# Comment on Post-Quantum Cryptography Requirements and Evaluation Criteria

D. J. Bernstein <djb@math.uic.edu>

Sat 9/17/2016 11:51 AM

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

Dear Dr. Chen,

Congratulations on initiating a standardization effort for post-quantum cryptography. In general the effort sounds useful and carefully planned, and I look forward to providing whatever assistance I can.

I have comments on several topics, which I have tried to sort here into decreasing order of importance.

1. Quantitatively comparing post-quantum public-key security levels is going to be a nightmare. I see only two ways that submitters a year from now can possibly be "confident that the specified security target is met or exceeded": (1) overkill; (2) overconfidence. Many users will not be satisfied with overkill, and NIST should not encourage overconfidence.

For comparison, let's ignore quantum computers for a moment. Imagine asking someone to choose DSA key sizes to be confident about reaching a pre-quantum 2^256 security target, the largest number in NIST's list of preselected security levels. Should be easy, right?

Here are some costs in the literature for computing multiplicative-group discrete logarithms by NFS index calculus, and thus breaking DSA:

* Exponent 2.080 from 1992 Gordon. What I mean here is that the cost is asymptotically L^(2.080...+o(1)), assuming standard conjectures, with the usual definition of L in NFS.

* Exponent 1.922 from 1993 Schirokauer.

* Exponent 1.901 from 2003 Matyukhin.

* Exponent 1.442 per target from 2006 Commeine--Semaev, after per-prime precomputation with exponent 1.901.

* Exponent 1.231 per target from 2013 Barbulescu, after per-prime precomputation with exponent 1.638, after one-time precomputation with exponent 1.901. (I doubt that 1.231 is optimal.)

All of these are exponents in an unrealistic model of computation where storage and communication are free. In a realistic model, I presume that 1.901 would go up to 1.976 by an adaptation of a factorization algorithm that I published in 2001, and 1.638 would go up to 1.704 for a batch of targets by an adaptation of my 2014 "Batch NFS" paper with Lange. I don't know what would happen to the 1.231.

One might try to argue that the 1.901 and 1.976 have been stable for a decade, and that multi-target attacks don't matter. But NIST's call explicitly, and sensibly, asks for "resistance to multi-key attacks". Multi-target attacks _do_ matter, and the current _exponent_ for the security of DSA against these attacks is only three years old.

Furthermore, a closer look shows that there are many more improvements that reduce concrete attack costs by reducing the "o(1)" quantities. Sloppily replacing o(1) with 0, as NIST apparently did to obtain its current recommendation of 15360 bits for DSA for >=2^256 security, is unjustified. Even for the simple case of single-key attacks, figuring out the o(1) with moderate precision at such large sizes is a difficult research project, extrapolating far beyond experimental ranges. At this point I wouldn't hazard a guess as to whether NIST's 2^256 is an overestimate or an underestimate.

If someone makes enough progress on this research project to announce that the single-key attack cost is actually between 2^245 and 2^250, will NIST withdraw its DSA standard? (What if there is a 2^100 attack against RSA-2048, rather than the commonly quoted 2^112?) If not, then what exactly is the point of asking people to be "confident" that 2^256 is "met or exceeded"? More to the point, experts are _not_ confident, even when multi-target attacks are ignored.

Some people are even more worried by the recent drastic reduction of security levels for pairing-based cryptography, by other index-calculus optimizations. Fortunately, DSA conservatively chose prime fields, avoiding the subfield/automorphism structures exploited in the latest attacks (and in the most complicated previous variants of NFS---one of the warning signals that led ECRYPT to recommend prime fields a decade ago). But the bigger picture is that index calculus is complicated and constantly improving. Would it really be so surprising to see another security loss that _does_ affect DSA?

keylength.com reports some sources making recommendations around the same 15360-bit level recommended by NIST, but also reports Lenstra and Verheul recommending 26000 or 50000 bits. The big difference here is Lenstra and Verheul leaving a security margin in case there is progress in cryptanalysis.

So, with all these numbers in mind, how should we choose DSA key sizes to be "confident" about >=2^256 pre-quantum security? Should we take NIST's overconfident 15360 bits, which definitely flunks the multi-key requirement, and isn't a solid bet even for a single key? How about 26000 bits? 50000 bits? Much bigger, considering Barbulescu's 1.231? What happens if 1.231 is improved further?

What NIST is asking post-quantum submitters to figure out is far more difficult than this DSA example, for several reasons:

* As one would expect given the history of how cryptanalytic effort has been allocated, the security picture for most post-quantum public-key algorithms is even less stable than the security picture for DSA. Example: Current algorithms for the famous shortest-vector problem take (conjecturally) time 2^((0.29...+o(1))d) in dimension

d, a vast improvement compared to 2^((0.40...+o(1))d), the best
result known just a few years ago.

* At this point we have only crude guesses as to the ultimate costs
of different quantum operations. I understand that NIST wants to
define 2^b post-quantum security as 2^b quantum AES computations,
but what is the relative cost of a quantum AES computation and a
lookup in an N-entry table using a quantum index? How does this
depend on N? How much harder is it if the table entries are
themselves quantum?

* I agree with NIST's comment that a 256-bit preimage search with
Grover is actually harder than a 256-bit collision search (even if
qubits are magically as cheap as bits), since Grover parallelizes
poorly. I agree that the optimum value of T*sqrt(S), subject to
ST=2^128 and T>=sqrt(S), is 2^(2*128/3). But T*sqrt(S) is not a
user-comprehensible cost metric, and not a metric for which many
subroutines have been analyzed.

* Algorithm designers benefit tremendously from being able to try out
their algorithms on small-scale and medium-scale problems. An
experiment can show with minimal effort that an algorithm doesn't
produce the desired outputs, or that it doesn't run at the desired
speed. Designers of quantum algorithms don't have this tool yet.

How is a submitter supposed to be confident of reaching, e.g., 2^128
post-quantum security? Submitters will end up making ill-informed random
guesses of which parameters to assign to which security levels. Security
analysis will then throw some submissions into the Scylla of being
"broken", while others will have thrown themselves into the Charybdis of
being "inefficient", even though those submissions might simultaneously
be _more secure and more efficient_ than other submissions that simply
happened to make luckier initial guesses of target security levels.

To summarize: Well-informed long-term security assessments will not
simply supersede obsolete guesswork. The guesswork will continue having
an influence long after it should have been thrown away. This is a
serious failure mode for the evaluation process.

Does "meet or exceed each of five target security strengths" mean that
each submission has to separately target all five levels, giving
designers five chances to be artificially thrown into the rocks? Is it
enough to target just the top level, namely 2^256 pre-quantum security
and 2^128 post-quantum security?

I found it particularly striking that this choice of top target security
level was based on the security achieved by a secret-key system (in this
case AES-256, for some reason ignoring multi-target attacks), rather
than on any attempt to assess what users actually need. I'm reminded of
the ludicrous requirement of 2^512 preimage resistance for SHA-3,
forcing permutation-based SHA-3 submissions such as Keccak to be much
larger and slower than they would otherwise have been.

If a public-key system naturally has 2^2b pre-quantum security and more
than 2^b post-quantum security (I predict that this will be a common
case), then choosing parameters to successfully target 2^256 pre-quantum
security will be overkill for 2^128 post-quantum security---and also

overkill for what users actually need. Why is this a sensible target?

If a public-key system naturally has 2^b post-quantum security and more than 2^2b pre-quantum security (I know one example like this), then choosing parameters to successfully target 2^128 post-quantum security will be overkill for 2^256 pre-quantum security. Why should the designer have to bother evaluating the pre-quantum security level?

Let me suggest a different approach:

* Leave it up to submitters to decide exactly what post-quantum security level to aim for.

* Tell them that security levels <2^64 will be viewed as "breakable", and that security levels >2^128 are unlikely to be viewed as more valuable than security level 2^128, except possibly as a buffer against future cryptanalytic progress.

* Ask them to do the most accurate job that they can of analyzing post-quantum security. Don't ask for fake confidence.

* Scrap the requirement of a pre-quantum security analysis. Users will use cheap ECC hybrids to obtain the pre-quantum security that they want.

Of course, many submissions will do a pre-quantum security analysis and then say "We don't think Grover will reduce the exponent by a factor beyond 2". Is there any problem with this? Should the number of submissions be limited by the current availability of expertise in quantum cryptanalysis?

Followup analysis will improve our understanding of the actual post-quantum security levels of various algorithms, and then NIST will look at a two-dimensional plot of speed vs. security level and decide which options are most interesting.

2. My understanding is that NIST is asking for two specific types of encryption, which NIST labels as "public-key encryption" and "key exchange". This is too narrow: it omits important variants of public-key encryption that people should be allowed to submit.

What I suspect will be most important in the long run is a CCA2-secure "KEM". A KEM can be viewed as a limited form of public-key encryption: the only thing a ciphertext can do is communicate a random session key. As a simple pre-quantum example, Shoup's "RSA-KEM" chooses a random number r mod pq and transmits a session key SHA-256(r) as the ciphertext r^3 mod pq. This is easier to design and analyze and implement than, say, RSA-OAEP.

(Proponents of RSA-OAEP will respond that RSA-OAEP can encrypt a short user-specified message as a ciphertext with the same length as pq. Some applications will notice the bandwidth difference. Obviously NIST should continue to allow public-key encryption as a target.)

One can easily combine a KEM with an authenticated cipher to produce a full-fledged public-key encryption scheme. But this understates the

utility of a KEM: the same session key can be reused to encrypt any
number of messages in both directions, whereas wrapping the KEM in a
public-key encryption scheme hides this functionality. Using this
public-key encryption scheme to encrypt another level of a shared
session key would be frivolous extra complexity. Why not let submitters
simply submit a KEM, skipping the cipher?

Sometimes people reduce the security goals and design KEMs to encrypt
just one message, _without_ chosen-ciphertext security. Here is the
application to keep in mind:

* a client generates a KEM public key;
* a server uses this to transmit a random session key;
* messages are signed by long-term keys for authentication;
* the KEM private key and session key are erased after the session.

This is how New Hope works inside TLS. The signatures (if handled
properly) prevent attackers from choosing any ciphertexts. So why not
let people submit single-message non-CCA2-secure KEMs?

(I don't like the TLS/SIGMA approach to secure sessions: it is
error-prone and excessively complex. This is not a broadcast scenario;
authentication does not require signatures. I prefer the simplicity of
using pure encryption: the long-term key is an encryption key, and the
soon-to-be-erased short-term key is another encryption key. This
requires multiple-message support and CCA2 security, but my current
impression is that this robustness has only minor costs, and I wouldn't
be surprised if the New Hope team decides to move in this direction.
However, if they instead decide that CCA2 security is too expensive,
they shouldn't be rejected for targeting TLS!)

What NIST calls "key exchange" in the draft sounds to me like a poorly
labeled KEM with intermediate security requirements: chosen-ciphertext
security seems to be required, but the interface sounds like it allows
only one message before the key is thrown away. NIST should make clear
if it instead meant a full-fledged KEM allowing any number of
ciphertexts. Either way, NIST should explicitly allow non-CCA2-secure
single-message KEMs such as New Hope.

Calling any of these systems "key exchange" is deceptive for people who
expect "key exchange" to be a drop-in replacement for DH key exchange.
In DH, Alice and Bob both know a shared secret as soon as they see
public keys from Bob and Alice respectively, with no additional
communication. As a concrete example, consider the very small number of
network round trips needed to efficiently authenticate data from hidden
client identities in the "CurveCP" and "Noise_XK" protocols. Here's
Noise_XK using ECC:

* Alice sends her ephemeral public key eG to Bob. New session key:
  hash of ebG, where b is Bob's long-term key.

* Bob responds with his ephemeral public key fG, encrypted and
  authenticated. New session key: hash of ebG and efG.

* Alice sends her long-term public key aG to Bob, encrypted and
  authenticated. New session key: hash of ebG, efG, and afG.

This third packet can already include data authenticated under the last session key, and Bob immediately knows that the data is from Alice. Pure public-key encryption (without signatures) needs another round trip for authentication: Bob has to send data to Alice's long-term public key and see the reply before Bob knows it's Alice talking.

There is one notable post-quantum example of the DH data flow, namely isogeny-based crypto. Security analysis of isogeny-based crypto is clearly in its infancy, but if isogeny-based crypto does survive then the data flow will be an interesting feature. People who submit isogeny-based crypto should be allowed to submit it in a way that makes this data flow clear, rather than having to wrap it in public-key encryption.

I understand that for signatures NIST explicitly decided to disallow one data flow of clear interest, namely stateful signatures, since there is already separate ongoing standardization of stateful hash-based signatures, which are the main reason for interest in this data flow. (The security of hash-based signatures is much better understood than the security of most other public-key systems.) But for encryption I don't see how a similar limitation could be justified.

To summarize, there are at least three clearly different types of data flow of interest: public-key encryption, KEMs, and DH. Within KEMs, there are at least two security targets of interest: passive security for one message, and chosen-ciphertext security for many messages. I suggest that NIST explicitly allow

* all four of these targets;
* also the intermediate type of KEM labeled as "key exchange" in the current draft, if NIST has an application in mind; and
* any further encryption targets that NIST identifies this year as being useful.

I also suggest defining some standard conversions that NIST will apply automatically: e.g., converting a CCA2-secure KEM into CCA2-secure PKE by composition with AES-256-GCM, and converting the other way by encrypting a random 256-bit key. NIST won't want to listen to pointless arguments such as "yes we know we're worse than this PKE but it wasn't submitted to the KEM category" from KEM submitters, and won't want to have to wade through artificially bloated PKE+KEM submissions that are really just one design but want to compete in every category.

3. I have three suggestions regarding terminology.

First, the draft refers frequently to "key exchange", which as noted above ends up deceiving people. I suggest scrapping this terminology in favor of more precise terminology such as KEM and DH. (There's already a NIST standard introducing relevant names such as "C(0,2)", but I don't know how many people are familiar with these names.)

Second, the draft uses "forward secrecy" (even worse, "perfect forward secrecy") to refer to the obvious security benefits of erasing a private key. This terminology also ends up deceiving people. Last week I was speaking with a banker who thought that TLS's "perfect forward secrecy" would protect his communications against future quantum computers. I suggest avoiding this terminology and instead saying something like "Fast key generation is useful for high-frequency generation of new key pairs, which in turn allows each private key to be promptly erased."

Third, the draft says that post-quantum cryptography is "also called quantum-resistant or quantum-safe cryptography", and makes occasional use of the "quantum-resistant" terminology after that. It's true that Google finds some hits for "quantum-resistant cryptography" and "quantum-safe cryptography" (1630 and 4340, compared to 47100 for "post-quantum cryptography"), but I'm not at all sure that the people using these terms are using them with the same meaning as post-quantum cryptography, and I predict that users seeing algorithms labeled as "resistant" and "safe" will be deceived into thinking that we are more confident than can be scientifically justified.

As a concrete example, research by Makarov et al. has convincingly shown that ID Quantique's QKD products are breakable at low cost, but one of the top hits for "quantum-safe cryptography" appears to refer to those products as "provably secure quantum-safe" cryptography. I presume that snake-oil peddlers choose this terminology precisely because it is deceptive; for the same reason, I suggest that NIST avoid the terminology. As an analogy, FIPS 186-4 has the sensible title "Digital signature standard", not "Safe digital signature standard" or "Attack-resistant digital signature standard".

4. Requiring submissions to be sent by postal mail will penalize some submitters for reasons that are not connected to the quality of their submissions. For example, as far as I know, the lowest-cost way to guarantee two-day delivery of a 1kg package from Bangalore to NIST is a Fedex International Priority Pak, which costs half a week's salary for a typical Indian professor.

I understand that NIST needs a signed printed statement regarding patents etc., but this statement is not urgent: it can be sent by mail later, or hand-delivered to NIST at the first workshop.

On a related note, requiring fax numbers and telephone numbers is silly.

5. The draft needs a general round of proofreading. For example, Wiener is not "Weiner", the JoC97 link does not work, and 4.B.4 is incomplete.

---D. J. Bernstein