1 Lattice Geometry for ML-DSA (Intuition First)

This note explains lattice geometry without heavy formulas first, then connects it to the short-vector problems used in lattice cryptography.

1.1 What is a lattice?

Everyday analogy. Imagine tiling a floor with parallelogram-shaped tiles. The corners of every tile form a regular, infinite pattern — that pattern is a lattice. You pick a starting point (the origin) and two "step" directions; every point you can reach by taking whole-number steps along those directions is a lattice point.

More precisely, given m linearly independent vectors $\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_m \in \mathbb{R}^n$, the lattice they generate is:

$$\mathcal{L}(B) = \{z_1\boldsymbol{b}_1 + z_2\boldsymbol{b}_2 + \dots + z_m\boldsymbol{b}_m : z_i \in \mathbb{Z}\} = \{B\boldsymbol{z} : \boldsymbol{z} \in \mathbb{Z}^m\} \tag{1}$$

where $B = [\boldsymbol{b}_1 \mid \boldsymbol{b}_2 \mid \dots \mid \boldsymbol{b}_m]$ is the basis matrix (columns are basis vectors).

Key properties:

- Discrete: lattice points are isolated — there is a minimum distance between any two distinct points.
- Periodic: the pattern repeats in every basis direction.
- Rank $\boldsymbol{m}$, dimension $\boldsymbol{n}$: the lattice lives in $\mathbb{R}^n$ but is spanned by m vectors. When $m = n$ the lattice is called full-rank.

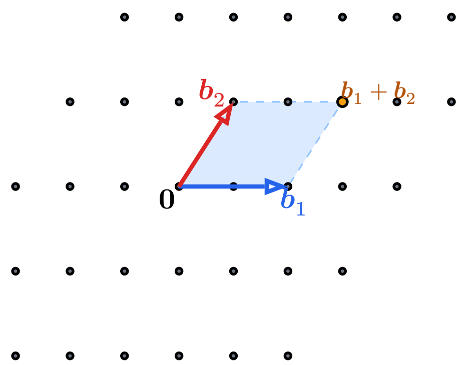

Figure 1: A 2D lattice generated by $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$. Every dot is an integer combination $z_1\boldsymbol{b}_1 + z_2\boldsymbol{b}_2$. The shaded region is the fundamental parallelogram — it tiles the plane with no gaps or overlaps.

Example in $\mathbb{Z}^2$. Take $\boldsymbol{b}_1 = (2, 0)$ and $\boldsymbol{b}_2 = (1, 3)$. Then:

- $1 \cdot \boldsymbol{b}_1 + 1 \cdot \boldsymbol{b}_2 = (3, 3)$ ✓ lattice point
- $(1.5, 1.5)$ is not a lattice point — no integer combination produces it.

1.2 Same lattice, different bases

A single lattice can be described by many different bases. Two bases B and B' generate the same lattice if and only if $B' = BU$ where U is a unimodular matrix (integer matrix with $\det(U) = \pm 1$).

$$\mathcal{L}(B) = \mathcal{L}(B') \quad \Longleftrightarrow \quad B' = BU, \quad U \in \mathbb{Z}^{m \times m}, \quad \det(U) = \pm 1 \tag{2}$$

Why this matters for cryptography:

- A good basis has short, nearly orthogonal vectors — problems like finding short vectors are easy.
- A bad basis has long, highly skewed vectors — the same problems become computationally hard.

Lattice cryptography works by publishing a bad basis (or equivalent public information) while keeping a good basis secret.

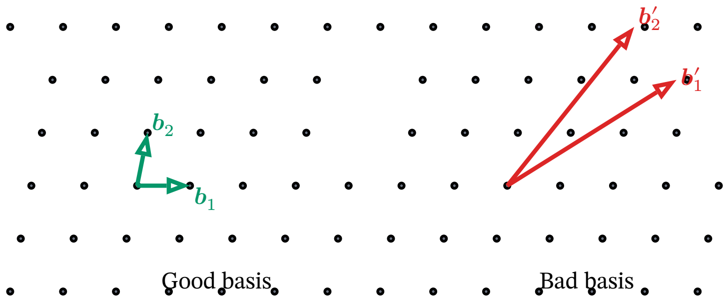

Figure 2: Two bases for the same lattice. Left: short, nearly orthogonal (good). Right: long, skewed (bad). The lattice points are identical.

1.3 Short vectors and the SVP

The minimum distance of a lattice is the length of its shortest nonzero vector:

$$\lambda_1(\mathcal{L}) = \min_{\boldsymbol{v} \in \mathcal{L} \setminus \{\boldsymbol{0}\}} \|\boldsymbol{v}\|_2 \tag{3}$$

Shortest Vector Problem (SVP). Given a basis B of lattice $\mathcal{L}$, find a nonzero $\boldsymbol{v} \in \mathcal{L}$ such that $\|\boldsymbol{v}\|_2 = \lambda_1(\mathcal{L})$.

In practice, even the approximate version is hard:

Approximate SVP ($\mathbf{SVP}_\gamma$). Find nonzero $\boldsymbol{v} \in \mathcal{L}$ with $\|\boldsymbol{v}\|_2 \leq \gamma \cdot \lambda_1(\mathcal{L})$.

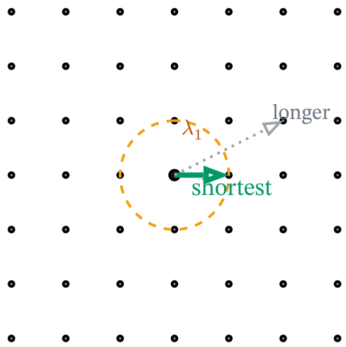

Figure 3: SVP intuition. The origin is the black dot; the green arrow is the shortest nonzero lattice vector. The dashed circle has radius λ_1 — no lattice point (except the origin) lies inside it.

Why SVP is hard:

Factor	Explanation
High dimension	In dimension $n \geq 500$, the number of candidate directions grows exponentially.
Exponential search space	Integer combinations $B\boldsymbol{z}$ for $\boldsymbol{z} \in \mathbb{Z}^m$ form an infinite discrete set.
Best algorithms are slow	The fastest known exact SVP algorithm runs in $2^{O(n)}$ time and space.
No quantum speedup	Unlike factoring, no efficient quantum algorithm for SVP is known.

1.4 Closest Vector Problem (CVP)

The CVP is the “sister problem” of SVP and is directly relevant to signatures.

CVP. Given a basis B of lattice $\mathcal{L}$ and a target point $\boldsymbol{t} \in \mathbb{R}^n$ (not necessarily on the lattice), find the lattice point closest to $\boldsymbol{t}$:

$$\text{find} \quad \boldsymbol{v}^* = \arg\min_{\boldsymbol{v}\in\mathcal{L}} \|\boldsymbol{v}-\boldsymbol{t}\|_2 \tag{4}$$

Connection to signatures: In ML-DSA, signing essentially requires solving a bounded-distance decoding problem — finding a lattice point within a certain radius of a target derived from the message hash. The signer can do this efficiently using the secret key (a good basis / trapdoor), while an attacker without the secret key faces a hard CVP instance.

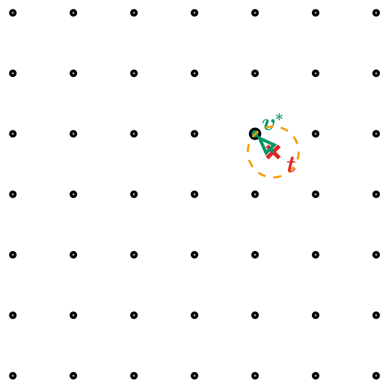

Figure 4: CVP: find the lattice point (green) closest to the target $\boldsymbol{t}$ (red cross). The dashed circle shows the distance to the nearest lattice point.

1.5 Short Integer Solutions (SIS)

Many lattice schemes (including ML-DSA verification) rely on the SIS problem.

SIS Problem. Given a random matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$ (with $m > n$), find a nonzero short vector $\boldsymbol{x} \in \mathbb{Z}^m$ such that:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0} \bmod q, \quad \|\boldsymbol{x}\|_\infty \leq \beta \tag{5}$$

The kernel $\{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0} \bmod q\}$ is a lattice. SIS asks: find a short vector in this lattice.

Why SIS is hard:
- Solutions exist (the kernel is a lattice of dimension $m - n$, so it has many vectors).
- Short solutions are rare — a random kernel vector has entries of order q, not β.
- Finding the short ones is as hard as worst-case lattice problems (Ajtai's theorem).

Geometric picture:

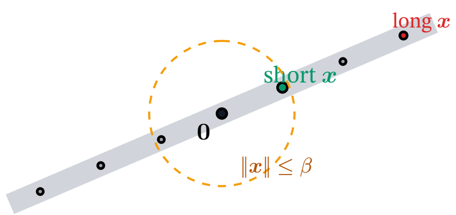

Figure 5: SIS geometry. The grey plane is the solution space $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$. Lattice points on this plane are plentiful, but only a few (green) are short — close to the origin.

Toy example over $q = 17$:

$$\boldsymbol{A} = \begin{pmatrix} 3 & 5 & 7 \end{pmatrix} \tag{6}$$

We need $3x_1 + 5x_2 + 7x_3 \equiv 0 \bmod 17$ with small x_i.

- $\boldsymbol{x} = (1, 2, 3)$: check $3 + 10 + 21 = 34 \equiv 0 \bmod 17$ ✓, and $\|\boldsymbol{x}\|_\infty = 3$ (short!).

- $\boldsymbol{x} = (0, 10, 5)$: also satisfies the equation, but $\|\boldsymbol{x}\|_\infty = 10$ (not short).

1.6 Learning With Errors (LWE) — the dual view

While SIS asks “find short $\boldsymbol{x}$ with $\boldsymbol{A}\boldsymbol{x} = \mathbf{0}$”, the LWE problem goes the other direction:

LWE Problem. Given $\boldsymbol{A} \in \mathbb{Z}_q^{n\times m}$ and $\boldsymbol{b} = \boldsymbol{A}^\top \boldsymbol{s} + \boldsymbol{e} \bmod q$ where $\boldsymbol{s}$ is secret and $\boldsymbol{e}$ is a short error vector, recover $\boldsymbol{s}$.

$$\boldsymbol{b} = \boldsymbol{A}^\top \boldsymbol{s} + \boldsymbol{e} \bmod q \tag{7}$$

The point $\boldsymbol{b}$ is close to the lattice generated by $\boldsymbol{A}^\top$ — it is a lattice point plus small noise. So LWE is essentially a CVP instance.

Problem	Given	Find
SIS	$\boldsymbol{A}$	short $\boldsymbol{x}$: $\boldsymbol{A}\boldsymbol{x} = \mathbf{0}$
LWE	$\boldsymbol{A}, \boldsymbol{b} = \boldsymbol{A}^\top \boldsymbol{s} + \boldsymbol{e}$	secret $\boldsymbol{s}$ (or distinguish from random)
SVP	basis B	shortest nonzero $\boldsymbol{v} \in \mathcal{L}(B)$
CVP	basis B, target $\boldsymbol{t}$	closest $\boldsymbol{v} \in \mathcal{L}(B)$ to $\boldsymbol{t}$

1.7 Module lattices (ML-DSA view)

ML-DSA does not use plain integer lattices — it uses module lattices over a polynomial ring. This gives both structure (for efficiency) and hardness (from the underlying lattice problems).

The ring:

$$R_q = \mathbb{Z}_q[X]/(X^n + 1), \quad n = 256, \quad q = 8380417 \tag{8}$$

Each element of R_q is a polynomial of degree < 256 with coefficients in $\mathbb{Z}_q$. Addition and multiplication follow polynomial arithmetic modulo $X^{\{256\}} + 1$.

From ring to module: A module vector $\boldsymbol{v} \in R_q^k$ has k polynomial entries, each with 256 coefficients. So $\boldsymbol{v}$ corresponds to $256k$ integers — a point in a high-dimensional lattice.

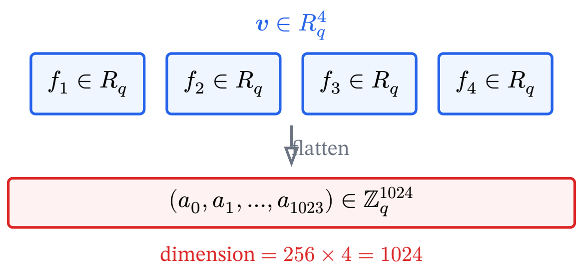

Figure 6: Module-lattice structure: each polynomial block contributes 256 coefficients, so a vector in R_q^k maps to a point in dimension $256k$.

Why modules?

Benefit	Explanation
Fast arithmetic	Polynomial multiplication via NTT in $O(n \log n)$ instead of $O(n^2)$.
Compact keys	A $k \times l$ matrix over R_q stores $256kl$ coefficients, but behaves like a $256k \times 256l$ integer matrix.

Benefit	Explanation
Tunable security	Increase k (module rank) to raise the lattice dimension without changing the ring.
Worst-case hardness	Module-SIS and Module-LWE reduce to worst-case lattice problems (under standard assumptions).

ML-DSA parameter sets:

Variant	(k, l)	Lattice dim	Security	NIST level
ML-DSA-44	$(4, 4)$	$256 \times 4 = 1024$	128 bit	2
ML-DSA-65	$(6, 5)$	$256 \times 6 = 1536$	192 bit	3
ML-DSA-87	$(8, 7)$	$256 \times 8 = 2048$	256 bit	5

1.8 How ML-DSA uses these ideas

ML-DSA is a Fiat-Shamir with Aborts signature scheme built on Module-LWE and Module-SIS. This section walks through every phase with figures.

1.8.1 Big picture

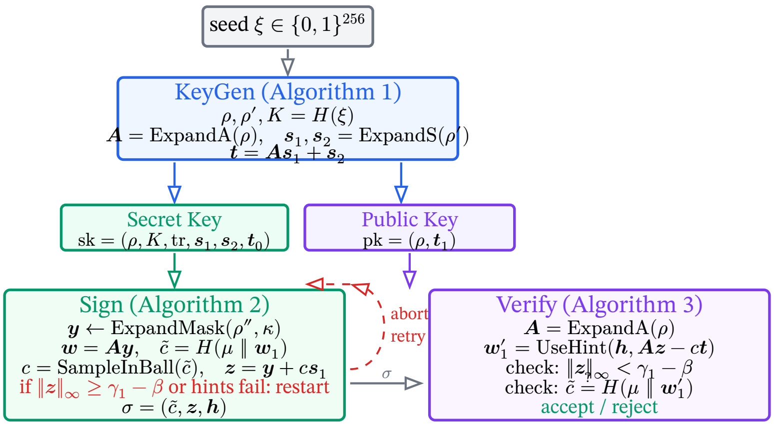

Figure 7: ML-DSA lifecycle. Three phases share the public matrix $\boldsymbol{A}$. The secret key enables efficient signing; the public key enables verification. The signature $(\tilde{c}, \boldsymbol{z}, \boldsymbol{h})$ is short — this is the lattice constraint.

1.8.2 Phase 1: Key Generation

Key generation creates a Module-LWE instance. The public key hides the short secrets.

Step by step (FIPS 204, Algorithm 1 / Algorithm 6):

1. Hash the 256-bit seed ξ to get three sub-seeds: ρ (for $\boldsymbol{A}$), ρ' (for secrets), K (for signing randomness).
2. Expand the public matrix: $\hat{\boldsymbol{A}} = \mathrm{ExpandA}(\rho) \in R_q^{k \times l}$. This is deterministic — anyone with ρ can reconstruct $\boldsymbol{A}$.
3. Sample short secrets: $\boldsymbol{s}_1 \in R_q^l$ and $\boldsymbol{s}_2 \in R_q^k$ from $\mathrm{ExpandS}(\rho')$. Each coefficient satisfies $|\mathrm{coeff}| \leq \eta$ (where $\eta \in \{2, 4\}$ depending on the parameter set).
4. Compute the public vector: $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s}_1 + \boldsymbol{s}_2 \in R_q^k$.
5. Split $\boldsymbol{t}$: write $\boldsymbol{t} = \boldsymbol{t}_1 \cdot 2^d + \boldsymbol{t}_0$ (Power2Round). Only $\boldsymbol{t}_1$ goes into the public key; $\boldsymbol{t}_0$ stays in the secret key.

$$\mathrm{pk} = (\rho, \boldsymbol{t}_1), \quad \mathrm{sk} = (\rho, K, \mathrm{tr}, \boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{t}_0) \tag{9}$$

where $\mathrm{tr} = H(\mathrm{pk})$ is a hash of the public key used during signing.

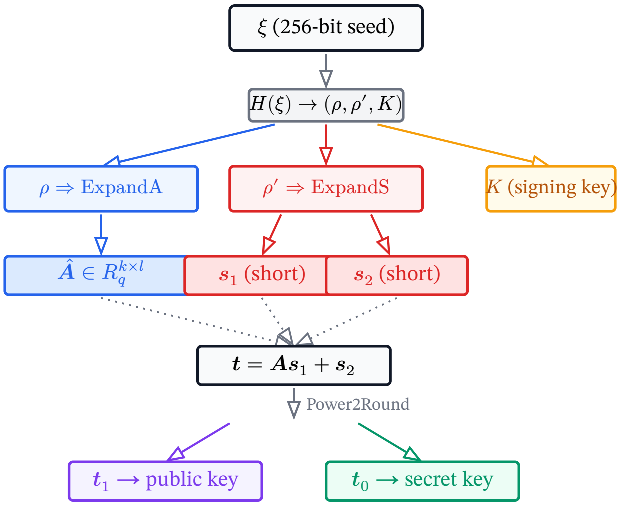

Figure 8: Key generation data flow. The seed ξ fans out into three sub-seeds. The public matrix $\boldsymbol{A}$ and short secrets $\boldsymbol{s}_1, \boldsymbol{s}_2$ combine into $\boldsymbol{t}$, which is split into public $\boldsymbol{t}_1$ and secret $\boldsymbol{t}_0$.

Geometric view — why the public key hides the secret:

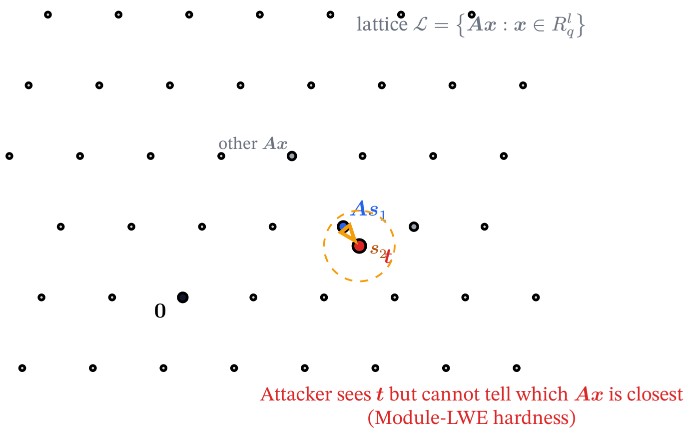

Figure 9: Module-LWE geometry. The lattice consists of all points $\boldsymbol{Ax}$ for $\boldsymbol{x} \in R_q^l$. The public key $\boldsymbol{t}$ equals $\boldsymbol{As}_1$ (a lattice point) plus the short error $\boldsymbol{s}_2$. Recovering $\boldsymbol{s}_1$ means finding which lattice point is closest to $\boldsymbol{t}$ — a hard CVP instance without the trapdoor.

1.8.3 Phase 2: Signing

Signing is the most intricate phase. It is an interactive proof made non-interactive via the Fiat-Shamir transform, with an abort mechanism to prevent secret leakage.

Step by step (FIPS 204, Algorithm 2 / Algorithm 7):

1. Compute $\mu = H(\mathrm{tr} \parallel M)$ where M is the message and $\mathrm{tr} = H(\mathrm{pk})$.
2. Compute $\rho'' = H(K \parallel \mathrm{rnd} \parallel \mu)$ — the per-signature randomness seed.
3. Initialize counter $\kappa = 0$.
4. Loop (may repeat multiple times):
   1. Mask: $\boldsymbol{y} = \mathrm{ExpandMask}(\rho'', \kappa) \in R_q^l$ with $|\mathrm{coeff}| < \gamma_1$.
   2. Commit: $\boldsymbol{w} = \boldsymbol{Ay}$, then decompose $\boldsymbol{w} = \boldsymbol{w}_1 \cdot 2\gamma_2 + \boldsymbol{w}_0$.
   3. Challenge: $\tilde{c} = H(\mu \parallel \mathrm{Encode}(\boldsymbol{w}_1))$, then $c = \mathrm{SampleInBall}(\tilde{c}) \in R_q$ (a sparse polynomial with τ nonzero ± 1 coefficients).
   4. Response: $\boldsymbol{z} = \boldsymbol{y} + c\boldsymbol{s}_1$.

5. Check 1: if $\|\boldsymbol{z}\|_\infty \geq \gamma_1 - \beta$, set $\kappa \leftarrow \kappa + l$ and go to step 4.
6. Check 2: compute $\boldsymbol{r}_0 = \boldsymbol{w} - c\boldsymbol{s}_2$, extract low bits. If $\|\boldsymbol{r}_0\|_\infty \geq \gamma_2 - \beta$, restart.
7. Hints: $\boldsymbol{h} = \mathrm{MakeHint}(\boldsymbol{r}_0, \boldsymbol{w})$. If the number of nonzero hints exceeds ω, restart.
8. Output: $\sigma = (\tilde{c}, \boldsymbol{z}, \boldsymbol{h})$.

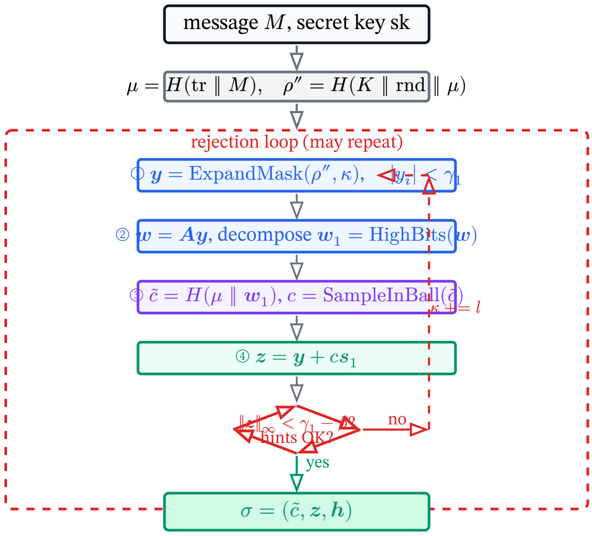

Figure 10: Signing flowchart. The inner loop (dashed red border) may execute multiple times due to rejection sampling. Each restart uses a fresh mask $\boldsymbol{y}$ via an incremented counter κ.

Geometric view — the mask-challenge-response triangle:

The key geometric insight is that $\boldsymbol{z} = \boldsymbol{y} + c\boldsymbol{s}_1$ is a vector sum. The mask $\boldsymbol{y}$ is large and random; the shift $c\boldsymbol{s}_1$ is small (because both c and $\boldsymbol{s}_1$ are short). The result $\boldsymbol{z}$ must land inside the acceptance region.

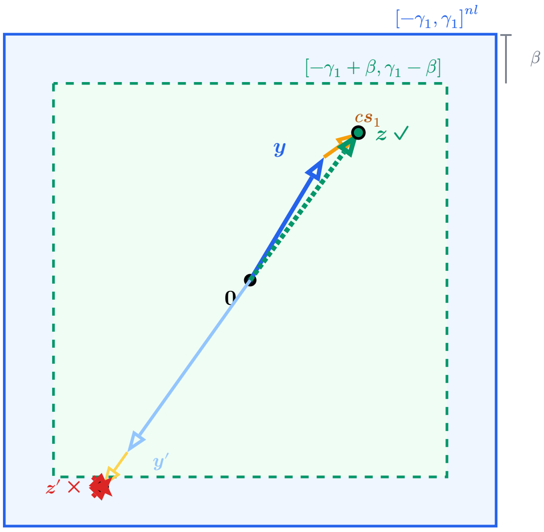

Figure 11: Signing geometry in coefficient space. The signer picks a random mask $\boldsymbol{y}$ (blue) uniformly in the large box $[-\gamma_1, \gamma_1]^{nl}$. The challenge shifts it by $c\boldsymbol{s}_1$ (orange, small). If the result $\boldsymbol{z}$ (green) stays inside the acceptance region $[-\gamma_1 + \beta, \gamma_1 - \beta]^{nl}$ (dashed), the signature is accepted. Otherwise the signer restarts.

Why rejection sampling is essential:

Without rejection, the distribution of $\boldsymbol{z} = \boldsymbol{y} + c\boldsymbol{s}_1$ depends on $\boldsymbol{s}_1$. An attacker who sees many signatures could statistically recover $\boldsymbol{s}_1$. Rejection sampling ensures that the conditional distribution of $\boldsymbol{z}$ (given that it is output) is uniform over the acceptance region, independent of $\boldsymbol{s}_1$.

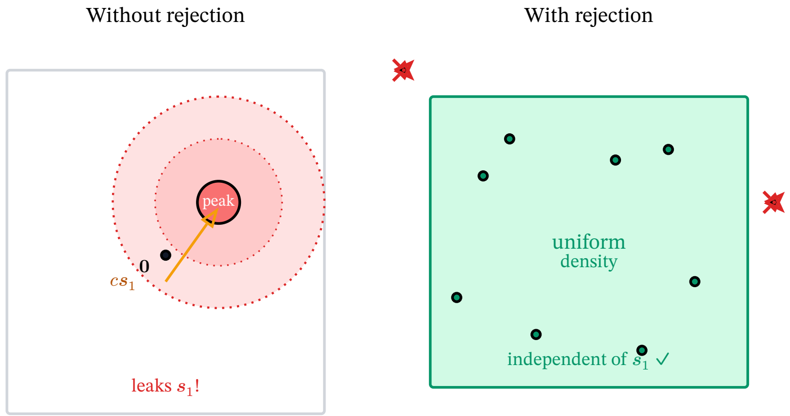

Figure 12: Rejection sampling removes secret-key dependence. Left: without rejection, the distribution of $\boldsymbol{z}$ is centered at $c\boldsymbol{s}_1$ (leaks the secret). Right: with rejection, only $\boldsymbol{z}$ values inside the acceptance box are output — the distribution is uniform and independent of $\boldsymbol{s}_1$.

1.8.4 Phase 3: Verification

Verification reconstructs the commitment $\boldsymbol{w}_1$ from the signature and public key, then checks the hash.

Step by step (FIPS 204, Algorithm 3 / Algorithm 8):

1. Reconstruct $\boldsymbol{A} = \mathrm{ExpandA}(\rho)$ from the public key.
2. Recover the challenge polynomial: $c = \mathrm{SampleInBall}(\tilde{c})$.
3. Compute $\boldsymbol{A}\boldsymbol{z} - c\boldsymbol{t}$ (using NTT for speed).
4. Apply hints: $\boldsymbol{w}_1' = \mathrm{UseHint}(\boldsymbol{h}, \boldsymbol{A}\boldsymbol{z} - c\boldsymbol{t})$.
5. Check norm: $\|\boldsymbol{z}\|_\infty < \gamma_1 - \beta$.
6. Check hash: $\tilde{c} \stackrel{?}{=} H(\mu \parallel \mathrm{Encode}(\boldsymbol{w}_1'))$.

Why it works — the algebra:

$$
\begin{aligned}
\boldsymbol{A}\boldsymbol{z} - c\boldsymbol{t} &= \boldsymbol{A}(\boldsymbol{y} + c\boldsymbol{s}_1) - c(\boldsymbol{A}\boldsymbol{s}_1 + \boldsymbol{s}_2) \\
&= \boldsymbol{A}\boldsymbol{y} + c\boldsymbol{A}\boldsymbol{s}_1 - c\boldsymbol{A}\boldsymbol{s}_1 - c\boldsymbol{s}_2 \\
&= \boldsymbol{A}\boldsymbol{y} - c\boldsymbol{s}_2 \\
&= \boldsymbol{w} - c\boldsymbol{s}_2
\end{aligned}
\tag{10}
$$

Since $c\boldsymbol{s}_2$ is short (both c and $\boldsymbol{s}_2$ are short), the high bits of $\boldsymbol{A}\boldsymbol{z} - c\boldsymbol{t}$ match the high bits of $\boldsymbol{w}$. The hints $\boldsymbol{h}$ correct any remaining carry discrepancies, so $\boldsymbol{w}_1' = \boldsymbol{w}_1$ exactly.

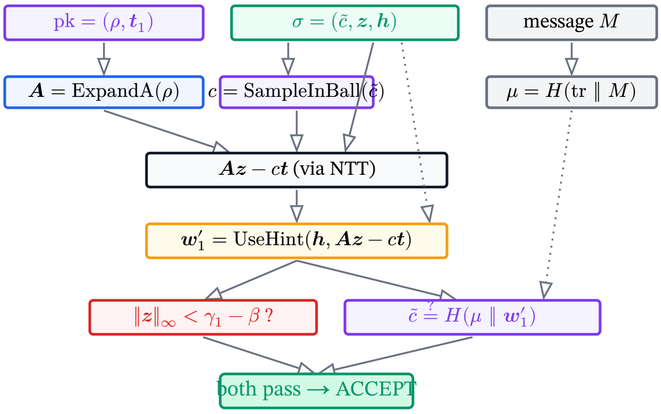

Figure 13: Verification data flow. The verifier uses only public information (pk, σ, message) to recompute $\boldsymbol{w}_1'$ and check the hash. The hints $\boldsymbol{h}$ correct rounding errors so that $\boldsymbol{w}_1' = \boldsymbol{w}_1$ exactly.

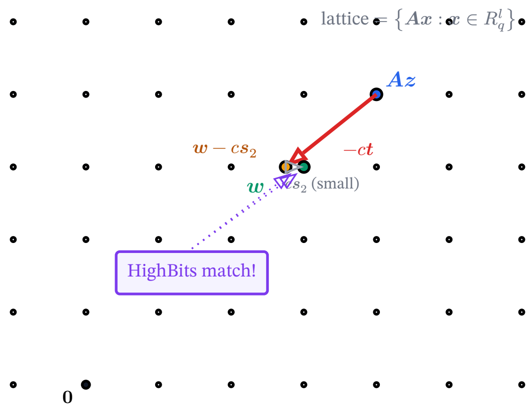

Figure 14: Verification algebra visualized. The verifier computes $\boldsymbol{Az}$ (a lattice point), subtracts $c\boldsymbol{t}$, and gets $\boldsymbol{w} - c\boldsymbol{s}_2$. Since $c\boldsymbol{s}_2$ is short, the high bits match $\boldsymbol{w}_1$, and the hash check succeeds.

1.8.5 Security: what an attacker faces

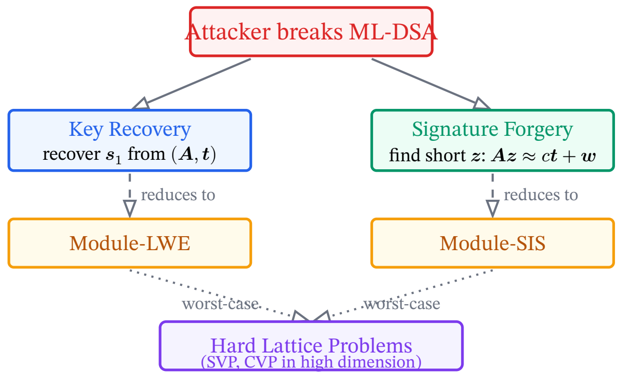

Figure 15: Security reduction tree. Any successful attack on ML-DSA implies an efficient algorithm for either Module-LWE or Module-SIS, both of which reduce to worst-case lattice problems believed to be hard even for quantum computers.

In summary:

- Key recovery ⇒ solve Module-LWE: given $\boldsymbol{A}$ and $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s}_1 + \boldsymbol{s}_2$, find the short $\boldsymbol{s}_1$.
- Forgery ⇒ solve Module-SIS: find a short $\boldsymbol{z}$ satisfying the linear constraint $\boldsymbol{Az} \equiv c\boldsymbol{t} + \boldsymbol{w} \bmod q$.
- Both Module-LWE and Module-SIS are at least as hard as worst-case lattice problems in dimension nk (Langlois & Stehlé, 2015).

- No known quantum algorithm provides a significant speedup for these problems.

1.9 Summary

Concept	Role in ML-DSA	Intuition
Lattice	Underlying algebraic structure	Infinite grid of integer-combination points
SVP / CVP	Source of computational hardness	Finding short / close vectors is hard in high dimensions
SIS	Signature verification	Short $\boldsymbol{x}$: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$ is hard to find
LWE	Key generation	$\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$ hides $\boldsymbol{s}$
Module structure	Efficiency + compact keys	Polynomial ring gives NTT speedup and smaller representations
Rejection sampling	Signature security	Ensures $\boldsymbol{z}$ leaks nothing about $\boldsymbol{s}_1$

Bottom line: Geometry gives the hardness (short vectors are hard to find). Algebra gives the efficiency (polynomial rings enable NTT). ML-DSA combines both into a practical post-quantum signature scheme.