The NIST MPTC Call

Post-Quantum Threshold Signatures and a Promising Approach to Threshold ML-DSA

Guilhem Niot

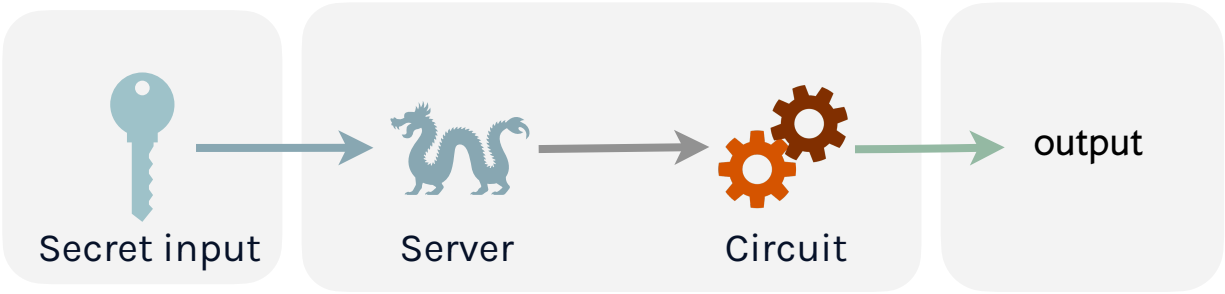
Séminaire C2 - 03/07/2026

Presentation Roadmap

I. Overview of the NIST MPTC Call, and the different dimensions considered.

II. In-depth introduction of Mithril, an efficient Threshold ML-DSA proposal.

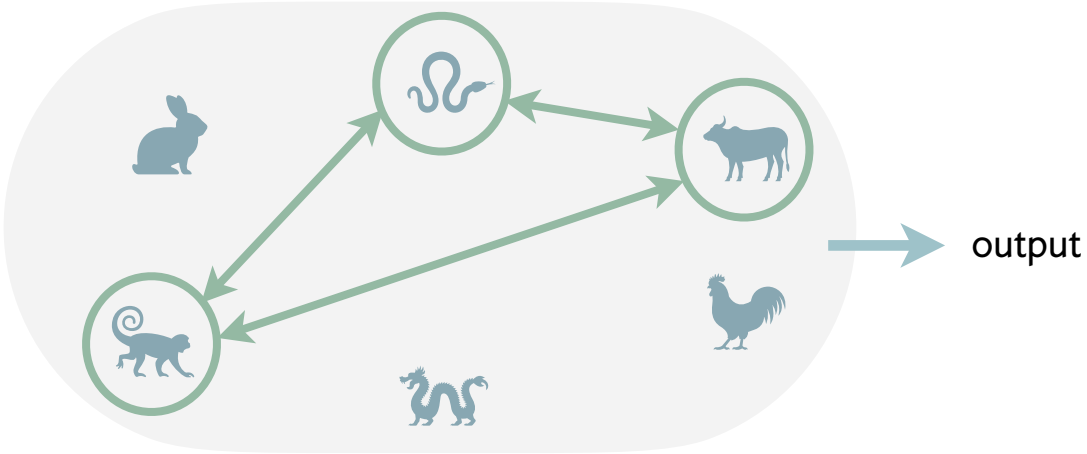
Secure Computation: The Centralized Setting



- What if the Server has limited trust? It may be malicious or hacked?
- What if the Server becomes available?

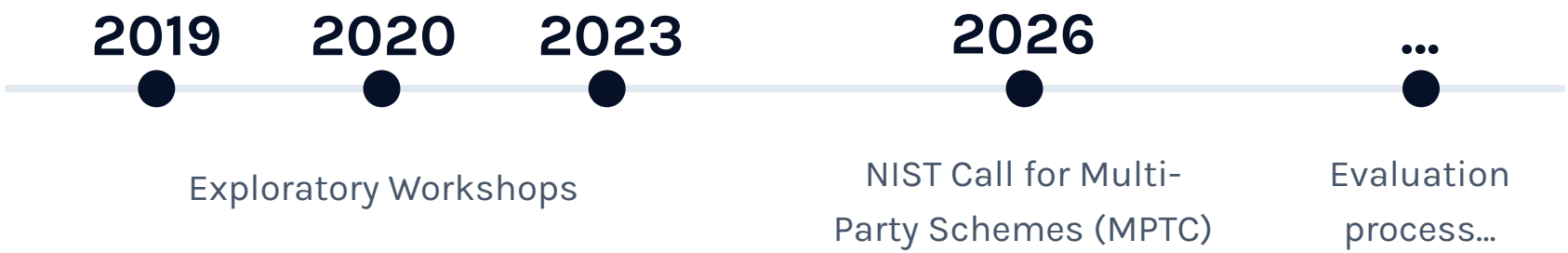
Secure Computation: The Multi-Party Setting

The secret input is shared among several parties.



They execute an interactive protocol to compute the circuit such that no single party learns the secret input.

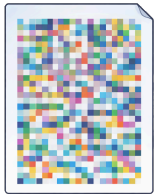
Multi-Party Cryptography and NIST



Stated goal: Gather high-quality reference material, and inform the development of future NIST recommendations and processes.

Multi-Party Cryptography and NIST: A Wide Scope

Threshold Signatures



Threshold Encryption

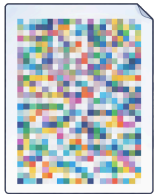
Generic techniques and Gadgets

FHE, Garbled circuits, threshold-friendly hash, etc.

Multi-Party Cryptography and NIST: A Wide Scope

Threshold Signatures

Focus of this talk



Threshold Encryption

Generic techniques and Gadgets

FHE, Garbled circuits, threshold-friendly hash, etc.

NIST MPTC: Flexible Requirements

Interchangeability

Threshold schemes should be compatible with subsequent operations (e.g. signature verification) of a non-threshold scheme.

Primary target

Alternative base scheme

… if provides significant advantages (e.g. more efficient, can detect misbehaviour, etc.)

NIST MPTC: Flexible Requirements

Interchangeability

Threshold schemes should be compatible with subsequent operations (e.g. signature verification) of a non-threshold scheme.

Security Model

How to model user interactions? Their corruptions?

NIST MPTC: Flexible Requirements

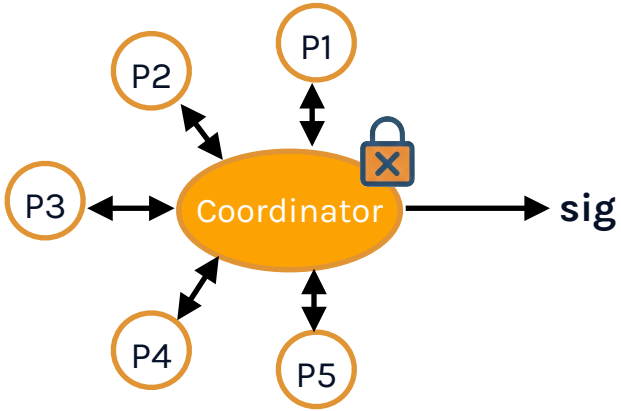
Interchangeability

Threshold schemes should be compatible with subsequent operations (e.g. signature verification) of a non-threshold scheme.

Security Model

How to model user interactions? Their corruptions?

Communication among parties is unspecified, for instance may be over (authenticated) pairwise channel, or broadcast channel.



NIST MPTC: Flexible Requirements

Interchangeability

Threshold schemes should be compatible with subsequent operations (e.g. signature verification) of a non-threshold scheme.

Security Model

How to model user interactions? Their corruptions?

Corruption model: As in the literature, several corruption models exist.

NIST MPTC: Flexible Requirements

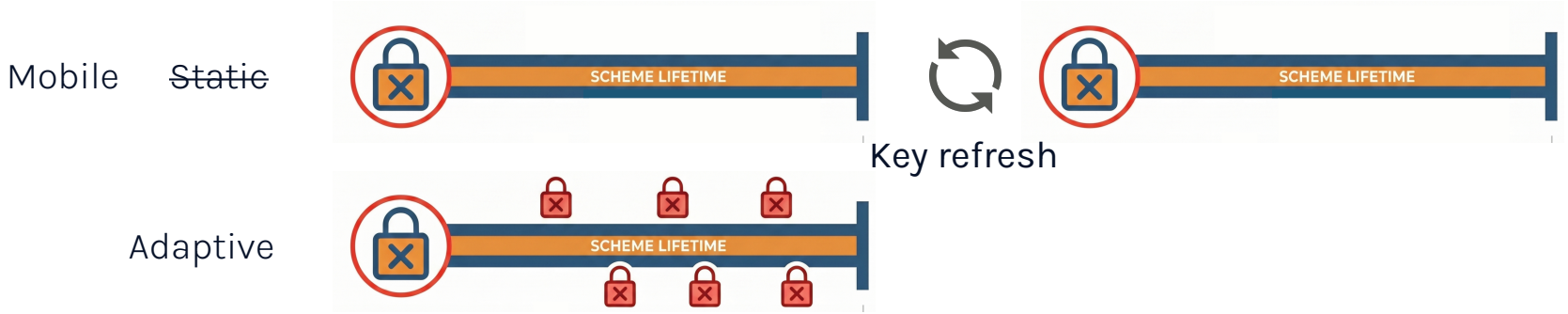
Interchangeability

Threshold schemes should be compatible with subsequent operations (e.g. signature verification) of a non-threshold scheme.

Security Model

How to model user interactions? Their corruptions?

Corruption model: As in the literature, several corruption models exist.



NIST MPTC: Flexible Requirements

Interchangeability

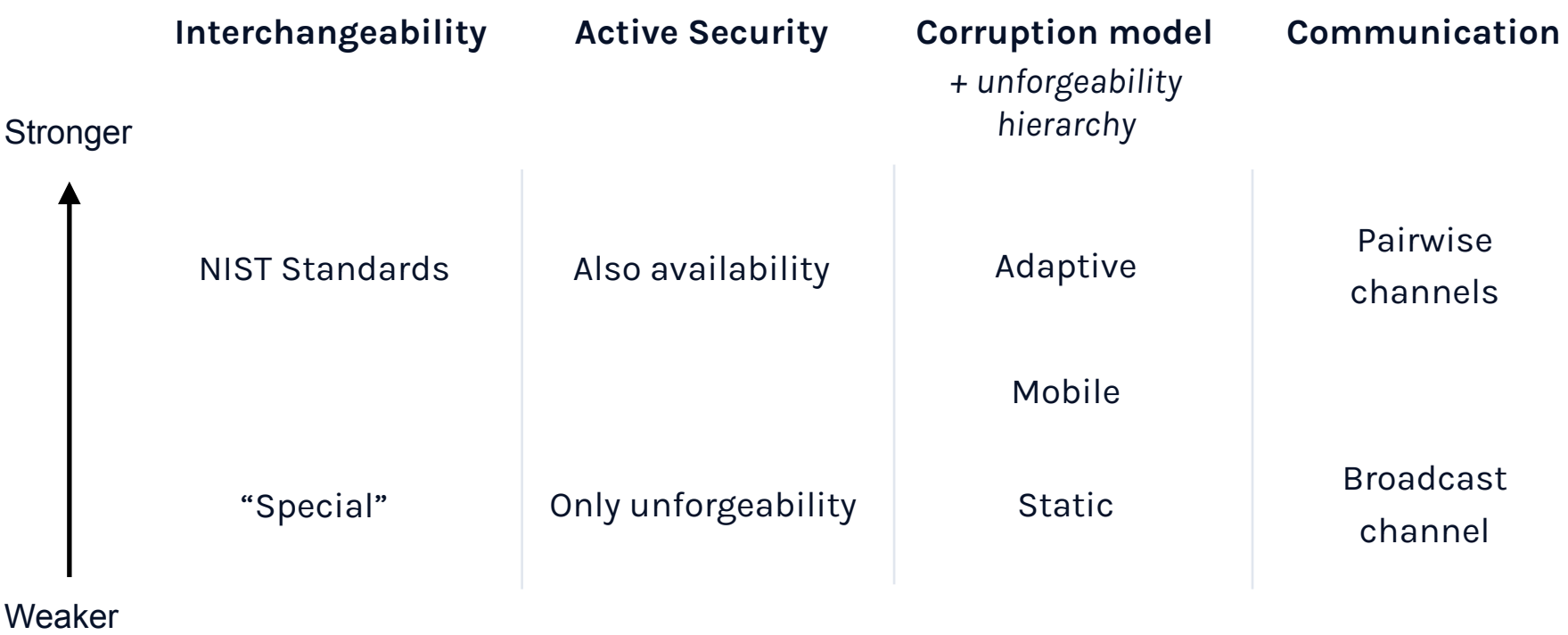
Threshold schemes should be compatible with subsequent operations (e.g. signature verification) of a non-threshold scheme.

Security Model

How to model user interactions? Their corruptions?

NIST requires active security (i.e. against active adversaries) for critical safety properties (i.e. unforgeability), but prevention of aborts induced by the attacker is optional.

NIST MPTC: Flexible Requirements



Threshold Signatures: The Classical Setting

In the classical setting, we have mature threshold solutions for all main standards:

EdDSA	ECDSA	BLS	RSA
FROST [KG20]	[CGGMP21]	[Bol03]	[Sho00]

Achieving adaptive security is still an active research field.
But good solutions are known for all other stronger properties mentioned.

Threshold Signatures: The Classical Setting

In the classical setting, we have mature threshold solutions for all main standards:

EdDSA	**ECDSA**	**BLS**	**RSA**
FROST [KG20]	[CGGMP21]	[Bol03]	[Sho00]

Achieving adaptive security is still an active research field.
But good solutions are known for all other stronger properties mentioned.

Proposals submitted to NIST MPTC for EdDSA, ECDSA and BLS.

Threshold Signatures: The PQ Standards?

It is desirable to design threshold solutions for the new NIST PQ standards.

ML-DSA

(Dilithium)

FN-DSA

(Falcon)

SLH-DSA

(SPHINCS+)

Threshold Signatures: The PQ Standards?

It is desirable to design threshold solutions for the new NIST PQ standards.

PoC solution with MPC[1], but huge latency (thousands of rounds) and high communication cost.

Theoretical solution exists, but impractical due to high storage

[1]. https://eprint.iacr.org/2026/1300

Threshold Signatures: The PQ Standards?

It is desirable to design threshold solutions for the new NIST PQ standards.

PoC solution with MPC[1], but huge latency (thousands of rounds) and high communication cost.

Theoretical solution exists, but impractical due to high storage

[1]. https://eprint.iacr.org/2026/1300

Threshold Signatures: The PQ Standards?

ML-DSA

(Dilithium)

There are several approaches to Threshold ML-DSA.

Paradigm	# parties supported	Comm./party	Computation	Detect Misbehavior
MPC (Quorus)	∞	**> 1 MB (> 20 rounds)**	Medium	✗
FHE[1]	∞	25 kB (2 rounds)	**High + Proprietary**	With NIZK
This talk (Mithril)	**< 10**	20 kB - 1MB (3 rounds)	Low	✗

[1]. https://eprint.iacr.org/2026/638

Threshold Signatures: The PQ Standards?

ML-DSA

(Dilithium)

- All approaches make some efficiency trade-offs.
- No satisfying way of detecting misbehaviour to prevent active attacks against availability.

Threshold Signatures: Alternative MPTC Proposals

… from other assumptions

In particular, Vinaigrette compatible with the NIST candidates MAYO/UOV.

… with state management.

Threshold variant of XMSS/LMS.

Or still from lattices, but… with high scalability, or efficient malicious behaviour detection.

Raccoon

Threshold-friendly signature scheme.
Although non standard and bigger signatures than ML-DSA (10 kB vs 2.5 kB).

Tanuki: up to 1024 parties supported.

Hermine: up to 64 parties supported, with efficient abort identification.

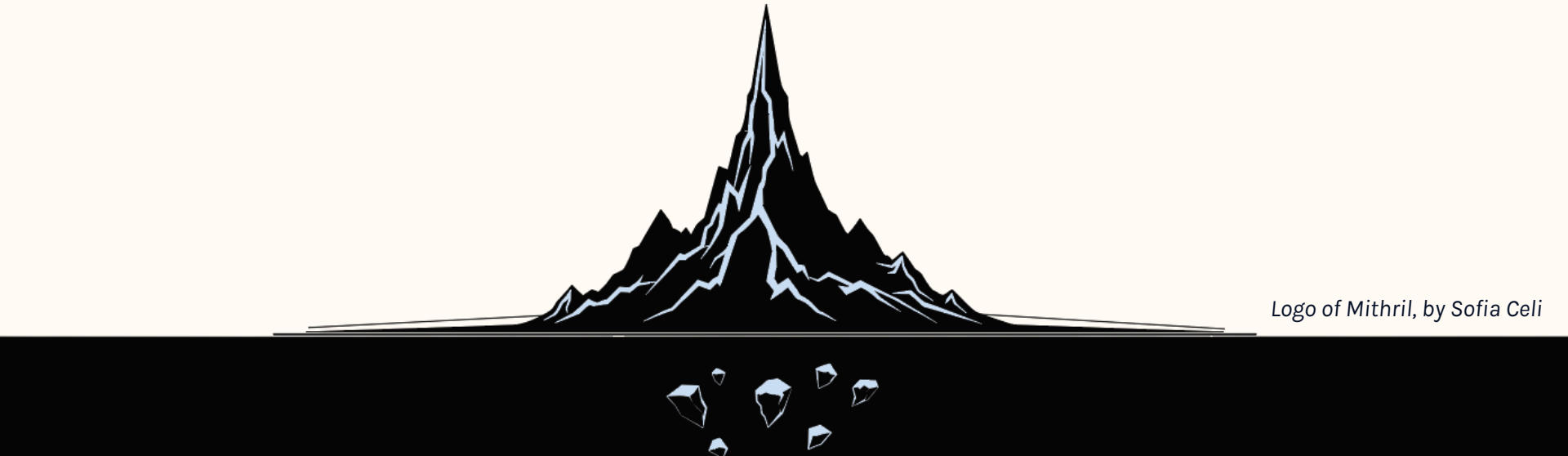
Threshold Signatures: Mithril

Mithril already attracts interest for designers who want to preserve compatibility with ML-DSA while having a favorable efficiency trade-off.

Notable use cases, with small number of parties, attacks against availability not in scope:

- Self-custody wallets
- Root of trusts

Mithril: An Efficient Threshold ML-DSA



Logo of Mithril, by Sofia Celi

ML-DSA Signatures

ML-DSA . Keygen() → sk, vk

- Sample uniform $\mathbf{A}$
- $\text{vk} = (\mathbf{A}, \mathbf{A} \cdot \text{sk} + \mathbf{e})$, for sk, $\mathbf{e}$ short

MLWE **assumption:** vk appears uniformly distributed.

Signing → Fiat-Shamir transform applied to protocol proving knowledge of sk.

1. **Randomness + Commitment:** Sample short $\mathbf{r}$, and commit $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} \rceil$.
2. **Challenge:** Derive challenge $c = H(\mathbf{w}, \text{msg})$.
3. **Response:** Compute $\mathbf{z} = c \cdot \text{sk} + \mathbf{r}$.

Verification → Check that $\mathbf{w}$ can be recovered from $\mathbf{z}$, and $\mathbf{z}$ is short.

2. Threshold ML-DSA

ML-DSA Signatures

ML-DSA . Keygen() → sk, vk

- Sample uniform $\mathbf{A}$
- $\text{vk} = (\mathbf{A}, \mathbf{A} \cdot \text{sk} + \mathbf{e})$, for sk, $\mathbf{e}$ short

MLWE **assumption:** vk appears uniformly distributed.

Signing → Fiat-Shamir transform applied to protocol proving knowledge of sk.

1. **Randomness + Commitment:** Sample short $\mathbf{r}$, and commit $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} \rceil$.
2. **Challenge:** Derive challenge $c = H(\mathbf{w}, \text{msg})$.
3. **Response:** Compute $\mathbf{z} = c \cdot \text{sk} + \mathbf{r}$.
   - If $\mathbf{z} \notin S$ → **restart**
4. Rejection sample:
   - If $\mathbf{z} \in S$ → **output**

Verification → Check that $\mathbf{w}$ can be recovered from $\mathbf{z}$, and $\mathbf{z}$ is short.

2. Threshold ML-DSA

ML-DSA Signatures

ML-DSA . Keygen() → sk, vk

- Sample uniform $\mathbf{A}$
- $\text{vk} = (\mathbf{A}, \mathbf{A} \cdot \text{sk} + \mathbf{e})$, for sk, $\mathbf{e}$ short

MLWE **assumption:** vk appears uniformly distributed.

Signing → Fiat-Shamir transform applied to protocol proving knowledge of sk.

1. **Randomness + Commitment:** Sample short $\mathbf{r}$, and commit $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} \rceil$.
2. **Challenge:** Derive challenge $c = H(\mathbf{w}, \text{msg})$.
3. **Response:** Compute $\mathbf{z} = c \cdot \text{sk} + \mathbf{r}$.
4. Rejection sample:
   - If $\mathbf{z} \notin S$ → **restart**
   - If $\mathbf{z} \in S$ → **output**

Verification → Check that $\mathbf{w}$ can be recovered from $\mathbf{z}$, and $\mathbf{z}$ is short.

2. Threshold ML-DSA

Distributing ML-DSA: Mithril At A High Level

Centralized

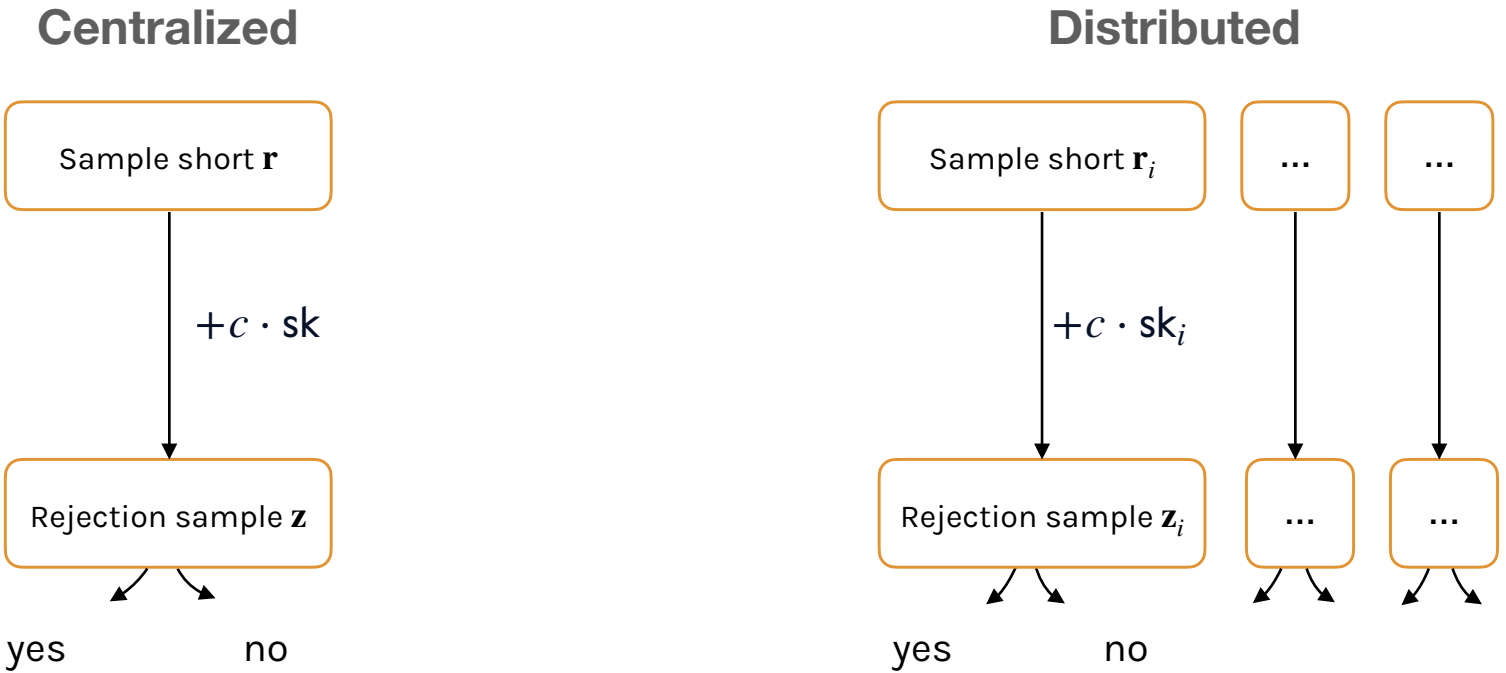
Sample short $\mathbf{r}$

Distributed

Sample short $\mathbf{r}_i$ | ... | ...

Aggregate $\mathbf{r} = \sum_i \mathbf{r}_i$

Distributing ML-DSA: Mithril At A High Level

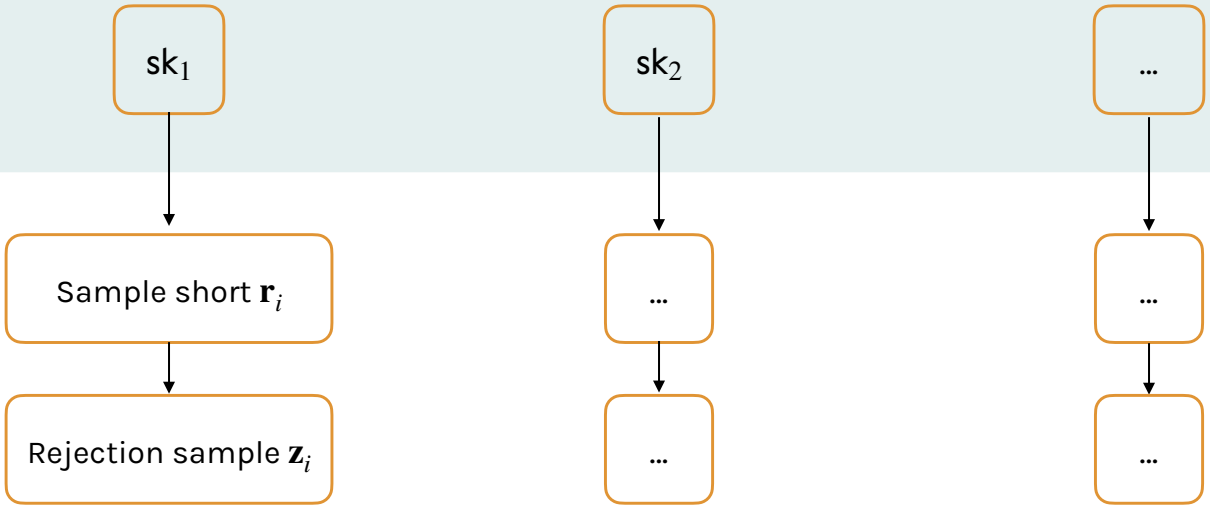


If all parties accepted, aggregate $\mathbf{z} = \sum \mathbf{z}_i$

Technique 1: (Short) Replicated Secret Sharing

For this to work, we need a **short partial secret** per party for each session.

$$\mathsf{sk} = \sum_i \mathsf{sk}_i$$



Distributing ML-DSA: Mithril At A High Level

ML-DSA signing

1. **Randomness + Commitment:** Sample short $\mathbf{r}$, and commit $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} \rceil$.
2. **Challenge:** Derive challenge $c = H(\mathbf{w}, \text{msg})$.
3. **Response:** Compute $\mathbf{z} = c \cdot \text{sk} + \mathbf{r}$. Rejection sample.

Our protocol

Mithril . Sign(msg) → sig

Round 1:

1. Sample short $\mathbf{r}_i, \mathbf{e}'_i$
2. $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
3. Broadcast $\text{commit}_i = H(\mathbf{w}_i)$

Round 2:

1. Broadcast $\mathbf{w}_i$

Round 3:

1. $\mathbf{w} = \sum_i \mathbf{w}_i$ + abort if inconsistent commit_i
2. $c = H(\lfloor \mathbf{w} \rceil, \text{msg})$
3. $\mathbf{z}_i = c \cdot \text{sk}_i + \mathbf{r}_i$
4. If $\mathbf{z}_i$ in S, **broadcast $\mathbf{z}_i$, else abort**

Combine:

1. $\text{sig} = (\sum_i \mathbf{z}_i, \lfloor \mathbf{w} \rceil)$
2. If sig not in S', **restart**
3. **return** sig

Technique 1: (Short) Replicated Secret Sharing

For this to work, we need a **short partial secret** per party for each session.

Use Replicated Secret Sharing.

ML-DSA* . Keygen() $\rightarrow$ sk, vk

1. For every possible set I of $N - T + 1$ parties
   2. $\mathsf{vk}_I = \mathbf{A} \cdot \mathsf{sk}_I + \mathbf{e}_I$, where $\mathsf{sk}_I, \mathbf{e}_I$ short
   3. Distribute $\mathsf{sk}_I, \mathbf{e}_I$ to parties in I
4. $\mathsf{vk} = \sum_i \mathsf{vk}_I$

1. When at most $T - 1$ parties are corrupted, at least one of these secrets remains hidden.
2. T parties can collaboratively reconstruct the full secret.

Partition $\sqcup_{i \in SS} m_i = \{I \text{ s.t. } |I| = N - T + 1\}$:

$$\mathsf{sk} = \sum_{i \in SS} \sum_{I \in m_i} \mathsf{sk}_I, \quad \mathbf{e} = \sum_{i \in SS} \sum_{I \in m_i} \mathbf{e}_I$$

Technique 2: Optimized Rejection Sampling

When T users sign → proba that all parties pass rejection sampling is p^T.

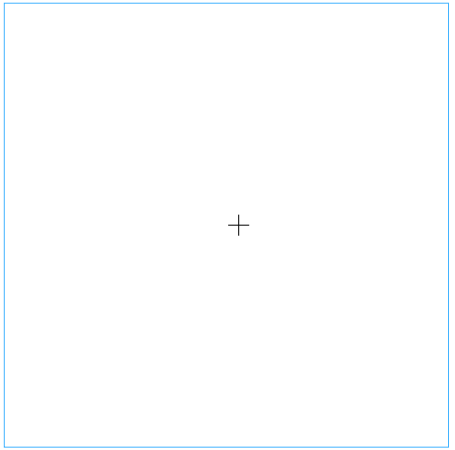
Exponential degradation over centralized setting.

Technique 2: Optimized Rejection Sampling

When T users sign → proba that all parties pass rejection sampling is p^T.

Exponential degradation over centralized setting.

Sample r in a centered **hypercube**.



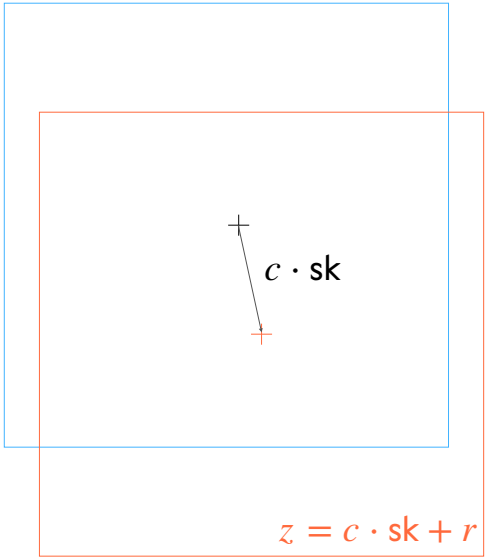
Technique 2: Optimized Rejection Sampling

When T users sign → proba that all parties pass rejection sampling is p^T.

Exponential degradation over centralized setting.

Sample r in a centered **hypercube**.

Then, the distribution of z depends on the secret.



Technique 2: Optimized Rejection Sampling

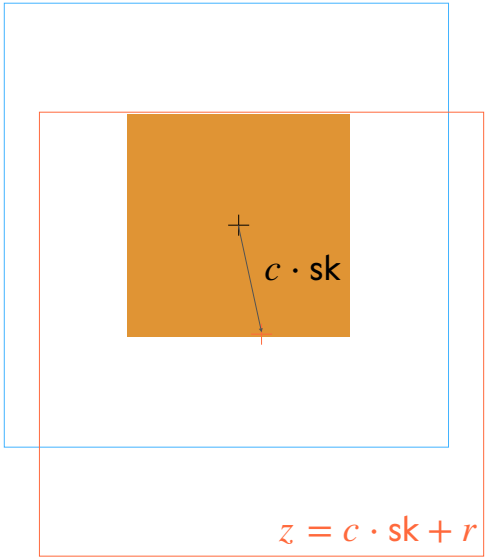
When T users sign → proba that all parties pass rejection sampling is p^T.

Exponential degradation over centralized setting.

Sample r in a centered **hypercube**.

Then, the distribution of z depends on the secret.

We reject any z outside of  .
The resulting distribution is independent of the secret.



Technique 2: Optimized Rejection Sampling

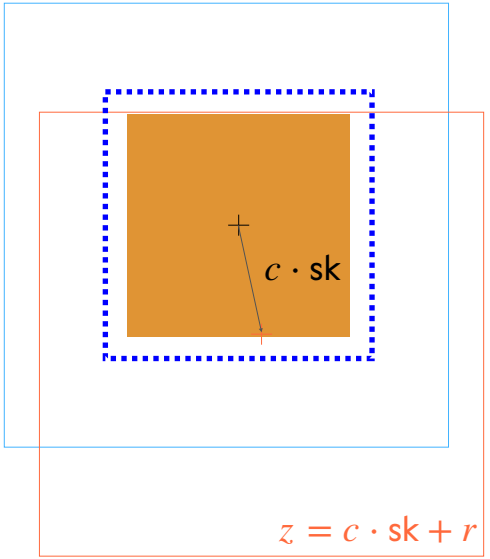
When T users sign → proba that all parties pass rejection sampling is p^T.

Exponential degradation over centralized setting.

Sample r in a centered **hypercube**.

Then, the distribution of z depends on the secret.

We reject any z outside of  .
The resulting distribution is independent of the secret.



Technique 2: Optimized Rejection Sampling

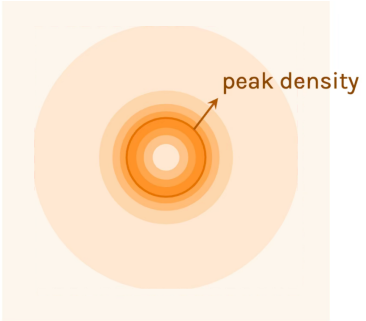
When T users sign → proba that all parties pass rejection sampling is p^T.

Exponential degradation over centralized setting.

Sample r from a **Gaussian** distribution.

Then, the distribution of z depends on the secret.

We rejection sampling $\mathbf{z}$ so that it is distributed according to a centered Gaussian independent of the secret.



Technique 2: Optimized Rejection Sampling

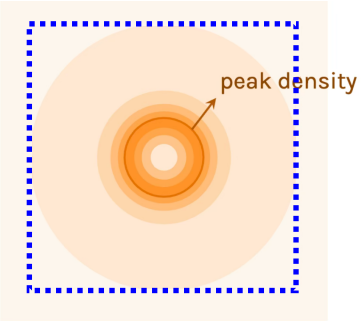
When T users sign → proba that all parties pass rejection sampling is p^T.

Exponential degradation over centralized setting.

Sample r from a **Gaussian** distribution.

Then, the distribution of z depends on the secret.

We rejection sampling $\mathbf{z}$ so that it is distributed according to a centered Gaussian independent of the secret.



Technique 2: Optimized Rejection Sampling

1. We can accept a somewhat low success probability by performing K attempts in parallel.

2. **Unbalanced constraints:** The aggregated signature must be small enough for ML-DSA verification.
   - For the first half $\mathbf{z}$: infinite norm constraint
   - For the second half $\mathbf{y}$ + rounding: (smaller) infinite norm constraint + deserialization constraint for the recovery of $\lfloor \mathbf{w} \rceil$

   → stronger constraint on second half: we want to use smaller $\mathbf{y}$ than $\mathbf{z}$

3. The size of the Gaussian used is **proportional to the norm of the partial secret** to hide: we minimize the number of secrets used by each party in a session.

Technique 2: Optimized Rejection Sampling

3

The size of the Gaussian used is **proportional to the norm of the partial secret** to hide: we minimize the number of secrets used by each party in a session.

$\binom{N}{N-T+1}$ secrets to partition among T parties.

Ideally, at most $\left\lceil \binom{N}{N-T+1} \Big/ T \right\rceil$ secrets each.

Technique 2: Optimized Rejection Sampling

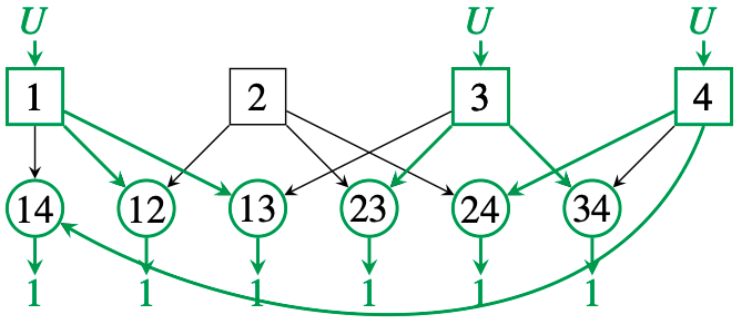
3

The size of the Gaussian used is **proportional to the norm of the partial secret** to hide: we minimize the number of secrets used by each party in a session.

$\binom{N}{N-T+1}$ secrets to partition among T parties.

Ideally, at most $\left\lceil \binom{N}{N-T+1} \Big/ T \right\rceil$ secrets each.

We find an optimal partition with a max-flow algorithm.



Proof Strategy

ML-DSA is proven secure in the centralized setting by replacing signatures revealed by samples independent of the secret key.

Proof Strategy

ML-DSA is proven secure in the centralized setting by replacing signatures revealed by samples independent of the secret key.

Possible in the Random Oracle Model (ROM):

- Sample challenge c, and response $\mathbf{z}$ first (independent of secret).
- Compute consistent commitment $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{z} - c \cdot \mathsf{vk} \rceil$.
- Program random oracle $H(\mathbf{w}, \mathsf{msg}) := c$.

Proof Strategy

ML-DSA is proven secure in the centralized setting by replacing signatures revealed by samples independent of the secret key.

Possible in the Random Oracle Model (ROM):

- Sample challenge c, and response $\mathbf{z}$ first (independent of secret).
- Compute consistent commitment $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{z} - c \cdot \mathsf{vk} \rceil$.
- Program random oracle $H(\mathbf{w}, \mathsf{msg}) := c$.

This works well because rejected values (dependent on the secret) are completely discarded.

Proof Strategy

In the threshold setting, $\mathbf{w}_i$ for discarded values is still revealed.

Proof Strategy

In the threshold setting, $\mathbf{w}_i$ for discarded values is still revealed.

Prior techniques:

- Left-over hash lemma / regularity lemma in [BTT22].
- Renyi divergence + search-to-decision reduction of MLWE in [DFPS23].

→ Does not apply to ML-DSA parameters.

Proof Strategy

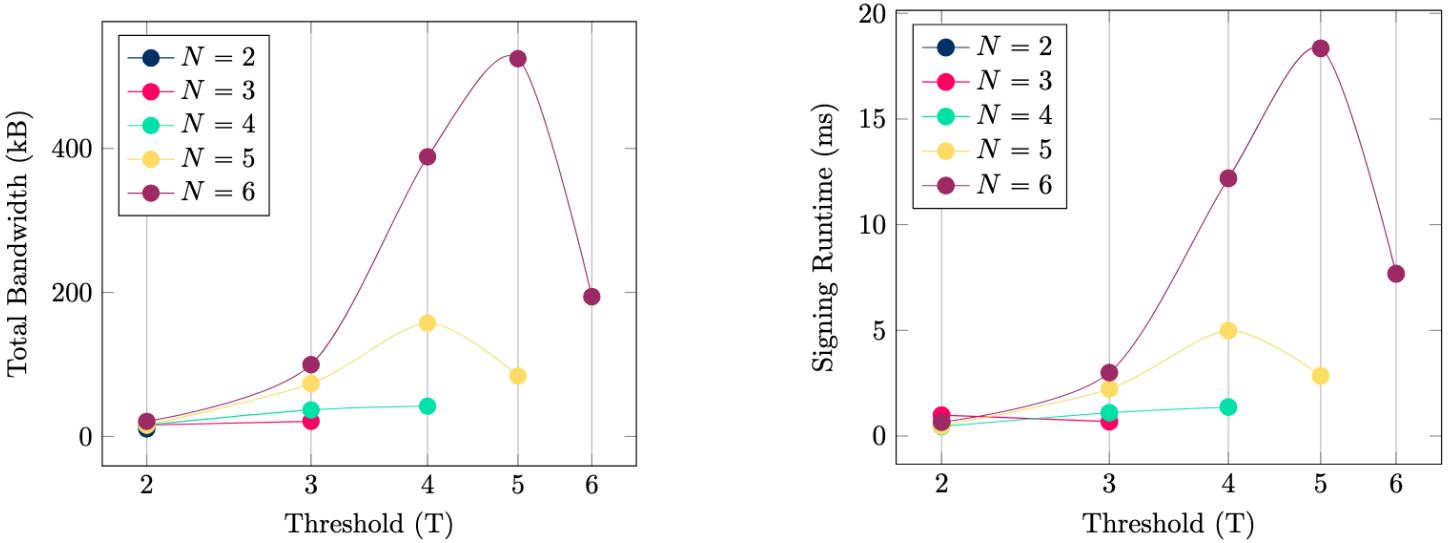
In the threshold setting, $\mathbf{w}_i$ for discarded values is still revealed.

Lemma: Rejected $\mathbf{w}_i$ is indistinguishable from uniform if:

- MLWE is hard over randomness and response distributions.

Performance: Bandwidth and latency

Parameters aim for a success probability 1/2 for each attempt (vs ~1/4 in original ML-DSA).



Bandwidth and latency of threshold signing for ML-DSA 44 (on a local network)

Parties are executed in parallel, and we average over successful attempts.

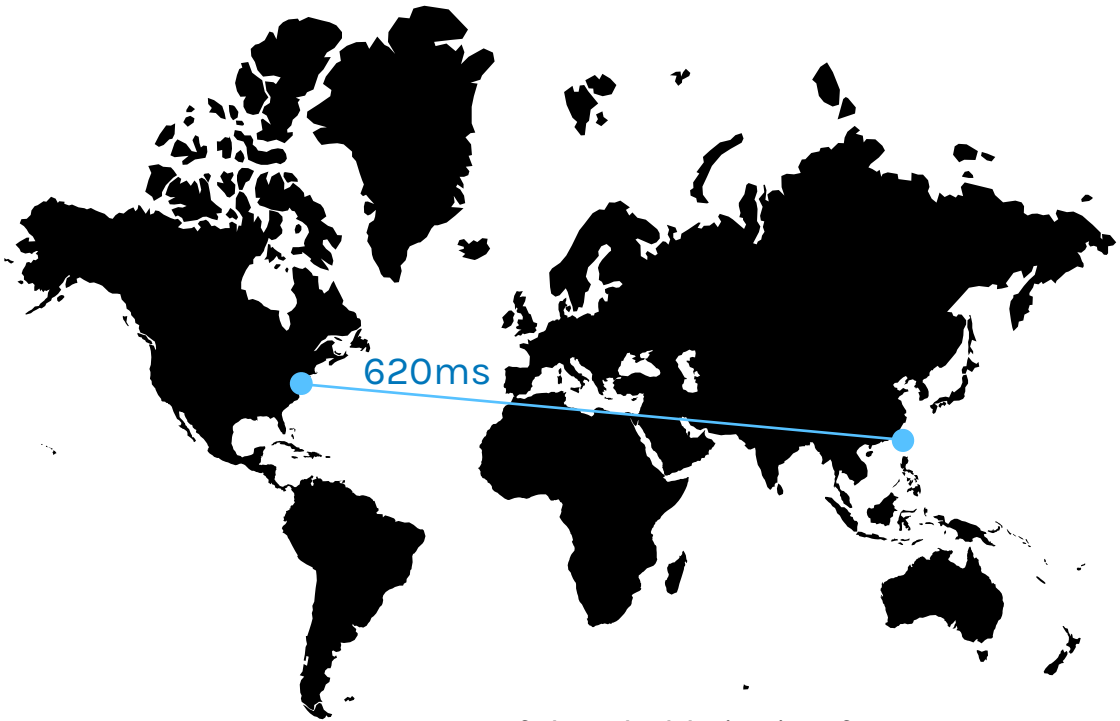
Performance: WAN latency

T=2, N=6

Latency of threshold signing for ML-DSA 44.
Parties are executed in parallel, and we average over successful attempts.

Performance: WAN latency

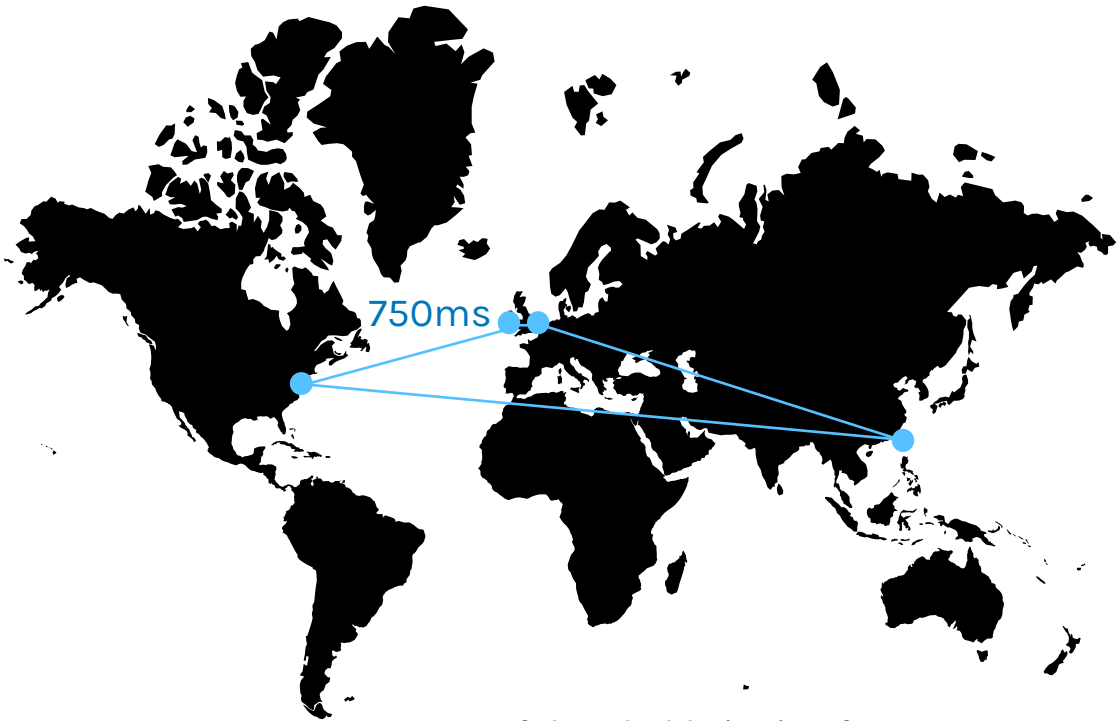
T=2, N=6



Latency of threshold signing for ML-DSA 44.
Parties are executed in parallel, and we average over successful attempts.

Performance: WAN latency

T=4, N=6



Latency of threshold signing for ML-DSA 44.
Parties are executed in parallel, and we average over successful attempts.

Threshold ML-DSA

Scheme	Feature	Signature Size
Mithril	Standard compatibility	2.5 kB (ML-DSA)

- Signing protocol in 3 rounds.
- Support Distributed Key Generation.
- Communication between 20kB and 1MB for *a few parties*.

Open Problems

- For Mithril:
  - Formal verification
  - Optimized implementations (memory, side-channel resistance)
  - More properties (key refresh in Mithril, a posteriori key sharing in constant time)
- Threshold schemes at large
  - DKG and stronger security models
  - Efficient solutions for blockchain consensus (requires signature aggregation)
  - Threshold variants for more NIST Additional Signature Candidates?

Questions?

Bonus: Distributed Key Generation

ML-DSA* . Keygen() → sk, vk

- For every possible set I of $N - T + 1$ parties
  - $\mathsf{vk}_I = \mathbf{A} \cdot \mathsf{sk}_I + \mathbf{e}_I$, where $\mathsf{sk}_I, \mathbf{e}_I$ short
  - Distribute $\mathsf{sk}_I, \mathbf{e}_I$ to parties in I
- $\mathsf{vk} = \sum_i \mathsf{vk}_I$

Core idea: parties in each group I will agree on a shared secret $\mathsf{sk}_I, \mathbf{e}_I$.

Bonus: Distributed Key Generation

ML-DSA*.Keygen() → sk, vk

- For every possible set I of $N - T + 1$ parties
  - $\mathsf{vk}_I = \mathbf{A} \cdot \mathsf{sk}_I + \mathbf{e}_I$, where $\mathsf{sk}_I, \mathbf{e}_I$ short
  - Distribute $\mathsf{sk}_I, \mathbf{e}_I$ to parties in I
- $\mathsf{vk} = \sum_i \mathsf{vk}_I$

Core idea: parties in each group I will agree on a shared secret $\mathsf{sk}_I, \mathbf{e}_I$.

Challenge: we rely on the honest sampling of secrets to bound the norm of several secrets.

→ We can rely on the ROM to enforce honest sampling.

Bonus: Distributed Key Generation

ML-DSA*.Keygen() → sk, vk

- For every possible set I of $N - T + 1$ parties
  - $\mathsf{vk}_I = \mathbf{A} \cdot \mathsf{sk}_I + \mathbf{e}_I$, where $\mathsf{sk}_I, \mathbf{e}_I$ short
  - Distribute $\mathsf{sk}_I, \mathbf{e}_I$ to parties in I
- $\mathsf{vk} = \sum_i \mathsf{vk}_I$

Core idea: parties in each group I will agree on a shared secret $\mathsf{sk}_I, \mathbf{e}_I$.

Challenge: we rely on the honest sampling of secrets to bound the norm of several secrets.

→ We can rely on the ROM to enforce honest sampling.

Round 1: parties agree on a shared secret for each group I (ensure secrecy).

Round 1-2: Parties agree on a global **seed** (ensure honest sampling)

At the end of round 2, parties derive secrets as $\mathsf{sk}_I, \mathbf{e}_I = H(\mathsf{ss}_I, \mathsf{seed}, I)$

Round 3-4: Parties reveal partial public keys $\mathsf{vk}_I = \mathbf{A} \cdot \mathsf{sk}_I + \mathbf{e}_I$, and aggregate them in the final key.