Comment on Post-Quantum Cryptography Requirements and Evaluation Criteria

Stebila, Douglas <stebilad@mcmaster.ca>

Fri 9/16/2016 5:00 PM

To: pqc-comments <pqc-comments@nist.gov>;

Comment on Post-Quantum Cryptography Requirements and Evaluation Criteria

September 16, 2016

Thank you for the opportunity to provide feedback on the upcoming NIST Post-Quantum Cryptography project.

The draft submission requirements set out a clear plan for a transparent process that will lead to the identification of one or more post-quantum technologies with confidence in the result. Please find below comments on six aspects of the submission requirements which I believe will further improve the process.

1) Security levels and refinement

The draft submission requirements ask for parameter sets at five target security levels, the lowest 3 of which are at 64, 80, and 96 bits of quantum security. Reducing the number of target security levels will simplify submissions and allow cryptanalytic research to focus on fewer parameters. I suggest omitting security levels below 128 bits of quantum security.

Given that some submissions will be based on mathematical problems for which cryptanalysis continues to advance, it seems plausible that security levels of parameter sets may evolve, either favourably or unfavourably, during the evaluation process. It would be unfortunate if promising submissions were disqualified because of cryptanalytic advances shaved e.g. 10 bits of security off of a 128-bit-level submission. I suggest that NIST aim to include some room for refinement of parameters in these scenarios, possibly (a) between the first and the second round, or (b) during a round with a minor update, or (c) by incorporating an even higher security level (say, 192-bit quantum security) to provide breathing room.

2) Royalty-free

NIST should require submitters to meet the same intellectual property requirements in the previous SHA-3 competition, namely that if their submission is selected then the submitters agree to place no restrictions on use of the algorithm, making it available on a worldwide, non-exclusive, royalty-free basis.

3) Key establishment / KEM security

Early text from NIST asked which model to use for key establishment; the current submission requirements say IND-CCA security. I support this security requirement. Among other reasons, it is compatible with the notion used in security proofs of TLS [Jager et al. CRYPTO 2012, Krawczyk et al. CRYPTO 2013].

4) Application-level performance

NIST should identify a variety of application scenarios and evaluate submissions in these scenarios. Given that many post-quantum schemes will involve trade-offs between runtime, memory, and communication, evaluating these schemes in

applications may provide surprising results compared to standalone evaluation. TLS should certainly be one of these application scenarios. Given the complexity of adding new algorithms to TLS implementations and fairly benchmarking such a system under realistic loads, NIST may want to apply this evaluation only to second round candidates, and NIST may also want to provide a standard interface and code for integration, rather than requiring submitters to each implement this themselves. For example, it should be possible to modify OpenSSL or another open source TLS implementation to include a ciphersuite that calls in to the NIST-specified PQC API.

5) Transition from traditional to post-quantum algorithms

In the years following this NIST process, standards and implementers will likely gradually transition to post-quantum cryptography, running traditional and post-quantum algorithms side-by-side. While the NIST process rightly focuses on evaluating the security and practicality of post-quantum algorithms, one aspect of practicality is enabling a smooth transition. NIST may want to include as a positive evaluation criteria any characteristics of the scheme which enable a smoother transition, for example a post-quantum scheme that can be easily run in a hybrid mode alongside a tradition algorithm and (safely) share some parameters.

6) Key exchange API

The current C API for key exchange has four functions:

crypto_keyestablishment_initiator_generate
crypto_keyestablishment_responder_generate
crypto_keyestablishment_initiator_recover
crypto_keyestablishment_responder_recover

The responder_generate function outputs a responder public key (ker) and a responder private key (skr), and then the responder_recover function outputs the shared secret (ss). This makes sense for plain Diffie-Hellman protocols, but may not make as much sense for some other protocols. KEMs generically are separated into 3 algorithms: KeyGen, Encaps, Decaps; the Encaps algorithm might be easily separable into responder_generate and responder_recover, but not necessarily.

(For example, in the Peikert [PQCrypto 14], BCNS [IEEE S&P 15], and NewHope [USENIX 2016] ring-LWE key exchange protocols, the responder's secret key is ostensibly s'; in responder_generate, the responder would compute the outgoing message and reconciliation data (u and r in NewHope); in responder_recover, the responder would need to recompute all of these values in order to then derive the shared secret (mu). Of course an implementation could use the skr output of responder_generate to store the shared secret and then just use responder_recover to output that, but this seems to go against the spirit of the API.)

I recommend revising the API for key exchange to have three algorithms: initiator_generate, responder, initiator_recover.

Thank for your consideration.

Sincerely,

Dr Douglas Stebila
Assistant Professor
Department of Computing and Software, Faculty of Engineering
McMaster University
Hamilton, Ontario, Canada