Hi all,

This isn’t an attack on NewHope, just another note on how bad it might be if you reuse private keys in the CPA version. It came out of discussion with Mark Marson and Mélissa Rossi. Sorry if this has been discussed before, but I thought it was interesting.

Let Alice’s public key be \((A, A_{s} + e)\), and Bob’s ciphertext be \((B, C)\), so that Alice is extracting a key based on \(C\)-Bs.

Suppose that Bob chooses \(B\) to be a zero divisor. For example, in the NTT domain, \(b\) might be zero in all coefficients except one. Then there are only \(q = 12289\) different possible keys that Alice can extract.

If for some reason Alice confirms the key first, by sending eg hash(key), then Bob can just check all 12289 keys to recover that component of NTT(s). He could even do this for eg 3 nonzero coefficients at a time, at a cost of 12289^3 work per coefficient, which would mean 340-ish chosen messages to recover the key. If the key confirmation is just hash(key), he might be able to accelerate this with a rainbow table. This is stronger than other known attacks, and it’s presumably possible to cut that down a little further by finishing with a lattice reduction or a combinatorial attack.

If Bob confirms the key first with SHA (which is the sane way to do it), then the attack is weaker than other known attacks, since it requires about 1024*12289/2 chosen messages. But if Bob confirms first with a polynomial MAC (eg poly1305 or GCM), and if there is some large-ish data earlier in the key exchange that Bob has control over, then Bob can compute a 12289-block message that verifies will all 12289 possible keys. Then after Alice confirms (or shares some function of the decrypted message with Bob) he can still mount the attack. So this attack wouldn’t work efficiently on TLS 1.3, but it might work with some custom protocol.

This is a good reason in general that you should confirm keys with SHA and not with AEAD (or with the FO transform), and preferably as soon as possible.

It’s easy to come up with countermeasures to this, such as rejecting capsules with too many coefficients equal to 0 (or \(q\)), but of course the only real countermeasure is don’t reuse keys in CPA mode.

Cheers,
— Mike
> It’s easy to come up with countermeasures to this, such as rejecting
> capsules with too many coefficients equal to 0 (or q)

The PQCrypto 2017 Gong--Zhao paper that introduced zero-divisor attacks
(https://eprint.iacr.org/2016/913) also put some effort into trying to obfuscate the attacks in a way that avoids some
simple countermeasures, although it’s not clear how strong the obfuscation is.

> the only real countermeasure is don’t reuse keys in CPA mode.

That has a more obvious effect, yes. Even more effective is to avoid standardizing any sort of "CPA mode" in the first
place.

Question for the people who are proposing both IND-CPA and IND-CCA2 modes for lattice systems: Are there any
publicly verifiable examples of applications where the extra cost of IND-CCA2 security is a significant part of the end
user’s total costs?

---Dan