From: Danny Niu <dannyniu@hotmail.com>
Sent: Sunday, October 4, 2020 1:12 AM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 3 OFFICIAL COMMENT: CRYSTALS-DILITHIUM

Hello Crystals team.

I was wondering if it's possible to use smaller parameter sets
to decrease the signature size, instead of relying on aggressive
compression techniques used in the current form of Dilithium.

Specifically, if we were to preserve the Z_q[X]/(X^256+1) module
and matrix row and column counts in the current specification, and
decrease 'q' and acceptance threshold of signature variable 'z', and
avoid public-key compression, would we be able to achieve comparable
bandwidth efficiency and security? Had the team done some calculation
on this?

Thanks.

From: Vadim Lyubashevsky <vadim.lyubash@gmail.com>
Sent: Monday, February 8, 2021 4:35 PM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 3 OFFICIAL COMMENT: CRYSTALS-DILITHIUM

Dear all,

Dustin and the NIST team pointed out to us on Friday that one of our internal hash functions, which converts the message to a digest (line 10 in Figure 4), is hashing to a 384-bit value. This was large enough for NIST level 3 security (which was our maximum security level in round 2), but would need to be 512 for NIST level 5 security. We forgot to make this change and we thank the NIST team for pointing it out. We have since updated our code and spec at

https://github.com/pq-crystals/dilithium
and
https://pq-crystals.org/dilithium/resources.shtml

Since this is an internal variable, it has no impact on the parameter sizes. The runtime differences are also negligible. Since this change does affect the KATs, we took it as an opportunity to make a few additional small changes that we were holding off on. To harmonize the randomness expansion function in the key generation and signing, we are now using SHAKE-256 with 512-bit secret seeds for both. We also reduced the output size of the public key hash (tr on line 7) to 256 bits. And we simplified the pseudo-code throughout the spec by merging the names of H and CRH to just H (since they both use SHAKE-256 with varying output sizes).

Best,

Vadim
(On behalf of the Dilithium team).

From: pqc-forum@list.nist.gov on behalf of Christ
Sent: Tuesday, October 12, 2021 2:51 AM
To: Markku-Juhani O. Saarinen
Cc: pqc-forum; cpei...@alum.mit.edu
Subject: Re: [pqc-forum] Re: ROUND 3 OFFICIAL COMMEN

Hi Markku,

Thank you for your feedback - this discussion is a dimens
schemes and managing keys that those schemes require.

> There are proposals such as [2] that diverge from the pub
team, and which have been evaluated in the NIST process.

We would like to point out that this work is focused entir
themselves. It has already found several issues with the s
preparation for a submission to IETF where a broader commu
feedback from authors of most schemes.

There was no guidance during the NIST process on how keys
serialized blob. Over the rounds serialization formats hav
Interoperability between early implementations has not bee
environments will not be able to manage even the smaller o
stored/transported in a compressed format. For these reaso
way to manage keys and we need a discussion on how to do t
discussion early and at the right venue will prevent many
divergence between standards.

Your feedback is useful in the context of security conside

Regards, on behalf of the team,

Christine

Op za 9 okt. 2021 om 22:57 schreef Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>:

> Hello!
>
> I think Chris's comment is a good illustration of the reasons why system integrators should be very careful when they interpret or modify PQC algorithms. There are usually excellent -- but sometimes non-obvious -- reasons for why the proposals do certain things the way they do them. Detail is important.
>
> For example, the security properties that Chris describes do not seem to hold with the simplified interpretation of Dilithium described in a recent ETSI document [1]. This is because (as far as I can see) the variant of Dilithium in [1] removes the public-key prefix binding from the hash function.
>
> It is tempting to use Dilithium, Falcon, and SP 800-208 HBS (LMS/XMSS) without their respective hash prefixes in PQC/PKI integration -- as essentially traditional hash-and-sign algorithms. This is because the internal APIs and certificate handling logic of PKI implementations is often built around that paradigm. In addition to breaking the "strong binding" property, the simplification introduces additional security risks stemming from hash function collision attacks.