Standardization efforts for PQC in OpenPGP in the Project PQC@Thunderbird

Stephan Ehlen\textsuperscript{BSI}, Andreas Hülsing\textsuperscript{TU/e}, Evangelos Karatsiolis\textsuperscript{MTG}, Stavros Kousidis\textsuperscript{BSI}, Johannes Roth\textsuperscript{MTG}, Falko Strenzke\textsuperscript{MTG}, Christian Tobias\textsuperscript{MTG}, Aron Wussler\textsuperscript{Proton}

\textbf{BSI:} German Federal Office for Information Security  \\
\textbf{MTG:} MTG AG, Germany  \\
\textbf{Proton:} Proton AG, Switzerland  \\
\textbf{TU/e:} University of Eindhoven
Design Criteria

Algorithm and Parameter Choices

Kyber-KEM

Signatures

Next Steps
Design Criteria

Algorithm and Parameter Choices

Kyber-KEM

Signatures
  PQC and Hash-and-Sign
  Multiple signatures on the protocol level

Next Steps
Design Criteria

- Use **composite** multi-algorithm (classic + PQC, a.k.a. hybrid) for Kyber and Dilithium, standalone for SPHINCS$^+$
- Backwards compatibility:
  - Having two different certificates (v4/v5)
  - Multiple signatures on the protocol level
- As classical algorithms we propose ECC:
  - “fix” all previously existing inconsistencies regarding data formats
    - i.e. native format for CFRG curves
Design Criteria

Algorithm and Parameter Choices

Kyber-KEM

Signatures
PQC and Hash-and-Sign
Multiple signatures on the protocol level

Next Steps
## Algorithm Choices

<table>
<thead>
<tr>
<th>Algorithm Combination</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kyber512 + X25519</td>
<td>MUST</td>
</tr>
<tr>
<td>Kyber1024 + X448</td>
<td>SHOULD</td>
</tr>
<tr>
<td>Kyber768 + ECDH-NIST-P-384</td>
<td>MAY</td>
</tr>
<tr>
<td>Kyber1024 + ECDH-NIST-P-521</td>
<td>MAY</td>
</tr>
<tr>
<td>Kyber768 + ECDH-brainpoolP384r1</td>
<td>MAY</td>
</tr>
<tr>
<td>Kyber1024 + ECDH-brainpoolP512r1</td>
<td>MAY</td>
</tr>
<tr>
<td><strong>Dilithium2 + Ed25519</strong></td>
<td>MUST</td>
</tr>
<tr>
<td>Dilithium5 + Ed448</td>
<td>SHOULD</td>
</tr>
<tr>
<td>Dilithium3 + ECDSA-NIST-P-384</td>
<td>MAY</td>
</tr>
<tr>
<td>Dilithium5 + ECDSA-NIST-P-521</td>
<td>MAY</td>
</tr>
<tr>
<td>Dilithium3 + ECDSA-brainpoolP384r1</td>
<td>MAY</td>
</tr>
<tr>
<td>Dilithium5 + ECDSA-brainpoolP512r1</td>
<td>MAY</td>
</tr>
<tr>
<td>SPHINCS(^{+})-simple-SHA2</td>
<td>SHOULD</td>
</tr>
<tr>
<td>SPHINCS(^{+})-simple-SHAKE</td>
<td>MAY</td>
</tr>
</tbody>
</table>
**SPHINCS+ Parameters**

- SPHINCS+\(-\text{simple-}\text{SHA2-128s}\) SHOULD
- SPHINCS+\(-\text{simple-}\text{SHA2-128f}\) SHOULD
- SPHINCS+\(-\text{simple-}\text{SHA2-192s}\) SHOULD
- SPHINCS+\(-\text{simple-}\text{SHA2-192f}\) SHOULD
- SPHINCS+\(-\text{simple-}\text{SHA2-256s}\) SHOULD
- SPHINCS+\(-\text{simple-}\text{SHA2-256f}\) SHOULD
- SPHINCS+\(-\text{simple-}\text{SHAKE-128s}\) MAY
- SPHINCS+\(-\text{simple-}\text{SHAKE-128f}\) MAY
- SPHINCS+\(-\text{simple-}\text{SHAKE-192s}\) MAY
- SPHINCS+\(-\text{simple-}\text{SHAKE-192f}\) MAY
- SPHINCS+\(-\text{simple-}\text{SHAKE-256s}\) MAY
- SPHINCS+\(-\text{simple-}\text{SHAKE-256f}\) MAY
Design Criteria

Algorithm and Parameter Choices

Kyber-KEM

Signatures
  PQC and Hash-and-Sign
  Multiple signatures on the protocol level

Next Steps
Kyber-KEM

Basic design paradigms:
  ▶ use ECDH / X25519 / X448 as KEMs
  ▶ omit the key derivation step and output a shared key
  ▶ derive the KEK from the ECDH and Kyber shared keys
  ▶ use SHA3-based simple concatenate-and-hash construction with some fixed info
Signatures

Design Criteria

Algorithm and Parameter Choices

Kyber-KEM

Signatures

PQC and Hash-and-Sign
Multiple signatures on the protocol level

Next Steps
Design Criteria

Algorithm and Parameter Choices

Kyber-KEM

Signatures
  PQC and Hash-and-Sign
    Multiple signatures on the protocol level

Next Steps
PQC and Hash-and-Sign

- SPHINCS$^+$ and Dilithium are not following the simple hash-and-sign paradigm
  - SPHINCS$^+$ uses randomized hashing, not to rely on the collision resistance of the hash function
  - Dilithium prepends the public key
- OpenPGP v5 signatures also features randomized hashing but the details differ
## Hashing in PQC Schemes vs. v5 Signatures

<table>
<thead>
<tr>
<th></th>
<th><strong>SPHINCS</strong>&lt;sup&gt;+&lt;/sup&gt;</th>
<th><strong>Dilithium</strong></th>
<th><strong>v5 signatures</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hash algo</strong></td>
<td>SHA2 / SHA3</td>
<td>SHA3</td>
<td>SHA2 / SHA3</td>
</tr>
<tr>
<td><strong>Salt size</strong></td>
<td>128, 192, or 256 bit</td>
<td>N/A</td>
<td>128 bit</td>
</tr>
</tbody>
</table>

- Depending on the SPHINCS<sup>+</sup> security level, SPHINCS<sup>+</sup> hash-and-sign v5 signatures will be weaker than original SPHINCS<sup>+</sup>.
- In order to preserve the security level of SPHINCS<sup>+</sup>, a larger salt value in v5 signatures is necessary for some parameters.
- Dilithium uses only SHA3 hashing. We considered binding the hash function to the algorithm ID.
Design Criteria

Algorithm and Parameter Choices

Kyber-KEM

Signatures

PQC and Hash-and-Sign

Multiple signatures on the protocol level

Next Steps
Signature Concatenation

The goal is backwards compatibility to legacy clients

- **E-mail: concatenate two signatures**
  - For instance Thunderbird and Proton clients currently process only the first signature
  - Classical signature followed by PQC

- **OpenPGP messages**
  - Multiple signatures already specified
  - State of implementation support apparently not optimal
  - Need additional testing in interoperability suite
Design Criteria

Algorithm and Parameter Choices

Kyber-KEM

Signatures

PQC and Hash-and-Sign

Multiple signatures on the protocol level

Next Steps
Next Steps

- Wait for publication of Kyber IP results from NIST
- Publication of the draft
  - Currently draft is still under construction
  - Expected publication Nov. or Dec. ’22
- Implementations
  - Proton already has an experimental go implementation
  - MTG will work on implementations:
    - in Libgcrypt/OpenPGP, Botan/RNP/Thunderbird
    - covering all algorithms proposed here
    - work from Jan. ’23 to Nov. ’23
- Improve the testing suite to include the missing tests
Questions for the WG

What do you think of:

- The algorithm selection?
- Binding the signature salt size to the hash ID?
- Binding the hash function to the algorithm ID?
Any feedback on the draft is very welcome!

An open discussion will follow in Mezzanine 12 starting at 14:50 (link for remote participants in the side-meetings wiki)