Post-Quantum Cryptography Standardization
Historical FAQs

Date Moved to Archive	Question #	Old Questions and/or Answers	Reason for Archive

9/5/17	15	**Q15: How does a submission obtain secure randomness?**
		
		A15: The function randombytes() will be available to the submitters. This is a function from the SUPERCOP test environment and should be used to generate seed values for an algorithm.
		
		For functional and timing tests a deterministic generator is used inside randombytes() to produce the seed values. If security testing is being done simply substitute calls to a true hardware RBG inside randombytes().
		
		Function prototype for randombytes() is:
		
		`// The xlen parameter is in bytes`
		`void randombytes(unsigned char *x,unsigned long long xlen)`
		
		The following demonstrate the use of the KAT and non-KAT versions of the functions to generate a key pair for encryption:
		
		`int crypto_encrypt_keypair_KAT(`
		`              unsigned char *pk,`
		`              unsigned char *sk,`
		`              const unsigned char *randomness`
		`         )`
		
		`int crypto_encrypt_keypair(unsigned char *pk, unsigned char *sk)`
		`{`
		`      unsigned char pk[CRYPTO_PUBLICKEYBYTES];`
		`      unsigned char sk[CRYPTO_SECRETKEYBYTES];`
		`      unsigned char seed[CRYPTO_RANDOMBYTES];`
		
		`      randombytes(seed, CRYPTO_RANDOMBYTES);`
		`      crypto_encrypt_keypair_KAT(pk, sk, seed);`
		`}`
			New answer removes last paragraph with code.

Date Moved to Archive	Question #	Old Questions and/or Answers	Reason for Archive
8/10/17	3	**Q3: What exceptions, if any, are there to the requirement for ANSI C source code? In particular, may C++ code or assembly optimizations be used?** **A3:** For both the mandatory reference implementation and the mandatory optimized implementations, all new code written by submitters for the submission should be written in as ANSI C-like a manner as possible, subject to some caveats. In particular, implementations that use NTL (see Question and Answer 16 for details on the use of third-party open source libraries) are necessarily allowed to be written in C++. However, the original and new code in this submission must still be as ANSI C-like as possible, and should only use C++ functionality where absolutely required in order to use NTL. In particular, as with code using any other third-party open source code, the submission must contain build scripts for both Windows and Linux that compile properly on the Intel x64 reference platform using version 6.4.0 of the GNU Compiler Collection (GCC). Furthermore, while submitters may not write their own new and original assembly (including inline assembly) code for either the mandatory referenced implementation or the mandatory optimized implementation, we are allowing the use of third party open-source libraries that themselves rely on assembly optimizations, subject to the constraints described in Question and Answer 16. Any optional additional implementations that submitters wish to include are subject to no constraints at all regarding the language and platform.	Question reworded and new clarified answer provided
8/10/17	4	**Q4: Will NIST consider platforms other than the "NIST PQC Reference Platform" when evaluating submissions?** **A4:** The reference platform was defined in order to provide a common and ubiquitous platform to verify the execution of the code provided in the submissions. NIST will include performance metrics from a variety of platforms in our evaluation, including: 64-bit "desktop/server class," 32-bit "mobile class," microcontrollers (32-, 16-, and where possible, 8-bit), as well as hardware platforms (e.g., FPGA). Submitters are encouraged to provide additional implementations for these platforms if possible. The reference platform should be treated as a single core machine. If an algorithm can make particular use of multiple cores or vector instructions, submitters are encouraged to provide additional implementations for these platforms.	New answer contains additional info

Date Moved to Archive	Question #	Old Questions and/or Answers	Reason for Archive
8/10/17	16	**Q16: Can third party open-source code be used in submissions?** **A16.** In both the mandatory reference implementation and the mandatory optimized implementation, submissions may use NTL Version 10.5.0 (http://www.shoup.net/ntl/download.html), GMP Version 6.1.2 (https://gmplib.org), and OpenSSL Version 1.10f (https://www.openssl.org/source). Submitters may assume that these libraries are installed on the reference platform and do not need to provide them along with their submissions. If a submitter wishes to use a third-party open source library other than the ones specified above, they must send a request to NIST at pqc-comments@nist.govby September 1st, 2017, with the name of the library and a link to the primary website hosting it from which it may be downloaded. NIST will either approve or deny this request within 2 weeks of receiving it. Should a request be approved, it will be added to the above list of acceptable third-party open source libraries provided in this FAQ. ….. …..	New answer adds library information in 1st paragraph
8/3/17	3	**Q3: Does the requirement for ANSI C source code preclude the use of assembly language optimizations?** **A3:** The optimized code required as part of the submission package should be ANSI C with no assembly (this includes inline assembly). This code is meant to be portable. If significant optimizations can be made with assembly, then it can be included as an additional implementation and discussed in the performance analysis.	Question reworded and new clarified answer provided

Date Moved to Archive	Question #	Old Questions and/or Answers	Reason for Archive
8/3/17	16	**Q16: Can third party open-source code be used in submissions?** **A16:** In short, they may be used, with the following caveats. 1. The library source code should be integrated into the submission package in a self-contained manner. This means that the submission package should contain build scripts which will allow for seamless "one-stop" building of the submitter's original code and all dependencies. For example, on a Linux platform, it should require no more work to build the than running the standard > ./configure [--options] > make > make install succession of commands. The build process should not require the installation of any new libraries that are not contained in the submission package. Separate build scripts should be included for the reference Windows platform and reference Linux platform that work using the GCC Compiler Collection (or ports thereof) and related tools as well as any platform-specific commands required. 2. As part of the written submission, the submitter shall describe in their own words the functionalities provided by any algorithms from third-party open-source libraries that are used in the implementations. **3.** The submitter is responsible for ensuring that they abide by all requirements of the license (if any) under which said library has been released.	Added two new paragraphs at start of answer and clarified the rest of the answer
4/26/17	15	**Q15: How does a submission obtain secure randomness?** **A15:** The function randombytes() will be available to the submitters. This is a function from the SUPERCOP test environment and should be used to generate seed values for an algorithm. Randombytes should only be used to seed a NIST-approved DRBG. As stated in the call for algorithms, the DRBG should be NIST approved. If a non-approved DRBG is used "the submitter shall provide an explanation for why a NIST-approved primitive would not be suitable." The length of the random value obtained from randombytes() should be selected to match one of the security categories in the call for algorithms. That is, if the call to generate a key pair is from category 1 the randomness value should be 192 bits (24 bytes), if the call is from category 2 or 3 it should be 256 bits (32 bytes) and if it is from category 4 or 5 it should be 320 bits (40 bytes). The DRBG will be used to expand that if necessary. For functional and timing tests a deterministic generator is used inside randombytes() to produce the seed values. If security testing is being done simply substitute calls to a true hardware RBG inside randombytes(). **…..** **…..**	Sentence "Randombytes should only be used…" removed from first paragraph.

Date Moved to Archive	Question #	Old Questions and/or Answers	Reason for Archive
4/11/17	15	**Q15: How does a submission obtain secure randomness?** **A15:** The function randombytes() will be available to the submitters. This is a function from the SUPERCOP test environment and should be used to generate seed values for an algorithm. If the algorithm needs additional randomness beyond the seed value a NIST-approved DRBG should be used. As stated in the call for algorithms, the DRBG should be NIST approved. If a non-approved DRBG is used "the submitter shall provide an explanation for why a NIST-approved primitive would not be suitable." The length of the random value obtained from randombytes() should be selected to match one of the security categories in the call for algorithms. That is, if the call to generate a key pair is from category 1 the randomness value should be 192 bits (24 bytes), if the call is from category 2 or 3 it should be 256 bits (32 bytes) and if it is from category 4 or 5 it should be 320 bits (40 bytes). The DRBG will be used to expand that if necessary. For functional and timing tests a deterministic generator is used inside randombytes() to produce the seed values. If security testing is being done simply substitute calls to a true hardware RBG inside randombytes().	Changes made to first paragraph only.
12/29/2016	OLD Q	**Q: Why are hash functions assigned fewer bits of quantum security than classical security?** **A:** Bernstein[1] is widely cited as demonstrating that the most efficient quantum algorithm for finding hash collisions is the classical algorithm given by Van Oorschot and Wiener[2] . NIST believes this analysis is correct. Nonetheless, NIST's security goal, that schemes claiming s bits of quantum security be at least as secure against cryptanalysis as a 2s bit block cipher leads to differing definitions for quantum and classical security. In particular, quantum search for a 2s bit key does not parallelize well. It is NIST's judgement that, since cryptanalysis in the real world tends to be most successful when it can take advantage of highly parallel implementations for attacks, finding collisions in a 2s bit hash function must be considered easier than searching for the key of a 2s-bit block cipher, even in a world with ubiquitous quantum computing. NIST therefore assigns fewer than s bits of quantum security against collision to 2s bit hash functions. [1] *Daniel J. Bernstein, Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?* *https://cr.yp.to/hash/collisioncost-20090517.pdf* [2] *Paul C. van Oorschot, Michael Wiener, Parallel collision search with cryptanalytic applications, Journal of Cryptology 12 (1999) http://people.scs.carleton.ca/~paulv/papers/JoC97.pdf*	Question removed from FAQ

Date Moved to Archive	Question #	Old Questions and/or Answers	Reason for Archive
11/30/2016	OLD Q	**Q: What is the rationale to convert time and space complexity of known attacks into a single number for quantum and classical security?** **A:** NIST's definition of s bits of quantum security is "as hard to break as a block cipher with a 2s bit key, assuming a relatively efficient and scalable quantum computing architecture is available." According to the analysis of Zalka[1] the best generic quantum attack on a 2s-bit block cipher requires a quantum circuit with depth*(squareroot (space)) proportional 2^s. This would suggest that quantum security should be defined as the minimum possible value of log(depth*(squareroot (space))) plus a constant (to put the quantum security of AES 128 at precisely 64 bits of quantum security,) accross all quantum and classical algorithms. This formula should only be taken as a rough guess, though, as there are additional factors to consider: Extremely serial and extremely parallel attacks are likely to be of limited practical relevance, even if the above formula rates them as most efficient. Likewise, even under the assumption that a relatively scalable and efficient quantum computing architecture is available, it is still likely that purely classical algorithms will be easier to implement than the formula suggests, and quantum algorithms that, unlike parallel versions of Grover's algorithms, cannot be divided into small, unentangled, subcircuits, will be harder to implement than the formula suggests. NIST plans to take these practical considerations into account when making its evaluations. Similarly, NIST's definition of s bits of classical security is "as hard to break as a block cipher with an s bit key, assuming quantum computers are not available." This suggests that classical security should be estimated as the minimum value of log(depth*space) plus a constant, over all classical attack algorithms. [1] *Christof Zalka, Grover's quantum searching algorithm is optimal, Physical Review A, 60:2746-2751, 1999* *http://arxiv.org/abs/quant-ph/9711070*	Question removed from FAQ