Quorus

Efficient, Scalable Threshold ML-DSA from MPC

Alexander Bienstock, **Leo de Castro**, Daniel Escudero, Antigoni Polychroniadou, Akira Takahashi

JPMC AlgoCrypt CoE, JPMC AI Research

https://eprint.iacr.org/2025/1163

This Talk

- We present an MPC protocol where parties can jointly generate ML-DSA signatures for **all** security levels.
  - This protocol scales to *any* number of parties.
  - Protocol UC-realizes ideal functionality F computing ML-DSA
  - Instantiated for honest-majority (no additional public key assumptions), but we give the building blocks for any access structure.
- **Main contribution:** variant of ML-DSA key generation & signing that is amenable to MPC.
  - This variant produces signatures that can be verified under the **unmodified** ML-DSA verification algorithm.
- **Paper**: *Quorus: Efficient, Scalable Threshold ML-DSA Signatures from MPC.* To appear at USENIX 2026. eprint 2025/1163.

Introduction: ML-DSA

- Primary NIST-standard for PQC signatures.
- Security assumption: Module Ring Learning with Errors
- Fiat-Shamir of Lyubashevsky ID protocol:
  - Relies on **rejection sampling** for security
  - The signing algorithm **fails with constant probability**.
  - In the normal setting, the signer simply retries until a signature is successfully produced.
  - In the MPC setting, we must handle these failures very carefully…

Threshold ML-DSA: Motivation

- Signing key for an ML-DSA public key can be threshold-shared amongst distinct parties.
- Public keys & signatures should look identical to regular (non-threshold) ML-DSA.
  - Verification should be **exactly** the same.
- Allows advanced crypto applications to interoperate with standard signatures/certificates.
  - Distributing credentials for a certificate authority.
  - Crypto wallets where a user can hold a signing key in one place or opt for a more secure version where the key is split across different parties.

Part 1: MPC-friendly Variant of ML-DSA

Lyubashevsky Identification Protocol

Prover

Secret key: $s \in R_q^{\ell}, e \in R_q^{k}$

1. Sample $y \in R_q^{\ell}$, $e' \in R_q^{k}$
2. Commit to y with w

$$w = Ay + e'$$

Verifier

Public key

$(A \in R_q^{k \times \ell}, t = A \cdot s + e)$

3. Sample random challenge $c \in R$

4. Compute masked opening

$$z = s \cdot c + y \in R_q^{\ell}$$

If $\|z\|_{\infty} \geq B$, ***ABORT***

Otherwise, send z

5. Check that $\|z\|_{\infty} < B$ and

$$Az \approx c \cdot t + w$$

Lyubashevsky Non-Interactive Identification Protocol (from the Fiat-Shamir transform)

Prover

Secret key: s, e

1. Sample y, e'
2. Commit to y with w

   $w = Ay + e'$

3. Compute $c = \mathrm{Hash}(A, t, w)$

4. Compute masked opening

   $z = s \cdot c + y$

If $\|z\|_\infty \geq B$, ***ABORT***

Otherwise, send z

For hash functions modeled as random oracles, this is safe.

w, c, z →

Verifier

Public key

$(A, t = A \cdot s + e)$

5. Check that $\|z\|_\infty < B$ and

   $Az \approx c \cdot t + w$ and

   $c = \mathrm{Hash}(A, t, w)$

MLDSA from Non-interactive ID Protocol

Signer

Secret key: s, e

Message $\boldsymbol{m}$

1. Sample y, e'
2. Commit to y with w
   $w = Ay + e'$

3. Compute $c = \text{Hash}(A, t, w, \boldsymbol{m})$

4. Compute masked opening
   $z = s \cdot c + y$

If $z \geq B$, ***ABORT***

Otherwise, send z

Just include the message in the hash.

w, c, z →

Verifier

Public key
$(A, t = A \cdot s + e)$

Message $\boldsymbol{m}$

5. Check that $z < B$ and
   $Az \approx c \cdot t + w$ and
   $c = \text{Hash}(A, t, w, \boldsymbol{m})$

MLDSA Standard

Signer

Secret key: s, e

Message $\boldsymbol{m}$

1. Sample y
2. Commit to y with
   $w = \lfloor Ay \rceil = \mathrm{HighBits}(Ay)$

3. Compute $c = \mathrm{Hash}(A, t, w, \boldsymbol{m})$

4. Compute masked opening

$z = s \cdot c + y$

If $z \geq B$, ***ABORT***

Otherwise, send the full signature.

No need to send both w and z.
Instead, send $h = \mathrm{diff}(w, Az - ct)$

No error in w, just rounding.
Security proof is ***only*** for passing signatures.

z, h, c

Verifier

Public key
$(A, t = A \cdot s + e)$

Message $\boldsymbol{m}$

5. Check that $z < B$.
Compute $\mathrm{w} = \mathrm{UseHint}(Az - c \cdot t, h)$.
Check that $c = \mathrm{Hash}(A, t, w, \boldsymbol{m})$.

MLDSA MPC Attempt #1: Run the full algorithm within the protocol.

Ideal Functionality for MLDSA Signer

MPC Parties

Secret key: s, e

Public key $(A, t = A \cdot s + e)$

Message $\boldsymbol{m}$

"Please sign $\boldsymbol{m}$"

1. Sample y
2. Commit to y with $w = \lfloor Ay \rceil$

3. Compute $c = \mathrm{Hash}(A, t, w, \boldsymbol{m})$

Problem: Hash() is determined by the standard and is ***not*** MPC-friendly. Extremely expensive to evaluate in MPC.

4. Compute masked opening

$$z = s \cdot c + y$$

If $z \geq B$, ***ABORT***. Send $\perp$.

Otherwise, send the full signature.

No need to send both w and z.
Instead, send $h = \mathrm{diff}(w, Az - ct)$

z, h, c OR $\perp$

5. Check that $z < B$.
Compute $\mathrm{w} = \mathrm{UseHint}(Az - c \cdot t, h)$.
Check that $c = \mathrm{Hash}(A, t, w, \boldsymbol{m})$.

Goal: Run Hash() outside the protocol.

MLDSA MPC Attempt #2: Return to the interactive protocol.

Ideal Functionality for MLDSA Signer

Secret key: s, e

1. Sample y
2. Commit to y with $w = \lfloor Ay \rceil$

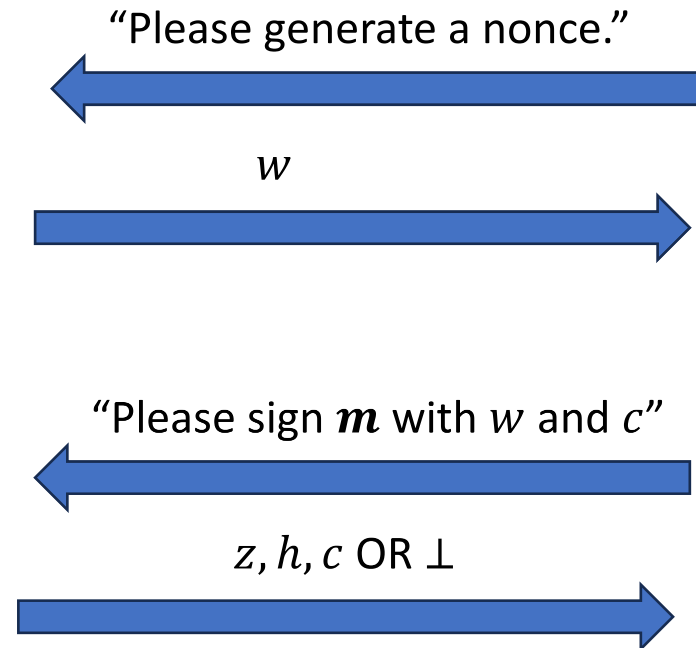
4. Compute masked opening

$$z = s \cdot c + y$$

If $z \geq B$, ***ABORT***. Send $\perp$.

Otherwise, send the full signature.

No need to send both w and z.
Instead, send $h = \text{diff}(w, Az - ct)$



MPC Parties

Public key
$(A, t = A \cdot s + e)$

Message $\boldsymbol{m}$

3. Compute $c = \text{Hash}(A, t, w, \boldsymbol{m})$

Prior work [BTT22] [DFPSX23] [dPN25] show standard Lyubashevsky ID can safely reveal $(w, c, \perp)$, but proving it for MLDSA is an open question [CGL+24].

MLDSA MPC Attempt #3: Return to the **secure** interactive protocol.

Ideal Functionality for MLDSA Signer

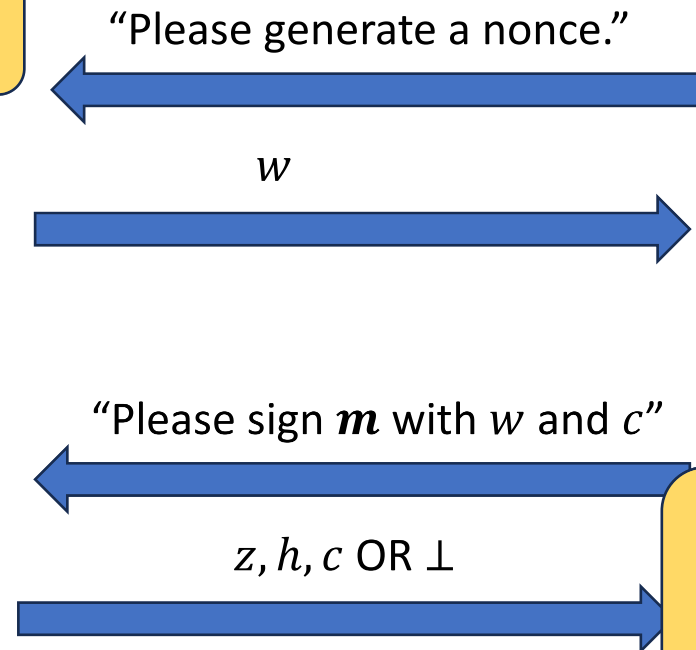
Secret key: s, e

Add large error to w

1. Sample y, e'
2. Commit to y with $w = Ay + e'$

MPC Parties

Public key
$(A, t = A \cdot s + e)$



Message $\boldsymbol{m}$

3. Compute $c = \mathrm{Hash}(A, t, w, \boldsymbol{m})$

4. Compute masked opening
$$z = s \cdot c + y$$
If $z \geq B$, ***ABORT***. Send $\bot$.

Otherwise, send the full signature.

No need to send both w and z.
Instead, send $h = \mathrm{diff}(w, Az - ct)$

Problem: This signature will not pass under the standardized parameters. The original error e' in the interactive protocol is too large.

MLDSA MPC Final Version: Shrinking the Error

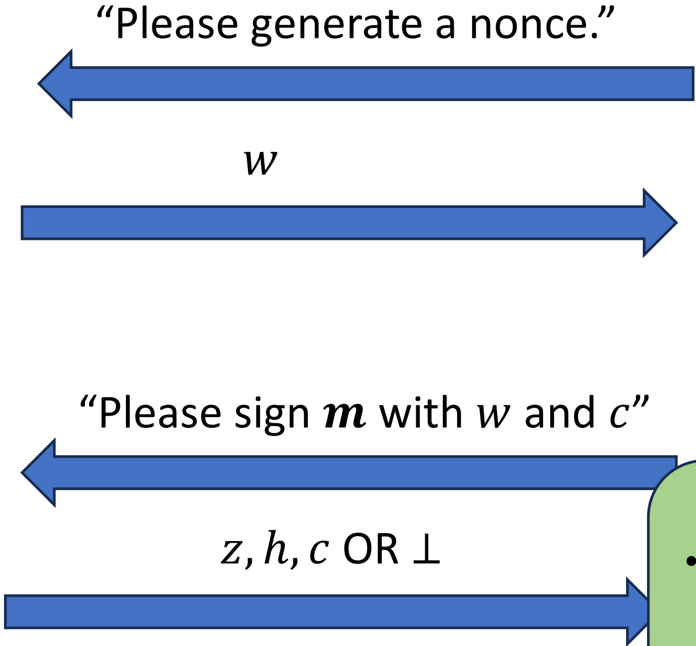
Ideal Functionality for MLDSA Signer

Secret key: s, e

1. Sample y, e'
2. Commit to y with

$w = \lfloor Ay + e' \rceil = \mathrm{HighBits}(Ay + e')$

Add small error to w



MPC Parties

Public key

$(A, t = A \cdot s + e)$

Message $\boldsymbol{m}$

3. Compute $c = \mathrm{Hash}(A, t, w, \boldsymbol{m})$

4. Compute masked opening

$z = s \cdot c + y$

If $z \geq B$, ***ABORT***. Send $\perp$.

Otherwise, send the full signature.

No need to send both w and z.
Instead, send $h = \mathrm{diff}(w, Az - ct)$

New security proof:

- This small error is safe for aborted signatures while still verifying under the standardized parameters.
- UF-CMA-$\perp$ (> UF-CMA): Hard to create forgery on $\boldsymbol{m}$ after seeing incomplete signatures on $\boldsymbol{m}$

Part 2: Protocol for the Signing Functionality

Version 0.1

More versions coming soon!

Protocol Design

- One-time setup phase:
  - Distribute secret shares of the signing key & public key error
- Per-signature Offline phase:
  1. Samples shares of the nonces
  2. Computes secret sharing of w

     Don’t open the sharing yet! Not known if ML-DSA is vulnerable to an ROS-style attack.
- Online phase (all signers hold the message):
  1. Open the shares of w.
  2. Signers locally compute the challenge
  3. Signers locally compute shares of z and error term (used to prevent leakage on public key error).
  4. Rejection sampling: Batched Comparison

MLDSA MPC Phases

Ideal Functionality for MLDSA Signer

Secret key: s, e

1. Sample y, e'
2. Commit to y with $w = \lfloor Ay + e' \rceil$

Per-Signature Offline Phase

4. Compute masked opening

$$z = s \cdot c + y$$

If $z \geq B$, ***ABORT***. Send $\perp$.

Otherwise, send the full signature.

No need to send both w and z.
Instead, send $h = \mathrm{diff}(w, Az - ct)$

One-time Setup

"Please generate a nonce."

w

"Please sign $\boldsymbol{m}$ with w and c"

z, h, c OR $\perp$

MPC Parties

Public key
$(A, t = A \cdot s + e)$

Message $\boldsymbol{m}$

3. Compute $c = \mathrm{Hash}(A, t, w, \boldsymbol{m})$

Per-Signature Online Phase

Background: edaBits [EGK+20]

- Very useful correlation in our protocol.
- An edaBit correlation has two pieces:
  - A secret-sharing of a value $r \in \mathbb{Z}_q$ over some arithmetic field $\mathbb{Z}_q$.
  - Secret sharings of the bits of r over some characteristic 2 field.
- When $r \in \{0,1\}$, this is a daBit.
- Very convenient to switch between arithmetic & Boolean operations.
- Very convenient to sample values from a power-of-two range.

Protocol Instantiation: Main Subroutines

- Per-signature Offline Phase
  - Sample shares of y and e' using edaBits.
  - Shares of $Ay + e'$ can be computed locally.
  - Rounding for $w = \lfloor Ay + e' \rceil$ can be implemented with bit decomposition via edaBits.
- Online Phase
  - Batched Comparison
    - edaBits + PrefixOr over characteristic-2 field
  - Batched Or
    - Uses daBits to lift shares of comparison output bits to a statistically large field

Complexity & Performance

- For the online phase, we give a "round optimized" and "communication optimized" versions of our protocol.
- The offline phase can be run in parallel, so only the communication optimized version is used.

# of parties		3	7	15	31	63	Rounds
MLDSA-44	C	0.31	0.31	0.31	0.31	0.31	29
	R	0.15	0.47	1.1	2.36	4.87	16
MLDSA-65	C	0.43	0.43	0.43	0.43	0.43	29
	R	0.21	0.65	1.5	3.24	6.7	16
MLDSA-87	C	0.59	0.59	0.59	0.59	0.59	29
	R	0.29	0.88	2.06	4.42	9.14	16

Table 3: Online communication in MB per party for a successful SIGN command. For n parties, we set $t = (n-1)/2$. The C rows are communication-optimized and the R rows are round-optimized.

# of parties	3	7	15	31	63	Rounds
MLDSA-44	6	15.9	40.6	110.4	330.1	86 – 108
MLDSA-65	4.1	11.9	34.4	108.4	369.9	81 – 103
MLDSA-87	5.5	16	46.3	146	499	81 – 103

Table 4: Offline communication in MB per party for a successful SIGN command. For n parties, we set $t = (n-1)/2$. We also report round-count, given as a range since for the offline phase the number of rounds increases as the number of parties grow (this is due to the edabits protocol).

Runtimes

- Across the 16 rounds of the online phase, the RTT of the network will dominate performance.
- **Takeaway**: the protocol is highly communication-bound

# of parties	3	7	15	31	63
ML-DSA-44	6.3	8.9	18.7	52.2	196.4
ML-DSA-65	8.7	12.5	25.7	70.4	261.2
ML-DSA-87	11.9	16.9	34.8	94.6	343.1

Table 6: Online compute times in ms ignoring network effects. All parties are evaluated in parallel.

Conclusion & Future Work

- Designed MPC-friendly MLDSA signing, proved its security, and defined reusable ideal functionality for threshold MLDSA
- Compared to Mithril
  - We define protocols UC-realizing MLDSA functionality vs game-based unforgeability
  - Scalable to any number of parties at the cost of larger round counts and communication
- Reduce the number of rounds
  - Current version has high wallclock times from the network RTT

- Approach: Improved correlations for comparison
  - Modifying edaBits to support more efficient comparisons

- Approach: Threshold FHE
  - Mentioned yesterday by Stehle: a two-round (online phase) threshold ML-DSA protocol from exact CKKS
  - Benchmarked with GPU acceleration

Thank you!

https://eprint.iacr.org/2025/1163