From: Mike Hamburg <mike@shiftleft.org>
Sent: Monday, September 16, 2019 2:11 PM
To: pqc-comments
Cc: pqc-forum
Subject: ROUND 2 OFFICIAL COMMENT: Round5

Hello Round5 authors,

While poking through the Round5 implementation, I noticed that encryption isn’t constant-time. In particular, sampling the secret with create_secret_vector takes an amount of time that depends on the seed.

This might or might not leak a useful amount of information during keygen and encrypt; with advanced timing attacks a surprising amount of data can leak from a single function call. But where it is really a problem is in re-encrypt with the FO transform. This enables an attacker to determine whether a given message decoded correctly or not.

Of course, a full attack is made more difficult by the presence of the code, but probably not as much as LAC, and it seems dangerous to assume that it is infeasible. At least, this would require another track of cryptanalysis.

Furthermore, it is difficult to re-implement around this issue without significantly hurting performance, because the sampling logic must be functionally equivalent for interoperability.

I suggest that the authors change the specification to use sorting, or on some other operation which is efficient to implement in constant time.

The same issue is present in LAC.

Regards,
— Mike

Kerman, Sara J. (Fed)

From:	Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent:	Monday, September 23, 2019 11:18 AM
To:	pqc-forum
Cc:	pqc-comments
Subject:	Re: ROUND 2 OFFICIAL COMMENT: Round5

On Monday, September 16, 2019 at 7:11:42 PM UTC+1, Mike Hamburg wrote:

> Hello Round5 authors,

Hello Mike,

Sorry for delayed answer; the Round5 team is large and would probably want to address the whole side-channel topic in more detail later (including things like EM and masking). Meanwhile, I'll just offer my response in a personal capacity (errors and omissions here are mine alone).

> While poking through the Round5 implementation, I noticed that encryption isn't constant-time. In particular, sampling the secret with create_secret_vector takes an amount of time that depends on the seed.

Sure, the default implementation of that function is based on rejection sampling and is therefore fundamentally not constant-time. This doesn't necessarily mean that there is leakage of secrets, of course, especially if you enable the CM_CACHE ("cache attack countermeasures") flag in the official implementation (submitted to NIST). Perhaps we (NIST and the community) should move to use some other term where appropriate ("secret-independent timing ?"). For example I recently reviewed the "new" Falcon implementation which is not "truly" constant time but carefully analyses leakage instead.

> This might or might not leak a useful amount of information during keygen and encrypt; with advanced timing attacks a surprising amount of data can leak from a single function call. But where it is really a problem is in re-encrypt with the FO transform. This enables an attacker to determine whether a given message decoded correctly or not.

This is a very good point -- I confess that I didn't think of the FO re-encryption "timing signature" issue too much previously. But one can certainly code a version of "create_secret_vector()" function that is truly constant time while being compatible with the current faster implementations.

I experimentally added this option to my "alternative" r5embed implementation; see the second r5_sparse_tern() implementation in https://github.com/r5embed/r5embed/blob/master/src/r5_ternvec.c After further code review and optimization this type of solution will be probably ported to the main codebase (as an option in the next release).

This sampler operates by maintaining the secret polynomial in bit-sliced coefficient form; since the polynomial coefficients are ternary { -1, 0, +1}, two bits is sufficient per coefficient. The bit-sliced tables are scanned fully very time to avoid cache timing leakage. There just needs to be extra "hi" rounds to guarantee that the Hamming weight reaches target h (can't have loop termination conditionals since those would not be strictly constant time).

The script https://github.com/r5embed/r5embed/blob/master/misc/ct_compute_hi.py is my humble numerical method for estimating the number of rounds, "hi", required to reach weight "h" with probability 1-2^-128 or whatever is considered appropriate -- perhaps one should match the security level due to FO/CCA proofs. Somebody let me know if I messed up the calculation badly.

Unfortunately, this exact sampling method does not extend to the non-ring variants in a compatible way as the XOF ("drbg") gets out of sync between rows in r5_create_secret_mat(). A simple fix is to re-initialize the XOF for each row with a combination of the seed and row number. This would also allow parallelization of this operation (leading to potential speedups) and I am hopeful that my Round5 coauthors will agree to this minor change.

> Of course, a full attack is made more difficult by the presence of the code, but probably not as much as LAC, and it seems dangerous to assume that it is infeasible. At least, this would require another track of cryptanalysis.

Sure, it makes sense to a have this type of a truly constant time code available in case someone wants it. Same as with various protection options available for the hardware implementation.

> Furthermore, it is difficult to re-implement around this issue without significantly hurting performance, because the sampling logic must be functionally equivalent for interoperability.

There is a performance impact for the ring version (aprx. 3x with my initial code), and we'll probably do the minor change required for the non-ring versions to have compatibility between faster and constant-time versions there as well. But I'd still like to keep the fast option available especially for cases where CPA is sufficient.

> I suggest that the authors change the specification to use sorting, or on some other operation which is efficient to implement in constant time.

We've considered sorting but I've been against it; sorting would certainly require more RAM in (embedded) software implementation, a much larger circuit in hardware, and would use more random bits (despite those extra iterations). Right now we can reach speeds and circuit sizes unattainable by other approaches, and I'd personally like to keep that advantage.

Cheers,
- Markku

Dr. Markku-Juhani O. Saarinen <mjos@pqshield.com> PQShield, Oxford UK.