My personal experience with the NIST PQC "competition"

Peter Schwabe

November 18, 2021
This talk is biased.
This talk is biased.

- Possible personal gain from (NIST) standardization of certain PQC schemes
This talk is biased.

- Possible personal gain from (NIST) standardization of certain PQC schemes
- I’m an “ivory tower” academic
- Luxury of ignoring certain real-world issues
- Still aim for real-world impact
This talk is biased.

- Possible personal gain from (NIST) standardization of certain PQC schemes
- I’m an “ivory tower” academic
- Luxury of ignoring certain real-world issues
- Still aim for real-world impact
- Various thoughts in this talk are meant as constructive criticism
This talk is biased.

- Possible personal gain from (NIST) standardization of certain PQC schemes
- I’m an “ivory tower” academic
- Luxury of ignoring certain real-world issues
- Still aim for real-world impact
- Various thoughts in this talk are meant as constructive criticism
- I’ll try to avoid finger-pointing
Why (I think) I was invited

• Involved in 7 submissions to the NIST PQC project
• All 7 advanced to round 2
• 5 advanced to round 3 (4 finalists, 1 alternate)
Why (I think) I was invited

- Involved in 7 submissions to the NIST PQC project
- All 7 advanced to round 2
- 5 advanced to round 3 (4 finalists, 1 alternate)
- ERC StG project “EPOQUE – Engineering post-quantum cryptography”
  - Efficient and secure implementation of PQC
  - Design of post-quantum protocols
KEMs

- Alkim, Avanzi, Bos, Ducas, de la Piedra, Pöppelmann, Schwabe, Stebila, Albrecht, Orsini, Osheter, Paterson, Peer, and Smart: NewHope
KEMs

- Alkim, Avanzi, Bos, Ducas, de la Piedra, Pöppelmann, Schwabe, Stebila, Albrecht, Orsini, Osheter, Paterson, Peer, and Smart: NewHope
- Avanzi, Bos, Ducas, Kiltz, Lepoint, Lyubashevsky, Schanck, Schwabe, Seiler, and Stehlé: Kyber
KEMs

• Alkim, Avanzi, Bos, Ducas, de la Piedra, Pöppelmann, Schwabe, Stebila, Albrecht, Orsini, Osheter, Paterson, Peer, and Smart: **NewHope**

• Avanzi, Bos, Ducas, Kiltz, Lepoint, Lyubashevsky, Schanck, Schwabe, Seiler, and Stehlé: **Kyber**

• Chen, Danba, Hoffstein, Hülsing, Rijneveld, Saito, Schanck, Schwabe, Whyte, Xagawa, Yamakawa, and Zhang: **NTRU**
7 submissions

KEMs

- Alkim, Avanzi, Bos, Ducas, de la Piedra, Pöppelmann, Schwabe, Stebila, Albrecht, Orsini, Osheter, Paterson, Peer, and Smart: **NewHope**
- Avanzi, Bos, Ducas, Kiltz, Lepoint, Lyubashevsky, Schanck, Schwabe, Seiler, and Stehlé: **Kyber**
- Chen, Danba, Hoffstein, Hülsing, Rijneveld, Saito, Schanck, Schwabe, Whyte, Xagawa, Yamakawa, and Zhang: **NTRU**
Signatures

• Chen, Hülsing, Rijneveld, Samardjiska, and Schwabe: MQDSS
Signatures

- Chen, Hülsing, Rijneveld, Samardjiska, and Schwabe: **MQDSS**
- Bai, Ducas, Kiltz, Lepoint, Lyubashevsky, Schwabe, Seiler, and Stehlé: **Dilithium**
Signatures

• Chen, Hülsing, Rijneveld, Samardjiska, and Schwabe: MQDSS
• Bai, Ducas, Kiltz, Lepoint, Lyubashevsky, Schwabe, Seiler, and Stehlé: Dilithium
• Hülsing, Aumasson, Bernstein, Beullens, Dobraunig, Eichlseder, Fluhrer, Gazdag, Kampanakis, Kölbl, Lange, Lauridsen, Mendel, Niederhagen, Rechberger, Rijneveld, Schwabe, and Westerbaan: SPHINCS+
Cryptographic Engineering

Designing and building cryptographic systems

The traditional approach

- Cryptography
  - Mathematics
  - Algorithms

- Implementation
  - Software
  - Hardware

- Scheme

- System
Designing and building cryptographic systems

A holistic approach
Designing and building cryptographic systems

A holistic approach
Examples, part I

Dilithium vs. Dilithium-G

• Dilithium uses uniform sampling
• Smaller signatures possible with discrete Gaussian
Dilithium vs. Dilithium-G

- Dilithium uses uniform sampling
- Smaller signatures possible with discrete Gaussian
- Hard to implement securely ("constant-time")
- Harder to test if sampling is correct
Dilithium vs. Dilithium-G

- Dilithium uses uniform sampling
- Smaller signatures possible with discrete Gaussian
- Hard to implement securely ("constant-time")
- Harder to test if sampling is correct

Parallel SHAKE in Kyber

- Kyber-768 needs to sample 9 polynomials uniformly mod $q$
- Use rejection sampling on SHAKE output
- 9 independent invocations of SHAKE (more output required!)
Dilithium vs. Dilithium-G

- Dilithium uses uniform sampling
- Smaller signatures possible with discrete Gaussian
- Hard to implement securely ("constant-time")
- Harder to test if sampling is correct

Parallel SHAKE in Kyber

- Kyber-768 needs to sample 9 polynomials uniformly mod $q$
- Use rejection sampling on SHAKE output
- 9 independent invocations of SHAKE (more output required!)
- Can vectorize SHAKE: much faster
- Easy to implement with low memory footprint on $\mu$c
Avoid fixed-weight sampling

• Fixed-weight sampling lowers failure prob. for lattice-based KEMs
• Avoid it in NewHope, Kyber, and NTRU-HRSS
Avoid fixed-weight sampling

- Fixed-weight sampling lowers failure prob. for lattice-based KEMs
- Avoid it in NewHope, Kyber, and NTRU-HRSS
- Typically computes secret permutation (e.g., through sorting)
- Problem: *simple* implementations leak through timing
Avoid fixed-weight sampling

- Fixed-weight sampling lowers failure prob. for lattice-based KEMs
- Avoid it in NewHope, Kyber, and NTRU-HRSS
- Typically computes secret permutation (e.g., through sorting)
- Problem: *simple* implementations leak through timing

Tweakable hash functions

- Software of SPHINCS\(^+\) modularized
- Abstraction layer: “tweakable hash functions”
- Idea: Use also to modularize the proof
Part I

Software implementations
NIST PQC software requirements

• Reference implementation “to promote understanding of how the submitted algorithm may be implemented”

• Optimized implementation “targeting the Intel x64 processor (a 64-bit implementation), is intended to demonstrate the performance of the algorithm”
NIST PQC software requirements

- Reference implementation “to promote understanding of how the submitted algorithm may be implemented”
- Optimized implementation “targeting the Intel x64 processor (a 64-bit implementation), is intended to demonstrate the performance of the algorithm”
- “Both implementations shall consist of source code written in ANSI C”
- Some permitted external dependencies: NTL, GMP, OpenSSL, XKCP
NIST PQC software requirements

- Reference implementation “to promote understanding of how the submitted algorithm may be implemented”
- Optimized implementation “targeting the Intel x64 processor (a 64-bit implementation), is intended to demonstrate the performance of the algorithm”
- “Both implementations shall consist of source code written in ANSI C”
- Some permitted external dependencies: NTL, GMP, OpenSSL, XKCP
- API following SUPERCOP API:
  - signing produces **signed message** instead of signature
  - randomness sampled internally
  - no batching of crypto operations
“NISTPQC, despite being an important and timely project, has produced the largest regression ever in the quality of cryptographic software. This will not be easy to fix.”

—Daniel J. Bernstein, Oct. 2018
Some nasty examples

- Function returning a pointer to local stack variable
Some nasty examples

- Function returning a pointer to local stack variable
- ... but that function was never used
• Function returning a pointer to local stack variable
• . . . but that function was never used
• Compression of keys, etc. not implemented
Some nasty examples

- Function returning a pointer to local stack variable
- ... but that function was never used
- Compression of keys, etc. not implemented
- FO transform without the public key
  - FO transform requires public key for re-encryption
  - Public key is not passed as argument to decaps
Some nasty examples

- Function returning a pointer to local stack variable
- . . . but that function was never used
- Compression of keys, etc. not implemented
- FO transform without the public key
  - FO transform requires public key for re-encryption
  - Public key is not passed as argument to decaps
  - Typical solution: store pk as part of sk
  - Here: read sk out of bounds, assume that pk is in memory just behind
- Out-of-bound reads, need zeroes in memory
- Out-of-bound writes, hope for the best
Some nasty examples

• Function returning a pointer to local stack variable
• ... but that function was never used
• Compression of keys, etc. not implemented
• FO transform without the public key
  • FO transform requires public key for re-encryption
  • Public key is not passed as argument to decaps
  • Typical solution: store pk as part of sk
  • Here: read sk out of bounds, assume that pk is in memory just behind
• Out-of-bound reads, need zeroes in memory
• Out-of-bound writes, hope for the best
Some more subtle examples

- Bug in noise sampling of Dilithium
- Bug in noise sampling of Falcon
Some more subtle examples

- Bug in noise sampling of Dilithium
- Bug in noise sampling of Falcon
- Decaps returning -1 on failure in Kyber
Some more subtle examples

- Bug in noise sampling of Dilithium
- Bug in noise sampling of Falcon
- Decaps returning -1 on failure in Kyber
- Timing leaks in many implementations
  - Often timing-attack resistance not claimed
  - Sometimes violating constant-time claims
Underlying reasons (?)

• Different understanding of “cryptographic software”
  • production quality, ready for real-world deployment
  • proof-of-concept, needs to just work
Underlying reasons (?)

- Different understanding of “cryptographic software”
  - production quality, ready for real-world deployment
  - proof-of-concept, needs to just work
- C make it easy to get things wrong
- ANSI C (i.e., without intrinsics) not useful to illustrate performance
Underlying reasons (?)

- Different understanding of “cryptographic software”
  - production quality, ready for real-world deployment
  - proof-of-concept, needs to just work
- C make it easy to get things wrong
- ANSI C (i.e., without intrinsics) not useful to illustrate performance
- Multiple teams without an “implementor”
Underlying reasons ()

- Different understanding of “cryptographic software”
  - production quality, ready for real-world deployment
  - proof-of-concept, needs to just work
- C make it easy to get things wrong
- ANSI C (i.e., without intrinsics) not useful to illustrate performance
- Multiple teams without an “implementor”
- New schemes: no existing test vectors, yet
• Different understanding of “cryptographic software”
  • production quality, ready for real-world deployment
  • proof-of-concept, needs to just work
• C make it easy to get things wrong
• ANSI C (i.e., without intrinsics) not useful to illustrate performance
• Multiple teams without an “implementor”
• New schemes: no existing test vectors, yet
• Unnecessarily hard to get it right
Some ideas how to do better

**Observation:** The more time you put into designing an exam, the less time you’ll need grading it.
Some ideas how to do better

Observation: The more time you put into designing an exam, the less time you’ll need grading it.

Invest more time:

• Provide some example code (simple, possibly pre-quantum scheme)
• Provide a (black-box) test harness
• Interactive system with feedback from remote test harness
Some ideas how to do better

**Observation:** The more time you put into designing an exam, the less time you’ll need grading it.

**Invest more time:**

- Provide some example code (simple, possibly pre-quantum scheme)
- Provide a (black-box) test harness
- Interactive system with feedback from remote test harness
- Provide access to benchmarking platforms (for optimized code)
PQClean

• Joint work with
  Matthias Kannwischer, Joost Rijneveld, John Schanck, Douglas Stebila, Thom Wiggers

• GitHub repo with extensive CI to ensure “clean” implementations:
  https://github.com/PQClean/PQClean

• Updated goal: focus on round-3 candidates

• Make it easy to use in other projects

• Make it easy to use as starting point for optimization

• Longer-term, if there is interest:
  • implementations with architecture-specific optimizations (WIP)
  • implementations in other languages?
• Joint work with
  Matthias Kannwischer, Joost Rijneveld, John Schanck, Douglas Stebila, Thom Wiggers
• GitHub repo with extensive CI to ensure “clean” implementations:
  https://github.com/PQClean/PQClean
• Goal: collect “clean C” code of all round-2 candidates
• Updated goal: focus on round-3 candidates
• Make it easy to use in other projects
• Make it easy to use as starting point for optimization
PQClean

• Joint work with Matthias Kannwischer, Joost Rijneveld, John Schanck, Douglas Stebila, Thom Wiggers

• GitHub repo with extensive CI to ensure “clean” implementations: https://github.com/PQClean/PQClean

• Goal: collect “clean C” code of all round-2 candidates

• Updated goal: focus on round-3 candidates

• Make it easy to use in other projects

• Make it easy to use as starting point for optimization

• Longer-term, if there is interest:
  • implementations with architecture-specific optimizations (WIP)
  • implementations in other languages?
The definition of “clean”

• Code is valid C99
• Passes functional tests
• API functions do not write outside provided buffers
• API functions do not need pointers to be aligned
• Compiles with -Wall -Wextra -Wpedantic -Werror with gcc and clang
• Compiles with /W4 /WX with MS compiler
• Consistent test vectors across runs
• Consistent test vectors on big-endian and little-endian machines
• Consistent test vectors on 32-bit and 64-bit machines
The definition of “clean”

• No errors/warnings reported by valgrind
• No errors/warnings reported by address sanitizer
• No errors/warnings reported by undefined-behavior sanitizer
• Only dependencies:
  • fips202.c
  • sha2.c
  • aes.c
  • randombytes.c
The definition of “clean”

• API functions return 0 on success, negative on failure
• No dynamic memory allocations
The definition of “clean”

- API functions return 0 on success, negative on failure
- No dynamic memory allocations
- Builds under Linux, MacOS, and Windows without warnings
- All exported symbols are namespaced with `PQCLEAN_SCHEMENAME_`
- Each implementation comes with license and meta information in `META.yml`
The definition of “clean” – the controversial bits

- No variable-length arrays (required to build under Windows)
The definition of “clean” – the controversial bits

- No variable-length arrays (required to build under Windows)
- Separate subdirectories (without symlinks) for each parameter set of each scheme
The definition of “clean” – the controversial bits

- No variable-length arrays (required to build under Windows)
- Separate subdirectories (without symlinks) for each parameter set of each scheme
- `#ifdefs` only for header encapsulation
The definition of “clean” – the controversial bits

• No variable-length arrays (required to build under Windows)
• Separate subdirectories (without symlinks) for each parameter set of each scheme
• #ifdefs only for header encapsulation
• No stringification macros
The definition of “clean” – the controversial bits

• No variable-length arrays (required to build under Windows)
• Separate subdirectories (without symlinks) for each parameter set of each scheme
• `#ifdefs` only for header encapsulation
• No stringification macros
• Dealing with controversial warnings (unary minus on unsigned integers)
The definition of “clean” – the controversial bits

• No variable-length arrays (required to build under Windows)
• Separate subdirectories (without symlinks) for each parameter set of each scheme
• #ifdefs only for header encapsulation
• No stringification macros
• Dealing with controversial warnings (unary minus on unsigned integers)
• Argument names consistent between .h and .c files
Limitations and lessons learned

• MS compiler does not support C99 → no variable-length arrays
Limitations and lessons learned

- MS compiler does not support C99 → no variable-length arrays
- Public CI services impose serious limitations through timeouts
- Not yet testing for "constant-time" behavior
- Could use valgrind with uninitialized secret data (dynamic)
  Alternative: ct-verif (static)
- Tricky to even find the right definition(s)
- Valgrind does not work with environments running on qemu
Limitations and lessons learned

- MS compiler does not support C99 → no variable-length arrays
- Public CI services impose serious limitations through timeouts
- Not yet testing for “constant-time” behavior
  - Could use valgrind with uninitialized secret data (dynamic)
  - Alternative: ct-verif (static)
Limitations and lessons learned

- MS compiler does not support C99 → no variable-length arrays
- Public CI services impose serious limitations through timeouts
- Not yet testing for “constant-time” behavior
  - Could use valgrind with uninitialized secret data (dynamic)
  - Alternative: \texttt{ct-verif} (static)
  - Tricky to even find the right definition(s)
Limitations and lessons learned

- MS compiler does not support C99 → no variable-length arrays
- Public CI services impose serious limitations through timeouts
- Not yet testing for “constant-time” behavior
  - Could use valgrind with uninitialized secret data (dynamic)
  - Alternative: `ct-verif` (static)
    - Tricky to even find the right definition(s)
- Valgrind does not work with environments running on qemu
• Main benchmark provider (?): eBACS/SUPERCOP
• Huge crypto benchmarking project by Bernstein and Lange
• See https://bench.cr.yp.to
Benchmarking software in NISTPQC

• Main benchmark provider (?): eBACS/SUPERCOP
• Huge crypto benchmarking project by Bernstein and Lange
• See https://bench.cr.yp.to
• Benchmarks on Cortex-M4: pqm4
• Joint work with Kannwischer, Petri, Rijneveld, and Stoffelen
• See https://github.com/mupq/pqm4
Benchmarking software in NISTPQC

- Main benchmark provider (?): eBACS/SUPERCOP
- Huge crypto benchmarking project by Bernstein and Lange
- See https://bench.cr.yp.to
- Benchmarks on Cortex-M4: pqm4
- Joint work with Kannwischer, Petri, Rijneveld, and Stoffelen
- See https://github.com/mupq/pqm4
Problem with this approach

- Benchmarks are not independent
- Massive overlap between sets of submitters and “benchmarkers”
- Unfair advantage for those submitters?
- Possibly more fair approach:
  - Fix a few target microarchitectures
  - Publish benchmarking software long ahead of time
  - If needed, give submitters access to platforms for optimizations
Beyond C

• How about Sage/Python for reference implementation?
  • Easier to write, easier to read
  • Less temptation to use in benchmarking
  • Downside: Less useful as starting point for optimization

• How about using Rust?
  • Harder to write
  • Easier to write safely
  • Easier to write correctly

• How about using hacspec?
  • Subset of Rust; see https://hacspec.github.io/
  • "A specification language for crypto primitives and more"
  • Interfaces to verification frameworks
  • Work towards goal of high-assurance crypto
• How about Sage/Python for reference implementation?
  • Easier to write, easier to read
  • Less temptation to use in benchmarking
  • Downside: Less useful as starting point for optimization

• How about using Rust?
  • Harder to write
  • Easier to write safely
  • Easier to write correctly
Beyond C

• How about Sage/Python for reference implementation?
  • Easier to write, easier to read
  • Less temptation to use in benchmarking
  • Downside: Less useful as starting point for optimization

• How about using Rust?
  • Harder to write
  • Easier to write safely
  • Easier to write correctly

• How about using hacspec?
  • Subset of Rust; see https://hacspe.github.io/
  • “A specification language for crypto primitives and more”
  • Interfaces to verification frameworks
  • Work towards goal of high-assurance crypto
Part II

Random thoughts
The pqc-forum mailing list

- A lot of very valuable, constructive, open, scientific discussion

- but also a lot of
- big egos
- personal attacks
- “sniping”

- Being loud does not necessarily correlate with being right (but also not with being wrong)

- Essentially all comments are subjective/biased:
  - Submitters have something to gain/lose
  - Companies have started implementing/investing

- Selection bias (optimize own scheme, attack low-hanging fruits)

- Very incomplete picture (e.g., for SCA)
The pqc-forum mailing list

• A lot of very valuable, constructive, open, scientific discussion
• . . . but also a lot of
  • big egos
  • personal attacks
  • “sniping”
The pqc-forum mailing list

• A lot of very valuable, constructive, open, scientific discussion
• ... but also a lot of
  • big egos
  • personal attacks
  • “sniping”
• Being loud does not necessarily correlate with being right
The pqc-forum mailing list

- A lot of very valuable, constructive, open, scientific discussion
- . . . but also a lot of
  - big egos
  - personal attacks
  - “sniping”
- Being loud does not necessarily correlate with being right (but also not with being wrong)
The pqc-forum mailing list

- A lot of very valuable, constructive, open, scientific discussion
- ... but also a lot of
  - big egos
  - personal attacks
  - "sniping"
- Being loud does not necessarily correlate with being right (but also not with being wrong)
- Essentially all comments are subjective/biased:
  - Submitters have something to gain/lose
  - Companies have started implementing/investing
The pqc-forum mailing list

• A lot of very valuable, constructive, open, scientific discussion
• . . . but also a lot of
  • big egos
  • personal attacks
  • “sniping”
• Being loud does not necessarily correlate with being right (but also not with being wrong)
• Essentially all comments are subjective/biased:
  • Submitters have something to gain/lose
  • Companies have started implementing/investing
  • Selection bias (optimize own scheme, attack low-hanging fruits)
  • Very incomplete picture (e.g., for SCA)
NIST approach: relate to AES/SHA2/SHA3 security

“computational resources may be measured using a variety of different metrics (e.g., number of classical elementary operations, quantum circuit size, etc.). In order for a cryptosystem to satisfy one of the above security requirements, any attack must require computational resources comparable to or greater than the stated threshold, with respect to all metrics that NIST deems to be potentially relevant to practical security.

NIST intends to consider a variety of possible metrics, reflecting different predictions about the future development of quantum and classical computing technology. NIST will also consider input from the cryptographic community regarding this question.”
Fix the attack-cost metric

- Hard to choose parameters for “unknown target”
- Proposals’ security claims use different metrics
- Hard to separate metrick discussion from scheme discussion
Fix the attack-cost metric

- Hard to choose parameters for “unknown target”
- Proposals’ security claims use different metrics
- Hard to separate metrick discussion from scheme discussion
- What is the cost of (quantum) access to (quantum) memory?
- How does the depth of a quantum circuit influence the cost?
Fix the attack-cost metric

- Hard to choose parameters for “unknown target”
- Proposals’ security claims use different metrics
- Hard to separate metric discussion from scheme discussion
- What is the cost of (quantum) access to (quantum) memory?
- How does the depth of a quantum circuit influence the cost?
- Also consider multi-target security?
Fix the attack-cost metric

- Hard to choose parameters for “unknown target”
- Proposals’ security claims use different metrics
- Hard to separate metric discussion from scheme discussion
- What is the cost of (quantum) access to (quantum) memory?
- How does the depth of a quantum circuit influence the cost?
- Also consider multi-target security?

Ignore classical attacks

- Concrete security of PQ schemes still not as stable as ECC
- Near-future deployment: use hybrid
- Long term: classical security won’t matter
Batch signing

- Most signatures are computed offline
- (prominent counter-example: TLS handshake signatures)
• Most signatures are computed offline
• (prominent counter-example: TLS handshake signatures)
• *Latency* does not matter for offline signatures
• *Throughput* may matter a lot (think of busy CAs)
Batch signing

- Most signatures are computed offline
- (prominent counter-example: TLS handshake signatures)
- *Latency* does not matter for offline signatures
- *Throughput* may matter a lot (think of busy CAs)
- Consider batch signing API:
  - Trivial to sign root of Merkle tree on top of OTS leaves
  - Can optimize "root-signing" scheme for size
  - Much (?) better performance in many relevant scenarios
Batch signing

- Most signatures are computed offline
- (prominent counter-example: TLS handshake signatures)
- *Latency* does not matter for offline signatures
- *Throughput* may matter a lot (think of busy CAs)
- Consider batch signing API:
  - Trivial to sign root of Merkle tree on top of OTS leaves
  - Can optimize “root-signing” scheme for size
  - Much (?) better performance in many relevant scenarios
- Curious to see more elaborate throughput-optimized signatures
But round 4 will still be limited to KEMs and Signatures, which is a great start but clearly limiting. The most obvious thing missing is maybe NIKE.”

—John Mattsson, Oct. 2021
“But round 4 will still be limited to KEMs and Signatures, which is a great start but clearly limiting. The most obvious thing missing is maybe NIKE.”

—John Mattsson, Oct. 2021

So far limited set of candidates

“But round 4 will still be limited to KEMs and Signatures, which is a great start but clearly limiting. The most obvious thing missing is maybe NIKE.”

—John Mattsson, Oct. 2021

So far limited set of candidates

• Azarderakhsh, Jao, and Leonardi (SAC 2017): “Post-Quantum Static-Static Key Agreement Using Multiple Protocol Instances”
“But round 4 will still be limited to KEMs and Signatures, which is a great start but clearly limiting. The most obvious thing missing is maybe NIKE.”

—John Mattsson, Oct. 2021

So far limited set of candidates

- Azarderakhsh, Jao, and Leonardi (SAC 2017): “Post-Quantum Static-Static Key Agreement Using Multiple Protocol Instances”
- Some work towards lattice-based NIKE; see
  - “Folklore” (Lyubashevsky): https://tinyurl.com/r5eb8su8
  - de Kock (Master’s thesis, TU/e, 2018): “A non-interactive key exchange based on ring-learning with errors”
  - Guo, Kamath, Rosen, Sotiraki (PKC 2020): “Limits on the Efficiency of (Ring) LWE Based Non-interactive Key Exchange”
Proofs

- Too many security proofs (of NISTPQC candidates) are wrong

• Round-1 Kyber public-key compression
• SPHINCS + assumptions on hash functions
• Lack of domain separation in various schemes (Bellare, Davis, Günther, Eurocrypt 2020)
• Kyber extra hash creates issues with QROM FO proof
• Too many security proofs (of NISTPQC candidates) are wrong
• Often theorems also wrong
• Sometimes security affected
• Too many security proofs (of NIST PQC candidates) are wrong
• Often theorems also wrong
• Sometimes security affected
• Some examples:
  • Round-1 Kyber public-key compression
  • SPHINCS$^+$ assumptions on hash functions
  • Lack of domain separation in various schemes (Bellare, Davis, Günther, Eurocrypt 2020)
  • Kyber extra hash creates issues with QROM FO proof
• Too many security proofs (of NISTPQC candidates) are wrong
• Often theorems also wrong
• Sometimes security affected
• Some examples:
  • Round-1 Kyber public-key compression
  • SPHINCS\textsuperscript{+} assumptions on hash functions
  • Lack of domain separation in various schemes (Bellare, Davis, Günther, Eurocrypt 2020)
  • Kyber extra hash creates issues with QROM FO proof

How confident can we be in proofs when schemes are standardized?
High-assurance crypto

Computer-verified proofs

• of security
• of implementation correctness
• of implementation security

Work in progress

• High-speed implementation of Kyber in jasmin
  (https://github.com/jasmin-lang/jasmin)
• Proof that implementation matches spec in EasyCrypt
  (https://github.com/EasyCrypt/easycrypt)
• Proof that spec achieves CCA security in EasyCrypt

See talk by Matthias Meijers at 3rd NIST PQC conference:
High-assurance crypto

Computer-verified proofs

- of security
- of implementation correctness
- of implementation security

Work in progress

- High-speed implementation of Kyber in jasmin (https://github.com/jasmin-lang/jasmin)
- Proof that implementation matches spec in EasyCrypt (https://github.com/EasyCrypt/easycrypt)
- Proof that spec achieves CCA security in EasyCrypt
- See talk by Matthias Meijers at 3rd NIST PQC conference: https://csrc.nist.gov/Presentations/2021/formal-verification-of-post-quantum-cryptography
Thank you!

https://cryptojedi.org

peter@cryptojedi.org

@cryptojedi