Implementing oqs-provider

or: How to add experimental PQC to OpenSSL

Michael Baentsch

Overview

- Background
- OpenSSL provider concept
- oqsprovider design principles
- History
- Features
- Lessons learned
- Outlook

Background (I)

- libOQS («Open Quantum Safe»)
  - Collection of PQC algs in NIST competition
  - C APIs separate for SIGs and KEMs
  - «No winners chosen»
- Application/library integrations
  - To facilitate «higher-level» PQC use/experimentation
  - e.g., openssh, openssl

Background (II)

- openssl fork adding liboqs support
  - Created 2018 by Microsoft Research (@christianpaquin)
  - Facilitating TLS and hybrid (classic+PQ) experimentation
  - Regularly updated to follow OpenSSL evolution (v111)
  - Used for app integrations (httpd, curl, haproxy, etc.)
  - Maintenance became unwildy over time
    - Following NIST changes
    - Following OpenSSL changes

OpenSSL providers

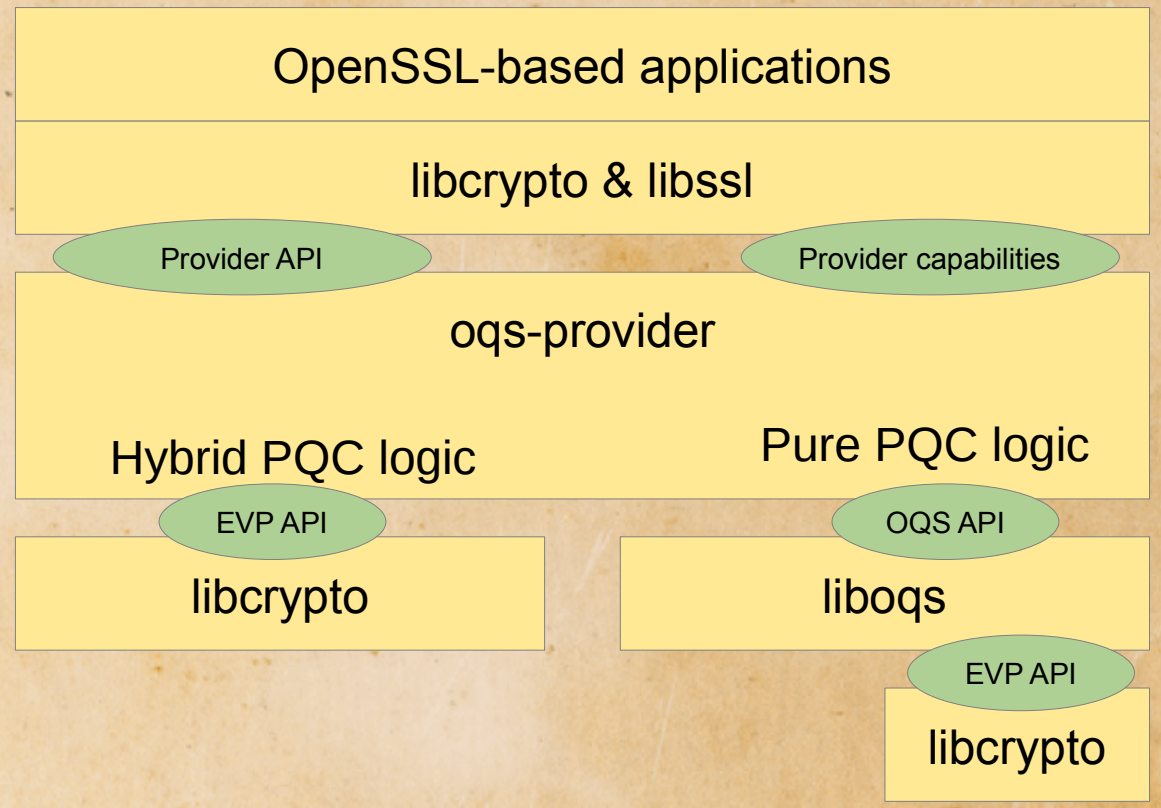
- Introduced in OpenSSL 3.0
- Permit extension of core crypto via defined APIs
- e.g., ciphers, digests, signatures, KEMs, RNGs, …
- Important for PQC: KEM & SIGs (& key mgmt)
- Dynamic or static add-on to OpenSSL installation
- Also used openssl-internally for default crypto, FIPS crypto (+deprecated/legacy crypto)

oqs-provider design principles

- «Choose no winner»: Treat all PQ candidate algs alike
  - using liboqs APIs → adding new algorithms to be trivial
- Add PQC in all OpenSSL versions >= 3.0
  - Evolving with OpenSSL feature set → remain useful over time
- As easy to maintain and use as possible
  - Generate as much code as possible & do not duplicate other FOSS work
  - Retain algorithm names across algorithm releases
  - Primary focus: shared lib deployment

  → Do not rely on internal OSSL crypto → useful in any openssl(3) install

oqs-provider design



History (I)

- 2021
  - Integration of all OQS KEMs (except ClassicMcEliece)
  - Linux only
  - «Extraction» from OpenSSL code base (@levitte)
  - Addition of hybrid KEMs (@bhess)
- 2022
  - Addition of all OQS SIG algs
  - First release
  - Enhancements to OpenSSL 3 (.2): pluggable sig logic added
  - Hybrid SIG support & export/import added (X.509)

History (II)

- 2023
  - More platforms supported (MacOS, Windows)
  - Static builds supported
  - Tooling support for IETF PQ cert hackathons
- 2024
  - Composite sigs added (Dilithium-only, @feventura)
  - PQCA: Enthusiastic adopters with a different perspective
- 2025
  - Still new algs added but reduction of feature set

Features

- 152 PQ algs enabled
  - For TLS 1.3 KEM ops
  - For X.509 cert gen & TLS 1.3 sig ops
  - Using standard openssl tooling & APIs
- Configuration-file driven code generation
  - (Auto-)Integrating any algorithm supported by liboqs
  - Creating arbitrary hybrid (PQ/classic crypto) variants
- (Auto)Adapting to OpenSSL 3.x runtime
- Runs on many Unix variants (incl. MacOS) & Windows

Lessons learned: Contributing

- Reach out to community & core team, as
  - some things may not be documented
  - some things may not work as intended
  - some features may be missing
  - some features may be buggy
- Do
  - Be courteous and considerate of alternative assumptions
  - Be diligent trying stuff first yourself
  - Explain your assumptions
  - Provide code samples

Lessons learned: oqs-provider

- Design concepts worked
  - Adding new PQ algorithms is trivial
  - «hybridization» to these, too
  - One binary can be used with any OpenSSL 3.x install
    - Adapting its own capabilities to those of base platform
    - «winner»/std PQ algs fall away automatically
- OpenSSL testing
  - oqs-provider as external «stress test»
- Per-algorithm optimizations not feasible

Lessons learned: FOSS cooperation

- Community (r)evolution: Make people meet
- Non-communicated assumptions are bad, e.g.
  - how project is meant to be used / whom it caters for
  - how to treat contributors, e.g., individual vs. corporate
- Mismatch of procedures to contributor base: size & type
- Clashes of open-source control philosophies
  - Trust the do-ers
  - Reign in management by non-contributors

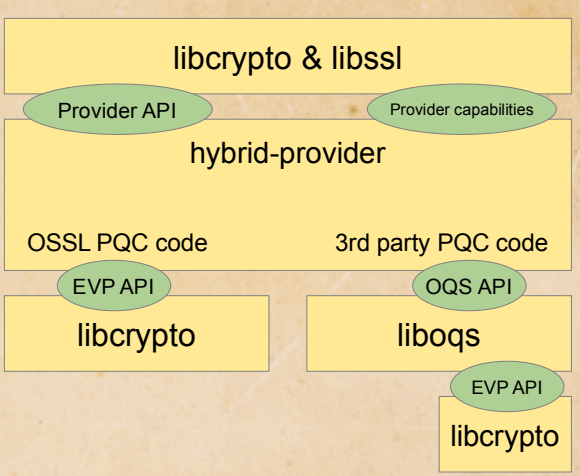
Challenges encountered

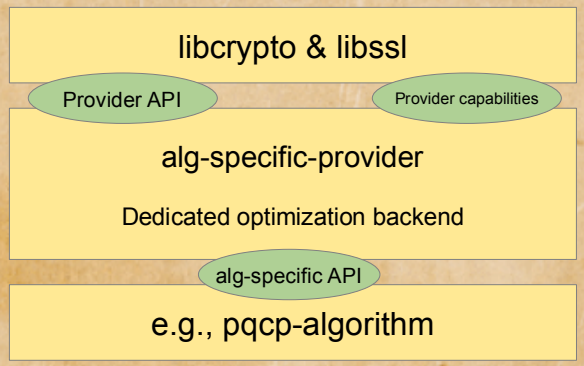
- Where to plug OpenSSL base support
  - e.g. hashes, classic KEMs in oqs-provider
  - e.g. RNG, SHA3 in liboqs
- Lots of boilerplate code
  - Heavy use of macros
  - Or maybe a generator like QUBIP/aurora
- Still a dependence on NIDs
  - Especially in X.509 code

Possible future work

- Completely get rid of NID logic
- Demonstrate best practice in the code
  - Not competing with commercially-supported FOSS
- Assist researchers in making better reference code
  - e.g., maintainability, upstream-ability
- Create additional layer to abstract location of PQC algorithm (underneath EVP API or liboqs API)
  - Would enable clean hybrid|composite implementations
  - Would enable use of std PQC in oqs-provider

alt-pqc provider design strawmen





Thank You!

- All Contributions VERY welcome
- Participate at **`github.com/open-quantum-safe/oqs-provider`**
  - Alternative design principles under discussion

    Discussion in GitHub plus virtual meeting on Oct 23, 2025
    (participation link at https://pqca.org/calendar)

- Questions?

Anytime via GH: @baentsch or email: info@baentsch.ch