PQC Algorithm Selection

Falko Strenzke
Johannes Roth
Stavros Kousidis
Aron Wussler

IETF 119
2024-03-20

Key Encapsulation

Original Proposal

ML-KEM-768 + X25519	MUST
ML-KEM-1024 + X448	SHOULD
ML-KEM-768 + ECDH-NIST-P-256	MAY
ML-KEM-1024 + ECDH-NIST-P-384	MAY
ML-KEM-768 + ECDH-brainpoolP256r1	MAY
ML-KEM-1024 + ECDH-brainpoolP384r1	MAY

Feedback from the List

- General support for X25519 and X448 with ML-KEM, one of them MUST
- Some voices for allowing ML-KEM with a single EC family, possibly moving the NIST/Brainpool to a separate doc
- Some support for at least one ML-KEM with NIST / Brainpool curve
- One voice for making one of the ML-KEM with NIST / Brainpool curves a SHOULD
- One voice for NTRU+X25519 and McEeliece+X448

Altered Proposal

draft-ietf-openpgp-pqc	
ML-KEM-768 + X25519	MUST
ML-KEM-1024 + X448	SHOULD

draft-?-openpgp-pqc-brainpool	
ML-KEM-768 + ECDH-brainpoolP256r1	?
ML-KEM-1024 + ECDH-brainpoolP384r1	?

Is there interest for draft-?-openpgp-pqc-nist?

Altered Proposal

- Keep ML-KEM + X25519 / X448, with one as MUST
- Split Brainpool to a specific draft

Key Derivation and Combination

Original Proposal

```
fixedInfo = algID

eccKeyShare = SHA3-256(sharedSecret || eccCT || eccPK)

eccData = eccKeyShare || eccCT

mlkemData = mlkemKeyShare || mlkemCT

encData = counter || eccData || mlkemData || fixedInfo

MB = KMAC256(domSeparation, encData, oBits,
customizationString)
```

Feedback from the List

- Do not specify an OpenPGP custom KEM combiner
- Include public keys in the KEM combiner
- Reuse an existing KDF for ECC
- KMAC is fine, but what are the advantages compared to SHA-3?

Altered Proposal

```
fixedInfo = algID || domSeparation

eccKeyShare = SHA3-256(sharedSecret || eccCT || eccPK)

eccData = eccKeyShare || eccCT || eccPK

mlkemData = mlkemKeyShare || mlkemCT || mlkemPK

MB = SHA3-256(counter || eccData || mlkemData ||
fixedInfo)
```

Altered Proposal

We keep on following the discussion on CFRG, for now:

- We keep the same ECC KDF as it's a simple Cramer–Shoup construction, identical to the existing X25519/X448, but using SHA-3
- We change to fixed-length output as AES-256 for key wrap is anyway hardcoded
- We remove KMAC because with with fixed-length it does not provide security advantages, but easier to implement

Digital Signature

Original Proposal

ML-DSA-65 + Ed25519	MUST
ML-DSA-87 + Ed448	SHOULD
ML-DSA-65 + ECDSA-NIST-P-256	MAY
ML-DSA-87 + ECDSA-NIST-P-384	MAY
ML-DSA-65 + ECDSA-BrainpoolP256r1	MAY
ML-DSA-87 + ECDSA-BrainpoolP384r1	MAY
SLH-DSA-SHA2	SHOULD
SLH-DSA-SHAKE	MAY

Composite vs Multiple Signatures

To compare the possible approaches, we laid out the solutions detailing:

- The required changes to OpenPGP
- A way to achieve backwards compatibility
- The provided security guarantees
- The relevant feedback from the list

Option 1: Separate Signatures

aka. "Migration like ECC"

We just standardize new codepoints for ML-DSA, standalone, no binding across keys.

Changes to OpenPGP: This option would simply imply two codepoints for ML-DSA-65 and ML-DSA-87.

Migration strategy: Over time users will generate new keys, and deprecate the old ones. Signers can sign the messages twice with ECC and PQ, and verifiers choose what to deem valid.

Security guarantee: No non-separability guarantee.

Feedback from the List in Favor of Option 1

- It is the most straightforward approach
- The ECC migration was done this way, and it was successful
- Independent signatures should be considered valid at time T (consider notarization)
- Define PQ-sig code-points now but we don't recommend using until years have passed
- This can be used with nested signatures
- Shifts responsibility onto applications giving greater flexibility to the protocol

Feedback from the List Against Option 1

- Some uses may not handle multiple signatures, e.g. PGP/MIME on some application layers or binding signatures
- The authors of Dilithium/ML-DSA recommend using the algorithm in a hybrid fashion
- Separate signatures might accidentally convey different information, leading to inconsistencies (e.g different sig time or different expiration)
- Can cause signature separability security issues, stripping attacks may look like a sender is capable only of one signature type (PQC or ECC)

Feedback from the List Against Option 1

- Shifts responsibility onto applications (e.g. MUAs) or users and this will cause interoperability issues.
- With two signatures we fundamentally don't know if AND vs OR behaviour is desired, but delays in updating existing systems make a consistent AND policy desirable.
- It will require two certificates for security reasons also for homogeneous environments (where you know everyone supports PQC)
- Nested signatures can only be used on a limited set of signature usages (not on detached, self-sig, or key binding)

Option 2: Composable Hybrid

We standardize new codepoints for ML-DSA as standalone, and introduce a binding across keys / signatures.

Changes to OpenPGP: two new codepoints as for (1), new (optional) mechanism to bind signatures. If a bound signature is found, the matching sig and two keys are required to verify it.

Migration strategy: Users who already have a v4 or v6 ECC certificate can generate a v6 PQ cert. The signer has to distribute two keys and the verifier needs to map the signatures across them to verify.

Security guarantee: Optional non-separability guarantee, if the signer binds two signatures they can't be verified independently.

Feedback from the List Against Option 2

- Binding on the protocol level will cause cross-layer dependencies and make implementation more complex
- Bad Signature = No Signature, one signature should be enough to validate
- Linking self-sigs/key binding in certificates is hard

Option 3: Composite Hybrid

We standardize new codepoints for ML-DSA hybrid with ECC (aka PQ/T), no binding across keys.

Changes to OpenPGP: two to six new code points for ML-DSA + EdDSA/ECDSA.

Migration strategy: Users who already have a v4 or v6 key would need to generate a new PQ/T key. The signer can sign twice and distribute two keys and the verifier needs just one of them to verify a signature, depending on their security policy.

Security guarantee: Strong non-separability. The component sigs are invalid when taken out of the composite context.

Feedback from the List in Favor of Option 3

- Multiple voices for desiring non-separable self-sig/key binding hybrid signatures
- Hybrid signatures reduce operational complexity (opsec harder than protocol-sec) or shouldn't introduce any operational complexity (handled at the lib level)
- Backwards compatibility transition and trust transition are different, linking the two migrations may be attractive but risky (C-R does not address transition v4-v6)
- Authentication protocols don’t take multiple signatures
- Library implementers seem not to have an issue with the hybrid complexity
- Standalone algorithms can be standardized later when confidence is higher (proposal for also adding standalone ML-DSA now was rejected)
- Allowing mixing standalone algorithms with AND policy will lead to application inventing their combination scheme at the protocol layer

Feedback from the List Against Option 3

- More complex to implement and users will need two keys anyway
- If we support non-composite PQ sig and a classic sig, then one has to figure out all the “how to” issues in any case, even if one also has composites
- Computationally more expensive
- Hybrid only offers value at the end of the ECC - PQC transition period
- Once we have CRQC, we have unnecessary overhead
- Signature validation policy (AND vs OR) should be up to the consumer
- Hybrid signatures increase operational complexity (a single key compromise is fatal)
- Risk of users reusing the same private key material for ECC, problem with revocation complexity
- Risk of algorithm code point explosion and interop issues

Other Feedback from the List for Option 3

- Hybrid sigs should be named differently from standalone
- One voice for limiting it to only one algorithm
- One voice for limiting it to two algorithms
- One voice for parameterization
- Multiple voices not to use parameterization in algorithms

ML-DSA Feedback

- Most people support both ML-DSA-65 and ML-DSA-87
- Some people support either ML-DSA-65 or ML-DSA-87, with -87 preferred

SLH-DSA Feedback

- One voice to keep at most one of the variants, as MAY, with one or two parameters
- One voice on MUST
- One voice for hybrid with curves

Other Feedback from the List

- Signature generation and verification may be at distant points in time
- Mandate (MUST) PQ algorithms based on two different underlying security assumptions
- Don't create a combinatorial zoo of algorithms
- Make it easy to introduce new algorithms catering to particular target audience
- Don't reduce security along the way
- Make one suite per draft
- It is too early to start issuing PQC signatures
- We need signatures now for CNSA 2.0, but different environments can have different transition periods
- Practical attacks against PQC may arise earlier than a CRQC

Altered Proposal

draft-ietf-openpgp-pqc	
ML-DSA-65 + Ed25519	MUST
ML-DSA-87 + Ed448	SHOULD
SLH-DSA-SHAKE-128f (?)	MAY
SLH-DSA-SHAKE-128s (?)	MAY
SLH-DSA-SHAKE-256s (?)	MAY

draft-?-openpgp-pqc-brainpool	
ML-DSA-65 + ECDSA-BrainpoolP256r1	?
ML-DSA-87 + ECDSA-BrainpoolP384r1	?

draft-?-openpgp-pqc-nist (?)

Altered Proposal

- Limit the number of hybrid algorithms to 2
- Limit the variant of SLH-DSA to 3: fast, compact, and high security.
- Move Brainpool to a separate draft
- Implement ML-DSA “wearing both belt and suspenders for now”

Thanks everyone for this vigorous discussion!

-dkg