The Falcon team, after discussing with NIST, envisions to modify the key pair generation implementation method by leveraging the techniques described in https://eprint.iacr.org/2023/290 and used in the HAWK scheme (HAWK is described on https://hawk-sign.info/ and was submitted to the additional round for post-quantum signatures). In a nutshell, the Falcon key pair generation process uses the following:

- The two polynomials f and g are generated with a fixed Gaussian distribution centred on zero. If the vector (f,g) has too large a norm, the process starts over.

- Polynomials F and G that fulfill the NTRU equation (fG - gF = q) are computed and reduced (with Babai’s round-off algorithm).

- If suitable F and G cannot be found then the process starts over. A solution can be easily verified to fulfill the equation, so there is no risk of accepting "wrong" (F,G).

In general, for given (f,g), there can be several reduced (F,G) solutions and neither is "more correct" than any other. The initial Falcon proposal uses floating-point operations in the reduction process, which raises some implementation issues with various hardware platforms, and slightly impairs test vector reproducibility since different platforms might apply distinct rounding and fall into a different (F,G) solution. The implementation used in HAWK uses only integer computations, making it much easier to reproduce exactly on many software platforms; it is also faster and uses less RAM. As a consequence, though, it can reject some (f,g) pairs for which suitable (F,G) solutions mathematically exist but are not found by the implementation, leading to regeneration of a new (f,g) pair. The rejection rate is less than 29% (about 8.2% for Falcon-512, 28.5% for Falcon-1024), and thus cannot have an impact on security worse than \log_2(1 - 0.29) bits, i.e. we may lose, theoretically, at worst 0.49 bits of security or so. No specific structure is known for key pairs that get rejected, but even if there were, it would not matter much for security.

The current implementation of the HAWK keygen uses the library code available at https://github.com/pornin/ntrugen, which also supports Falcon key pair generation directly. That library implements a second change, which is that the Gaussian distribution table used in the generation of (f,g) uses only 16 bits per entry, instead of 64 bits for the current Falcon code. This also slightly reduces the overall key pair generation time, by using fewer bits from the RNG. It should have no non-negligible impact on security.

The main effect of these changes is that for a given source seed, a different key pair is obtained; however, nothing in the rest of Falcon is modified, and the key pair generation methods are interoperable.

Note that we talk here about producing the (F,G) pair from (f,g); the generation of the Falcon tree is another process which can be considered to be a part of key pair generation, or to be instead a component of signature generation whose cost can be shared among distinct uses of the same key, since the tree depends only on the private key. The Falcon tree generation still uses floating-point operations.

While the proposed change is objectively good from a software engineering perspective (smaller, faster, more reproducible...) and seems innocuous for security, we would welcome any feedback from the community on this subject.

--Thomas Pornin, for the Falcon team
I welcome this change, and would like to see all the floating-point operations gone, making FALCON more similar to HAWK in that sense.

Thanks!
--
V/R,
Uri

There are two ways to design a system. One is to make it so simple there are obviously no deficiencies. The other is to make it so complex there are no obvious deficiencies.

- C. A. R. Hoare

The Falcon team, after discussing with NIST, envisions to modify the key pair generation implementation method by leveraging the techniques described in https://eprint.iacr.org/2023/290 and used in the HAWK scheme (HAWK is described on https://hawk-sign.info/ and was submitted to the additional round for post-quantum signatures). In a nutshell, the Falcon key pair generation process uses the following:

- The two polynomials f and g are generated with a fixed Gaussian distribution centred on zero. If the vector (f,g) has too large a norm, the process starts over.

- Polynomials F and G that fulfill the NTRU equation (fG - gF = q) are computed and reduced (with Babai's round-off algorithm).

- If suitable F and G cannot be found then the process starts over. A solution can be easily verified to fulfill the equation, so there is no risk of accepting "wrong" (F,G).

In general, for given (f,g), there can be several reduced (F,G) solutions and neither is "more correct" than any other. The initial Falcon proposal uses floating-point operations in the reduction process, which raises some implementation issues with various hardware platforms, and slightly impairs test vector reproducibility since different platforms might apply distinct rounding and fall into a different (F,G) solution. The implementation used in HAWK uses only integer computations, making it much easier to reproduce exactly on many software platforms; it is also faster and uses less RAM. As a consequence, though, it can reject some (f,g) pairs for which suitable (F,G) solutions mathematically exist but are not found by the implementation, leading to regeneration of a new (f,g) pair. The rejection rate is less than 29% (about 8.2% for Falcon-512, 28.5% for Falcon-1024), and thus cannot have an impact on security worse than log_2(1 - 0.29) bits, i.e. we may lose, theoretically, at worst 0.49 bits of security or so. No specific structure is known for key pairs that get rejected, but even if there were, it
Hi,

if you can eliminate all floating point operations from Falcon, I'd be very interested in the proposal.

If only some but not all of the floating points are removed, I struggle to see appeal though. You'd still be in a situation where no established leakage analysis and side-channel countermeasures exist for a key part of the algorithm and deployment options would be as limited as for the current Falcon spec.

Please correct me if I'm wrong but the changes do sound fairly extensive. I would very much encourage to produce a full spec and reference implementation for the proposed changes and then allow a generous amount of time (1 to 2 years?) for study and analysis. Ideally, I'd like to see a 3rd party security review, a HW implementation and research into side-channel security (at least one constant time implementation, preferably also a masked implementation) and fault resistance before a final FIPS document is released.

Even with Dilithium the hint unpacking issue was discovered by Mike Hamburg only after the end of the official comment period. Since we already* have Dilithium and Sphincs+ I believe there's no need to rush; better make sure we get this 100% right and provide developers with the knowledge to roll out high quality implementations.

Best,
Simon

*) The final FIPS documents are not published yet but close enough. For me that counts as already having the algorithms.
Hi all,

Let me start by saying that we do agree that removing FP operations is a good general direction for improvement on embedded platforms.

However, we agree with Simon here: we do not see a (significant) benefit from removing the (partial) FP operations of the key generation of Falcon.

Key generation is in many cases not even performed on an embedded device.

In many cases, keys are provisioned beforehand and therefore this change would not help with easing Falcon implementation.

Even in the cases where key generation would be performed on chip, the FP operations that remain are still a question mark for side-channel protection.

Without removing FP operations from signing (that does not impact performance as grossly as FP to Integer conversion does), we do not see a clear benefit for embedded context.

Cheers, on behalf of the NXP PQC team,

Christine

Op di 13 feb 2024 om 21:52 schreef Simon Hoerder <simon@hoerder.net>:

Hi,

if you can eliminate all floating point operations from Falcon, I'd be very interested in the proposal.

If only some but not all of the floating points are removed, I struggle to see appeal though. You'd still be in a situation where no established leakage analysis and side-channel countermeasures exist for a key part
One significant benefit of removing all the FP operation from FALCON would be simplification of its validation process – the fact that validation difficulties are explicitly called out in the NIST reports should tell something.

--

V/R,
Uri

Hi all,

Let me start by saying that we do agree that removing FP operations is a good general direction for improvement on embedded platforms.

However, we agree with Simon here: we do not see a (significant) benefit from removing the (partial) FP operations of the key generation of Falcon.

Key generation is in many cases not even performed on an embedded device.

In many cases, keys are provisioned beforehand and therefore this change would not help with easing Falcon implementation.

Even in the cases where key generation would be performed on chip, the FP operations that remain are still a question mark for side-channel protection.
Dear all,

I hope you’re doing well.

First, we would like to thank the community for the feedback on the potential change in the keygen of Falcon (FN-DSA). At this moment, NIST’s opinion on this change is not final, but we do lean a little bit towards accepting the modification of replacing Falcon Round 3 NTRUGen procedure by Hawk’s one as mentioned by Thomas’ Pornin in this thread. I will try to explain why in this email.

This change would bring some improvements:

1) Keygen without floating point (provided that the FFT part is done in the signature algorithm, which does not slow signature a lot according to Falcon’s team experiments)

2) Easier to get a constant time keygen

3) Keygen would use less RAM

4) Numerically stable keygen (given a seed, there is no ambiguity on the key a correct implementation will generate)

It is worth noting that these improvements come without impacting the security (as far as we are aware of), and they do not change the signature nor the verification process. In fact, only a small portion of the keygen as specified in the 3rd Round specs would be modified, namely the generation of F and G.

Item 4) above addresses validation’s concerns, as raised by Uri, and it is worth noting that besides NIST’s CAVP, testability is a priority for a lot of people/companies/institutions.

The use of floating point in FN-DSA and the challenges it raised could delay by a great deal its adoption, and so even if this tweak only solves the problem partially, we do agree on that, it is still an improvement that one gets almost “for free”.

Overall, NIST agrees with Falcon’s Team that this idea sounds reasonable, but we also want to ensure that the community supports the idea!

Thus, we encourage more people to answer on this thread to make sure that we’ve heard as much of the community’s opinion before we make a decision.

Feel free to reach out to us during the NIST 5th Standardization Conference as well.

Sincerely,

Dr. Maxime BROS
(on behalf of the NIST FIPS 206 - FN-DSA Team)
In relation to the new Falcon keygen implementation, I have also written a Python equivalent, which outputs the exact same key pairs as the C code, for the same input seed. This is a one-file implementation meant to support tests and debug, in particular printing out intermediate values. It is available there: 

Thomas

Le vendredi 5 avril 2024 à 20:49:31 UTC-4, Maxime Bros a écrit :

Dear all,

I hope you’re doing well.

First, we would like to thank the community for the feedback on the potential change in the keygen of Falcon (FN-DSA). At this moment, NIST’s opinion on this change is not final, but we do lean a little bit towards accepting the modification of replacing Falcon Round 3 NTRUgen procedure by Hawk’s one as mentioned by Thomas’ Pornin in this thread. I will try to explain why in this email.

This change would bring some improvements:

1) Keygen without floating point (provided that the FFT part is done in the signature algorithm, which does not slow signature a lot according to Falcon’s team experiments)

2) Easier to get a constant time keygen

3) Keygen would use less RAM

4) Numerically stable keygen (given a seed, there is no ambiguity on the key a correct implementation will generate)

It is worth noting that these improvements come without impacting the security (as far as we are aware of), and they do not change the signature nor the verification process. In fact, only a small portion of the keygen as specified in the 3rd Round specs would be modified, namely the generation of F and G.

Item 4) above addresses validation’s concerns, as raised by Uri, and it is worth noting that besides NIST’s CAVP, testability is a priority for a lot of people / companies / institutions.

The use of floating point in FN-DSA and the challenges it raised could delay by a great deal its adoption, and so even if this tweak only solves the problem partially, we do agree on that, it is still an improvement that one gets almost “for free”.

Overall, NIST agrees with Falcon’s Team that this idea sounds reasonable, but we also want to ensure that the
Dear Thomas,

Thank you very much for making it available. It'll be very useful.

Regards,

Anjan Roy

On Thu, 11 Apr 2024, 16:49 'Thomas Pornin via pqc-forum, <pqc-forum@list.nist.gov>' wrote:

In relation to the new Falcon keygen implementation, I have also written a Python equivalent, which outputs the exact same key pairs as the C code, for the same input seed. This is a one-file implementation meant to support tests and debug, in particular printing out intermediate values. It is available there:


Thomas

Le vendredi 5 avril 2024 à 20:49:31 UTC-4, Maxime Bros a écrit :

Dear all,

I hope you’re doing well.

First, we would like to thank the community for the feedback on the potential change in the keygen of Falcon (FN-DSA).

At this moment, NIST’s opinion on this change is not final, but we do lean a little bit towards accepting the modification of replacing Falcon Round 3 NTRUGen procedure by Hawk’s one as mentioned by Thomas’ Pornin in this thread. I will try to explain why in this email.

This change would bring some improvements:

1) Keygen without floating point (provided that the FFT part is done in the signature algorithm, which does not slow signature a lot according to Falcon’s team experiments)

2) Easier to get a constant time keygen

3) Keygen would use less RAM

4) Numerically stable keygen (given a seed, there is no ambiguity on the key a correct implementation will generate)

It is worth noting that these improvements come without impacting the security (as far as we are aware of), and they do not change the signature nor the verification process. In fact, only a small portion of the keygen as specified in the 3rd Round specs would be modified, namely the generation of F and G.
Dear all,

I hope you’re doing well.

We would like to thank the community for the feedback on the potential change in the keygen of Falcon (NIST FIPS 206 – FN-DSA) that we received in the email thread about this, and during our 5th Standardization Conference in Rockville, Maryland.

From this feedback and our internal discussions, we have decided to accept the change which consists of replacing a part of Falcon keygen with the equivalent part of Hawk’s keygen to avoid using floating-point arithmetic. Namely, we will change only the Reduce() algorithm in NTRUSolve() which will allow implementations to use fixed-point instead of floating-point as described in [1]. Please note that using fixed-point in the signing process would require too much memory (roughly 256 bits per value), thus it will not be considered.

In addition to this change, after extensive internal discussions and communications with the Falcon team, we decided to allow the use of emulated floating-point as specified in the Falcon reference implementation and in [2]. More precisely, it means that FIPS 206 will allow one to use native or emulated floating-point for the signing algorithm. Please note that the fixed-point (as previously mentioned) for the keygen will be a possibility, not a requirement. This means that one could generate the key AND sign using only emulated or native floating-point.

This choice was made to give more freedom to the implementers with respect to the side-channel resistance since emulated floating-point does have disadvantages, but unlike native floating-point it is more likely to be constant time as noted in [2]. However, we encourage the community to keep researching ways to implement side-channel-resistant floating-point.

Last but not least, in FIPS 206, the generation of the Falcon’s tree will be part of the signing process by default, but we will also allow it to be precomputed / stored with the private key as an option.

While this is our current plan for FIPS 206, we encourage feedback from the community.

Sincerely,

Dr. Maxime BROS
(on behalf of the NIST FIPS 206 - FN-DSA Team)


You note that you could use either floating point or fixed point in the key generation.

Does that mean that they both define the same transform “random coins -> public/private key pair”. Or, there are essentially two different key generation algorithms (and both are acceptable)?

Dear all,

I hope you’re doing well.

We would like to thank the community for the feedback on the potential change in the keygen of Falcon (NIST FIPS 206 – FN-DSA) that we received in the email thread about this, and during our 5th Standardization Conference in Rockville, Maryland.

From this feedback and our internal discussions, we have decided to accept the change which consists of replacing a part of Falcon keygen with the equivalent part of Hawk’s keygen to avoid using floating-point arithmetic. Namely, we will change only the Reduce() algorithm in NTRUSolve() which will allow implementations to use fixed-point instead of floating-point as described in [1]. Please note that using fixed-point in the signing process would require too much memory (roughly 256 bits per value), thus it will not be considered.

In addition to this change, after extensive internal discussions and communications with the Falcon team, we decided to allow the use of emulated floating-point as specified in the Falcon reference implementation and in [2]. More precisely, it means that FIPS 206 will allow one to use native or emulated floating-point for the signing algorithm. Please note that the fixed-point (as previously mentioned) for the keygen will be a possibility, not a requirement. This means that one could generate the key AND sign using only emulated or native floating-point.

This choice was made to give more freedom to the implementers with respect to the side-channel resistance since emulated floating-point does have disadvantages, but unlike native floating-point it is more likely to be constant time as noted in [2]. However, we encourage the community to keep researching ways to implement side-channel-resistant floating-point.

Last but not least, in FIPS 206, the generation of the Falcon’s tree will be part of the signing process by default, but we will also allow it to be precomputed / stored with the private key as an option.

While this is our current plan for FIPS 206, we encourage feedback from the community.

Sincerely,

Dr. Maxime BROS
(on behalf of the NIST FIPS 206 - FN-DSA Team)


You received this message because you are subscribed to the Google Groups "pqc-forum" group. To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.

To view this discussion on the web visit https://groups.google.com/a/list.nist.gov/d/msgid/pqc-forum/SJ0PR09MB96039CA15DE8C0A38E891C6999DDB2%40SJ0PR09MB9603.namprd09.prod.outlook.com.
Dear NIST,

Recently, it is shown that the emulated floating-point arithmetic does not behave as claimed in [1], see https://eprint.iacr.org/2024/321. The good news is that it doesn’t affect the complex FFT in the signature generation of Falcon with suitable range analysis. But this work only analyzes the FFT and not the full signature generation.

Should we regard the C implementation with double as the spec or the one with the emulated floating-point arithmetic as the spec?

Perhaps there are no distinctions in the context of Falcon, but currently, there is no rigorous justification between the double implementation and the emulated floating-point implementation.

Thanks,
Vincent Hwang

On Tue, Jul 9, 2024 at 4:53 PM ‘Bros, Maxime P. (IntlAssoc)’ via pqc-forum <pqc-forum@list.nist.gov> wrote:

Dear all,

I hope you’re doing well.

We would like to thank the community for the feedback on the potential change in the keygen of Falcon (NIST FIPS 206 – FN-DSA) that we received in the email thread about this, and during our 5th Standardization Conference in Rockville, Maryland.

From this feedback and our internal discussions, we have decided to accept the change which consists of replacing a part of Falcon keygen with the equivalent part of Hawk’s keygen to avoid using floating-point arithmetic. Namely, we will change only the Reduce() algorithm in NTRUSolve() which will allow implementations to use fixed-point instead of floating-point as described in [1]. Please note that using fixed-point in the signing process would require too much memory (roughly 256 bits per value), thus it will not be considered.

In addition to this change, after extensive internal discussions and communications with the Falcon team, we decided to allow the use of emulated floating-point as specified in the Falcon reference implementation and in [2]. More precisely, it means that FIPS 206 will allow one to use native or emulated floating-point for the signing algorithm. Please note that the fixed-point (as previously mentioned) for the keygen will be a possibility, not a requirement. This means that one could generate the key AND sign using only emulated or native floating-point.

This choice was made to give more freedom to the implementers with respect to the side-channel resistance since emulated floating-point does have disadvantages, but unlike native floating-point it is more likely to be constant time as noted in [2].

However, we encourage the community to keep researching ways to implement side-channel-resistant floating-point.

Last but not least, in FIPS 206, the generation of the Falcon’s tree will be part of the signing process by default, but we will also allow it to be precomputed / stored with the private key as an option.
It’s the second case: the two keygen algorithms are different, in that from the same seed (more specifically, from the same \((f,g)\) polynomials) you may get different key pairs (there are multiple solutions \((F,G)\) for a given \((f,g)\) pair, and they are not trivially derived from each other), but none of them is better or worse than the others, as far as security is concerned.

In fact there are many keygen algorithms here. The formal description is a single reduction operation which formally uses infinite precision, but practically speaking there is no infinite that fits in a computer, and in fact you’ll more or less need to perform multiple passes of reduction, with a limited-precision implementation, to gradually reduce the \((F,G)\) candidate with regard to \((f,g)\). In the code as was submitted with Falcon, IEEE754 binary64 values are used (i.e. the usual "double" type) and multiple passes are applied, each being assumed to skim off about 25 bits from the polynomial coefficients; this already diverges from the strict formal description of Reduce(), since it does not use infinite precision and there can always be some threshold effects which make the reduction end up with a different (but still valid) solution. In the "new code" (ntrugen library, used in Hawk, also applicable to Falcon and BAT), a fixed-point 64-bit representation is used instead, which makes things a lot faster on low-end hardware without a FPU, but also leads to again different (but valid) solutions.

Thomas

Le mardi 9 juillet 2024 à 11:09:02 UTC-4, Scott Fluhrer (sfluhrer) a écrit :

You note that you could use either floating point or fixed point in the key generation.

Does that mean that they both define the same transform “random coins -> public/private key pair”. Or, there are essentially two different key generation algorithms (and both are acceptable)?

Dear all,

I hope you’re doing well.

We would like to thank the community for the feedback on the potential change in the keygen of Falcon (NIST FIPS 206 – FN-DSA) that we received in the email thread about this, and during our 5th Standardization Conference in Rockville, Maryland.

From this feedback and our internal discussions, we have decided to accept the change which consists of replacing a part of Falcon keygen with the equivalent part of Hawk’s keygen to avoid using floating-point arithmetic. Namely, we will change only the Reduce() algorithm in NTRUSolve() which will allow implementations to use fixed-point instead of floating-point as described in [1]. Please note that using fixed-point in the signing process would require too much memory (roughly 256 bits per value), thus it will not be considered.
As far as I know, the implementation is not the spec. The spec is the spec. It seems to me that NIST is not usually providing reference implementations, only specifications (as PDF documents) and test vectors. The existing implementation of Falcon is part of the submission, not of the specification. This is also the dynamic we can observe with other algorithms, e.g. Kyber and Dilithium, where (as far as I know) the last submitted algorithm and the current (draft) spec differ on a couple of details.

When NIST publishes a draft specification I'll adjust the existing C code to match it and make a not-really-reference-but-still-close-enough implementation, since in practice it can be expected that people wanting to use the algorithm will first go for some existing code, so there should be a reasonably reusable implementation somewhere. Ideally I'll also make a Rust or Go implementation, too.

Thomas

Le mardi 9 juillet 2024 à 11:10:06 UTC-4, Vincent Hwang a écrit :

Dear NIST,

Recently, it is shown that the emulated floating-point arithmetic does not behave as claimed in [1], see https://eprint.iacr.org/2024/321. The good news is that it doesn't affect the complex FFT in the signature generation of Falcon with suitable range analysis. But this work only analyzes the FFT and not the full signature generation.

Should we regard the C implementation with double as the spec or the one with the emulated floating-point arithmetic as the spec?

Perhaps there are no distinctions in the context of Falcon, but currently, there is no rigorous justification between the double implementation and the emulated floating-point implementation.

Thanks,

Vincent Hwang

On Tue, Jul 9, 2024 at 4:53 PM 'Bros, Maxime P. (IntlAssoc)' via pqc-forum <pqc-...@list.nist.gov> wrote:

Dear all,

I hope you're doing well.

We would like to thank the community for the feedback on the potential change in the keygen of Falcon (NIST FIPS 206 – FN-DSA) that we received in the email thread about this, and during our 5th Standardization Conference in Rockville, Maryland.

From this feedback and our internal discussions, we have decided to accept the change which consists of replacing a part of Falcon keygen with the equivalent part of Hawk’s keygen to avoid using floating-point arithmetic. Namely, we will change only the Reduce() algorithm in NTRUSolve() which will allow implementations to use fixed-point instead of floating-point as described in [1]. Please note that using fixed-point in the signing process would require too much memory (roughly 256 bits per value), thus it will not be considered.

In addition to this change, after extensive internal discussions and communications with the Falcon team, we decided to allow the use of emulated floating-point as specified in the Falcon reference implementation and in [2]. More precisely, it means that FIPS 206 will allow one to use native or emulated floating-point for the signing algorithm. Please note that the fixed-point (as previously mentioned) for the keygen will be a possibility, not a requirement. This means that one could generate the key AND sign using only emulated or native floating-point.
Hello all!

We implemented the calculation of the digital signature using a fixed point representation (total data size of 64 bits, consisting of: 1 bit for the sign, 37 bits for the integer part, and 26 bits for the fractional part).

We applied this implementation to the first 100 keys of the author's implementation for \( n=1024 \). All signatures were successfully verified for both calculation options (dynamic and pre-tree calculation).

The size of the digital signature after packing, excluding the nonce, message for the signature, and additional two bytes for the length, varies as follows:

- For double data: 1154 to 1165 bytes
- For fixed point data: 1229 to 1243 bytes

The increase in the length of the digital signature does not exceed 7%. Since 64 bits were used for the polynomial coefficient assignment, the memory size for the polynomial assignment does not depend on the data type (double or fixed point).

To create the digital signature, the author's algorithm was used, which excludes arithmetic for data with a fixed point, thus not requiring additional memory when using a fixed point representation.

Thanks

Olena Kachko
Yuri Gorbenko

https://iit.com.ua/