PQSRC highlights

Daniel J. Bernstein

PQSRC, FVPQS, and Crypto Frontiers

Post-Quantum Software Research Center (PQSRC) is hosted at the University of Illinois at Chicago.

One of the Intel Crypto Frontiers Research Center projects: Fast Verified Post-Quantum Software (FVPQS), a joint project between

- Daniel J. Bernstein at PQSRC and
- Tung Chou, Bow-Yaw Wang, and Bo-Yin Yang at Academia Sinica in Taiwan.

This talk: Highlights of PQSRC's work on FVPQS.
For more on FVPQS, see Bow-Yaw Wang's talk later today.

Examples of the motivation for FVPQS

2018 announcement re Dilithium: software bug meant that "reuse of randomness can easily be exploited to recover the secret key".

Examples of the motivation for FVPQS

2018 announcement re Dilithium: software bug meant that "reuse of randomness can easily be exploited to recover the secret key".

2019 announcement re Falcon: software bug meant that "signatures were valid but leaked information on the private key . . . the traditional development methodology (i.e. 'being super careful') has failed".

Examples of the motivation for FVPQS

2018 announcement re Dilithium: software bug meant that "reuse of randomness can easily be exploited to recover the secret key".

2019 announcement re Falcon: software bug meant that "signatures were valid but leaked information on the private key . . . the traditional development methodology (i.e. 'being super careful') has failed".

2020 announcement re FrodoKEM: software timing leak allowed a demo extracting "the secret key for all security levels using about 2^{30} decapsulation calls".

Examples of the motivation for FVPQS

2018 announcement re Dilithium: software bug meant that "reuse of randomness can easily be exploited to recover the secret key".

2019 announcement re Falcon: software bug meant that "signatures were valid but leaked information on the private key . . . the traditional development methodology (i.e. 'being super careful') has failed".

2020 announcement re FrodoKEM: software timing leak allowed a demo extracting "the secret key for all security levels using about 2^{30} decapsulation calls".

Another 2020 announcement re FrodoKEM: "the FrodoKEM team also fixed the timing oracle [GJN20] badly and caused a more serious security problem while trying to do that."

More examples of the motivation for FVPQS

2023.12 KyberSlash1 announcement: under some compilers, reference implementation leaks secrets through div timing.

More examples of the motivation for FVPQS

2023.12 KyberSlash1 announcement: under some compilers, reference implementation leaks secrets through div timing.

2023.12 KyberSlash2 announcement: under some compilers, reference implementation leaks secrets through div timing in another way, via reencryption.

More examples of the motivation for FVPQS

2023.12 KyberSlash1 announcement: under some compilers, reference implementation leaks secrets through div timing.

2023.12 KyberSlash2 announcement: under some compilers, reference implementation leaks secrets through div timing in another way, via reencryption.

Key-recovery attacks have been demonstrated for both leaks.

More examples of the motivation for FVPQS

2023.12 KyberSlash1 announcement: under some compilers, reference implementation leaks secrets through div timing.

2023.12 KyberSlash2 announcement: under some compilers, reference implementation leaks secrets through div timing in another way, via reencryption.

Key-recovery attacks have been demonstrated for both leaks.

14 Kyber libraries issued patches against KyberSlash.

More examples of the motivation for FVPQS

2023.12 KyberSlash1 announcement: under some compilers, reference implementation leaks secrets through div timing.

2023.12 KyberSlash2 announcement: under some compilers, reference implementation leaks secrets through div timing in another way, via reencryption.

Key-recovery attacks have been demonstrated for both leaks.

14 Kyber libraries issued patches against KyberSlash.

2024.06 announcement re Kyber: under some compilers, reference implementation leaks secrets through conditional-branch timing.

The pursuit of speed

Official Keccak (SHA-3) code package:

- `KeccakP-1600-reference.c`,
- `KeccakP-1600-x86-64-shld-gas.s`,
- `KeccakP-1600-AVX2.s`,
- `KeccakP-1600-AVX512.s`,
- `KeccakP-1600-times8-SIMD512.c`,
- `...`

"Why so many implementations?" — People want more speed than an "optimizing" compiler obtains from reference code.

The pursuit of speed

Official Keccak (SHA-3) code package:

- `KeccakP-1600-reference.c`,
- `KeccakP-1600-x86-64-shld-gas.s`,
- `KeccakP-1600-AVX2.s`,
- `KeccakP-1600-AVX512.s`,
- `KeccakP-1600-times8-SIMD512.c`,
- ...

"Why so many implementations?" — People want more speed than an "optimizing" compiler obtains from reference code.

Post-quantum crypto is more complicated than Keccak, and post-quantum software includes large volumes of hand-optimized software.

PQSRC highlight: `saferewrite`

Tool from https://pqsrc.cr.yp.to/downloads.html.
Has been used to verify many software optimizations.

PQSRC highlight: `saferewrite`

Tool from https://pqsrc.cr.yp.to/downloads.html.
Has been used to verify many software optimizations.

Easy-to-use interface. Examples included in latest package:
707 implementations of 274 simple functions. Automated.

PQSRC highlight: `saferewrite`

Tool from https://pqsrc.cr.yp.to/downloads.html.
Has been used to verify many software optimizations.

Easy-to-use interface. Examples included in latest package:
707 implementations of 274 simple functions. Automated.

Tries, often successfully, to answer the question of whether optimized code matches reference code for *all* inputs:
prove yes, or find an input where the outputs are different.
Also scans code for timing variations, including `mul` and `div`.

Example: automatically catches the bug in FrodoKEM's `cmp_64xint16`, and automatically verifies the fixed code.

PQSRC highlight: `saferewrite`

Tool from https://pqsrc.cr.yp.to/downloads.html.
Has been used to verify many software optimizations.

Easy-to-use interface. Examples included in latest package:
707 implementations of 274 simple functions. Automated.

Tries, often successfully, to answer the question of whether optimized code matches reference code for *all* inputs:
prove yes, or find an input where the outputs are different.
Also scans code for timing variations, including `mul` and `div`.

Example: automatically catches the bug in FrodoKEM's `cmp_64xint16`, and automatically verifies the fixed code.

Automatically analyzes binaries: e.g., can find compiler differences, or differences between C and assembly language.

PQSRC highlight: `cryptoint`

Almost-header-only C library distributed as module inside SUPERCOP, `lib25519`, `libmceliece`, `libntruprime`.
Liberally licensed to allow widest possible reuse.

PQSRC highlight: `cryptoint`

Almost-header-only C library distributed as module inside SUPERCOP, `lib25519`, `libmceliece`, `libntruprime`. Liberally licensed to allow widest possible reuse.

Library provides functions for constant-time comparisons, bit extractions, etc. on `{int,uint}{8,16,32,64}`. Advantages over previous work: more functions; the implementation is designed to protect against compilers introducing timing variations; all functions are verified using `saferewrite`.

PQSRC highlight: `cryptoint`

Almost-header-only C library distributed as module inside SUPERCOP, `lib25519`, `libmceliece`, `libntruprime`. Liberally licensed to allow widest possible reuse.

Library provides functions for constant-time comparisons, bit extractions, etc. on `{int,uint}{8,16,32,64}`. Advantages over previous work: more functions; the implementation is designed to protect against compilers introducing timing variations; all functions are verified using `saferewrite`.

For x86-64 and aarch64, library uses inline assembly, including a new `readasm` tool to improve auditability and to avoid common classes of inline-assembly bugs. For portable code, library uses `optblocker`, a volatile zero.

PQSRC highlight: `nttcompiler`

Tool from https://pqsrc.cr.yp.to/downloads.html.
Usable for many lattice systems: e.g., used in `libntruprime`.

PQSRC highlight: `nttcompiler`

Tool from https://pqsrc.cr.yp.to/downloads.html.
Usable for many lattice systems: e.g., used in `libntruprime`.

`nttcompiler` input: size-2^k NTT strategy.
Output: (1) portable C; (2) C with AVX2 intrinsics.
`nttcompiler` verifies that the AVX2 binaries
produce correct NTT outputs for all possible inputs.

PQSRC highlight: `nttcompiler`

Tool from https://pqsrc.cr.yp.to/downloads.html.
Usable for many lattice systems: e.g., used in `libntruprime`.

`nttcompiler` input: size-2^k NTT strategy.
Output: (1) portable C; (2) C with AVX2 intrinsics.
`nttcompiler` verifies that the AVX2 binaries
produce correct NTT outputs for all possible inputs.

"Don't compilers produce poor speeds?"

PQSRC highlight: `nttcompiler`

Tool from https://pqsrc.cr.yp.to/downloads.html.
Usable for many lattice systems: e.g., used in `libntruprime`.

`nttcompiler` input: size-2^k NTT strategy.
Output: (1) portable C; (2) C with AVX2 intrinsics.
`nttcompiler` verifies that the AVX2 binaries
produce correct NTT outputs for all possible inputs.

"Don't compilers produce poor speeds?" — Yes for general-purpose languages, but often a domain-specific compiler does a good job starting from a domain-specific language. See generally "The death of optimizing compilers".

PQSRC highlight: `nttcompiler`

Tool from `https://pqsrc.cr.yp.to/downloads.html`.
Usable for many lattice systems: e.g., used in `libntruprime`.

`nttcompiler` input: size-2^k NTT strategy.
Output: (1) portable C; (2) C with AVX2 intrinsics.
`nttcompiler` verifies that the AVX2 binaries
produce correct NTT outputs for all possible inputs.

"Don't compilers produce poor speeds?" — Yes for general-purpose languages, but often a domain-specific compiler does a good job starting from a domain-specific language. See generally "The death of optimizing compilers".

Closest previous work: SPIRAL DSL for floating-point FFTs.
But NTTs raise new codegen+verification questions.

PQSRC highlight: `safegcd2`

Formal proofs distributed as part of the HOL Light prover.
Used in, e.g., Amazon's new formally verified speedups
for X25519 software and Ed25519 software in LibCrypto.
Also applicable in many post-quantum systems.

PQSRC highlight: `safegcd2`

Formal proofs distributed as part of the HOL Light prover.
Used in, e.g., Amazon's new formally verified speedups
for X25519 software and Ed25519 software in LibCrypto.
Also applicable in many post-quantum systems.

These are proofs that $1 + \lfloor 9437b/4096 \rfloor$ iterations of the
2019 Bernstein–Yang "divstep" iteration, starting with $\delta = 1/2$
and $0 \leq g \leq f \leq 2^b$, compute gcd, modular inverse, etc.
This iteration is convenient for fast constant-time code.

PQSRC highlight: `safegcd2`

Formal proofs distributed as part of the HOL Light prover.
Used in, e.g., Amazon's new formally verified speedups
for X25519 software and Ed25519 software in LibCrypto.
Also applicable in many post-quantum systems.

These are proofs that $1 + \lfloor 9437b/4096 \rfloor$ iterations of the 2019 Bernstein–Yang "divstep" iteration, starting with $\delta = 1/2$ and $0 \leq g \leq f \leq 2^b$, compute gcd, modular inverse, etc.
This iteration is convenient for fast constant-time code.

Planning paper on proof technique and software integration;
joint work with Harrison, Maxwell, Wang, Wuille, Yang.

PQSRC highlight: goppadecoding

HOL Light formal proofs, Lean formal proofs, and paper:
see https://cr.yp.to/papers.html#goppadecoding.
Verifies formulas used in deployed Classic McEliece software.

PQSRC highlight: goppadecoding

HOL Light formal proofs, Lean formal proofs, and paper:
see https://cr.yp.to/papers.html#goppadecoding.
Verifies formulas used in deployed Classic McEliece software.

Context: Classic McEliece uses a much more powerful decoder
than typical lattice systems, giving it much higher security
at each ciphertext size, making it the most efficient choice
for applications using static post-quantum keys.

PQSRC highlight: goppadecoding

HOL Light formal proofs, Lean formal proofs, and paper:
see https://cr.yp.to/papers.html#goppadecoding.
Verifies formulas used in deployed Classic McEliece software.

Context: Classic McEliece uses a much more powerful decoder
than typical lattice systems, giving it much higher security
at each ciphertext size, making it the most efficient choice
for applications using static post-quantum keys.

These proofs guarantee correctness for a specific decoder,
the decoder used in the Classic McEliece software.
Unexpected spinoff: can skip reencryption after this decoder
(although reencryption might still rescue security after faults).

Big step towards complete verification of the software.