LECTURE 3: ML-KEM AND ML-DSA

Post-Quantum Cryptography

Florida Atlantic University

Fall 2026



Outline

01 Key encapsulation

02 Toy Kyber

03 From toy Kyber to ML-KEM

04 Signatures: from Schnorr to Dilithium

05 Toy Dilithium

06 From toy Dilithium to ML-DSA

07 Summary

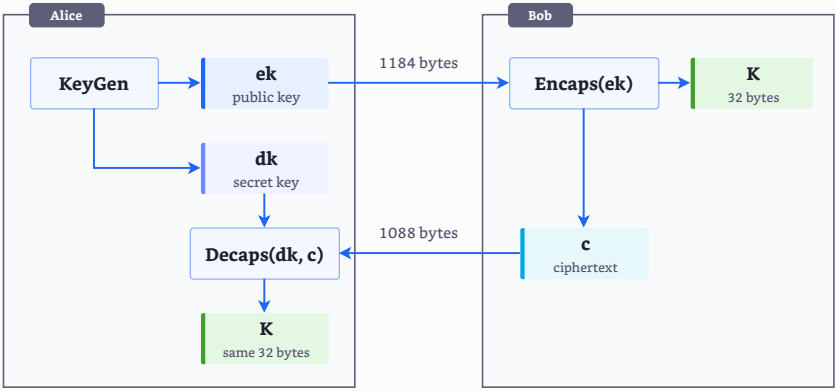
We have MLWE and fast multiplication. How do we exchange a key and sign a message with them?

01

Key encapsulation

A KEM instead of Diffie-Hellman

- Diffie-Hellman: both sides send a share and combine them. A **KEM** (key encapsulation mechanism) works differently: Bob picks the key and sends it to Alice, locked with her public key.



- Three algorithms: KeyGen, Encaps, Decaps. Sizes shown for ML-KEM-768.

ML-KEM is already in use

- TLS 1.3 in current browsers and at Cloudflare uses a **hybrid**: X25519 (ECDH) and ML-KEM-768 together. The session key mixes both, so an attacker must break both.

Bytes on the wire	**client → server**	**server → client**
X25519 alone	32	32
ML-KEM-768 alone	1184	1088
hybrid X25519MLKEM768	1216	1120

- More than 30 times the data of ECDH alone, about 2.3 KB per handshake instead of 64 bytes.

02

Toy Kyber

Key generation: the Lecture 1 example

- KeyGen is an MLWE instance: public key $(\mathbf{A}, \mathbf{t})$, secret key $\mathbf{s}$. We reuse the toy instance from Lecture 1 ($q = 17$, $n = 4$, $k = 2$):

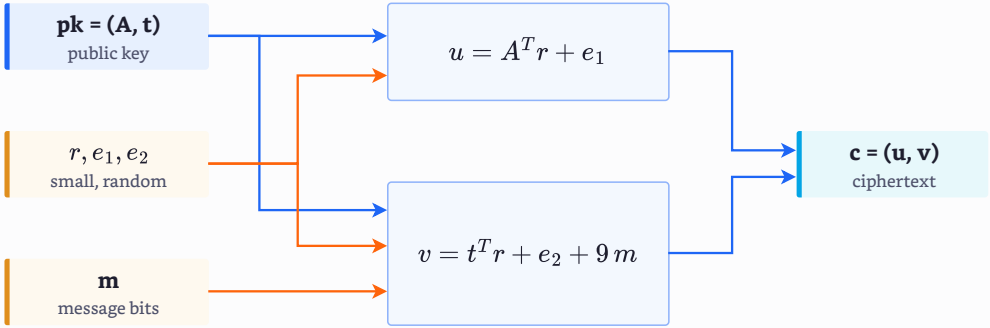
$$\begin{bmatrix} (3, 10, 0, 7) & (12, 1, 5, 9) \\ (6, 14, 2, 0) & (4, 8, 11, 16) \end{bmatrix} \cdot \begin{bmatrix} (1, 0, -1, 2) \\ (0, -2, 1, 1) \end{bmatrix} + \begin{bmatrix} (1, -1, 0, 0) \\ (0, 1, 0, -1) \end{bmatrix} = \begin{bmatrix} (13, 12, 1, 6) \\ (10, 10, 2, 4) \end{bmatrix}$$

public, random | secret, small | noise, small | public

- Alice publishes $\mathbf{A}$ and $\mathbf{t}$, keeps $\mathbf{s}$, and throws $\mathbf{e}$ away.

Encryption

- Bob picks fresh small values $\mathbf{r}, \mathbf{e}_1, e_2$ and computes two things:



- **u** is a new MLWE sample, so it hides **r**. v hides the message under $\mathbf{t}^T\mathbf{r}$, which looks random.

Encryption with numbers

- Message $m = (1, 0, 1, 1)$. Bob adds each bit as 0 or 9 (about $q/2$), so $9m = (9, 0, 9, 9)$.

Bob's small random values

$\mathbf{r} = (1 + x,\ 1 - x)$

$\mathbf{e}_1 = (1 + x,\ x^3)$

$e_2 = x^2 + x^3$

Ciphertext

$u_0 = A_{00}r_0 + A_{10}r_1 + e_{1,0} = (3, 5, 15, 5)$

$u_1 = A_{01}r_0 + A_{11}r_1 + e_{1,1} = (6, 0, 9, 3)$

$v = t_0r_0 + t_1r_1 + e_2 + 9m = (13, 8, 15, 2)$

- Every product is a ring multiplication from Lecture 1, done with the NTT from Lecture 2 in real code.
- Bob sends $c = (\mathbf{u}, v)$: 12 numbers that look random. The message bits are not visible in v.

Encryption with numbers

- Message $m = (1, 0, 1, 1)$. Bob adds each bit as 0 or 9 (about $q/2$), so $9m = (9, 0, 9, 9)$.

Bob's small random values

$\mathbf{r} = (1 + x,\ 1 - x)$

$\mathbf{e}_1 = (1 + x,\ x^3)$

$e_2 = x^2 + x^3$

Ciphertext

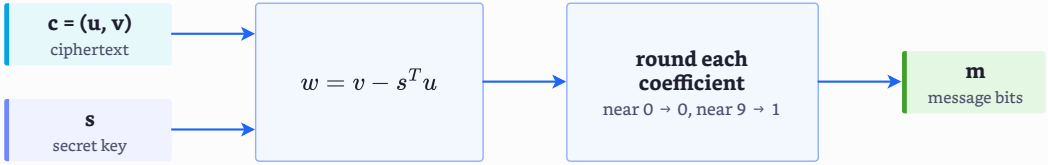
$u_0 = A_{00}r_0 + A_{10}r_1 + e_{1,0} = (3, 5, 15, 5)$

$u_1 = A_{01}r_0 + A_{11}r_1 + e_{1,1} = (6, 0, 9, 3)$

$v = t_0r_0 + t_1r_1 + e_2 + 9m = (13, 8, 15, 2)$

- Every product is a ring multiplication from Lecture 1, done with the NTT from Lecture 2 in real code.
- Bob sends $c = (\mathbf{u}, v)$: 12 numbers that look random. The message bits are not visible in v.

Decryption with numbers



Alice computes

$s_0 u_0 + s_1 u_1 = (5, 7, 5, 11)$

$w = v - \mathbf{s}^T\mathbf{u} = (13-5,\ 8-7,\ 15-5,\ 2-11) = (8, 1, 10, 8)$

round: $8 \to 1,\ \ 1 \to 0,\ \ 10 \to 1,\ \ 8 \to 1 \qquad m = (1, 0, 1, 1)$

Why decryption works

- Use $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ and expand:

$$\begin{aligned} w = v - \mathbf{s}^T\mathbf{u} &= (\mathbf{t}^T\mathbf{r} + e_2 + 9m) - \mathbf{s}^T(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1) \\ &= \cancel{\mathbf{s}^T\mathbf{A}^T\mathbf{r}} + \mathbf{e}^T\mathbf{r} + e_2 + 9m - \cancel{\mathbf{s}^T\mathbf{A}^T\mathbf{r}} - \mathbf{s}^T\mathbf{e}_1 \\ &= 9m + \underbrace{\mathbf{e}^T\mathbf{r} + e_2 - \mathbf{s}^T\mathbf{e}_1}_{\text{small noise}} \end{aligned}$$

In our example

$\text{noise} = (-1, 1, 1, -1), \quad 9m + \text{noise} = (8, 1, 10, 8) = w$

- The big term $\mathbf{s}^T\mathbf{A}^T\mathbf{r}$ cancels only for the owner of $\mathbf{s}$. Everyone else sees random-looking numbers.

Why decryption works

- Use $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ and expand:

$$\begin{aligned} w = v - \mathbf{s}^T\mathbf{u} &= (\mathbf{t}^T\mathbf{r} + e_2 + 9m) - \mathbf{s}^T(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1) \\ &= \cancel{\mathbf{s}^T\mathbf{A}^T\mathbf{r}} + \mathbf{e}^T\mathbf{r} + e_2 + 9m - \cancel{\mathbf{s}^T\mathbf{A}^T\mathbf{r}} - \mathbf{s}^T\mathbf{e}_1 \\ &= 9m + \underbrace{\mathbf{e}^T\mathbf{r} + e_2 - \mathbf{s}^T\mathbf{e}_1}_{\text{small noise}} \end{aligned}$$

In our example

$\text{noise} = (-1, 1, 1, -1), \quad 9m + \text{noise} = (8, 1, 10, 8) = w$

- The big term $\mathbf{s}^T\mathbf{A}^T\mathbf{r}$ cancels only for the owner of $\mathbf{s}$. Everyone else sees random-looking numbers.

When decryption fails

- The noise is a sum of many small products. With $q = 17$ there is almost no room:

Scheme	Room for noise ($q/4$)	Chance a decryption fails
our toy, $q = 17$	4	54% to 66% (depends on the message)
ML-KEM-512	832	2^{-139}
ML-KEM-768	832	2^{-165}
ML-KEM-1024	832	2^{-175}

- We picked the toy values by hand so that our example works.
- $q = 3329$ leaves enough room that a failure is never expected in practice.

When decryption fails

- The noise is a sum of many small products. With $q = 17$ there is almost no room:

Scheme	Room for noise ($q/4$)	Chance a decryption fails
our toy, $q = 17$	4	54% to 66% (depends on the message)
ML-KEM-512	832	2^{-139}
ML-KEM-768	832	2^{-165}
ML-KEM-1024	832	2^{-175}

- We picked the toy values by hand so that our example works.
- $q = 3329$ leaves enough room that a failure is never expected in practice.

03

From toy Kyber to ML-KEM

What changes

Part	Toy	ML-KEM-768 (FIPS 203)
ring	$q = 17, n = 4, k = 2$	$q = 3329, n = 256, k = 3$
matrix **A**	written out	expanded from a 32-byte seed ρ
small values	chosen by hand	sampled from a binomial distribution
multiplication	schoolbook	NTT, 7 stages
ciphertext	full numbers	compressed
use	encrypt a message	KEM with a re-encryption check

- The core, $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$ and $w = v - \mathbf{s}^T\mathbf{u}$, stays exactly the same.

A seed instead of a matrix

- **A** is 3×3 polynomials of 256 coefficients, 12 bits each: 3,456 bytes.
- Instead, Alice publishes a random 32-byte seed ρ. Both sides feed it into SHAKE128 and read the coefficients of **A** from the output.

Two details

- SHAKE128 gives 12-bit numbers. Values of 3329 or more are skipped, so every coefficient is uniform mod q.
- **A** is random anyway, so it is read directly as NTT values (Lecture 2) and never transformed.

- The public key becomes $(\hat{\mathbf{t}}, \rho)$: $3 \cdot 384 + 32 = 1184$ bytes.

A seed instead of a matrix

- **A** is 3×3 polynomials of 256 coefficients, 12 bits each: 3,456 bytes.
- Instead, Alice publishes a random 32-byte seed ρ. Both sides feed it into SHAKE128 and read the coefficients of **A** from the output.

Two details

- SHAKE128 gives 12-bit numbers. Values of 3329 or more are skipped, so every coefficient is uniform mod q.
- **A** is random anyway, so it is read directly as NTT values (Lecture 2) and never transformed.

- The public key becomes $(\hat{\mathbf{t}}, \rho)$: $3 \cdot 384 + 32 = 1184$ bytes.

A seed instead of a matrix

- **A** is 3×3 polynomials of 256 coefficients, 12 bits each: 3,456 bytes.
- Instead, Alice publishes a random 32-byte seed ρ. Both sides feed it into SHAKE128 and read the coefficients of **A** from the output.

Two details

- SHAKE128 gives 12-bit numbers. Values of 3329 or more are skipped, so every coefficient is uniform mod q.
- **A** is random anyway, so it is read directly as NTT values (Lecture 2) and never transformed.

- The public key becomes $(\hat{\mathbf{t}}, \rho)$: $3 \cdot 384 + 32 = 1184$ bytes.

Small values from coin flips

- ML-KEM-768 needs coefficients between -2 and 2, with small values more likely.
- Centred binomial distribution with $\eta = 2$: flip 4 random bits $b_1 b_2 b_3 b_4$ and output $(b_1 + b_2) - (b_3 + b_4)$.

value	-2	-1	0	1	2
probability	$1/16$	$4/16$	$6/16$	$4/16$	$1/16$

- Example: bits $1, 1, 0, 1$ give $(1 + 1) - (0 + 1) = 1$.
- The bits come from SHAKE256 of a secret 32-byte seed, so the whole secret key can be regenerated from 32 bytes.

Compression: fewer bits on the wire

- Decryption tolerates noise, so the ciphertext does not need all 12 bits per coefficient. Keep only the top d bits:

$$\text{Compress}_d(x) = \left\lceil \frac{2^d}{q}\, x \right\rfloor \bmod 2^d, \qquad \text{Decompress}_d(y) = \left\lceil \frac{q}{2^d}\, y \right\rfloor$$

$d = 10$ **(used for u)**

$2000 \to 615 \to 1999$

error 1

$d = 4$ **(used for v)**

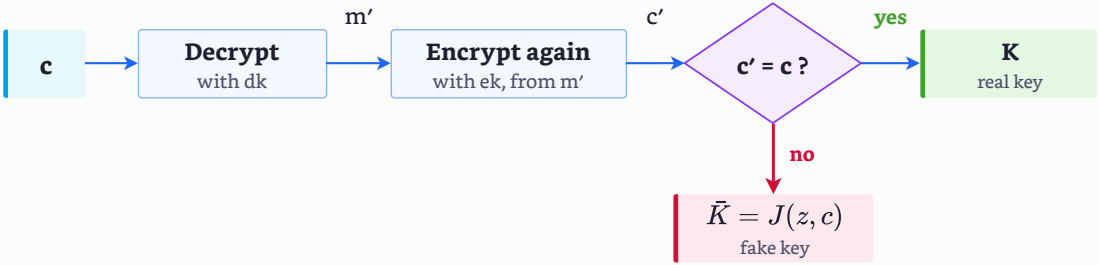
$2000 \to 10 \to 2081$

error 81, still far below $q/4 = 832$

- Ciphertext for ML-KEM-768: $3 \cdot 256 \cdot 10 + 256 \cdot 4$ bits $= 960 + 128 = 1088$ bytes, instead of 1536.

From encryption to a KEM

- Encaps: pick a random 32-byte m (256 bits, one per coefficient). Hash it into the key and the randomness: $(K, r) = G(m \,\|\, H(\mathsf{ek}))$. Encrypt m using r.
- Decaps: decrypt to get m', then rebuild everything from m' and check:



- This re-encryption check is the Fujisaki-Okamoto (FO) transform.

Why the re-encryption check?

- Without it, an attacker can send Alice slightly changed ciphertexts and watch whether the session works.
- Each answer leaks a little about $\mathbf{s}$. Enough queries recover the whole key.

With the check

- A changed ciphertext never re-encrypts to itself, so it is always caught.
- Alice then returns a fake key $\bar{K} = J(z, c)$ from a secret seed z. To the attacker it looks like any other key, so even the failure reveals nothing.

Why the re-encryption check?

- Without it, an attacker can send Alice slightly changed ciphertexts and watch whether the session works.
- Each answer leaks a little about $\mathbf{s}$. Enough queries recover the whole key.

With the check

- A changed ciphertext never re-encrypts to itself, so it is always caught.
- Alice then returns a fake key $\bar{K} = J(z, c)$ from a secret seed z. To the attacker it looks like any other key, so even the failure reveals nothing.

Where the NTT is used

Step	Computation
KeyGen	$\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \mathrm{NTT}(\mathbf{s}) + \mathrm{NTT}(\mathbf{e})$ (stays in NTT form)
Encrypt	$\hat{\mathbf{r}} = \mathrm{NTT}(\mathbf{r}),\ \mathbf{u} = \mathrm{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1,\ v = \mathrm{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \mu$
Decrypt	$w = v - \mathrm{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \mathrm{NTT}(\mathbf{u}))$

- $\circ$ is the pointwise product of Lecture 2 (BaseCaseMultiply on 128 pairs).
- In optimized software, a large share of the time goes to SHAKE, the hashing of seeds.

ML-KEM parameter sets

Set	k	η	(d_u, d_v)	ek	dk	ciphertext	security
ML-KEM-512	2	3	(10, 4)	800	1632	768	128-bit
ML-KEM-768	3	2	(10, 4)	1184	2400	1088	192-bit
ML-KEM-1024	4	2	(11, 5)	1568	3168	1568	256-bit
X25519 (ECDH)				32	32	32	128-bit
RSA-2048				256	256	256	112-bit

- Sizes in bytes. All three sets share $n = 256$ and $q = 3329$; η is for **s** and **e**. The shared key is always 32 bytes.

04

Signatures: from Schnorr to Dilithium

Recap: Schnorr signatures

The scheme

secret x, public $\mathrm{pk} = g^x$

Sign: random y, $w = g^y$,
$c = H(M, w)$, $z = y + c\,x$

Verify: $w' = g^z \cdot \mathrm{pk}^{-c}$,
accept if $H(M, w') = c$

Numbers: $p = 23$, $g = 5$

$x = 6$, $\mathrm{pk} = 5^6 = 8$

$y = 3$, $w = 5^3 = 10$

suppose H gives $c = 7$

$z = 3 + 7 \cdot 6 = 45 \equiv 1 \pmod{22}$

$w' = 5^1 \cdot 8^{-7} = 5 \cdot 2 = 10$ ✓

- Same key as Alice in the Lecture 1 Diffie-Hellman example. Exponents work mod 22, the order of g.

The same three moves with lattices

	Schnorr	**Dilithium (toy)**
secret	x	small $\mathbf{s}_1, \mathbf{s}_2$
public key	g^x	$\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
commit	$w = g^y$	$\mathbf{w} = \mathbf{A}\mathbf{y}$
challenge	$c = H(M, w)$	$c = H(M, \mathrm{HighBits}(\mathbf{w}))$, a small polynomial
response	$z = y + c\,x$	$\mathbf{z} = \mathbf{y} + c\,\mathbf{s}_1$
check	$g^z\,\mathrm{pk}^{-c} = w$	$\mathbf{A}\mathbf{z} - c\,\mathbf{t} = \mathbf{w} - c\,\mathbf{s}_2 \approx \mathbf{w}$

- Two new problems: the check is only approximate ($\approx$), and $\mathbf{z}$ could leak $\mathbf{s}_1$.

The same three moves with lattices

	Schnorr	**Dilithium (toy)**
secret	x	small $\mathbf{s}_1, \mathbf{s}_2$
public key	g^x	$\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
commit	$w = g^y$	$\mathbf{w} = \mathbf{A}\mathbf{y}$
challenge	$c = H(M, w)$	$c = H(M, \mathrm{HighBits}(\mathbf{w}))$, a small polynomial
response	$z = y + c\,x$	$\mathbf{z} = \mathbf{y} + c\,\mathbf{s}_1$
check	$g^z\,\mathrm{pk}^{-c} = w$	$\mathbf{A}\mathbf{z} - c\,\mathbf{t} = \mathbf{w} - c\,\mathbf{s}_2 \approx \mathbf{w}$

- Two new problems: the check is only approximate ($\approx$), and $\mathbf{z}$ could leak $\mathbf{s}_1$.

Two problems, two fixes

Problem 1: the verifier gets $\mathbf{w} - c\,\mathbf{s}_2$, not $\mathbf{w}$

Fix: hash only the *high bits* of $\mathbf{w}$. A small change ($c\,\mathbf{s}_2$) leaves the high bits alone, unless $\mathbf{w}$ sits near a boundary. The signer checks for that.

Problem 2: $\mathbf{z} = \mathbf{y} + c\,\mathbf{s}_1$ can reveal $\mathbf{s}_1$

$\mathbf{y}$ is random in a fixed range. Values of $\mathbf{z}$ near the edge of that range happen more often for one sign of $c\,\mathbf{s}_1$ than for the other.

Fix: **rejection sampling**. Publish $\mathbf{z}$ only if all its coefficients stay well inside the range. Otherwise start over with a new $\mathbf{y}$.

Two problems, two fixes

Problem 1: the verifier gets $\mathbf{w} - c\,\mathbf{s}_2$, not $\mathbf{w}$

Fix: hash only the *high bits* of $\mathbf{w}$. A small change ($c\,\mathbf{s}_2$) leaves the high bits alone, unless $\mathbf{w}$ sits near a boundary. The signer checks for that.

Problem 2: $\mathbf{z} = \mathbf{y} + c\,\mathbf{s}_1$ can reveal $\mathbf{s}_1$

$\mathbf{y}$ is random in a fixed range. Values of $\mathbf{z}$ near the edge of that range happen more often for one sign of $c\,\mathbf{s}_1$ than for the other.

Fix: **rejection sampling**. Publish $\mathbf{z}$ only if all its coefficients stay well inside the range. Otherwise start over with a new $\mathbf{y}$.

05

Toy Dilithium

Toy parameters and keys

- Signatures need more room than our KEM toy, so: $q = 97$, $n = 4$, and one polynomial per key part ($k = \ell = 1$).

Parameters

$\mathbf{y}$ coefficients in $(-16, 16]$ $(\gamma_1 = 16)$

high bits in steps of 16 $(\gamma_2 = 8)$

c has two coefficients ± 1 $(\tau = 2)$

$\mathbf{s}_1, \mathbf{s}_2$ in $\{-1, 0, 1\}$ $(\eta = 1)$

$\beta = \tau\eta = 2$: the largest size of $c\,\mathbf{s}$

Keys

$a = (23, 61, 8, 45)$

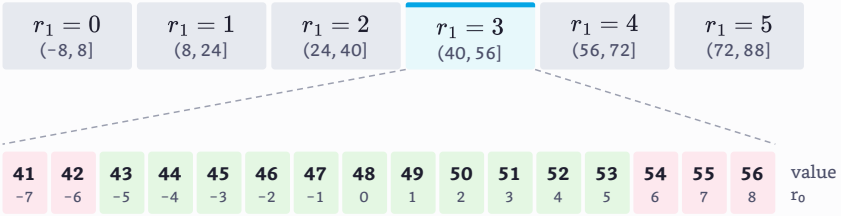
$s_1 = (1, -1, 0, 1)$

$s_2 = (0, 1, -1, 0)$

$t = a\,s_1 + s_2 = (7, 31, 95, 60)$

High bits and low bits

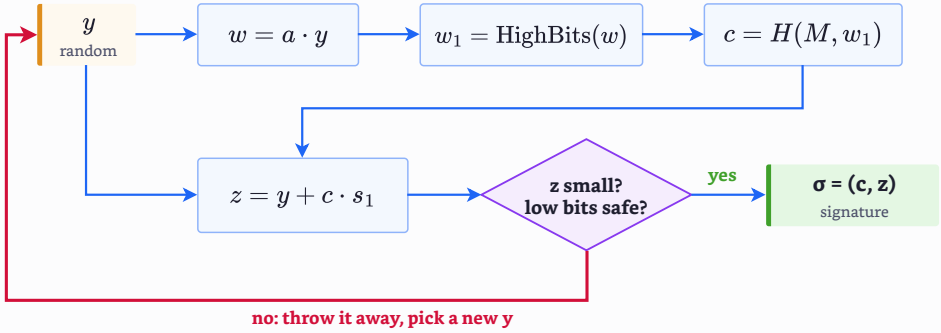
- Write every coefficient as $r = r_1 \cdot 16 + r_0$ with $r_0 \in (-8, 8]$. r_1 is the high part, r_0 the low part.
- Examples: $53 = 3 \cdot 16 + 5$, so $(r_1, r_0) = (3, 5)$. $\quad 45 = 3 \cdot 16 - 3$, so $(3, -3)$.



green: $|r_0| < \gamma_2 - \beta = 6$, safe · red: near the edge

- If $|r_0| < \gamma_2 - \beta = 6$, adding anything of size at most $\beta = 2$ keeps r in the same block.

Signing



- "z small": every coefficient of z below $\gamma_1 - \beta = 14$ in size.
- "low bits safe": every coefficient of $\text{LowBits}(w - c\,s_2)$ below $\gamma_2 - \beta = 6$ in size.

Signing with numbers: try 1

Message M = ``hello''

$y = (0, 7, 1, 14)$

$w = a\,y = (84, 4, 14, 51)$, $\quad w_1 = \mathbf{HighBits}(w) = (5, 0, 1, 3)$

our toy H gives $c = -x + x^3$, $\quad c\,s_1 = (2, -1, 0, 1)$

$z = y + c\,s_1 = (2, 6, 1, 15)$

Rejected

$|15| \geq 14$. A value this close to the edge could leak the sign of $c\,s_1$. Throw everything away and draw a new y.

- Nothing from a rejected try is ever published.

Signing with numbers: try 1

Message M = ``hello''

$y = (0, 7, 1, 14)$

$w = a\,y = (84, 4, 14, 51)$, $\quad w_1 = \mathsf{HighBits}(w) = (5, 0, 1, 3)$

our toy H gives $c = -x + x^3$, $\quad c\,s_1 = (2, -1, 0, 1)$

$z = y + c\,s_1 = (2, 6, 1, 15)$

Rejected

$|15| \geq 14$. A value this close to the edge could leak the sign of $c\,s_1$. Throw everything away and draw a new y.

- Nothing from a rejected try is ever published.

Signing with numbers: try 2

New y

$y = (-4, -12, 1, -14)$

$w = a\,y = (33, 32, 83, 45), \quad w_1 = (2, 2, 5, 3)$

$c = -x - x^2, \quad c\,s_1 = (1, 0, 0, 1), \quad z = (-3, -12, 1, -13)$

Both checks pass

z: all coefficients below 14 in size ✓

$w - c\,s_2 = (34, 32, 84, 45)$, low bits $(2, 0, 4, -3)$: all below 6 in size ✓

Signature: $\sigma = (c, z) = (-x - x^2,\ (-3, -12, 1, -13))$

Signing with numbers: try 2

New y

$y = (-4, -12, 1, -14)$

$w = a\,y = (33, 32, 83, 45), \quad w_1 = (2, 2, 5, 3)$

$c = -x - x^2, \quad c\,s_1 = (1, 0, 0, 1), \quad z = (-3, -12, 1, -13)$

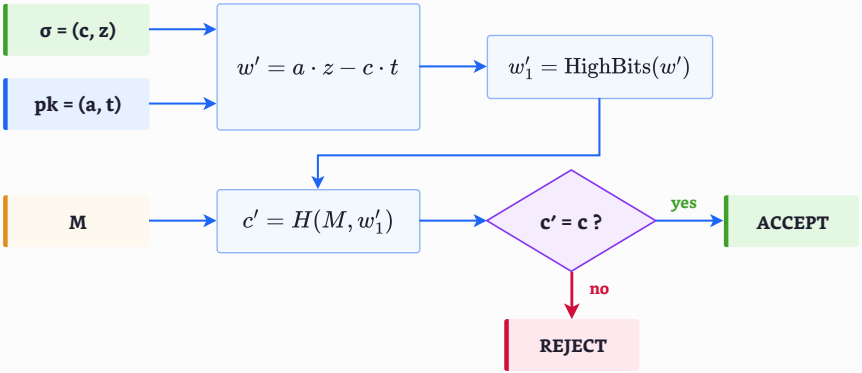
Both checks pass

z: all coefficients below 14 in size ✓

$w - c\,s_2 = (34, 32, 84, 45)$, low bits $(2, 0, 4, -3)$: all below 6 in size ✓

Signature: $\sigma = (c, z) = (-x - x^2,\ (-3, -12, 1, -13))$

Verification



With numbers

$a\,z = (92, 85, 46, 16), \quad c\,t = (58, 53, 59, 68)$

$w' = a\,z - c\,t = (34, 32, 84, 45), \quad \text{HighBits}(w') = (2, 2, 5, 3) = w_1$, so $c' = c$ ✓

Why verification works

$$\begin{aligned} a\,z - c\,t &= a\,(y + c\,s_1) - c\,(a\,s_1 + s_2) \\ &= a\,y + c\,a\,s_1 - c\,a\,s_1 - c\,s_2 \\ &= w - c\,s_2 \end{aligned}$$

- **Numbers:** $w = (33, 32, 83, 45)$ **and** $w' = (34, 32, 84, 45)$ **differ by** $c\,s_2 = (-1, 0, -1, 0)$**.**
- The signer already checked that the low bits of $w - c\,s_2$ are safe, so its high bits equal those of w. The verifier gets the same w_1 and the same c.
- A forger without s_1 cannot find a small z that passes. That would solve a hard lattice problem close to MLWE.

Why verification works

$$\begin{aligned} a\,z - c\,t &= a\,(y + c\,s_1) - c\,(a\,s_1 + s_2) \\ &= a\,y + c\,a\,s_1 - c\,a\,s_1 - c\,s_2 \\ &= w - c\,s_2 \end{aligned}$$

- Numbers: $w = (33, 32, 83, 45)$ and $w' = (34, 32, 84, 45)$ differ by $c\,s_2 = (-1, 0, -1, 0)$.
- The signer already checked that the low bits of $w - c\,s_2$ are safe, so its high bits equal those of w. The verifier gets the same w_1 and the same c.
- A forger without s_1 cannot find a small z that passes. That would solve a hard lattice problem close to MLWE.

How many tries?

- Each try passes both checks with some fixed probability, so the number of tries is random.

Scheme	Average tries per signature
our toy	about 9
ML-DSA-44	4.25
ML-DSA-65	5.1
ML-DSA-87	3.85

- Signing time varies from one signature to the next. Verification always runs once.

06

From toy Dilithium to ML-DSA

What changes

Part	Toy	ML-DSA-44 (FIPS 204)
ring	$q = 97$, $n = 4$	$q = 8380417$, $n = 256$, full NTT
sizes	1×1	$\mathbf{A}$ is 4×4, from a 32-byte seed
challenge c	two ± 1	39 coefficients ± 1 (SampleInBall, SHAKE256)
bounds	$\gamma_1 = 16$, $\gamma_2 = 8$	$\gamma_1 = 2^{17}$, $\gamma_2 = (q-1)/88$
public key	full $\mathbf{t}$	only the high part $\mathbf{t}_1$, plus hints
message	M	$\mu = H(H(\mathsf{pk}) \,\|\|\, M)$

- The sign and verify logic is the same as in the toy.

Shrinking the public key with hints

- $\mathbf{t}$ has 23-bit coefficients. ML-DSA splits $\mathbf{t} = \mathbf{t}_1 \cdot 2^{13} + \mathbf{t}_0$ and publishes only $\mathbf{t}_1$ (10 bits each).
- ML-DSA-44 public key: $32 + 4 \cdot 256 \cdot 10/8 = 1312$ bytes, instead of about 2,976.

The cost

The verifier now computes $\mathbf{A}\mathbf{z} - c\,\mathbf{t}_1 2^{13} = \mathbf{w} - c\,\mathbf{s}_2 + c\,\mathbf{t}_0$, a little further from $\mathbf{w}$.

The signer adds **hint bits $\mathbf{h}$** that say which high bits to move by one. At most $\omega = 80$ hints are allowed.

Shrinking the public key with hints

- $\mathbf{t}$ has 23-bit coefficients. ML-DSA splits $\mathbf{t} = \mathbf{t}_1 \cdot 2^{13} + \mathbf{t}_0$ and publishes only $\mathbf{t}_1$ (10 bits each).
- ML-DSA-44 public key: $32 + 4 \cdot 256 \cdot 10/8 = 1312$ bytes, instead of about 2,976.

The cost

The verifier now computes $\mathbf{Az} - c\,\mathbf{t}_1 2^{13} = \mathbf{w} - c\,\mathbf{s}_2 + c\,\mathbf{t}_0$, a little further from $\mathbf{w}$.

The signer adds **hint bits h** that say which high bits to move by one. At most $\omega = 80$ hints are allowed.

Deterministic or hedged signing

- The random **y** comes from SHAKE256 of the secret key, the message and a 32-byte value `rnd`.

Hedged (default)

`rnd` = 32 fresh random bytes

same message, different signature each time

harder to attack with side channels and faults

Deterministic

`rnd` = 32 zero bytes

same message, same signature

for devices without a good random number generator

ML-DSA parameter sets and sizes

Scheme	(k, ℓ)	η	τ	public key	secret key	signature
ML-DSA-44	(4, 4)	2	39	1312	2560	2420
ML-DSA-65	(6, 5)	4	49	1952	4032	3309
ML-DSA-87	(8, 7)	2	60	2592	4896	4627
ECDSA P-256				64	32	64
RSA-3072				384	384	384
FN-DSA-512 (Falcon)				897		666
SLH-DSA-128s				32	64	7856

- Sizes in bytes. ML-DSA signatures are about 40 times larger than ECDSA, and signing and verifying are fast.

07

Summary

Summary

ML-KEM

- Key: an MLWE sample $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$.
- Encrypt: hide $\lceil q/2 \rfloor \cdot m$ under $\mathbf{t}^T\mathbf{r}$.
- Decrypt: $v - \mathbf{s}^T\mathbf{u}$, then round.
- KEM: random m, hash to K, re-encryption check.

ML-DSA

- Schnorr with lattices: commit, challenge, response.
- Hash only the high bits of $\mathbf{w}$.
- Reject any $\mathbf{z}$ that could leak $\mathbf{s}_1$.
- Verify: $\text{HighBits}(\mathbf{A}\mathbf{z} - c\,\mathbf{t}) = \mathbf{w}_1$.

Under both

Polynomials in $\mathcal{R}_q$ (Lecture 1), multiplied with the NTT (Lecture 2), and the MLWE problem.

Thank you!