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 $R=Z_q[X]/(X^{256}+1)$ for q=8,380,417
- All randomness during singing is generated uniformly from a power-of-2 range (i.e. easy sampling, no gaussians)

Security Level	Public Key (Bytes)	Signature (Bytes)
128 (NIST II)	1312	2420
192 (NIST III)	1952	3293
256 (NIST V)	2592	4595

Dilithium

(high-level overview)

$$\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 = \mathbf{t}_0 + \mathbf{b}\mathbf{t}_1$$

Sign(μ)

$\mathbf{y} \leftarrow$ Coefficients in $[-\gamma, \gamma]$

$c := H(\text{high}(\mathbf{A}\mathbf{y}), \mu)$

$\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$

If $|\mathbf{z}| > \gamma - \beta$ or $|\text{low}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2)| > \gamma - \beta$

restart

Create carry bit hint vector h

Signature = $(\mathbf{z}, h, c)$

Verify($\mathbf{z}$, c, μ)

Check that $|\mathbf{z}| \leq \gamma - \beta$

and

$c = H(\text{high}(h \text{ “+” } \mathbf{A}\mathbf{z} - c\mathbf{b}\mathbf{t}_1), \mu)$

$\text{high}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2)$

Some FAQs

1. The verifier learns the high order bits of Ay – do we need LWR hardness?
   - No. The verifier only learns the high order bits of Ay 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 $\mathbf{A}\mathbf{s}_1+\mathbf{s}_2$, (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 , msg)

Randomized Dilithium derives its randomness r from the system

Change the randomized Dilithium randomness to be r = H(r', K, 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