum] Officiel comment: FALCON

Dear El Hassane,

On 15/09/2019 16:23, EL HASSANE LAAJI wrote:

> 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).

Thank you for this remark. The line 11 of Algorithm 6 is actually exact, however there is a typo in line 12, which should be G <-- G'(x^2) f'(x^2) / f(x) instead of the current G <-- G'(x^2) g'(x^2) / g(x). We will correct this typo in the next iteration of the specification.

A quick explanation of "why this works": by induction hypothesis, it holds that f'(x^2) G'(x^2) - g'(x^2) F'(x^2) = q mod(x^n + 1) (hence f', g', F', G' satisfy what we call the NTRU equation mod (x^{n/2} + 1)). On the other hand, f'(x^2) = f(x) f(-x) and g'(x^2) = g(x) g(-x). Therefore we can rewrite the NTRU equation as f(x) [f(-x) G'(x^2)] - g(x) [g(-x) F'(x^2)] = q mod(x^n + 1). Taking F = g(-x) F'(x^2) = F'(x^2) g'(x^2) / g(x) and G = f(-x) G'(x^2), it is clear that f, g, F, G satisfy the NTRU equation mod (x^n + 1) (which is exactly the purpose of Algorithm 6). A more detailed discussion on this algorithm is given in https://ia.cr/2019/015 (in particular Algorithm 4 of https://ia.cr/2019/015 is almost exactly Algorithm 6 of Falcon's specification).

> 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!

We initially tried to do something along these lines, which would indeed be faster and simpler. However an issue with this approach is that G is not guaranteed to be an integer polynomial. Doing that modulo q would not work either: it would guarantee that f G - g F = 0 mod (x^n + 1, q), however we want to enforce the stronger equation f G - g F = q mod (x^n + 1).

> Best regards

From: EL HASSANE LAAJI <e.laaji@ump.ac.ma>
Sent: Monday, September 16, 2019 4:29 PM
To: Thomas Prest
Cc: pqc-forum; pqc2019; pqc-comments
Subject: Re: [pqc-forum] Officiel comment: FALCON

Hi Tomas,
You wrote."..However an issue with this approach is that G is not guaranteed to be an integer polynomial..."

I think G should be integer polynomial:
- we compute the inverse of f mod q ---> f^-1 coefficients are in Z.
- then G=(q-gF)(f^-1) in Z
Or we compute all polynomials in NTT form.
Thanks

Le lundi 16 septembre 2019, Thomas Prest <thomas.prest@pqshield.com> a écrit :

> Dear El Hassane,
>
> On 15/09/2019 16:23, EL HASSANE LAAJI wrote:
>
> > 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).
>
> Thank you for this remark. The line 11 of Algorithm 6 is actually exact, however there is a typo in line 12, which should be G <-- G'(x^2) f'(x^2) / f(x) instead of the current G <-- G'(x^2) g'(x^2) / g(x). We will correct this typo in the next iteration of the specification.
>
> A quick explanation of "why this works": by induction hypothesis, it holds that f'(x^2) G'(x^2) - g'(x^2) F'(x^2) = q mod(x^n + 1) (hence f', g', F', G' satisfy what we call the NTRU equation mod (x^{n/2} + 1)). On the other hand, f'(x^2) = f(x) f(-x) and g'(x^2) = g(x) g(-x). Therefore we can rewrite the NTRU equation as f(x) [f(-x) G'(x^2)] - g(x) [g(-x) F'(x^2)] = q mod(x^n + 1). Taking F = g(-x) F'(x^2) = F'(x^2) g'(x^2) / g(x) and G = f(-x) G'(x^2), it is clear that f, g, F, G satisfy the NTRU equation mod (x^n + 1) (which is exactly the purpose of Algorithm 6). A more detailed discussion on this algorithm is given in https://ia.cr/2019/015 (in particular Algorithm 4 of https://ia.cr/2019/015 is almost exactly Algorithm 6 of Falcon's specification).
>
> > 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!
>
> We initially tried to do something along these lines, which would indeed be faster and simpler. However an issue with this approach is that G is not guaranteed to be an integer polynomial. Doing that modulo q would not work either: it would guarantee that f G - g F = 0 mod (x^n + 1, q), however we want to enforce the stronger equation f G - g F = q mod (x^n + 1).
>
> > Best regards

From: Thomas Prest <thomas.prest@pqshield.com>
Sent: Tuesday, September 17, 2019 2:21 AM
To: EL HASSANE LAAJI
Cc: pqc-forum; pqc2019; pqc-comments
Subject: Re: [pqc-forum] Officiel comment: FALCON

Dear El Hassane,

You need two conditions:

1. The coefficients of f, g, F, G should have integer coefficients
2. The equation f G - g F = q should hold mod (x^n + 1) (so not mod q)

With the approach you describe, the condition 2 would hold mod (x^n + 1, q). But we want it to hold mod (x^n + 1), which is a stronger condition.

Regards,

Thomas

On 16/09/2019 22:28, EL HASSANE LAAJI wrote:

> Hi Tomas,
> You wrote."..However an issue with this approach is that G is not guaranteed to be an integer polynomial..."
>
> I think G should be integer polynomial:
> - we compute the inverse of f mod q ---> f^-1 coefficients are in Z.
> - then G=(q-gF)(f^-1) in Z
> Or we compute all polynomials in NTT form.
> Thanks
>
> Le lundi 16 septembre 2019, Thomas Prest <thomas.prest@pqshield.com> a écrit :
>
> > Dear El Hassane,
> >
> > On 15/09/2019 16:23, EL HASSANE LAAJI wrote:
> >
> > > 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).
> >
> > Thank you for this remark. The line 11 of Algorithm 6 is actually exact, however there is a typo in line 12, which should be G <-- G'(x^2) f'(x^2) / f(x) instead of the current G <-- G'(x^2) g'(x^2) / g(x). We will correct this typo in the next iteration of the specification.
> >
> > A quick explanation of "why this works": by induction hypothesis, it holds that f'(x^2) G'(x^2) - g'(x^2) F'(x^2) = q mod(x^n + 1) (hence f', g', F', G' satisfy what we call the NTRU equation mod (x^{n/2} + 1)). On the other

From: EL HASSANE LAAJI <e.laaji@ump.ac.ma>
Sent: Tuesday, September 17, 2019 4:08 AM
To: Thomas Prest
Cc: pqc-forum; pqc2019; pqc-comments
Subject: Re: [pqc-forum] Officiel comment: FALCON

Dear Tomas.
I want to say, the (mod q) is only for computing the inverse of f. Not for all equation 2.
Thanks.

Le mardi 17 septembre 2019, Thomas Prest <thomas.prest@pqshield.com> a écrit :

> Dear El Hassane,
>
> You need two conditions:
>
> 1. The coefficients of f, g, F, G should have integer coefficients
> 2. The equation f G - g F = q should hold mod (x^n + 1) (so not mod q)
>
> With the approach you describe, the condition 2 would hold mod (x^n + 1, q). But we want it to hold mod (x^n + 1), which is a stronger condition.
>
> Regards,
>
> Thomas
>
> On 16/09/2019 22:28, EL HASSANE LAAJI wrote:
>
> > Hi Tomas,
> > You wrote."..However an issue with this approach is that G is not guaranteed to be an integer polynomial..."
> >
> > I think G should be integer polynomial:
> > - we compute the inverse of f mod q ---> f^-1 coefficients are in Z.
> > - then G=(q-gF)(f^-1) in Z
> > Or we compute all polynomials in NTT form.
> > Thanks
> >
> > Le lundi 16 septembre 2019, Thomas Prest <thomas.prest@pqshield.com> a écrit :
> >
> > > Dear El Hassane,
> > >
> > > On 15/09/2019 16:23, EL HASSANE LAAJI wrote:
> > >
> > > > 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).

From: Thomas Prest <thomas.prest@pqshield.com>
Sent: Tuesday, September 17, 2019 7:31 PM
To: EL HASSANE LAAJI
Cc: pqc-forum; pqc2019; pqc-comments
Subject: Re: [pqc-forum] Officiel comment: FALCON

Dear El Hassane,

The fact that you compute the inverse of f mod q prevents this approach from satisfying condition 2. This can be easily checked in sage.

Regards,
Thomas

On 17/09/2019 10:07, EL HASSANE LAAJI wrote:

> Dear Tomas.
> I want to say, the (mod q) is only for computing the inverse of f. Not for all equation 2.
> Thanks.
>
> Le mardi 17 septembre 2019, Thomas Prest <thomas.prest@pqshield.com> a écrit :
>
> > Dear El Hassane,
> >
> > You need two conditions:
> >
> > 1. The coefficients of f, g, F, G should have integer coefficients
> > 2. The equation f G - g F = q should hold mod (x^n + 1) (so not mod q)
> >
> > With the approach you describe, the condition 2 would hold mod (x^n + 1, q). But we want it to hold mod (x^n + 1), which is a stronger condition.
> >
> > Regards,
> >
> > Thomas
> >
> > On 16/09/2019 22:28, EL HASSANE LAAJI wrote:
> >
> > > Hi Tomas,
> > > You wrote."..However an issue with this approach is that G is not guaranteed to be an integer polynomial..."
> > >
> > > I think G should be integer polynomial:
> > > - we compute the inverse of f mod q ---> f^-1 coefficients are in Z.
> > > - then G=(q-gF)(f^-1) in Z
> > > Or we compute all polynomials in NTT form.

From: 'Thomas Pornin' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Wednesday, September 18, 2019 10:18 AM
To: pqc-forum
Subject: [pqc-forum] OFFICIAL COMMENT: Falcon (bug & fixes)

Hello,

to my greatest shame, the new Falcon implementation published on 2019-08-02 had two severe bugs in the sampler; one table was wrong, and a scaling factor was applied at the wrong place. The consequences of these bugs are the following:

- Produced signatures were valid but leaked information on the private key.
- Performance was artificially inflated: the rejection sampling was not rejecting often enough, and the signatures were a bit shorter than they should. With the fixed code, signature performance on x86 (with AVX2) is at about 470k cycles (instead of 390k cycles previously) for Falcon-512, with an average signature size of about 657 bytes (instead of 651 bytes).

The implementation has been fixed, and is published on the Falcon Web site: https://falcon-sign.info/

The report on the implementation has been updated: https://falcon-sign.info/falcon-impl-20190918.pdf
This includes a new section on the issues and their correction.

The fact that these bugs existed in the first place shows that the traditional development methodology (i.e. "being super careful") has failed. I had previously predicted that lattice-based cryptography would entail many implementation issues because samplers are hard to test; at that time I did not envision to find myself at the short end of that prediction, but there we are. Falcon is a rather extreme case because it needs a precise, non-trivial distribution (Gaussian distribution, with varying non-integral centres and standard deviations); however, any random sampler is susceptible to be misimplemented in ways that can be hard to detect automatically. In the Falcon implementation, I have added unit tests that check the inner CDT and perform some chi-square tests that should detect the most egregious deviations, but this is far from providing complete assurance of correctness of the implementation. Other members of the Falcon team are currently working on a much more comprehensive framework for statistical tests on samplers.

New packages will soon be sent to interested parties in their respective formats (NIST, SUPERCOP, PQClean...). The fix to the sampler implies changes to the test vectors, and since performance is impacted, new benchmarks should be run.

Special thanks to Markku-Juhani O. Saarinen, who found the bugs and made some extensive statistical analyses.

Thomas

From: Elsa Velazquez <elve5895@colorado.edu>
Sent: Sunday, September 22, 2019 3:54 PM
To: pqc-forum
Subject: [pqc-forum] Re: OFFICIAL COMMENT: Falcon (bug & fixes)

Hello,
Do the fixes include testing to ensure the FPE is being used correctly, or are the two things checked completely independently of each other? If so, do the KEMs verify the FPE is done correctly? If so, is there a way to do so without KEMs?
And, is there any impact on key and stack sizes (aside performance) with these fixes?
Many thanks,
Elsa

On Wednesday, September 18, 2019 at 8:17:36 AM UTC-6, Thomas Pornin wrote:

> Hello,
>
> to my greatest shame, the new Falcon implementation published on 2019-08-02 had two severe bugs in the sampler; one table was wrong, and a scaling factor was applied at the wrong place. The consequences of these bugs are the following:
>
> - Produced signatures were valid but leaked information on the private key.
> - Performance was artificially inflated: the rejection sampling was not rejecting often enough, and the signatures were a bit shorter than they should. With the fixed code, signature performance on x86 (with AVX2) is at about 470k cycles (instead of 390k cycles previously) for Falcon-512, with an average signature size of about 657 bytes (instead of 651 bytes).
>
> The implementation has been fixed, and is published on the Falcon Web site: https://falcon-sign.info/
>
> The report on the implementation has been updated: https://falcon-sign.info/falcon-impl-20190918.pdf
> This includes a new section on the issues and their correction.
>
> The fact that these bugs existed in the first place shows that the traditional development methodology (i.e. "being super careful") has failed. I had previously predicted that lattice-based cryptography would entail many implementation issues because samplers are hard to test; at that time I did not envision to find myself at the short end of that prediction, but there we are. Falcon is a rather extreme case because it needs a precise, non-trivial distribution (Gaussian distribution, with varying non-integral centres and standard deviations); however, any random sampler is susceptible to be misimplemented in ways that can be hard to detect automatically. In the Falcon implementation, I have added unit tests that check the inner CDT and perform some chi-square tests that should detect the most egregious deviations, but this is far from providing complete assurance of correctness of the implementation. Other members of the Falcon team are currently working on a much more comprehensive framework for statistical tests on samplers.
>
> New packages will soon be sent to interested parties in their respective formats (NIST, SUPERCOP, PQClean...). The fix to the sampler implies changes to the test vectors, and since performance is impacted, new benchmarks should be run.
>
> Special thanks to Markku-Juhani O. Saarinen, who found the bugs and made some extensive statistical analyses.
>
> Thomas

From: Thomas Prest <thomas.prest@pqshield.com>
Sent: Friday, September 27, 2019 12:58 PM
To: Elsa Velazquez; pqc-forum
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: Falcon (bug & fixes)

Hello,

Le 22/09/2019 à 20:54, Elsa Velazquez a écrit :

> Hello,
> Do the fixes include testing to ensure the FPE is being used correctly, or are the two things checked completely independently of each other?

The bug and the fixes are independent of floating-point emulation (FPE). FPE is checked (see below), and we are working on providing a comprehensive test suite for checking Gaussian sampling as well.

> If so, do the KEMs verify the FPE is done correctly? If so, is there a way to do so without KEMs?

Do you mean KEMs or KATs? We don't use KATs to test FPE, instead we implement a test function (see **test-FP_block()** in https://falcon-sign.info/impl/test_falcon.c.html) to check the correct behavior of FPE.

> And, is there any impact on key and stack sizes (aside performance) with these fixes?

The fixes have no impact on key and stack sizes.

I hope this answered your questions.

> Many thanks,
> Elsa

Cheers,
Thomas

> On Wednesday, September 18, 2019 at 8:17:36 AM UTC-6, Thomas Pornin wrote:
> > Hello,
> >
> > to my greatest shame, the new Falcon implementation published on 2019-08-02 had two severe bugs in the sampler; one table was wrong, and a scaling factor was applied at the wrong place. The consequences of these bugs are the following:
> >
> > - Produced signatures were valid but leaked information on the private key.
> > - Performance was artificially inflated: the rejection sampling was not rejecting often enough, and the signatures were a bit shorter than they should. With the fixed code, signature performance on x86 (with AVX2) is at about 470k cycles (instead of 390k cycles previously) for Falcon-512, with an average signature size of about 657 bytes (instead of 651 bytes).
> >
> > The implementation has been fixed, and is published on the Falcon Web site: https://falcon-sign.info/
> >
> > The report on the implementation has been updated: https://falcon-sign.info/falcon-impl-20190918.pdf
> > This includes a new section on the issues and their correction.