CRYSTALS - Dilithium

Shi Bai
Leo Ducas
Eike Kiltz
Tancrede Lepoint

Vadim Lyubashevsky
Peter Schwabe
Gregor Seiler
Damien Stehle
Dilithium

- “Schnorr-like” lattice-based signature scheme
- Based on the hardness of Module-SIS and Module-LWE
- All operations over $R=\mathbb{Z}_q[X]/(X^{256}+1)$ for $q=8,380,417$
- All randomness during signing is generated uniformly from a power-of-2 range (i.e. easy sampling, no gaussians)

<table>
<thead>
<tr>
<th>Security Level</th>
<th>Public Key (Bytes)</th>
<th>Signature (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128 (NIST II)</td>
<td>1312</td>
<td>2420</td>
</tr>
<tr>
<td>192 (NIST III)</td>
<td>1952</td>
<td>3293</td>
</tr>
<tr>
<td>256 (NIST V)</td>
<td>2592</td>
<td>4595</td>
</tr>
</tbody>
</table>
Dilithium
(high-level overview)

\[ As_1 + s_2 = t_0 + bt_1 \]

**Sign(μ)**

- \( y \leftarrow \) Coefficients in \([-γ, γ]\)
- \( c \leftarrow H(\text{high}(Ay), \mu) \)
- \( z \leftarrow y + cs_1 \)
- If \( |z| > γ - β \) or \( |\text{low}(Ay - cs_2)| > γ - β \)
  - restart
- Create carry bit hint vector \( h \)

**Signature** = (\( z, h, c \))

**Verify(\( z, c, μ \))**

- Check that \( |z| \leq γ - β \)
- \( c = H(\text{high}(h + "\) Az - cbt_1), \mu) \)
- \( \text{high}(Ay - cs_2) \)
Some FAQs

1. The verifier learns the high order bits of $A_y$ – do we need LWR hardness?
   • No. The verifier only learns the high order bits of $A_y$ when we don’t abort. The $w$ can be derived from the signature and the challenge, and because the aborting condition allows us to perfectly simulate the signature, this $w$ gives no extra information.

2. Does the rejection probability leak side-channel information about the secret?
   • No. The probability of rejection is the same for all secrets and challenges.

3. The low-order bits of $A_{s_1+s_2}$, (i.e. $t_0$) are not given to the verifier as part of the public key. Is this secret?
   • No. The verifier does not need them to verify, but they are not secret. Some information is leaked with every signature. The security proof assumes that $t_0$ is public.
Recommended Change 1

Deterministic Dilithium derives randomness $r = H(K, \text{msg})$
Randomized Dilithium derives its randomness $r$ from the system

Change the randomized Dilithium randomness to be $r = H(r', K, \text{msg})$ where $r'$ comes from the system.

Advantage: randomized Dilithium still secure even if the system randomness is bad.
Disadvantage: basically none (recall that msg is already the compressed message digest, so there is virtually no slowdown).
Recommended Change 2

Switch to 12-round Keccak for public key expansion

Advantages:
- signing is much faster especially on devices that can’t keep the whole expanded key in memory and regenerate it with every rejection
- In some cases, 80% of verification is Keccak

Disadvantages:
- none? Just need to assume that x-round Keccak generated lattice instances are still hard. Don’t need the function to be a “cryptographic hash function”
Decision that needs to be made

Keep the 90’s version of Dilithium along with the Keccak version
(AES and SHA-256 instead of Keccak)

Having only have the Keccak version is more elegant
... but ...

AES is significantly more efficient than Keccak (partly due to hardware acceleration)

Seems like keeping both makes sense, or is it a short-sighted decision?
Could even use a round-reduced AES to generate the lattice
CRYSTALS – Dilithium

https://pq-crystals.org/dilithium/index.shtml

https://github.com/pq-crystals/dilithium

https://github.com/pq-crystals/security-estimates