Public Comments on draft FIPS 204

Comment period: August 24, 2023 – November 22, 2023

On August 24, 2023 NIST requested comments on the initial draft FIPS 204, *Module-Lattice-Based Digital Signature Standard*. The comments that NIST received during the comment period are collected below.

LIST OF COMMENTS

1. Comments from Joe Hobo, August 24, 2023

The Kyber and Dilithium specs feel very different, despite having much in common. The NIST standards should try to fix this and provide a uniform presentation on their overlap. [Why not use the same notation (e.g. for bit-reversal)? Why is FIPS 204 in a different font? Why not consistently use \mathsf?]

- line 503 ff.: Why write the needlessly complicated $w(\pm\zeta_i)$
, $\zeta_i=\zeta^{\mathsf{brv(128+i)}}$, $i\in[0,128)$ instead of the simpler
$w(\zeta^{2\mathsf{brv}(i)+1})$, $i\in[0,256)$? The NTT is the "mathiest" part of the specs and should be explained as uniformly as possible across both Kyber and Dilithium. If it isn't obvious that these are the same, one can check

```python
def brv(i):
    j = 0
    for b in range(8):
        x = i%2
        j += x*(2**(7-b))
        i = i//2
    return j

q = 8380417
z = 1753

for i in range(256):
    e = i % 2
    print((z**(2*brv(i)+1)) % q, (((-1)**e) * (z**brv(128 + i//2))) % q)
```

- line 581: The suffix numbers refer to module dimensions in Dilithium, but refer to four times the security strength in Kyber (or something similar, not specified). Couldn't these be made uniform?

- Algorithm 4, first line: Correct "(using in little-endian order)."

- Algorithms 6/7: Why not use uniform notation between Kyber/Dilithium?

- line 822 ff., Hashing and Pseudorandom Sampling: Wouldn't it be easier to modify FIPS 202 to include a byte stream or generator/stateful SHAKE instead of only the fixed-length XOF? Again, there is a disconnect between Kyber and Dilithium specs, although both have the same goal of using SHAKE for rejection sampling.

- Algorithm 34: Put spaces after "Input:" and "Output:" and use consistent capitalization of Boolean.

- Line 907 display: Same comment as above; just write $\zeta^{2\mathsf{brv}(i)+1}$ instead of the more complicated expression(s).

- line 1089 ff., Appendix B: Why an appendix on Montgomery reduction and not, e.g., Barrett reduction (simpler in my opinion)?

2. Comments from Simon Hoerder, August 24, 2023

Thanks for the draft standards, it is very good to have them. However, I notice that none of them contain a test vector. I believe it would be very useful to have at least a test vector included, maybe even at a suitably high level a worked example. This would help immensely with interoperability concerns.

Please note that I put the FIPS-20x-comments@nist.gov addresses in BCC to avoid list replies accidentally getting sent there.

Best regards,
Simon

3. Comments from Erkan Uslu, August 25, 2023

Dear NIST PQC team,

According to part 1.3.2 the size of c^{tilde} which is a part of signature is increased for ML-DSA-65 and ML-DSA-87. However in table 2, it can be seen that the signature sizes of the ML-DSA-65 and ML-DSA-87 is still same as Dilithium-3 and Dilithium-5 respectively.

I think the actual signature sizes of these parameter sets should be
3309 (3293 + 16) bytes for ML-DSA-65 and 4627 (4595 + 32) bytes for
ML-DSA-87.

If I am wrong can you please correct me?

By the way thanks for your afford on these standardization process and
great works.

Best,
Erkan USLU
Middle East Technical University

4. Comments from Danny Niu, August 26, 2023

In draft for ML-DSA (aka Dilithium) on line 529, it says:

> When the module Znq in LWE and SIS is replaced by a module over a ring larger than Zq (such as Rq),

This seems to be the other way around. Shouldn't it be:

> When the ring Zq (such as Rq) in LWE and SIS is replaced by a module Znq over a larger set,

I can't do a good articulation, and if it's indeed an editorial error, I hope someone can come up with better wording.

Regards, DannyNiu/NJF.

Maybe "Rq" in the parentheses in my proposed (mis-)wording is wrong, but the general idea I've been suggesting is: it should be "ring" replaced by *module*.

5. Comments from Anubhab Baksi, August 28, 2023

Dear NIST,

Regarding the recent FIPS reports (FIPS 203, FIPS 204 and FIPS 205), we have noticed that the estimates on the Grover's search complexity on AES-128/-192/-256 are taken from the Eurocrypt'20 paper by Jaques et al. However, their implementations contained some bugs (due to inherent issues with Q#), and consequently the estimates were erroneous. We have contacted the authors of this paper, and they agreed about the presence of the bug.

As we have shown in Table 11 of our paper ("Quantum Analysis of AES" by Jang et al.), the complexities with their bug-fixed code for AES-128/-192/-256 are respectively 2^{157.33}, 2^{222.76}, 2^{287.28} (we considered multiple bug-fixing and optimizations on top of their base codes and those were the best results).

Apart from that, our own implementations in the same paper achieve the complexity estimates of 2^{156.64}, 2^{221.99} and 2^{286.48} respectively; which are the currently best-known results to the best of our knowledge.

Therefore, we would be grateful if you kindly consider having a look at our work. We would be obliged to discuss with you, should there be any necessity.

Thanking you,
Anubhab Baksi (for, and on behalf of other authors),
Singapore
On behalf of: Kyungbae Jang, Hyunji Kim, Gyeongju Song, Hwajeong Seo, Anupam Chattopadhyay

6. Comments from Elaine Barker, August 29, 2023

1. Lines 67-68: FIPS 186-5 should also be listed here.
2. Line 95: Are the allowed algorithms in FIPS 140-3 necessarily in an approved status for govt. use?
3. Line 255: Refer to Section 3.2 for an explanation of the MLWE problem.
4. Lines 262-263: Replace with "...signature generation (Section 6), and signature verification (Section 7)..."
5. Sections 1.2 and 1.3: Consider not including these sections in the final FIPS, but providing the content in 1.2, 1.3, and Appendix A in a separate document (e.g., a NISTIR).
6. Line 280, fault-tolerant: provide an explanation of what this is .
7. Line 297: Remove the comma.
8. Line 298, "increases the length of c˜ to 384 and 512 bits, respectively": Some text seems to be missing. What are the two values for?
9. Line 370: Change to "For digital signatures, the private key...".
10. Lines 390 and 402: Remove the commas.
11. Line 436: Should the A and T be bolded?
12. Lines 440-441: Provide examples (as is done for other NIST documents for these notations.
13. Lines 449-450: Shouldn't the definition say something about the variables reversing but the coefficients don't?
14. Lines 469-472: Don't like this notation. y<-- f(x) looks OK to me.
15. Lines 517-518, "the adversary cannot create any additional valid signatures based on the signer’s public key": This is a bit weird, since the public key is not used for signing. Maybe say something about not being able to determine the private signing key from the public key? Clarify what is meant.
16. Line 525: Define "noisy linear equation".
17. Lines 538-541: This looks like proof of possession and could be used in SP 800-89.
18. Line 581: Delete "we use" and insert "are used" after "ML-DSA-87".
19. Line 598: Change "value" to "seed".
20. Line 611: Replace "can" with "could".
21. Line 622: Should "public key" be changed to "private key" here?
22. Lines 648-649: Should the categories be listed in Table 1?
23. Line 660, "XOF (namely, SHAKS-256": Say that this appears as H(tsi, 1024) in step 1 or refer to Section 8.3.
24. Line 668: Change "thought of" to "considered"?

25. Lines 676 and 693, "a 512-bit hash...": Suggest reordering the phrase to "*tr*, a 512-bit hash of the public key for use...". As written, it sounds like tr is hashed rather than the public key.
26. Line 680-681: Include "M" in Section 2.3, indicating that M is not necessarily the data to be signed but may a message digest.
27. Line 683: Remove "In addition,".
28. Lines 725-726: Will the CAVP need to test this alternative implementation? Or do they end up with the same result?
29. Lines 728-729: Why bother with this notation (i.e., the "<<" and ">>"). These values are just computed and used later in the same Algorithm? Using just "c$\mathbf{s}_1$" works.
30. For the various algorithms: Consider pointing to the pseudocode algorithms routines by number or a link to make them easy to get to.
31. Line 735: Should "section 1" be changed to "Table 1"?
32. Section 7.1: Pre-hashing seems to be optional. Will there be OIDs to distinguish the use or non-use of pre-hashing? Would pre-hashing be the default? What if a verifier can only verify a pre-hashed message but the signer did not pre-hash? Either provide guidance or settle on always pre-hashing.
33. Line 767, re using a XOF: This seems to say that using a XOF is acceptable for pre-hashing, but lines 836-840 say it is not. We have not yet approved using a XOF as a general-purpose hash function. Need to resolve this conflict.
34. (8.1) and (8.2) and line 830: Define this notation in Section 2.3.
35. Line 830: Define "little endian" in Section 2.

7. Comments from Lauren Brandt, September 7, 2023

Hello,

I have noticed a discrepancy between the encoded private key size in Algorithm 18 (skEncode) and the sizes (in bytes) of the private key listed in Table 2.

In Algorithm 18, the size of the encodedSk is 32+32+64+32*((K+L)*bitlen(2*ETA)+D*K) bytes.

For ML-DSA-44:
K = 4, L = 4, ETA = 2, D = 13.
32+32+64+32*((4+4)*bitlen(2*2)+13*4)
=2560

For ML-DSA-65:
K = 6, L = 5, ETA = 4, D = 13
32+32+64+32*((6+5)*bitlen(2*4)+13*6)
= 4032

For ML-DSA-87
K = 8, L=7, ETA = 2, D = 13
32+32+64+32*((8+7)*bitlen(2*2)+13*8)
= 4896

The values in Table 2 for the Private Key Bytes are:
ML-DSA-44: 2528
ML-DSA-65: 4000
ML-DSA-87 : 4864

The values from Table 2 are all 32 bytes less than output size of Algorithm 18. In Dilithium 3.1, tr is 32 bytes. In FIPS 204, this size or tr was increased by 32 bytes to 64 bytes. I think this could explain the differences between Algorithm 18 and Table 2.

Best,
Lauren Brandt

8. Comments from Vadim Lyubashevsky, September 8, 2023

Dear NIST, dear all,

At this week's Post-Quantum Cryptography Summit in Oxford (thank you to the organizers!), a group of us went through the exercise of implementing the Dilithium draft standard from https://csrc.nist.gov/pubs/fips/204/ipd in Python and then comparing it to our C reference implementation from https://github.com/pq-crystals/dilithium/tree/standard. This reference implementation is a modification of the pre-standard Dilithium which includes the proposed changes from the draft standard document. The result was that modulo one small difference (and some inconsistencies in the standard algorithms which could be easily fixed), the algorithms appear to match. Since we implemented the algorithms from https://csrc.nist.gov/pubs/fips/204/ipd as verbatim as possible, they are extremely slow, and so we could not perform many tests. So we are just mildly confident that everything matches.

Below, we explain the minor difference and give other suggestions for the standard.

Best,
Vadim

- The ExpandMask function (Algorithm 28) already uses a different input for SHAKE for every polynomial. It should thus use the output from the beginning of the hash stream and not start at an offset (i.e. the r is not needed in the expansion in line 4). This currently differs from our reference C implementation and leads to different test vectors. Removing the r (or setting it to 1) makes things more efficient. This is the version that we tested against our reference code.

- As in the ML-KEM and SLH-DSA standards, we strongly recommend to use byte arrays as inputs and outputs, and not bit strings. This should not result in any changes in the test vectors. The way things are now, very often there are many consecucive BytesTobits and BitsTobytes calls -- especially when submitting inputs to hash functions, which input/output bytes -- that cancel each other out.

- Please consider including the whole table of powers of zeta for the NTT. Then inside the NTT description simply use \zeta_j or \zeta[j]. Of course add an explanation of how the entries are computed.

- Move discussions about how to implement certain math operations (like rounding) to a dedicated "Notes for Implementors" section

- The majority of us thought that the entire variable \tilde{c} should be used in the SampleInBall function rather than just \tilde{c_1}.

This would remove the need for variables \tilde{c_1} and \tilde(c_2), where \tilde(c_2) isn't actually ever used. The performance impact of the change will essentially be zero, but possibly make things less confusing.

Typos and inconsistencies:

Algorithm 1: Power2Round should not take d as an input

Algorithm 5: Should take bit length as input because other functions call it with it (Algorithms 12 and 13).

Algorithm 7: d is used as the length of z, but d is a public value

Algorithms 21 and 22: For consistency with previous algorithm descriptions, the k and l in lines 1 and 3 should be in the "double exponent"

Algorithm 25: eta should not be a parameter in calling the function in lines 5 and 6

Algorithms 23, 35, 36: k is used as a local loop variable, but it was previously defined as global

Algorithms 35 and 36: the brv function should be written out as a separate algorithm

Line 436: Symbol for transposing matrices is inconsistent with the one in Kyber standard. Suggest remove it from Dilithium standard since it is not used in this document

No floating point arithmetic: Comment similar to the one in FIPS 203 (line 703) is applicable to FIPS 204.
Implementations of ML-DSA doesn't require and should not use floating-point arithmetic.

9. Comments from Lauren Brandt, September 18, 2023

Hello,

I think there is a typo in the output of Algorithm 22 of the ML-DSA final draft.

Algorithm 22 states that the output size is a bit string representation, w_1 \in {0,1}^(32*k*bitlen((q-1)/(2*GAMMA2)-1).

However, this function is dependent on BytesToBits and SimpleBitPack. For each call to SimpleBitPack, the return will be a byte string of length 32* bitlen((q-1)/(2*GAMMA2)-1). BytesToBits will then produce a bit string from this byte string. As a byte is 8 bits, each iteration of the for loop will produce 32* bitlen((q-1)/(2*GAMMA2)-1)*8 bits. As this for loop is run k times, there should be a total of 32*k* bitlen((q-1)/(2*GAMMA2)-1)***8** bits produced. So I think the output should instead say w_1 \in {0,1}^(32*k*bitlen((q-1)/(2*GAMMA2)-1)*8.

Best,
Lauren

10.Comments from Stephan Mueller, September 18, 2023

Dear ladies and gentlemen,

Considering the request for comments on the FIPS 204 draft from August 24, 2023, please find below comments.

FIPS 204 Comments

Error in ctilde Length Specification

At various places, including the final length of the signature, the c~ length is defined to be 32 bytes instead of lambda / 4.

Signature Generation: Reverse Hashing of TR and M

Algorithm 2 step 6 specifies:

mu <- H(tr||M, 512)

which implies the hashing of the input message and the public key hash TR.

It is suggested to reverse the order of the input data to the following proposal:

mu <- H(M||tr, 512)

Cryptographic impact: The chosen hash algorithm and the resulting strength of the message digest is agnostic of the order of the input data provided that all input data is used for the hashing. Of course the resulting message digest changes, but its cryptographic strength, i.e. the property relied on by the signature, is unaffected. Thus, the change is considered to not affect the cryptographic properties of the signature.

Implementation impact: With the change, an implementation of signature operation may become significantly simpler and fully consistent with currently existing signature schema implementations. The reason is the following: Some signature algorithm implementations provide 3 interfaces to perform the signature generation operation:

1. signature init - this operation effectively initialize the message hashing (akin to a hash init operation)

2. signature update - this operation effectively adds a message to the hash state (akin to a hash update operation); this operation is allowed to be called multiple times until all message parts are added to the hash state

3. signature final - this operation calculates the message digest and from it the signature (akin to a hash final operation immediately followed by the signature generation operation)

Examples for such approaches are found in contemporary algorithm implementations like OpenSSL (EVP_DigestSignInit, EVP_DigestSignUpdate, EVP_DigestSignFinal) or leancrypto (lc_dilithium_sign_init, lc_dilithium_sign_update, lc_dilithium_sign_final)

The purpose of this segmentation is that it allows the caller to apply the signature operation on a message that is held multiple memory locations (i.e. the message is NOT held in a contiguous memory block). Without the allowance of step 2 from above, a caller must copy a potentially large message into a contiguous memory block and then perform the signature operation which may cause the message to be present in memory twice. As messages may be very large (e.g. ISO disk images which may multiple of gigabytes in size), such data duplication leads to waste of resources.

With the FIPS 204 stipulated calculation of mu, it is required that the TR value must be added to the hash state before the actual message. This implies that the TR value must be known to the signature init step (or at least before the first message part is added to the hash state). The TR value can only be obtained from the secret key which implies that the secret key must be supplied to the signature init step in addition to the signature final step.

By reversing the hash input to calculate mu, the TR value is the last data that is added to the hash state. As the message digest is calculated in the signature final operation, the addition of the TR to the hash state can be equally performed in the signature final state. This implies that the secret key only needs to be provided to the signature final operation and not to the signature init operation.

With the proposed change therefore the signature init operation can be now implemented the same way as all existing signature schemes, i.e. without the need to provide the sk.

Signature Verification: Reverse Hashing of TR and M

When the signature generation operation is changed as outlined above, the corresponding part in the signature verification must change, too.

Algorithm 3 step 7 specifies:

mu <- H(tr||M, 512)

which implies the hashing of the input message and the public key hash TR.

It is suggested to reverse the order of the input data to the following proposal:

mu <- H(M||tr, 512)

Aside from the necessity to change the hash input parameters following the signature generation operation, the very same arguments given for the signature generation change apply here, too.

Storage of Seed instead of Key Pair

Chapter 4, past paragraph allows the implementor to store the seed zeta instead of the full public/private key pair to reduce storage space.

Please clarify whether the absolute zeta that was used to successfully

generate a key pair is only allowed to be stored. The goal of the question is to establish whether it is permissible that zeta is stored as follows:

1. Zeta is generated by a DRBG as outlined in section 3.5.1.

2. Zeta is inserted into a KDF, such as SP800-108 with zeta being the key, but allowing an arbitrary label / context. The result of the KDF is used as the zeta that is inserted into the key generation specified in chapter 5. Adding to this, please specify whether it is permissible that a chain of KDFs is allowed where the result of one KDF is used as the key for the next KDF. The output of the final KDF operation is uses as the zeta for the Dilithium key generation outlined in chapter 5.

Ciao
Stephan

11.Comments from Danny Niu, October 3, 2023

Greetings, all.

The current draft for ExpandMask function says:

v <- (H(rho||n)[[32rc]],H(rho||n)[[32rc+1]],...

which means in addition to adding the nonce n to the seed rho,
the previous r polynomial modules are skipped (thus wasted)
for each s[r] generated - even though there had already been
domain separation.

What's NIST's response on this?

Thanks.

12.Comments from Simon Hoerder, October 5, 2023

Hi,

In Algorithm 3 of FIPS 204, the inputs to the UseHint() function in line 11 are vectors. In Algorithm 34, the pseudo code for UseHint(), the inputs are single values instead.

My interpretation is that a developer should iterate over the input vectors and apply algorithm 34 to each pair of elements, i.e:

```
for idx in range(0, len(h)):
    w1dash[idx] = UseHint(h[idx], wdashapprox[idx])
```

(The range check on just h is atrocious but it's supposed to illustrate my understanding, not be decent code.)
Maybe the pseudo code should be specific about the iteration and not leave things to developer interpretation?

Best,
Simon

13.Comments from Conrado Gouvea, October 6, 2023

Hi,

Section 7.1 specifies that if desired, a hash of the message should be signed. However, simply doing that is not usually advisable because a signature of a hashed message will be identical to a signature of a sequence of bytes that is equal to the hash of the message.
I suggest using domain separation in the underlying hash, to differentiate prehashed from regular signatures, like it's done for EdDSA vs HashEdDSA in FIPS 186-5.

Thanks,

Conrado Gouvea

14.Comments from Isabela Rossales, October 10, 2023

Dear NIST Team,

In Section 7.1 of the draft, there is a mention of a prehash version of ML-DSA, albeit with limited details provided. This has sparked some questions regarding its implementation.

In the prehash version, will the step 6 of Algorithm 2 be removed from the *ML-DSA_PH.Sign* algorithm and be set as a parameter *ML-DSA_PH.Sign(sk, mu)*?
Or will a *phflag* (similarly to EdDSA-PH) be concatenated in step 6's hash *mu=H(tr||M||phflag, 512)* ? Or will step 6 stay the same, except that when calling the Sign function, the caller sends H(M) instead of M?

Sincerely,
Isabela Rossales

15.Comments from Paul Duncan, October 23, 2023

Hi Everyone,

The wording in the description of bitlen() on line 438 of section 2.3 in the FIPS 204 draft is confusing.

The description says "For a positive integer a, the minimum number of binary digits required to represent a. For example, bitlen 32 = 6 and bitlen 31 = 5."

I think the phrase "the minimum number of binary digits required to represent a" is confusing because it implies (to me, anyway) that the definition of bitlen(a) is "ceil(log_2(a))" rather than the actual definition used in the FIPS 204 draft, which is "the number of integers in the range [0, a]" or "ceil(log_2(a + 1))".

The actual definition only becomes apparent once the reader considers the examples in the second sentence of the description.

I think the phrase "the minimum number of binary digits required to represent a" in the first sentence of the description of bitlen() should be changed to read "the minimum number of binary digits required to represent the number of integers in the range [0, a]".

With that change, the full description of bitlen() would look like this:

> "For a positive integer a, the minimum number of binary digits
> required to represent the number of integers in the range [0, a].
> For example, bitlen 32 = 6 and bitlen 31 = 5."

This wording is consistent with the examples in the second sentence of the description and also consistent with how bitlen() is used throughout the FIPS 204 draft.

It might also help to add the mathematical definition to the description of bitlen() as well. In that case, the full description of bitlen() would read as follows:

> "For a positive integer a, ceil(log_2(a + 1)), or the minimum number
> of binary digits required to represent the number of integers in the
> range [0, a]. For example, bitlen 32 = 6 and bitlen 31 = 5."

Thanks,
--
Paul Duncan

16.Comments from Paul Duncan, October 23, 2023

Hi Everyone,

The value in the "Private Key" column for all three rows of "Table 2. Sizes (in bytes) of keys and signatures of ML-DSA." in section 4 of the FIPS 204 draft (at the top of page 14) appear to be off by 32 bytes.

* ML-DSA44: Table 2 = 2528 bytes, Expected = 2560 bytes
* ML-DSA65: Table 2 = 4000 bytes, Expected = 4032 bytes
* ML-DSA87: Table 2 = 4864 bytes, Expected = 4896 bytes

I suspect the key sizes in the table were not updated when the length of tr was increased from 32 bytes in Dilithium 3.1 to 64 bytes for the FIPS 204 draft (as described on line 297 in section 1.3.2).

According to the output of Algorithm 1 ML-DSA.KeyGen() and the output of Algorithm 18 skEncode(), the private key size (in bytes) is defined as:

sklen = len(rho) + len(K) + len(tr) + 32 * ((k+l) * bitlen(2*eta) + d * k)
(bitlen(a) = ceil(log_2(a + 1))

Substituting in the ML-DSA44 parameters (k=4, l=4, eta=2), I get:

= 32 + 32 + 64 + 32 * ((4 + 4) * bitlen(2*2) + 13 * 4)
= 2560 bytes

ML-DSA65 (k=6, l=5, eta=4):

= 32 + 32 + 64 + 32 * ((6 + 5) * bitlen(2*4) + 13 * 6)
= 4032 bytes

ML-DSA87 (k=8, l=7, eta=2):

= 32 + 32 + 64 + 32 * ((8 + 7) * bitlen(2*2) + 13 * 8)
= 4896 bytes

Thanks,
--
Paul Duncan

17.Comments from Paul Duncan, October 26, 2023

Hi Everyone,

The description of centered modular reduction (mod +/-) on line 444 in section 2.3 of the FIPS 204 draft is incomplete because it does not define how to handle an odd modulus.

Two reasonable interpretations of the description in the FIPS 204 draft ("floor(a/2)" and "round(a/2)") could lead to a implementation that is different than the one provided in section 2.1 of the Dilithium 3.1 spec.

For example, if an implementer interprets "alpha/2" as "floor(alpha/2)" for an odd modulus alpha = 7 using the description in the FIPS 204 draft:

```
x = y mod^{+-} 7
-floor(7/2) < r' <= floor(7/2)
-3 < m' <= 3

4 = y mod^{+-} 7
y = (no solution)
```

Versus the description from section 2.1 of the Dilithium 3.1 spec:

```
x = y mod^{+-} 7
-((7-1)/2) <= r' <= ((7-1)/2)
-3 <= r' <= 3

4 = y mod^{+-} 7
y = -3
```

If an implementer interprets "alpha/2" in the centered modular reduction description in the FIPS 204 draft as "round(alpha/2)", then the problem is even worse:

```
x = y mod^{+-} 7
-round(7/2) < r' <= round(7/2)
-4 < m' <= 4

4 = y mod^{+-} 7
y = -3 or 4 (two solutions)
```

Suggested solutions:

1. Update the centered modular reduction description on line 444 in section 2.3 of the FIPS 203 draft to explain how to handle an odd modulus, similar to section 2.1 of the Dilithium 3.1 spec (including the clarifying footnote).
2. Add floor() to the centered modular reduction description, so it reads "floor(-alpha/2) < m' <= floor(alpha/2)".

Thanks,

--
Paul Duncan

18.Comments from Paul Duncan, October 26, 2023

Hi Everyone,

It would be helpful if the calls to H() and H_128() in the algorithms in the FIPS 204 draft were hyperlinks to section 8.3.

Instances:

* Algorithm 1, lines 2 and 8
* Algorithm 2, lines 6, 8, and 15
* Algorithm 3, lines 6, 7, and 12
* Algorithm 23, lines 4, 7, and 9
* Algorithm 24, line 4
* Algorithm 25, line 4
* Algorithm 28, line 4

Thanks,

--
Paul Duncan

19.Comments from Gustavo Banegas, October 31, 2023

Dear NIST,
Regarding the choice of XOF used in ML-DSA and ML-KEM (Dilithium and Kyber), if we take the example of SPHINCS+, it is possible to implement it with different hash functions. However, this choice seems to have already been firmly decided upon for Kyber and Dilithium.

An alternative could be an XOF based on a symmetric primitive (e.g. ASCON). For security levels beyond 128 bits of security, it's worth exploring the potential of upcoming, yet-to-be-specified variants of ASCON or a different secure XOF besides SHAKE, there were previous discussions concerning the performance of SHAKE/SHA-3.

Additionally, with regard to performance optimization, we would like to advocate for the implementation of randomization in the context of Dilithium. This topic was previously deliberated on May 5th and other occasions this has been discussed into modify the XOF, and we agree with the previous discussion in this topic that it would be beneficial to replace SHAKE to produce pseudo-randomness with an SP 800-90C certified DRBG.

Do you plan to give any comment or modifications regarding this matter?

All the best,
Gustavo

20.Comments from Graham Costa, November 16, 2023

Hi,

Thank you for the opportunity to review this standard. Our team has reviewed the standard from the perspective of a module developer who regularly submits modules to CMVP for validation. With that in mind, our main focus is on how requirements and statements in the standard may be interpreted by developers, test labs and certifiers.

Comment 1:

Lines 592 - As SP 800-89 is now a dated publication (published in 2006 and largely referencing FIPS PUB 186-3 exclusively using SHA1) we'd suggest references are updated to the specific section that is expected to be applied. In particular, as written, SP 800-89 explicitly covers key validation for DSA, ECDSA and RSA - as such it's not immediately obvious what parts of that standard are expected to apply in the case of validating public keys from ML-DSA.

Proposed fix - Add an explicit reference in the text to the sections from SP 800-89 that FIPS 204 expects to apply in the context of ML-DSA. Independent of this, SP 800-89 should be updated given its recent reference from FIPS PUB 186-5 and proposed reference from FIPS PUB 204 and 205.

Comment 2:

Lines 588-589 - To avoid potential confusion with available options to an 'Approved RBG' we'd propose explicitly stating here that "seed … shall be generated using output of an approved RBG using one of the architectures defined in SP 800-90C." That standard in turn references SP 800-90A and SP 800-90B which don't need to be included in this standard.

There is potential for confusion by referencing all three standards. In particular, whether outputs from an SP 800-90B noise source could be used without a DRBG and/or whether a platform supporting an approved DRBG must also include its own noise source in addition to a DRBG. SP 800-90C makes it clear that all architectures require use of a DRBG and separately makes it clear that the DRBG may be seeded using external noise sources.

Comment 3:

Line 598 - Linked to the comment above, we propose removing the word 'freshly' from the sentence The seed ξ shall be **freshly** generated using an approved RBG,…'. Provided entropy has been protected, there is no requirement or benefit to requiring entropy be generated at the point of generating a signature and where this has the potential to impact the performance of many modules where entropy is generated during periods of inactivity and stored in an

entropy pool awaiting use. This standard shouldn't introduce any additional requirements on entropy (intentional or not) that aren't present in the NIST CTG standards for Approved RBG.

If the worry is that entropy in storage could be compromised or corrupted, the same would apply to keys used by this algorithm.

Should you have any further questions, please do not hesitate to contact us.

Graham COSTA (he/him)
Security and Certifications Manager
Digital Identity and Security
Thales

21.Comments from Danny Niu, November 17, 2023

I'm testing my implementation of ML-DSA/Dilithium against the Oct 2023 example values provided by NIST.

So far, verification test passes (i.e. verification subroutine recognizes correct signature-message-publickey tuple).

However, I cannot test signature generation. One problem I've identified so far, is that the private key is too short to be correct and acceptable. I've calculated the required length of private keys for category I and III parameters, and multiply them by 2 to account for hexadecimal encoding, and the length of the "sk" lines from the example files are shorter than them.

Would NIST consider updating the example values? Or provide guidance on workarounds so that signature generation can continue to be tested?

Thanks.
-- DannyNiu/NJF.

22.Comments from Beat Heeb, November 17, 2023

The verify algorithm can be slightly simplified by removing the last check ("[[number of 1's in h is <= omega]]") from line 13 in algorithm 3 and the corresponding text on lines 756 and 757. Rationale: The check is irrelevant because the given signature encoding does not allow to encode more than omega one bits in h (there is space for no more than omega one bit indices). A correctly encoded h vector therefore cannot contain more than omega one bits and the check is always true. The correctness of the encoding is already checked in sigDecode() on line 2 of algorithm 3.

Regards
Beat Heeb
Oberon microsystems

23.Comments from John Gray, November 17, 2023

In section 9.4 of SLH-DSA and 7.1 there is mention of accommodating pre-hashing. This is great to see! I am requesting that you define separate Object Identifiers for the pre-hashed versions verses the non pre-hashed versions.

Since you accommodate for prehash in FIPS 204 and 205, having a separate OID definition or defined domain separation would prevent application incompatibility, otherwise applications would need to have pre-shared information amongst themselves (or try multiple times in the case of failures, which isn't helpful). So please define either different OIDs or domain separated versions in those drafts. I think using different OIDs would be the easiest for implementors otherwise the format of the message would have to first be parsed to determine which method to use (and anytime we implementors have to parse stuff opens up the possibility of errors)…

24.Comments from Danny Niu, November 18, 2023

The current formula for signature length in Algorithm 2 ML-DSA.Sign and Algorithm 3 ML-DSA.Verify is:

32 + l*32*(1+bitlen(gamm_1+1))+omega+k.

However, the actual length of challenge string packed into the signature is 2*lambda bits, (which happens to equal k*8 octets).

This should be corrected.

I should've said more clearly. I mean the "32" at the beginning should be changed to "k*8", or whatever variable/expression NIST see fit, since the length of the challenge packed into the signature is not fixed.

25.Comments from John Preuß Mattsson, November 20, 2023

Dear NIST,

Thanks for your continuous efforts to produce well-written open-access security documents. Please find attached our comments on FIPS 203, 204, and 205.

Best Regards,
John Preuß Mattsson,
Expert Cryptographic Algorithms and Security Protocols

Comment attached.

Date: November 20, 2023

Ericsson AB
Group Function Technology
SE-164 80 Stockholm
SWEDEN

Comments on the draft versions of FIPS 203 (ML-KEM), FIPS 204 (ML-DSA), and FIPS 205 (SLH-DSA)

Dear NIST,

Thanks for your continuous efforts to produce well-written open-access security documents. Please find below our comments on FIPS 203 (Draft), FIPS 204 (Draft), and FIPS 205 (Draft).

General comments on all three draft specifications:

- *"secure even against adversaries who possess a quantum computer"*
  "including after the advent of quantum computers"
  "If large-scale quantum computers are realized"
  "resistance to attacks from a large-scale quantum computer"
  "adversaries in possession of a large-scale quantum computer"

  A lot of adversaries already have small, error prone, and currently quite useless quantum computers. "Large-scale" is better but is also not a good term as in addition to being large, the quantum computer must have a sufficiently low error rate to be relevant. We suggest that NIST uses the established and excellent term Cryptographically (or Cryptanalytically) Relevant Quantum Computer (CRQC). This aligns with CNSA 2.0 [1].

- "A number associated with the security strength of a post-quantum cryptographic algorithm"

  We approve NIST sparingly using the term "post-quantum" at all in the draft standards. It is a quite bad term as quantum-resistant algorithms need to be deployed before the advent of CRQCs. We suggest that NIST removes the last few "post-quantum" and replace them with the established and excellent term "quantum-resistant". This aligns with CNSA 2.0 [1].

- "Key search on block cipher with 128-bit key"

  Attacking any sort of 128-bit symmetric cryptography (block cipher or not) requires a drastic number of computational resources comparable to attacking AES-128. As concluded in [2–3], at least cubic (n^3) or quartic (n^4) speedups are required for a practical quantum advantage. The

viability of quantum advantage with cubic speedups is still ambiguous [3]. Algorithms with quadratic (n^2) speedup like Grover's algorithm (which is proven to be optimal) will not provide any practical quantum advantage for breaking symmetric cryptography or any other problems.

NIST will soon standardize Ascon-128 [4] which is quantum-resistant but not a block cipher. Stream ciphers like SNOW 3G and sponge-based key derivation and authentication functions like TUAK with 128-bit keys used for protection in 4G and 5G mobile networks are also quantum-resistant. In fact, we would expect that key search on SNOW 3G requires more gates than attacking AES-128. But the exact gate counts are not very important practically. While SNOW 3G and TUAK are not NIST algorithms, we think NIST has an important role in educating the general public that also these types of algorithms are and will remain quantum-resistant.

We suggest that NIST changes the definition of security category 1 to "Key search on cipher with 128-bit key". This aligns with the statement from UK NCSC [5]:

"the security of symmetric cryptography is not significantly impacted by quantum computers, and existing symmetric algorithms with at least 128-bit keys (such as AES) can continue to be used. The security of hash functions such as SHA-256 is also not significantly affected, and secure hash functions can also continue to be used."

We also suggest that NIST provides a statement in some SP or FIPS document estimating for how many decades security category 1 and 128-bit symmetric cryptography will be allowed. The current text in SP 800-57 just states that security strengths 128, 192, and 256 are acceptable beyond 2030. When RSA-1024 was disallowed in 2010 it could almost be broken by the world's fastest supercomputer [6–7]. When RSA-2048 is disallowed in 2030 it is expected to take another 30 years until it can be broken by a supercomputer [6–7]. Using similar security margins (0–30 years), security category 1 should be allowed until 2060–2090. A suggested very conservative statement would be "security category 1 can be used at least until 2060". That would give some needed guidance for industry, but also enable NIST to allow security category 1 to be used longer if Moore's law slows down as many people predict.

- We think it is excellent that ML-KEM and ML-DSA only use SHA-3/Keccak. As stated by Mike Hamburg "SHAKE has a more appropriate interface, comparable or better performance, and is easier to make side-channel resistant" [8]. Hash functions should be designed to provide indifferentiability from a random oracle [9]. Looking at hash performance figures for small output strings it is easy to think that SHA-2 is slightly faster on some platforms, but this is often completely negated by the fact that to use SHA-2 in a secure you need a lot of complex and heavy constructions only designed to overcome the severe shortcomings of SHA-2 such as HMAC, HKDF, MGF1, HASH_DRBG, some function to get short output (NIST defines several ways for SHA-2), and often a mix of SHA-256 and SHA-512. Doing anything secure with SHA-2 is very complex. SHA-2 is not robust.

- We strongly think NIST should produce negative test vectors for all algorithms. Negative test vectors are very important for catching bugs that might have security implications. We think all future algorithm and protocols standards should be accompanied with negative test vectors. It is often claimed that security agencies participate in standardization and production of

cryptographic modules with the explicit goal of sabotaging security to enhance their surveillance capabilities. Taking a strong stance on finding security threatening implementation bugs would increase the trust in NIST as a global SDO for cryptography. Two functions that require negative test vectors are ML-KEM.Encaps and ML-KEM.Decaps. FIPS validation shall not be achievable without input validation.

- *"At present, ML-KEM is believed to be secure even against adversaries who possess a quantum computer."*
  "ML-DSA is believed to be secure even against adversaries in possession of a large-scale quantum computer."
  "SLH-DSA is expected to provide resistance to attacks from a large-scale quantum computer."
  "ML-DSA is designed to be strongly existentially unforgeable"

  We suggest removing "At present", which is not suitable for a document that will live for many years. The terms believed to, expected to, and designed to are all used when talking about security properties. We suggest removing "believed". A huge amount cryptanalytic effort targeting lattice-based cryptography has been done before and during the NIST PQC project and US government is planning to protect all national security systems using lattice-based cryptography. Maybe only use the term "designed to", which seems to be the most common in FIPS 203—205 and other NIST specifications.

- "1.3 Differences From the … Submission"

  These sections are probably better suited as appendixes in the final standards.

Comments on FIPS 203 (Draft):

- We strongly disagree with suggestions that NIST should remove ML-KEM-512 based on a heavily contested claim regarding the gate count in one theoretical memory model [10]. We agree with NIST that the cost of breaking ML-KEM-512 is higher than the cost to break AES-128. We think that it is excellent that NIST has specified ML-KEM-512. If ML-KEM-512 is slightly above or below the theoretical security level of AES-128 in one theoretical model is practically completely irrelevant. The important thing practically is that ML-KEM-512 is approximately as hard to break as AES-128. We believe ML-KEM-512 offer a significant security margin for many applications, especially if used in hybrid mode with Curve25519. As stated by UK NCSC [5], ML-KEM-512 provides an acceptable level of security for personal, enterprise, and government information.

  The maximum transmission unit (MTU) on the Internet is typically just around 1300 bytes. The encapsulation key and ciphertext are 800 and 768 bytes in ML-KEM-512 versus 1184 and 1088 bytes in ML-KEM-768. The size difference means that when using ML-KEM-512, a lot more additional information can be sent in the same packet. This significantly reduces latency, which is very important in many applications. We believe that the availability of ML-KEM-512 will increase the adoption rate of quantum-resistant cryptography. We believe most implementations will support all of the security levels so applications should be able to change the security level quickly if needed.

- We are strongly against replacing SHA-3/Keccak in ML-KEM with SHA-2 as suggested in [8]. Such a big change would risk introducing various kinds of security problems, decrease trust in ML-KEM and NIST, lower performance on most/all platforms, and significantly delay deployment of ML-KEM. Using Keccak should not be a problem for organizations that have invested in cryptographic agility.

- "makes the encapsulation key available to Party B. Party B then uses Party A's encapsulation key to generate one copy of a shared secret key along with an associated ciphertext. Party B then sends the ciphertext to Party A over the same channel"

  It does not have to be the same channel. It is quite common that a different channel is used.

- "used by two parties to establish a shared secret key over a public channel"
  "A shared secret key is computed jointly by two parties (e.g., Party A and Party B)"
  "randomness used by the two parties"

  ML-KEM is also very useful for quantum-resistant protection of data at rest using for example Hybrid Public Key Encryption (HPKE) [11]. In such use cases the party encapsulating and decapsulation may be one and the same, i.e., there is only one party. We think this should be mentioned in the specification.

- "As a result, ML-KEM is believed to satisfy so-called IND-CCA security"

  We think it should be described that the encapsulation key can be used several times. That this follows from the IND-CCA security is likely not obvious to most readers. We think this information should be mentioned in FIPS 203 and not just in the future SP 800-227.

- We think it would be good if FIPS 204 also discusses additional security properties. E.g., is ML-KEM believed to have key commitment or not? How does reusing the encapsulation key affect the security bounds. Can anything be said about multi-key security?

- "The scheme K-PKE is not sufficiently secure"

  We suggest that NIST explains is some detail why NIST believes that K-PKE is not sufficiently secure.

Comments on FIPS 204 (Draft):

- Very good that NIST embraced the suggestion to include hedged signatures [12] and made it the default mode. We believe that making this mode the default will increase the practical security in deployed systems.

- Rejection sampling is new to most users of digital signatures. FIPS 203 has an excellent table showing decapsulation failure rate. We think FIPS 204 should have a similar table showing the probability for one or more rejections in the signing algorithm. This is not trivial for most readers to calculate. Users will want to know if the variable signing time is something they need to care

about or if the probabilities are so low that the variable signing time can be ignored.

– "ML-DSA is designed to be strongly existentially unforgeable under chosen message attack"

  We suggest also adding the abbreviation SUF-CMA, i.e., "ML-DSA is designed to be strongly existentially unforgeable under chosen message attack (SUF-CMA)". This allows the reader to search for "SUF-CMA" or "CMA".

– "ML-DSA is also designed to satisfy additional security properties beyond unforgeability, which are described in [6]"

  We suggest that FIPS 204 lists the additional security properties that ML-DSA is designed to satisfy. The current text does not say if ML-DSA satisfies all or a subset of the properties, and the paper [13] analyzes a non-standardized version of ML-DSA. It is not clear to most readers if the analysis is still valid for ML-DSA.

Comments on FIPS 205 (Draft):

– *"The 12 parameter sets included in Table 1 were designed to meet certain security strength categories defined by NIST in its original Call for Proposals [21] with respect to existential unforgeability under chosen message attack (EUF-CMA)".*

  We think this is a good selection of parameters. Sections 10.1, 10.2, and 10.3 provide a clear illustration of how much easier it is to work with SHAKE instead of SHA2. The SHA-2 versions are downright inelegant and complexity like this often leads to specification and implementation bugs. We think NIST should also mention if SLH-DSA is (believed to be) SUF-CMA or not. i.e., given a number of different message-signature pairs (m_i, σ) can an attacker create a new signature (m_i, σ') for an already signed message m_i.

– For ML-DSA, hedged signatures are the default, the value rnd should be generated by an approved RBG, and the deterministic mode should not be used on platforms where side-channel attacks are a concern. For SLH-DSA, hedged signatures are not the default, opt_rand does not require use of an approved RBG, and for devices that are vulnerable to side-channel attacks opt_rand may be set to a random value. We suggest that NIST aligns the SLH-DSA specification with use the stronger ML-DSA requirements alternatively explain why it is acceptable for SLH-DSA to have much weaker requirements.

– We think FIPS 205 should describe how the security depends on the number of times the SLH-DSA private key is used to generate signatures. We think it would be helpful for the reader to understand if there are no practical limits for the number of signatures that can be generated or if systems producing a very large number of signatures should change the SLH-DSA private key periodically to keep a high security level. In ECDSA the collision probability can be ignored while AES-256-GCM with r random IVs only provide $\approx 97 - \log_2 r$ bits of security due to the collision probability [14].

- "finding such a collision would be expected to require fewer computational resources than specified for the parameter sets' claimed security levels in all cases except SLH-DSA-SHA2-128f and SLH-DSA-SHAKE-128f."

  We suggest that FIPS 205 describes how much fewer resources would be needed. An application might require different security levels for different properties. It would also be good if NIST stated that it is unknown if SLH-DSA provides the properties exclusive ownership and non re-signability [13].

- "Don't support component use."
  "cryptographic modules should not make interfaces to these components available to applications"

  We would suggest that this is softened or rewritten. That some hardware implementations do not support important building blocks like the AES round function and the KECCAK-p permutation has turned out to be very limiting for innovation, significantly decreasing performance (or security) of future standards like ML-KEM and meaning that the acceleration cannot be used for algorithms like AEGIS [15], Rocca-S [16], Snow 5G [17], and Simpira [18] that make use the AES round function. We think it is very important that many types of hardware implementations do support component use to enable future innovation and standards. In general, we strongly think that NIST should encourage hardware implementations such as CPUs to have flexible APIs supporting component use.

Best Regards,
John Preuß Mattsson,
Expert Cryptographic Algorithms and Security Protocols

[1] NSA, "Announcing the Commercial National Security Algorithm Suite 2.0"
https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF

[2] Hoefler, Häner, Troyer, "Disentangling Hype from Practicality: On Realistically Achieving Quantum Advantage"
https://cacm.acm.org/magazines/2023/5/272276-disentangling-hype-from-practicality-on-realistically-achieving-quantum-advantage/fulltext

[3] Babbush, McClean, Newman, Gidney, Boixo , Neven, "Focus beyond Quadratic Speedups for Error-Corrected Quantum Advantage"
https://arxiv.org/pdf/2011.04149.pdf

[4] NIST, "NIST Selects 'Lightweight Cryptography' Algorithms to Protect Small Devices"
https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices

[5] UK NCSC, "Next steps in preparing for post-quantum cryptography"
https://www.ncsc.gov.uk/whitepaper/next-steps-preparing-for-post-quantum-cryptography

[6] CRYPTEC, "Cryptographic Technology Evaluation Committee Activity Report"
https://www.cryptrec.go.jp/symposium/2023_cryptrec-eval.pdf

[7] CRYPTEC, "Japan CRYPTREC Activities on PQC"
https://events.btq.li/Japan_CRYPTREC_Activities_on_PQC_Shiho_Moriai.pdf

[8] NIST PQC Forum, "Comments on FIPS 203/204"
https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/SPTpYEP7vRg

[9] Maurer, Renner, Holenstein, "Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology"
https://eprint.iacr.org/2003/161

[10] NIST PQC Forum, "Kyber security level?"
https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/W2VOzy0wz_E

[11] IETF RFC 9180, "Hybrid Public Key Encryption"
https://www.rfc-editor.org/rfc/rfc9180.html

[12] Preuß Mattsson, "OFFICIAL COMMENT: CRYSTALS-Dilithium"
https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/1mQjngj_2Po/m/-p4RKXGQAwAJ

[13] Cremers, Düzlü, Fiedler, Fischlin, Janson, "BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures"
https://eprint.iacr.org/2020/1525.pdf

[14] Preuß Mattsson, Smeets, Thormarker, "Proposals for Standardization of Encryption Schemes"
https://csrc.nist.gov/csrc/media/Events/2023/third-workshop-on-block-cipher-modes-of-operation/documents/accepted-papers/Proposals%20for%20Standardization%20of%20Encryption%20Schemes%20Final.pdf

[15] IRTF, "The AEGIS Family of Authenticated Encryption Algorithms"
https://datatracker.ietf.org/doc/draft-irtf-cfrg-aegis-aead/

[16] IRTF, "Encryption algorithm Rocca-S"
https://datatracker.ietf.org/doc/draft-nakano-rocca-s/

[17] Ekdahl, Johansson, Maximov, Yang, "SNOW-Vi: an extreme performance variant of SNOW-V for lower grade CPUs"
https://eprint.iacr.org/2021/236

[18] Gueron, Mouha, "Simpira v2: A Family of Efficient Permutations Using the AES Round Function"
https://eprint.iacr.org/2016/122

26.Comments from the National Security Agency, November 20, 2023

As part of our review process, NSA Cybersecurity had a team programming up the NIST draft standards, including software engineers who were chosen specifically as non-experts. The idea was that experts would likely implement everything correctly, but we wanted to make sure average software engineers could also follow the standards. This process led to some mis-implementations and potential security issues. Many of the comments below suggest clarifications and minor tweaks to the draft standards document to hopefully make the standards easier to securely and correctly implement and to avoid some of the issues we saw during our study. We also include comments which would help increase usability and adoption rate for Federal systems. Note that some of these issues have already been addressed in the PQC forum since NIST released the drafts on August 24, 2023, but we still include them here for completeness. Comments which relate to both FIPS.203 and FIPS.204 will be included in both places.

The following are directly related to issues our prototypers had in their experiments, where we believe clarification could help prevent mis-implementation.

- **One common issue was implementing NTT.**
    - In FIPS.204 Algorithm 35, the zeta functions are explicitly mod q, but other parts of the arithmetic are not. Because this involves multiplying and adding numbers, it is very easy to overflow integer types if you don't regularly do modular reductions. We suggest this at least be mentioned in the NTT discussion or could be made explicit in the pseudocode. Mathematically this is not an issue, but it was a point of confusion in practice.
- **Another issue some of our implementers had was which mod to use.**
    - Either because of a typo or a pdf issue, FIPS.204 line 463 defines the infinity norm to be **w mod±q**, missing an important absolute value.
    - In general, there was some confusion amongst implementers of **mod± versus mod**, especially in parts of FIPS.204 which don't specify which to use. Suggest making this more explicit in FIPS.204 and removing mod± in FIPS.203 since it is used only one place (4.7) where an interval would have worked instead. Having two separate mod functions ended up leading a lot of errors/problems in our tests.
- **Bits versus bytes was perhaps the most consistent source of initial errors in our study.**
    - In FIPS.204, the function **Bitlen** can be interpreted multiple ways because it says 'minimum number of bits needed' but there is a hidden assumption about integer types. Moreover, bitlen is only used on constants (constant for a given security level) so have constants as answers. To avoid issues, we suggest simply putting in the correct constants rather than using the Bitlen function to calculate them.
    - **Bit Strings/padding** – we had multiple implementers who ended up writing out a bit string as a byte string where each byte was padded to 8 bits. We suggest adding some wordage of bit strings versus byte strings and to clarify that bits are in adjacent

positions without padding. This could be in the definition section of FIPS.204 and ideally should be the same in FIPS.203.

- **The final common type of error we came across related to Pass-by-reference:**
    - FIPS.203 says to assume no pass-by-reference in the pseudo-code (FIPS.204 doesn't specify) but in practice pass-by-reference will be used in real implementations. We suggest either repeating this where it matters to remind readers or tweaking pseudo-code to be safe with either method, for example with explicit copies. As an example of the danger, the BytetoBits() function in FIPS.203 will zeroize the encryption key if you use pass-by-reference with the pseudo code.

The following are comments which we felt were important but didn't fall into the previous category.

- **Seed size**. During FIPS.204 keygen, starting seed is 256-bit true random extended to 512 bits; this should be 512 if worried about leftover hash lemma and larger than 256 if worried about multi-target attacks.
- **Prehash**. FIPS.204 discusses pre-hashing with size 2*security. It says 'needs' rather than shall or should which could be misinterpreted (we were asked about it). The standard should make clear this is a suggestion and not a requirement since the pre-hash is out-of-scope of this signature standard. There will, for example, likely be cases where SHA384 is used with level 5 Dilithium.
- **Harmonization of FIPS.203 and FIPS.204.** By design, the background information and many functions of Kyber and Dilithium are mathematically the same. It would help developers if the standards lined up more closely. Some possible changes/tweaks:
    - At minimum, the level 1,3,5 parameter names should have the same naming conventions ; using the security target for naming would be one way to simplify rather than ML-DSA-87 vs ML-KEM-1024.
    - The organization of FIPS.203 and FIPS.204 could be closer – for example putting together Auxiliary functions in the same portions of the documents.
    - Although the NTT functions are slightly different, most of the mathematical discussion and background information could be the same in both FIPS.203 and FIPS.204.
    - Some functions like ExpandA in FIPS.204 and SampleNTT in FIPS.203 are doing the exact same thing (they are both making the matrix Ahat in KeyGen) but the pseudocode is very different, with is broken up into multiple functions in DFIPS.204. A lot of the bit/byte-manipulation and sampling functions could be the same or very similar in both standards.
- **Hash choices.**
    - In the ML-KEM specification, the cryptographic functions are defined in an early section (4.1 Cryptographic Functions) which makes the different instantiations of hashes easier to keep track of than in ML-DSA. We strongly suggest ML-DSA adopt a

similar section and the language be consistent between the two. As such, the remainder of these comments apply to both ML-DSA and ML-KEM.

- In ML-KEM section 4.1, the function XOF is defined as SHAKE and comments suggest that one can generate bits as needed by "squeezing" the sponge function to output more bits. While mathematically this is true, it is not quite consistent as written with FIPS 202 which defines the SHA3 and SHAKE functions. In section 6.2 of FIPS 202, SHAKE is explicitly defined as taking a digest length d in addition to a message m, and so FIPS implementations cannot necessarily be used in the was described. In the appendix A.2 they make it clear that SHAKE(m,d) and SHAKE(m,d+e) will share the same first d bits of output, the property being used in ML-KEM. We offer two options to fix this:
  - a) Keep SHAKE as the XOF. One can select a very large d to call for SHAKE and if more than d bits of random are needed, have the algorithm return failure. An appendix can then note that in practice for the sake of efficiency fewer bits can be called and more can be squeezed out as needed. Alternatively, have verbiage to the effect that if you need e additional bits, call SHAKE(m,d+e) and discard the first d bits.
  - b) If one wants to keep the functionality of generating bits as you need them, that is much closer to the description of the 800-90B standards for DRBG. Given that ML-KEM relies heavily elsewhere on secure hashes, we would suggest defining Hash_DRBG as the XOF used in ML-KEM/ML-DSA.
- It can be impractical to call multiple hash functions when one is taking advantage of previously-validated FIPS modules because frequently modules only implement a subset of the FIPS they are tested against. Hence, we suggest only one hash primitive be used. If SHAKE continues to be chosen for the XOF then it should be used for the hash as well. If Hash_DRBG is used, then the hash primitive chosen for Hash_DRBG should be the same hash primitive chosen to construct all of PRFs and hashes in the standard.
- Further, we suggest that the hash function being invoked in the above discussion be SHA2 with a size appropriate to the security level. The reasons to select SHA2 are manifold:
  - SHA2 family algorithms are ubiquitous in the federal space and SHA-384/SHA-512 are the algorithms used for National Security Systems as specified in CNSA. Because SHA2 is used so often in low-level parts of the computing trust infrastructure, from today's commercial and government signing infrastructure, to HMAC, to Hash_DRBG, nearly any FIPS-compliant device in use by the government that utilizes hardware acceleration for cryptography will need to have SHA2 present for the next decade. For example, Federal PKIs (including the DoD PKI) have transitioned to SHA2 and so FIPS-validated SHA2 modules are on all PKI-enabled devices in the federal

government as well as any device being purchased in the medium term. SHA3/SHAKE, on the other hand, has very limited uptake in this space.
- Due to this ubiquity, SHA2 will generally not require additional hardware if one is upgrading a system to ML-KEM/ML-DSA while the use of SHA3 may. On many near-term systems, SHA3 and SHAKE require more hardware space than SHA2 because it is an add-on.
- If the module is implemented in software, SHA2 is substantially faster than SHA3 or SHAKE.

It should be noted that we believe that hash functions are well enough understood and that SHA2 is ubiquitous enough, that a change from SHA3 to SHA2 would be a minor change. Even our non-expert implementers were able to swap hash functions in less than an hour with a few lines of code and we have already seen in SPHINCS+ that the community has high confidence in both SHA2 and SHA3 for PQ solutions. For why we think this overall comment is important to make for faster federal adoption, we would like to provide some context for how dominant SHA2 is over SHA3 when it comes to market penetration in the government sector. We consider as a proxy the number of cryptographic hardware products that have active NIST Cryptographic Module Validation Program (CMVP) validation credentials in the CMVP public database. This hardware represents the acceleration available to federal agencies as they try to adopt FIPS 203 and 204 over the next several years.

In a search performed on 10/13/23, there were 429 hardware modules implementing FIPS 180-4 (SHA2) while 34 have FIPS 202 (SHA3 or SHAKE). This order-of-magnitude difference is not solely the result of the longer life of FIPS 180-4: if we restrict to hardware modules validated in the last calendar year, 10 have FIPS 202 while 102 contain FIPS 180-4. Further, of those 10, only a single one of the hardware modules containing FIPS 202 actually had a FIPS-validated SHAKE. This dominance also extends to digital signatures. A similar search of the Cryptographic Algorithm Validation Program (CAVP) database shows that in 2023 there are 284 CAVP validated implementations of FIPS 186 signature schemes using SHA2, 2 using SHAKE, and none using SHA3.

The ubiquity of SHA2 applies to internet protocols as well. For example, most TLS, IPSec, and SSH implementations will make use of SHA2 to generate session keys. Further, these protocols often use SHA2 as part of the integrity function alongside a non-AEAD cipher. For either use case, the supporting IETF RFCs do not define SHA3 as an option - the use of SHA3 (or SHAKE) is primarily limited to signatures, which as seen above, are typically not FIPS-compliant.

While several products will likely be able to transition to ML-KEM regardless of the hash used (such as software defined services on general purpose devices), constrained or embedded form factors as well as high performance gear may delay

their transition until hardware acceleration is available. That acceleration is presently ubiquitous for SHA2, in limited availability for SHA3, and nearly unavailable in SHAKE.

To summarize, in order to ease transition for large commercial enterprises, reduce Size/Weight/Power requirements, and make it more likely that the whole of government can comply to the greatest extent possible with the requirements in National Security Memo 10 we suggest that both ML-KEM and ML-DSA adopt SHA2. For the largest parameter sets (and potentially for all) this could be something similar to:

- XOF(p,i,j):=sequential bits of SHA512_DRBG(p||i||j) as instantiated by Hash_DRBG_Generate_algorithm.
- H(s) := truncated(SHA-512(s), 256)
- J(s) := truncated(SHA-512(s), 256)
- G(c) := SHA-512(c)
- Domain separation if desired can be achieved by prepending a domain-specific value, i.e. G(c)=SHA-512('G'||c).

We want to be clear that this comment is proposing a minor change to section 4.1, and not an addition to it. Under no circumstance should there be multiple versions of ML-KEM of a particular security level that differ only by the choice of XOF. These would not interoperate with each other and would only confuse customers as they try to comply with mandates to transition to a particular version. While several different approaches can be used for e.g. key generation in RSA or ECDH, ML-KEM is unique in that due to the FO transform a change in key generation breaks interoperability with other implementations.

Other clarifying suggestions which we think would help developers:

- **Polynomial multiplication over Rings.** FIPS.204 mentions NTT(ab) = NTT(a) ◦NTT(b) which then leads developers to want to use this as a test. For both FIPS.203 and FIPS.204, we suggest explaining how polynomial multiplication works in an appendix or explicitly calling out a reference.
- **Explicit functions**. Several auxiliary functions (brv in FIPS.204, bitrev7, Compress, decompress in FIPS.203) are given pseudocode but not explicit functions. We suggest giving them full functions.
- Similarly, **NTT_mult** is an explicit function in FIPS.203 but somewhat hidden in the mathematical notation section in FIPS.204. Suggest it becomes an explicit function in FIPS.204.
- **NTT tables**. Algorithm 35 in FIPS.204 mentions that usually zeta is pre-computed. Suggest adding the pre-computed table to FIPS.203 and FIPS.204 since this is one of the more complicated operations in the algorithm and also acts as an extra validation test.

- **Type Clarification.** The input and output objects of some operations aren't immediately clear to non-experts. For example, NTT in FIPS.204 is defined on polynomials and then used on vectors in Algorithm 1. Places like Algorithm 1 line 5 would be clearer with e.g. an explicit NTT-matrix-mult function so someone could look at the exact expected input and outputs. Alternatively, there could be more exposition near line 5 and similar locations.

Typos and Minor comments in FIPS.204:

- Line 304: "ML-DSAstandard" should have a space.
- 528: odd to give "Gaussian elimination" as an example of best known techniques to attack LWE.
- 557: "scheme: As with" - As should be as.
- 570: comma should go after rho, not before
- 671: t should be \bm{t}
- 756: lists the verifier's range for coefficients of z as closed (square brackets), while the algorithm 3 requires an open interval.
- 779/algorithm 4: "using in little-endian order" has an extra 'in'
- Algorithm 22: output should be 256k not 32k.
- Algorithm 25: For clarity, could replace "Input: A seed $\rho\in {0,1}^{528}" with "Input: A bitstring $\rho \in {0,1}^{512+16}$". Also Algorithm 24.
- Algorithm 28 line 4 (expandMask): 'r' is both in the seed of the hash and in the choice of bits (a 32rc term), leading to unnecessary computation and most bits being ignored.
- Notation re-use – algorithm 6 uses c which has a different meaning in the main sign/verify; might be clearer to change. Algorithm 23 uses variable k which is also a system parameter.

27.Comments from Leif Ibsen, November 21, 2023

FIPS 204 (Draft) comment - inconsistent signature sizes:

Algorithm 2 (Sign) and Algorithm 3 (Verify) specify the signature size as

32 + l * 32 * (1 + bitlen(gamma1 - 1)) + omega + k

Algorithm 20 (sigEncode) and Algorithm 21 (sigDecode) specify the signature size as

lambda / 4 + l * 32 * (1 + bitlen(gamma1 - 1)) + omega + k

which is different when lambda is 192 (ML-DSA-65D3) or 256 (ML-DSA-87)

FIPS 204 (Draft) comment - inconsistent private key sizes:

Algorithm 1 (KeyGen) specifies the private key size as

32 + 32 + 64 + 32 * ((l + k) * bitlen(2 * eta) + d * k)

which amounts to

2560, 4032 and 4896, respectively

However, table 2 on page 14 specifies 2528, 4000 and 4864

Regards Leif Ibsen

28.Comments from the Infineon PQC team, November 21, 2023

Dear NIST PQ team,
we have the following comments on the current FIPS204 draft, we omit points that were already publicly discussed on the mailing list.

- Algorithm 2: we suggest to use the entire c in SampleInBall. Using only c1 does not appear to have any benefits and introduces unnecessary complication.
- Algorithm 1 and Algorithm 2: Key generation and signing both use a variable rho', but these variables do not refer to the same value. All other variable names are consistent between the algorithms. We suggest to rename one of the two variables.
- We suggest to use a different naming scheme for the parameter sets. The current names (44, 65, 87) might be unintuitive, hard to remember, and could be misinterpreted as the security level in bits. Also, a potential future introduction of further parameter sets with increased (k,l) could lead to irregularities (e.g., ML-DSA-1110)
- We welcome the introduction of the hedged mode and the discouragement to use the deterministic setting.

The following comments relate to minor textual clarifications:

- Line 685ff: in a fully deterministic variant, fault attacks are a more severe threat (compared to side-channel attacks)

Best regards,
the Infineon PQC team

29.Comments from the Canadian Centre for Cyber Security, November 21, 2023

Hello,

Please find attached our comments for the FIPS 204, Module-Lattice-Based Digital Signature Standard. If posted publicly, please attribute our comments to the “Canadian Centre for Cyber Security”, and refrain from publishing my name.

If you have any questions, please don’t hesitate to reach out to me directly.

Regards,

Comment attached.

FIPS 204: ML-DSA Comments

Small edits

The prefix to each comment is in the format **Section / Page / Line numbers** (or Equation, or Algorithm where relevant)

- 1.1 / 1 / 245: Since signatures are not used for protection of data, we recommend replacing "protection" with "confirmation of integrity".
- 1.3.2 / 2 / 304: "ML-DSAstandard" should be "ML-DSA standard".
- 2.1 / 4 / 368 and 377: There is some inconsistency in the use of hyphens in "public key" versus "public-key".
- 2.3 / 6 / 420: In the definition of the set B, we recommend clarifying that this is the set of integers represented by a byte.
- 2.3 / 7 / 463: Absolute value signs are missing. "For the element w\in Z_q, ||w||_{\infinity} = w mod^{+-}q." should be "For the element w\in Z_q, ||w||_{\infinity} = |w mod^{+-}q|."
- 2.3 / 7 / 464-465: The notation w_i and w[i] which is used here is not defined until lines 492 and 497, respectively. We recommend moving the definitions of w_i and w[i] from section 2.4 into 2.3.
- 2.3 / 7 / 475 and 2.4 / 8 / 493: There is duplication in notation used. Please replace Z_d by Z_m and replace R_d by R_m.
- 2.4 / 8 / 488: "then the product is also understood" should be "then the product or sum is also understood".
- 2.4 / 8 / 489-491: These lines may be removed as they are repetitive and not grammatically correct.
- 3.1 / 9 / 516-519: The bracketed sentence is too long and should stand on its own. We recommend to remove the brackets and replace "i.e" with "That is,".
- 3.2 / 9 / 525: The notation Z_q^n should be previously defined (in section 2).
- 3.2 / 9 / 525-527: Since the zero vector needs to be excluded from being a valid solution, change "a solution t" to "a non-zero solution t".
- 3.3 / 10 / 547-548: The quantifier here that coefficients be short should only apply to S_1 and S_2, not A.
- 3.3 / 10 / 548, 550, 552 and possibly elsewhere: It should be "short" for vectors and "small" for integers (e.g. integer coefficients can't be short).
- 3.3 / 10 / 553: z should be bold.
- 3.3 / 10 / 555-561: The portion of the sentence defining public and private keys from the very end of the paragraph should be moved up to the top.
- 3.3 / 10 / 575: 'tr' can be removed from the sentence here.
- 3.3 / 10 / 571 and 579: It should be t_1 instead of t_0.
- 3.3 / 10 / 579: A bullet is missing.
- 3.5 / 11 / 597-603: This is hard to parse as text, so we suggest making it into a table instead.

- 3.5 / 11 / 613: "differs from its specified length" should be "differ from their specified length".
- 4 / 13 / 640-647: We suggest to remove this paragraph except the last two sentences, which can then stay here or be moved to Appendix A.
- 4 / 14 / Table 2: The private key sizes listed are all incorrect. ML-DSA-44 private key size should be 2560, ML-DSA-65 private key size should be 4032, ML-DSA-87 private key size should be 4896.
- 4 / 14 / Table 2 and 6 / 17 / Algorithm 2: There is an error in the calculations of the signature length. In the Output of Algorithm 2, change "32 + ..." to "λ/4 + ...". In Table 2, change the Signature Size of ML-DSA-65 from "3293" to "3309". In Table 2, change the Signature Size of ML-DSA-87 from "4595" to "4627".
- 5 / 14 / 660: We recommend changing "as needed using an XOF (namely, SHAKE-256) to produce" to "as needed using an XOF (namely, SHAKE-256), denoted by *H*, to produce".
- 5 / 14 / 664: We recommend changing "the subset of polynomial vectors whose coefficients are short" to "the subset of polynomial vectors whose coordinate polynomials have short coefficients".
- 5 / 14 / 671: t should be bold.
- 5 / 15 / Algorithm 1 and 8.4 / 33 / 866: There is inconsistency in including d as a variable input of Power2Round(). Remove d in the input of Power2Round [4, 15, Algorithm 1] and [8.4, 33, 866].
- 6 / 17 / Algorithm 2, line 8: The variable name in Sign() \rho' should be changed to \rho'' to avoid confusion with the \rho' used in Key Generation which is a different value.
- 6 / 17 / Algorithm 2, line 16: This line introduces c_1 and c_2 but the accompanying text is implicitly only about c_1 . We suggest that either the accompanying text should explain both c_1 and c_2, or explicitly say that c_1 is the first 256 bits.
- 7 / 18 / 739: "and" after the semi-colon can be changed to "then".
- 7 / 19 / Algorithm 3 : Similar to 4 / 14 / Table 2 and 6 / 17 / Algorithm 2. In the Input of Algorithm 3, the length of the signature should be “λ/4 + …” instead of “32 + …”
- 8.1 / 22 / 788-789: The reference to SimpleBitPack can be removed as it is unnecessary.
- 8.1 / 23 / Algorithms 12 and 13: In line 4 of both Algorithm 12 and 13 the function BitsToInteger is given a second input, "c". This input is not part of the definition of BitsToInteger in Algorithm 5 and should be removed.
- 8.1 / 24 / Algorithm 14: In the **Input** description, the reference to "... the coefficients in **h** ..." is confusing since **h** is a vector of polynomials. The input description can be changed to "A polynomial vector **h** \in R_2^k such that the polynomials **h**[0], **h**[1], ...,**h**[k-1] have collectively at most \omega nonzero coefficients."
- 8.2 / 24 / 806-807: The last sentence that starts with "These procedures..." does not substantively clarify anything and can be removed.
- 8.2 / 25 / Algorithm 16 : There’s an extra parenthesis at the end of the Input line.
- 8.2 / 25 / Algorithm 17 : There’s an extra parenthesis at the end of the Output line and an extra parenthesis on line 4.
- 8.2 / 26 / Algorithm 18: The input lists **s_1** and **s_2** as elements of the wrong space. Replace " **s_1** \in R^{\ell} " by " **s_1** \in R_q^{\ell} " and replace " **s_2** \in R^k " by " **s_2** \in R_q^k".

- 8.2 / 27 / Algorithm 19: Change Algorithm 19 to output **s_1** and **s_2** as elements of R_q^{\ell} and R_q^k respectively.
- 8.2 / 28 / Algorithm 20: Change the Algorithm 20 input to have "**z** \in R_q^{\ell}".
- 8.2 / 28 / Algorithm 20: Change the input description from "**z** \in R_q^{\ell} with coefficients in" to "**z** \in R_q^{\ell} whose polynomial coordinates have coefficients in".
- 8.2 / 28 / Algorithm 21: Change the output description from "**z** \in R_q^{\ell} with coefficients in..." to "**z** \in R_q^{\ell} whose polynomial coordinates have coefficients in".
- 8.2 / 28 / Algorithm 20 and 21: The bound inputs into BitPack and BitUnpack in algorithm 20 and 21 are not as accurate as they could be. We suggest the following changes:
  In algorithm 20:
  - line 3: change BitPack(z[i], \gamma_1-1,\gamma_1) to Bitpack(\gamma_1 - \beta -1, \gamma +\beta +1)
  - Ouput: change signature length to B^{\lambda /4+\ell*32*(1+bitlen(\gamma_1-\beta-1))+w+k

  In algorithm 21:
  - line 4: change BitUnpack(x_i, \gamma_1-1,\gamma_1) to BitUnpack(\gamma_1 -\beta -1, \gamma +\beta +1]
  - Input: change signature length to B^{\lambda /4+\ell*32*(1+bitlen(\gamma_1-\beta-1))+w+k
- 8.2 / 28 / Algorithm 22: Change the input description from "**w_1** \in R^{k} with coefficients in" to "**w_1** \in R^{k} whose polynomial coordinates have coefficients in".
- 8.3, 29, 852: The footnote 5 should be moved into the main text.
- 8.3 / 30 / Algorithm 23: The notation for the XOF used on line 9 is incorrect. Replace H(p)[i+t-256] by H(p,k+1)[i+t-256).
- 8.3 / 31 / Algorithm 25, lines 5 and 6: the function CoeffFromHalfByte is called with two inputs, but is defined to only take one input. Remove \eta from being an input to the function.
- 8.3 / 32 / Algorithm 27: Change "**s_1** \in R_q^{\ell} and **s_2** \in R_q^k, each with coefficients in the interval" to "**s_1** \in R_q^{\ell} and **s_2** \in R_q^k, each with polynomial coordinates whose coefficients are in the interval".
- 8.3 / 32 / Algorithm 28: The output **s** can be changed to **y** as that is the variable that is used in signing.
- 8.3, 32, Algorithm 28: The length of *v* does not exactly match the formula for the length of *v* given as input to BitUnpack in line 5. As such we recommend changing c to ensure the size of *v* matches exactly for any \gamma_1 value. In line 1, change c ← 1+ bitlen(\gamma_1-1) to c← bitlen(2*\gamma_1-1).
- 8.4 / 33 / 864: Change "In this case, the functions are applied coefficientwise" to "In this case, the functions are applied coefficientwise to the polynomials in the vectors."
- 8.4 / 33 / 870 and 871: There's 2 extra parenthesis in the formula the one before the comma and the one at the end.
- References / 38 / 912-914 and References / 39 / 976-979: Note that [1] and [21] are not cited the way NIST recommends that they be cited (from within the papers themselves).

- References / 38-39: There is some inconsistency in the formatting of names in the references. For example, Elaine B. Barker versus Elaine Barker or John M. Kelsey versus John Kelsey or C.P. Schnorr.
- Appendix B / 44: There is inconsistency between variable names for normal form versus the Montgomery form. We suggest to take a, b, c for normal form and r,s,t for Montgomery form (then use u instead of t in lines 2 and 3 of the algorithm since t is taken).
- Appendix B, 44: In the description of algorithm 37, replace " Converts from Montgomery form to regular form" by "This algorithm can be used in 2 situations:
  1) Converts from Montgomery form to regular form
  2) Perform the "reduction" step in Montgormery multiplication so that Montgomery_Reduce(integer_multiplication(r,s)) = Montgomery_Multiplication(r,s)"

Suggested changes to presentation:

- We recommend adding a note that one can pre-compute the finite set of bitlen values and use them as needed, instead of computing them every time. This is possible since there is a finite set of inputs to the function bitlen in all uses in the specification and all of the inputs rely on parameters of the scheme (not intermediate values).
- Throughout document but specifically in section 4, we recommend keeping n(=256) symbolic. In the NTT, it can then say to take a (2n)-th root of unity instead of 512 to make it clear where the numbers come from.
- The value \zeta is set be a constant, 1753, for all parameter sets. This should be explicitly listed along with other values as a system parameter that does not change between parameter sets.
- Pseudocode is written using latex symbols instead of ASCII names to represent both variables and parameters. Occasionally this has the potential for confusion or even collisions when names are rendered in ASCII in a straightforward manner. For example, in the NTT and NTT^-1 Algorithms 35 and 36, the symbol "\zeta" represents a system constant set to 1753 and "zeta" is an intermediate variable. Furthermore, the use of ASCII names can help clarify the role of different variables. For example, renaming ρ to something like publicseed helps implementers understand both the role of this variable.
- The draft standard for ML-KEM contains many comments in the auxilliary algorithms indicating data types or clarifying some lines of pseudocode. The presentation of some of the auxilliary functions in ML-DSA could benefit from similar comments.
- In parts of the specification for which there is a high degree of overlap with ML-KEM, some notation could be made more consistent. For example, ML-DSA uses 'brv' to denote bit reversal of a byte and ML-KEM uses 'BitRev_7' to denote reversal of seven bits.
- On line 290-292, in the discussion of increasing the bit length of the private random seed and message representative, it would be helpful to cite a reference for the security flaw argument and correction details.

- In Section 2.3, the notation H(x, n) to mean the n-bit output hash of x should be introduced.
- The circle operator ◦ is defined in section 2.3 for multiplication in the ring T_q and in section 2.5 it is also explained that it can be used to denote matrix multiplication of matrices with entries in T_q. However, in several places it is used to multiply two elements that are not in T_q or are not two matrices with elements in T_q (see Algorithms 1, 2, and 3). All possible uses of the circle operator should be explicitly listed in the document and include explanations of how they work in each use case. This should either be included in the definition of the circle operator or in a (possibly new) subsection in the notation section.
- In Algorithms 6 and 7, the description of each algorithm in italics and the Output lines should be consistent in amount of detail; for example, Output line should specify the length of output for each algorithm.

Suggested clarifications:

- Throughout the document, it should be made explicit when central mod representation of an element of Z_q is used versus standard representation. For example, the definition of R_q could be interpreted to be that all elements are in the standard [0,q-1] representation; however, in many cases the elements of R_q are defined to have coefficients in intervals that include negative values.
- The destruction of intermediate values is given a "shall" directive, emphasizing its importance. To help an implementer follow this directive, which intermediate data needs deletion at which point should be clarified, as the guidance here only states "as soon as it is no longer needed".
- For the purposes of FIPS validation and patent protection, NIST should clarify whether alternative private key formats are allowed. For example, to save on storage space, one may wish to store the smaller root seed \xi instead of values derived from it.
- Clarification on whether it is acceptable to use the same key pair for both the randomized ("hedged") and deterministic signing variants could be added. Note that in the FIPS 186-5 Digital Signature Standard, ECDSA also has deterministic and randomized versions, and section 6.2 explicitly states that “(Deterministic) ECDSA keys **shall** only be used for the generation and verification of (deterministic) ECDSA digital signatures.” We recommend providing a similar clarification in this standard.
- In section 2.3, page 7, lines 459-460, it could be unclear whether the meaning of "i^{th}" starts with counting from 0 or 1. We recommend to specify "w[[i]] denotes the byte at index i of w". Similarly below with "bit at index j" instead of "j^{th}".
- In section 2.3, page 6, line 439, we suggest specifying how one might implement the bitlen function. In particular, while it does not appear that bitlen(0) is ever called, defining bitlen 0 = 1 might avoid confusion.
- Algorithm 15 cannot return an h with more than \omega 1's, but Algorithm 3 has an explicit check that the number of 1's in h is less than or equal to \omega (this is also reiterated on lines 756-757). We suggest either excluding this check or explain why it is explicitly there.

- On line 886, we suggest adding context to explain why *r_1* changing by more than 1 is an issue; specifically, that if *r_1* changes by more than 1, then the hint could not be 1 bit and keeping hints to a single bit minimizes the size it adds to the signature.
- For Algorithm 35 and 36, we suggest adding an explanation in section 8.5 about how to apply NTT and NTT^-1 to vectors of elements over the space or make it explicit in the algorithm steps how to deal with such inputs.

Functional change:

- 5 / 15 / Algorithm 1: Change the key generation seed \xi to at least 40 bytes to prevent a multi-target attack. (Note that we have the same comment for ML-KEM about the seed used in Key generation).

30.Comments from Cloudflare, November 21, 2023

Please find our public comments on FIPS IPD 203, 204, and 205 attached.

Best,

Bas

Comment attached.

To: fips-203-comments@nist.gov; fips-204-comments@nist.gov; fips-205-comments@nist.gov
Subject: Comments on FIPS 203, 204, and 205

A submission from Cloudflare, Inc., in response to the National Institute of Standards and Technology's (NIST) 24 August 2023 request for public comments on initial public drafts of FIPS 203 "Module-Lattice-Based Key-Encapsulation Mechanism Standard", FIPS 204 "Module-Lattice-Based Digital Signature Standard", and FIPS 205 "Stateless Hash-Based Signature Standard".

Cloudflare appreciates this opportunity to comment on the National Institute of Standards and Technology's (NIST) request for public comments on the initial public drafts of FIPS 203, 204, and 205. Cloudflare submits the following comments, which will address our own experience with post-quantum cryptography and protocol design, our view on measured cryptographic agility as it applies to the present drafts, and specific comments for each.

Introduction and Cloudflare Background

Cloudflare is a leading connectivity cloud company. It empowers organizations to make their employees, applications, and networks faster and more secure everywhere, while reducing complexity and cost. Cloudflare's connectivity cloud delivers a full-featured, unified platform of cloud-native products and developer tools, so any organization can gain the control they need to work, develop, and accelerate their business.

Powered by one of the world's largest and most interconnected networks, Cloudflare blocks billions of threats online for its customers every day. It is trusted by millions of organizations – from the largest brands to entrepreneurs and small businesses to nonprofits, humanitarian groups, and governments across the globe. Cloudflare is used by nearly 20% of all Internet websites.[1]

At Cloudflare we have helped increase security and privacy on the Internet by pushing the envelope on cryptographic design and deployment, by contributing to among others TLS 1.3, MLS, DNS-over-HTTPS, Encrypted ClientHello. From 2019 onwards we have executed several internal and external large-scale experiments to determine the real-world deployability of post-quantum cryptography.[2] This has allowed us to deploy an early version ML-KEM (FIPS 203) in production, which at the time of writing, is used by 1.7% of all our inbound TLS 1.3 connections.

---

[1] See W[3]Techs, Usage Statistics and Market Share of Reverse Proxy Services for Websites, https://w3techs.com/technologies/overview/proxy.
[2] https://blog.cloudflare.com/the-tls-post-quantum-experiment/
https://blog.cloudflare.com/sizing-up-post-quantum-signatures/
https://blog.cloudflare.com/post-quantum-to-origins/

Measured cryptographic agility

The need for cryptographic agility

We believe it is crucial that protocols for use on the Internet are designed with the flexibility to replace its underlying cryptographic primitives gradually and securely in case of compromise. The threat of quantum computers is the obvious example, and we applaud NIST's efforts to standardize post-quantum cryptography.

We also feel it has been a prudent choice of NIST to standardize SHA3, so that it can replace SHA2 in case a weakness is found (for which there is no indication).

The cost of cryptographic agility

However, we also believe that due restraint should be exercised in adopting new primitives in *existing* protocols. The case for post-quantum cryptography is clear, but, for instance, *at the moment* there is no point replacing SHA2 with SHA3 for the key schedule of TLS 1.3: there is no indication that SHA2 is weak, and adopting SHA3 comes with a significant cost. If both are available, some users will end up enabling one, and disabling the other. Those that want to be compatible with all, will need to deploy both. This incurs a significant cost in development, testing, and increased surface for implementation mistakes. The situation is more pertinent for constrained use cases, such as embedded devices.

That does not mean SHA3 will see no deployment. To the contrary: for new use cases, we prefer SHA3, or to be more precise, SHAKE, as it is a more versatile primitive that is easier to use correctly (no need for HMAC, HKDF, or MGF1). We are pleased to see SHAKE used in these drafts.

Call for continued restraint

We appreciate the choices NIST has made so far, showing restraint where differences between variants are minimal, but allowing different options where it matters:

- NIST has removed many variants in the present drafts as compared to the submissions: the AES-based variants have been removed, leaving only the SHA3-based variants for ML-KEM and ML-DSA.
- NIST has picked only a single KEM for now. We understand a second KEM might be picked from the fourth round, in case cryptanalysis against structured lattices improves.
- Three signature schemes have been chosen for standardization, of which standards have been drafted for two. NIST is looking to standardize more in an on-ramp. Considering the widely varying performance and security characteristics of the available schemes, we feel it was the right choice.

We see one opportunity to reduce the number of variants:

- We do not see the need for both a SHAKE and a SHA2-based variants of SLH-DSA.

We ask NIST to continue exercising restraint, and reject calls to standardize *additional* variants using different symmetric primitives.

CLOUDFLARE

12-round SHAKE / SHA3

For all schemes, not just SLH-DSA, hashing accounts for a significant, if not often the majority, of the computation.

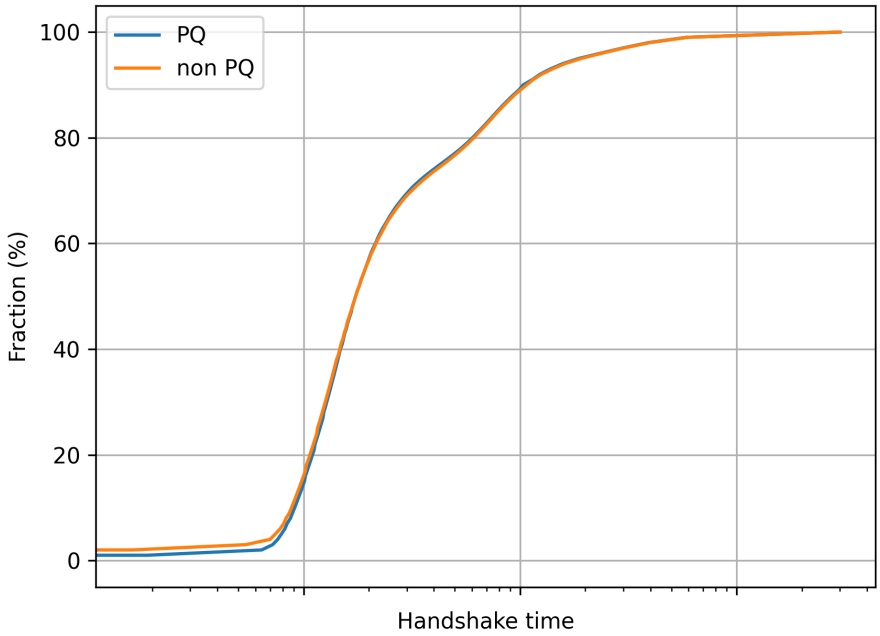
The designers of SHAKE / SHA3 have reiterated their 2016 proposal to use a variant of 12 rounds as was originally proposed, which has now been dubbed “TurboSHAKE”. We are comfortable with the wide security margin of the 12-round variant, and would welcome NIST *replacing* SHAKE / SHA3 in these drafts with their “Turbo” variants.

Specific comments on FIPS 203 (ML-KEM)

Of the three drafts, we have the most hands-on experience with ML-KEM, which is now deployed in production worldwide.

We expect ML-KEM to be a practical replacement for classical key agreement on the Internet. We like its simple design and great performance on most platforms. Informed by our 2019 experiment with Google, we were concerned that a significant portion of connections would break because of the large key sizes of ML-KEM. Fortunately, this fraction has decreased dramatically since then, and it does not seem to be a blocker.

At the time of writing, 7% of all Chrome 118+ TLS 1.3 connections to Cloudflare hosted websites are using *X25519Kyber768Draft00*, a hybrid of the classical key exchange X25519 and an early version of ML-KEM. The following graph shows the cumulative distribution function of TLS handshake times of those connections (PQ) compared to those Chrome 118+ connections that did not use Kyber (non PQ).



This shows excellent performance *on average*. Although promising, we cannot yet conclude that performance is great for all users.

Specific comments on FIPS 204 (ML-DSA)

None of the signature schemes submitted to the competition are an ideal drop-in replacement for classical signature schemes. ML-DSA's drawback is the size of its public key and signatures. This makes it impractical for some applications, and unfavorable in many. On the other hand, ML-DSA is reasonably easy to implement, and is computationally relatively cheap.

Despite its drawbacks, standardization of ML-DSA is a good choice and absolutely necessary, so that those that need to adopt early can move forward despite the costs.

We applaud NIST's ongoing efforts to standardize additional signature schemes that fit use cases for which ML-DSA is ill-suited.

Specific comments on FIPS 205 (SLH-DSA)

(No specific comments.)

31.Comments from Dev Null, November 21, 2023

This letter contains comments on the chapter “Appendix A — Security Strength Categories” identical among the 3 proposed FIPS standards NIST FIPS 203 ipd (Lines 1159 to 1238), NIST FIPS 204 ipd (Lines 1014 to 1088), NIST FIPS 205 ipd (Lines 1134 to 1213).

Definitions
To avoid confusion and distinguish between:
A) the Security Strength categories listed in FIPS ipd 203, 204, 205
B) the Security Strength method of using bits of security from “Recommendation for Key Management” SP 800-57 rev. 5 and earlier.
(https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final)

The following terms will be used:
“Strength Categories” – The new proposed method in FIPS ipd 203, 204, 205.
“Bits of Security Method” – The established method in SP 800-57.
Since both of them are called “Security Strength” in their respective documents.

Table of content:

Dev Point 1 – The problem of using sliding scales as a base of measurements

The Security Strength Categories are units that change over time with advances in technology and crypto-analysis.
Using AES-256 today (pre-quantum) has a higher security level (in respect to effort and money needed to compromise a key)
than AES-256 will have post-quantum plus years of Moore’s law making computing power cheaper.

Therefore, I propose a Security Category scale of 10 with room for future improvements or reverting back to the Bits of Security Method for a more granular and uncapped approach.

Using specific algorithms as metering sticks is like having one person’s arm being the base of all measurements of lengths.
If that reference person then has a sword accident shortening their arm, everything using that reference scale need to change values, to compare with the new arm length. With everything increasing in numerical length because of the new shortened measurement base. But everything (apart from the arm) has not changed size, yet increase in length because of the reference getting shorter.
If a new vulnerability is found in AES, all the algorithms in Strength Categories 1, 3 and 5 not affected by that vulnerability need re-evaluation because they suddenly are stronger in

comparison to AES and may change to a higher Strength Category, yet in practice those other algorithms have had no increase in security.
Science moved towards physical constants for units of measurements in order to avoid constant changes as updating a basic measurement unit affects all derivatives of that unit. Hence metrologists generally try to keep units of measurement stable.
Please keep the standard for how to measure the security of cryptographic algorithms as stable as possible, we use these high-quality standards all over the world; they are crucial for the global economy.

Dev Point 2 – The future will have lower security than what is available today

The proposed Strength Category method is capped at level 5, which is how secure AES 256 is today.
NISTIR 8105 http://dx.doi.org/10.6028/NIST.IR.8105 Table 1 explicitly mentions a reduction in security level for AES and SHA-3 as an impact of a large-scale quantum computer.
Furthermore the advancement in computing technology will make computing resources cheaper each year, making all the Strength Categories weaker year by year when looking at the capital investment needed to break the probabilistic security strength. Breakthroughs in crypto-analysis will decrease the cost even further.

In order to reach the same security level as pre-quantum, there needs to be a Security Strength Category that in the future post-quantum world will display the equivalent of the current day strength of AES-256.

Dev Point 3 – The world needs a standard for something better than AES-256

I know of companies that are trying to make home-brewed AES-512 but their implementations often end up only being AES-257 because of an incorrect understanding of bits of security.
An official AES-512 would be a good first step to establish a post-quantum security level that is at least as good or better than our current pre-quantum AES-256.

Here is a good reference on how to count bits of security by cryptographer Daniel J. Bernstein October 2023:
https://blog.cr.yp.to/20231003-countcorrectly.html
An article also touching on the Strength Categories method.

I am not an expert on Grover's algorithm, meaning that I have to rely on sources like "NIST IR 8105 - Report on Post-Quantum Cryptography", that on page 8 states:

"*Grover's algorithm provides a quadratic speed-up for quantum search algorithms in comparison with search algorithms on classical computers. We don't know that Grover's algorithm will ever be practically relevant, but if it is, doubling the key size will be sufficient to preserve security.*"

Following this quoted recommendation a current pre-quantum system using AES-128 can “just” upgrade to using AES-256 post-quantum and maintain the same or higher level of security (using the Bits of Security Method to count security strength and ignoring Moore’s law for an easier analogy).

But a current pre-quantum system using AES-256 has no way to stay at the same level of security post-quantum. The security level will just be decreased with the amount a practical implementation of Grover’s algorithm will compromise it with.

While I am no expert on Grover’s algorithm, I am an expert on cybersecurity risk management and in many safety-critical systems (IEC 61508 provides guidance) a decrease in safety level is not allowed when proposing a new implementation or updating a system.
The probability of failure must be the same or lower than the existing system (in some countries).
Many safety-critical systems and critical infrastructure (defined in EU Directive 2555 from 2022; NIS2) rely on cryptography to provide safety.
A remotely controlled rail switch or drinking water pumps are good examples.

But as noted, there is no such option to keep the same security/safety level post-quantum for existing systems reliant on using AES-256.
Furthermore, as noted in Dev Point 2 the advancement in computing technology and crypto-analysis will decrease the price of the needed computing resources each year, making all the Strength Categories weaker year by year when looking at the capital investment needed to break the probabilistic security strength.

This conflicts with the risk management strategy of always striving for equal or better safety and security over time.

As AES-128 with time needs to be replaced with AES-256.
&
RSA and ECC needs to be replaced or combined with PQC (like in X25519Kyber768Draft00).
&
Hashing functions need larger outputs.

The IT industry is already moving towards increased modularity in all crypto systems, from software over MFA tokens to HSMs.
Many systems are more ready than ever to handle change to new cryptographic algorithms without downtime. By some called the Crypto Agility approach.
This change to a more flexible approach in the IT sector also opens up for the option that organisations wanting a stable risk level can steadily increase the key and hash output lengths if there are algorithms that allow for it.
But with the current suggested cap of Strength Category level 5 and the level in practice being decreased with advances in technology and crypto-analysis it is hard (if not even impossible) for an organisation to keep a stable risk level in respect to the usage of cryptographic algorithms

detailed in NIST standards.

I appreciate the huge work done by NIST, scientists and the industry in the making of these PQC standards, it has been a great journey, my thanks go out to everyone involved.

Kind Regards
-Dev Null
November 22nd, 2023
Dev Null
Quantum Key Distribution Network Engineer
at the Technical University of Denmark

32.Comments from Gideon Samid, November 21, 2023

Dear NIST officials,

Your invitation to the public to speak on this important matter is much appreciated. The wisdom of crowds. It is the best antidote against groupthink. Cyber security today is based on the assumption that the attacker is not smarter than expected. It is a tenuous assumption as history indicates. It calls for a broader examination of the challenge ahead. I have come to cryptography from the field of innovation appraisal, specializing in estimating the innovation load associated with an innovation challenge. Cracking a cipher may be represented as an innovation challenge. This off-side approach to cryptography has led me to the following observation.

The very process managed by NIST to identify a post quantum algorithm points to its inherent flaw. Selection of algorithms is based on absence of a published breach. The longer an algorithm stays in the public domain without a public breach, the more it is deemed fit. This selection process suffers from two critical flaws expressed in the two following prospective scenarios: (i) a public breach may surface in the near or far future, (ii) an undetected private breach will undermine the purpose of the selection process.

The first scenario is well accounted for by NIST, preparing standby alternatives, and managing a modular configuration to ease the process of cipher replacement. The second scenario is (i) more likely, (ii) more damaging, and (iii) leading to more narrow adoption. All in all, the high likelihood of a private breach of any selected algorithm presents a strong need to re-imagine our cyber defense.

Ahead in this comment, I will elaborate on the unacceptable risk of the private breach scenario, and present a cryptographic alternative that responds to the challenge of quantum computers even when combined with the challenge of a smarter mathematician. This alternative approach is based on pattern-devoid cryptography, which has peer-reviewed accounts (including a book to be published next month), is expressed in a few dozen US patents, and has been vetted by the German national institute of standard, TÜV.

The motivation of the hundreds, perhaps thousands, of cryptographers to publish a breach of any selected algorithm, is personal reputation, which indeed keeps very smart people awake many a night. These mathematicians work with very limited computing power. By contrast, would be private breachers are all the countries in the world, to name a subset of interested breach parties. *The ideal state for any country in the world today is for it to have a secret private breach for an algorithm that no lesser than the US NIST declares secure*. Carefully handled such a secret breach will remain hidden for a long time, and offer substantial political and economical advantages to the breach holder -- in times of peace. In times of war the breach holder will impress deep confusion and paralytic mayhem on its adversary. Especially if the adversary is the United States that is disproportionally cyber reliant. All countries in the world, therefore, will regard the prospect of identifying a private breach to a NIST declared

secure PQC, as a prime (secret) national objective. Countries will assemble large teams of their best mathematicians and provide them with powerful computing machines, to which academic cryptographers are not privy. We will never know if any of the algorithms NIST lists as qualified has been privately breached by one or more of the countries of the world bent on so doing. This specter will make the entire NIST operation a supportive tool for US adversaries.

Looking from the opposite side, countries of the world will suspect that NIST selected algorithms have all privately been breached by the NSA, and therefore be apprehensive of adoption.

The reason that all the proposed algorithms are breach-open is that they are all based on mathematical complexity. The more complex a mathematical challenge, the more avenues for mathematical shortcuts are associated with it. Obviously, such shortcuts that simplify the math and enable a breach, lie far off on the territory that is traversed with human imagination. Otherwise, these simplifications would have been spotted right away. Indeed, these mathematical breach procedures may require so much mathematical imagination that no one, no country will have what it takes to extract them in a timely fashion. Namely these algorithms will serve their purpose.

Both the existence of such mathematical shortcuts as well as the measure of mathematical imagination needed for their extraction, are matters that pose a great challenge for being credibly estimated. Using a methodology I started to develop in my PhD dissertation at the Technion in Israel the results for mathematical complexities of the NIST candidates are alarming. Transforming flat (Cartesian) lattice representation to an unbound geometry (non metric space) may enable a much simpler computational dynamics leading to an effective breach. It is therefore that we should strive to achieve security with ciphers that are mathematically so simple that there is no room for a shortcut. Security then is constructed through lavish use of randomness. Pattern devoid cryptography accounts for ciphers which limit their cryptanalyst to brute force attack, and then deny the attacker success by using a dynamic key where size and content grow with use, and by deploying AI-guided use of ciphertext dilution that is readily reversed by the intended recipient, but remains inherently confusing to the omnipotent attacker. Pattern devoid cryptography is less elegant. It is using larger, secret size, secret geometry keys, and is communicating extra-long ciphertexts. But what is gained by this inelegance is mathematically proven security.

As described pattern devoid ciphers don't fit neatly into modern day cyber dynamics, the way this NIST candidate algorithm does, but given the unrelenting risk of private breach, it is well advised to install a pattern devoid cipher at least in the mode of "Lifeboats on the Titanic" -- namely for the eventuality when the elegant cryptography fails. The pattern devoid, clumsy cipher will kick in and save the day.

Reference:

1. "Tesla Cryptography:" Powering Up Security with Other Than Mathematical Complexity https://eprint.iacr.org/2023/803

2. “Pattern Devoid Cryptography” https://eprint.iacr.org/2021/1510

3. “Artificial Intelligence Assisted Innovation” https://www.intechopen.com/chapters/75159

4. “AI Resistant (AIR) Cryptography” https://eprint.iacr.org/2023/524

5. “The UnEnding CyberWar” https://www.amazon.com/Unending-Cyberwar-Gideon-Samid/dp/0963522043

6. “The Cipher Who Came in from the Cold” https://www.amazon.com/dp/B0B8PFGZSB/

Gideon Samid

33. Comments from Falko Strenzke, November 22, 2023

The following proposal is a result of the recent discussion on the LAMPS list about safe signature-separability defences and safe optional pre-hashing for the PQC signature schemes with contributions from David Cooper, Markku Saarinen, Darren Johnson and others. However, with this specific form of proposal I still can only claim to speak for myself.
I propose to build the following two features into ML-DSA (FIPS 204) and SLH-DSA (FIPS 205):

1. Introduction of a binary flag in the algorithm's interface that gives domain separation between the two uses of directly signing the message and signing the hash of the message (pre-hashing).
2. Introduction of a context string into the algorithm's interface of a length between 0 and 255 octets that allows the protocol or application to define a custom domain separation string.

This proposal is meant to be equivalent to the construction of the respective features of Ed25519ctx/Ed25519ph or Ed448/Ed448ph in RFC 8032.
The reason for the first point is the need to avoid any ambiguity with respect to what is the signed message given the alternate options of direct-signing and sign-pre-hashed-message. When lacking this domain separation feature in ML-DSA, protocols that allow both variants and cannot provide authentic information about the variant during signature verification will be subjected to signature forgery attacks.
The reason for the second point is that currently there are plans to design composite signature schemes that fulfil non-separability notions, i.e. make it impossible that a signature is stripped from the set of signatures in a composite signature and the remaining signature(s) appear as (a) valid signature(s) of the message. Existing protocols that cannot provide for authentic information about the nature (composite/standalone) of the signature algorithm during verification can achieve non-separability with a means of domain separation in the signature algorithm. The protocol can thus use the context string to ensure domain separation between composite and standalone use and possibly realize further security features such as domain separation between applications.
- Falko

34.Comments from the NXP PQC team, November 22, 2023

Dear NIST team,

The NXP PQC team would like to make the following comments on the FIPS 204 draft:

1) We would like to clarify whether implementations can add a finite For loop in Line 11 of ML-DSA.Sign, instead of an infinite While loop. If there is any error during the signing routine, the device will get stuck in an infinite loop. Rather, it would be preferable to return an error after a certain large number of attempts that happens with negligible probability (e.g., 2^-128). This can of course also be achieved by a time-out on the signing time, but it is a more complicated solution as it will be device-specific.

2) We would like to clarify whether a NIST-approved DRBG / PRNG can be used instead of the ExpandMask function in Line 12 of ML-DSA.Sign. The security of Dilithium remains the same whether y is pseudo-randomly generated, or randomly generated. The cost of ExpandMask is large in scenarios where SHA3 requires side-channel protection, as bit-leakage on y accumulated over multiple signature results in key recovery attacks. Masking this function adds a significant performance penalty to the signing routine. Instead, equivalent behavior can be achieved by sampling y using a NIST-approved DRBG (e.g., from NIST SP 800-90x) at potentially lower cost on systems where such an efficient DRBG is available.

NIST comments in the FIPS 204 draft that “For every computational procedure that is specifed in this standard, a conforming implementation may replace the given set of steps with any mathematically equivalent set of steps”. This would imply replacement of ExpandMask by a DRBG is allowed, as the output distribution is the same. We would like to confirm that this is indeed the case.

Thanks for all the hard work.

Kind regards,
NXP PQC team

35.Comments from Markku-Juhani Saarinen, November 22, 2023

Dear NIST,

Section 7.1 "Prehash ML-DSA" of FIPS 204 IPD allows *"signing of a message rather than signing the message directly."* I propose that this option be removed.

In the case of ML-DSA, the existence of the prehash option is seemingly motivated by the perceived speed difference between SHA2 and SHA3, which is arbitrary and platform-dependent. However, there are significant disadvantages:

1. The option opens up ML-DSA to non-targeted collision attacks and contradicts the BUFF claim of Section 3.3.

Proof: Signature generation and verification use 512-bit variable mu = H(tr || M) as a starting point (Line 6, Alg 2 and Line 7, Alg 3), where tr is derived from the public key.

We observe that arbitrary H(m1) = H(m2) collisions are also "universal" H(tr || H(m1)) = H(tr || H(m2)) collisions for any value of tr, and hence ML-DSA is vulnerable to them if simple prehashing is allowed. However, creating H(tr || m1) = H(tr || m2) collision would require a targeted collision attack using tr = H(pk).]

2. Since the signing and verification functions don't know if the input is a message or a hash of a message, the option potentially makes the same signature valid for both M and H(M) -- depending on context -- and resulting in an existential forgery: I query a signature for H(M) and then submit M'=H(M) as a second signature.

Discussion: Comparison to SLH-DSA.

In another message, I am suggesting that

- for FIPS 205 / SLH-DSA only a specific "prehash" M = H(pk || M') version is allowed instead of plain M for signature computation.

- for ML-DSA I am suggesting that only the non-prehashed versions mu = H(tr || M) are permitted instead prehashed. For ML-DSA prehashing would imply mu = H(tr || H(M)).

In both cases, removing the (unauthenticated) input option removes the need for domain separation padding. In both cases the end result -- a concatenation of a public key prefix with the message -- is consistent.

We note that SLH-DSA (FIPS 205 IPD), Algorithm 18 slh_sign(M, SK) requires two passes over entire M without prehash:

Line 7: R <- PRFmsg (SK.prf, opt_rand, M)
Line 10: digest <- Hmsg (R, PK.seed, PK.root, M)

We observe that to compute "digest" in SLH-DSA, one will need to hash the entire M first to obtain "R," and then one will have to hash the entire M again to obtain "digest" on line 10. This is entirely sequential due to chaining of "R." In practice, this means that one can't compute a signature over streaming data, rendering the "pure" scheme unusable in many use cases. So, there is a substantial need for "prehash" in SLH-DSA.

Additionally (as discussed above) I would argue that using plain H(M) is non-ideal as its use makes any scheme vulnerable to untargeted collision attacks. Hence, I proposed to use mu = H(pk || M) for SLH-DSA as well -- in that case the public key is already a pair of compact hashes, so this is analogous to "tr" and the countermeasure (BUFF transform) used by ML-DSA. The main motivation for double-hashing was collision attacks, which the public key prefix mitigates.

However, in ML-DSA (FIPS 204 IPD), the signatures are computed from mu = H(tr || M) (Line 6, Alg 2 and Line 7, Alg 3). There is no need to process input data M twice when processing large inputs; one only needs to know the seed tr = H(pk) when one starts the process. Hence prehashing is merely a security problem here, with no major performance advantages.

Best Regards,
Dr. Markku-Juhani O. Saarinen <markku-juhani.saarinen@tuni.fi>
Professor of Practice, Tampere University

36. Comments from P. Kampanakis, J. Massimo, T. Lepoint, AWS, November 22, 2023

Dear Dustin, NIST PQ Project team,

We appreciate NIST's leadership in standardizing schemes that the whole world can use. We know the process is tough and can sometimes be heavily scrutinized. The new schemes will be with us for years, and we can understand the impact this whole project led by NIST will have.

We would like to provide feedback on the ML-DSA FIPS-204 draft too:

First, as with ML-KEM, we want to bring up the security level mappings to ML-DSA parameters. It is unclear how the ML-DSA parameters are mapped to the MAXDEPTH levels. For example, we are not sure how to reach to that ML-DSA-65 maps to about "*2^221/MAXDEPTH quantum gates or 2^207 classical gates* ". This could be done in Section 4 or Appendix A, but we think you want to reference section C.5.1. of the Dilithium Round 3 submission.

Secondly, line 767 suggests "t*he digest that is signed needs to be generated using an approved hash function or extendable-output function (XOF) (e.g., from FIPS 180 [8] or FIPS 202 [7]) that provides at least λ bits of classical security strength against both collision and second preimage attacks [7, Table 4]*". The document should rather prescribe SHAKE256 with 64byte output as a conservative choice for all uses. SHAKE256 aligns with Dilithium which uses it internally as well. Also on the predigesting topic. There was a recent discussion in the NIST list about updating ML-DSA and SLH-DSA to include some domain separation for when signing M and H(M) so that an application which supports both cannot suffer existential forgeries where two different messages M and H(M) can be validated by the same signature (depending on the verifier logic). Although this would be a problem for use-cases that need to be able to sign M and H(M), these are not many. X.509 does not need to sign M and H(M), it can only do M and that is what draft-ietf-lamps-dilithium-certificates specifies. Other applications are similar, they don't need to support both M and H(M). Similarly, CMS allows for both, but in the case of H(M) you can add any arbitrary attributes for domain separation in the optional SignedAttributes. For context, in the past EdDSA took a similar path which ended up being futile. It specified pureEdDSA and prehashEdDSA. That ended in unnecessary complications in the signature although in the end pureEdDSA was mostly adopted in various standards. Thus, we are against tweaking ML-DSA which would complicate it. Applications that need to support both signing M and H(M) should specify any domain separation they need. We do suggest for NIST to update section 7.1 for ML-DSA to spell out the application existential forgery concern and suggest domain separation for H(M) to prevent it in the application that uses ML-DSA.

We want to suggest addressing the keypair vs pseudorandom value storage issue. This has come up previously in other fora. The public/private keypair can be reconstructed by storing the 32 random byte value ξ. In some cases some parties may prefer to store this only and reconstruct the keypair when they need it. Using this 32-byte value instead of a few kilobytes of

(pk,sk) could be briefly discussed as an option for the keypair owner in Section 5.

Additionally, a few minor suggestions

- The naming convention for the security levels of ML-DSA (-44, -65, -87) are an anti-pattern with how we traditionally name algorithms with their security level. While it is true that 44 65 and 87 come from the dimensions of the matrix A, the numbers may be interpreted as bit strength, or more tenuously that ML-DSA-65 may be 21 “security levels“ more than ML-DSA-44, when there should be no connotation. In short, should the dimensions of matrix A be the distinguishing numerical value we assign to each parameter set? Will this cause misunderstandings?
- Line 304: Typo in “*ML-DSAstandard*”.

Regards,
Panos Kampanakis, Jake Massimo, Tancrede Lepoint
Amazon Web Services (AWS)

37.Comments from Michael Ravnitzky, November 22, 2023

Comment attached.

Comments on FIPS 204: ML-DSA

FIPS 204 specifies a module-lattice-based digital signature algorithm, ML-DSA, for applications that require authentication and non-repudiation. A digital signature algorithm is a method of generating and verifying signatures on messages, which can prove the identity and the integrity of the sender. A module-lattice-based algorithm is an algorithm that is based on hard mathematical problems that involve sets of points that form a regular grid in a high-dimensional space. These problems are believed to be secure against both classical and quantum computers, which are devices that can perform complex calculations very fast. Quantum computers are expected to be able to break some of the current digital signature algorithms, such as RSA or DSA, which are based on hard mathematical problems that quantum computers can solve easily.

The National Institute of Standards and Technology (NIST) has demonstrated commendable foresight and expertise in the development of FIPS 204. Their efforts to standardize a secure module-lattice-based digital signature algorithm underscore their commitment to advancing cybersecurity in an era where quantum computing poses new challenges. NIST's proactive approach in addressing these challenges ensures that the integrity of digital communications will be preserved, maintaining trust and security in digital transactions for years to come.

What follows are several observations on FIPS 204.

Clarifying EUF-CMA Security (Unforgeability) in ML-DSA

The draft document does not sufficiently detail the mechanisms by which ML-DSA ensures existential unforgeability under a chosen message attack (EUF-CMA), a critical security property for digital signature schemes. This property guarantees that even if an adversary can choose messages and obtain their signatures, they cannot forge a signature for a new, unchosen message, thereby preventing impersonation and data tampering.

While the document mentions ML-DSA's foundation on the Falcon scheme—known for its EUF-CMA security within the quantum random oracle model (QROM)—it fails to address the nuances introduced by ML-DSA's unique aspects. These include its reliance on module lattices rather than generic lattices, a distinct error distribution, and an alternative compression function. Given these differences, the direct applicability of Falcon's security proof to ML-DSA remains ambiguous.

To enhance the document's utility and credibility, it is essential to provide a thorough explanation and justification of ML-DSA's EUF-CMA security. This should include a discussion of how ML-DSA's design variations influence its security proof relative to Falcon. Such clarification will enable readers to confidently assess ML-DSA's security foundations and effectively compare it with other digital signature schemes.

Rationalizing ML-DSA Parameter Sets Based on Security Standards

The draft outlines three parameter sets for ML-DSA reflecting security equivalence to established federal security standards to cater to different quantum resistance levels. These sets are designed to strike a balance between ensuring robust security and maintaining performance efficiency. However, as far as I have been able to discern the document does not fully articulate the basis for these selections. It lacks detail on the specific security objectives targeted against quantum threats, the complexity of anticipated quantum attacks, the projected capabilities of quantum hardware, and the confidence intervals for the security margins.

To provide clarity and aid readers in comprehending the impact of these parameters on ML-DSA's security and performance, to the extent practicable, the document should expound on the selection process. It should detail the criteria and assumptions that guided the choice of parameters, ensuring they are in line with current security objectives.

Addressing Practical Limitations of ML-DSA

The draft primarily concentrates on the theoretical underpinnings of ML-DSA, including its mathematical foundation, algorithmic structure, and theoretical security. However, it overlooks the practical challenges that may surface during the real-world implementation and deployment of ML-DSA. Critical considerations such as susceptibility to side-channel attacks, requisite countermeasures, interoperability with established standards, and the nuances of testing and validation are absent.

To bridge this gap, the document should incorporate a comprehensive discussion on the practical aspects of ML-DSA, detailing how potential vulnerabilities can be mitigated and what measures can be taken to ensure seamless integration into existing systems. It would also be beneficial to include examples demonstrating ML-DSA's application across various domains, such as secure email communication, digital certification, and blockchain technology.

Enhancing the Section on Security Proofs and Estimates for ML-DSA

The draft asserts that ML-DSA meets the stringent security criterion of existential unforgeability under chosen message attack (EUF-CMA) within the quantum random oracle model (QROM). Despite this, the document lacks formal security proofs or quantitative estimates to substantiate this claim—both of which are pivotal for a thorough evaluation and validation of the algorithm's security. To remedy this, the document should incorporate detailed security proofs or, at a minimum, provide references to existing literature that offers such validation. This addition would significantly aid readers in assessing the security level and margin of ML-DSA, facilitating a more informed comparison with other digital signature schemes.

Refining the Section on Compatibility and Interoperability for ML-DSA

The document currently omits a critical discussion on the compatibility and interoperability of ML-DSA, which are essential for its successful integration with existing standards and protocols. While it provides an in-depth look at ML-DSA's mathematical structure, algorithmic details, and security considerations, it neglects the practical aspects of system integration and the broader network environment. This oversight includes a lack of information on specific requirements, feedback mechanisms, and evaluation processes for compatibility and interoperability, as well as standards, protocols, and support resources. To rectify this, the document should include a detailed examination of ML-DSA's compatibility and interoperability challenges and opportunities.

Assessing Unconventional Attack Vectors on ML-DSA

ML-DSA, a post-quantum digital signature algorithm, is predicated on the module learning with errors (MLWE) problem—a sophisticated extension of the learning with errors (LWE) problem. While MLWE is reputedly resistant to both classical and quantum computational attacks, it would seem that ML-DSA could potentially be vulnerable to certain non-standard or brute force attacks that target specific aspects or frailties of the algorithm or its foundational problem. This section aims to explore such potential attacks and their implications for ML-DSA's security. By scrutinizing these unconventional attack vectors, we can better understand the potential security risks facing ML-DSA and develop strategies to fortify its defenses against such threats.

Exploiting the Compression Function: A conceivable attack vector is the exploitation of the compression function employed in ML-DSA's signing process. This function, which serves to condense the signature size that could otherwise be unwieldy due to MLWE's inherent randomness, inadvertently introduces a degree of information loss and distortion. Consequently, this could diminish the signature's entropy, making it more susceptible to attacks, or induce anomalies that might inadvertently leak details about the secret key or the message. Theoretically, a quantum computer capable of executing quantum search operations could leverage this vulnerability to decipher the secret key or message by sifting through all compatible values or solutions aligned with the compressed signature. Such an attack, while highly speculative and contingent on a quantum computer and algorithm sophisticated enough to manipulate the compression function—and a quantum attack model surpassing QROM—is not beyond the realm of possibility.

Distinguishing Valid from Invalid Signatures: Another potential attack could arise from the error correction mechanism utilized in ML-DSA's verification algorithm. This mechanism, designed to accommodate signatures with inherent noise, introduces additional complexity and potential security concerns. For instance, it could lead to an enlarged signature size, increased verification computation time, or heightened risk of verification failure. Moreover, it might reduce the signature's randomness, rendering it

more predictable. A statistical analysis method capable of differentiating valid signatures from invalid ones could exploit these error correction characteristics to compromise the algorithm's integrity.

Mitigating Chosen Message Attacks and Distinguisher Risks in ML-DSA

The document should address the potential for chosen message attacks (CMA) on ML-DSA, where an adversary could exploit statistical methods to differentiate between valid and invalid signatures. Such an attack threatens the existential unforgeability under chosen message attack (EUF-CMA) security, a cornerstone of digital signature integrity. This security ensures that an adversary, despite having the ability to obtain signatures on chosen messages, cannot forge a signature on a new message, thereby safeguarding against impersonation and unauthorized data alterations.

Distinguisher Concerns: The error correction technique within ML-DSA's verification algorithm could inadvertently serve as a distinguisher. This technique, intended to accommodate signatures with inherent noise, might produce discernible patterns or errors that could expose sensitive information, such as the secret key or the message content. A sophisticated statistical analysis method capable of identifying these patterns could potentially compromise the algorithm's security by extracting the secret key or message from the signature.

Attack Feasibility: Executing such an attack would necessitate not only a statistical method adept at distinguishing between valid and invalid signatures but also a substantial volume of chosen message queries. This requirement could render the attack impractical or easily detectable in many scenarios. Additionally, it would demand a CMA attack model more robust than the QROM model, which ML-DSA is designed to withstand. While the complexity and uncertainty of this attack make it a challenging endeavor, it cannot be deemed impossible.

Conclusion

In light of these considerations, it is imperative for the document to elaborate on the strategies and safeguards implemented within ML-DSA to mitigate the risks posed by CMAs and distinguishers. By proactively addressing these vulnerabilities, the document will reinforce the confidence in ML-DSA's security measures and its resilience against both conventional and unconventional cryptographic threats.

Michael Ravnitzky
Silver Spring, Maryland