Mithril: Efficient Threshold ML-DSA from Short Secret Sharing

NIST MPTS Workshop 2026 - 01/2026

Rafael del Pino, Sofía Celi, Gustavo Delerue, Thomas Espitau, **Guilhem Niot**, Thomas Prest

Efficient Post-Quantum Threshold Signatures?

Raccoon

Threshold Raccoon: Practical Threshold Signatures from Standard Lattice Assumptions

Rafael del Pino[1], Shuichi Katsumata[1,2], Mary Maller[1,3], Fabrice Mouhartem[4], Thomas Prest[1], Markku-Juhani Saarinen[1,5]

Ringtail**: Practical Two-Round Threshold Signatures from Learning with Errors**

Cecilia Boschini
ETH Zürich, Switzerland

Darya Kaviani
UC Berkeley, USA

Russell W. F. Lai
Aalto University, Finland

Giulio Malavolta
Bocconi University, Italy

Akira Takahashi
JPMorgan AI Research & AlgoCRYPT CoE, USA

Mehdi Tibouchi
NTT Social Informatics Laboratories, Japan

Two-Round Threshold Signature from Algebraic One-More Learning with Errors

Thomas Espitau[1], Shuichi Katsumata[1,2], Kaoru Takemure* [1,2]

Efficient Post-Quantum Threshold Signatures?

Raccoon

Ringtail: Practical Two-Round Threshold Signatures from Learning with Errors

Cecilia Boschini, *ETH Zürich, Switzerland*; Darya Kaviani, *UC Berkeley, USA*; Russell W. F. Lai, *Aalto University, Finland*; Giulio Malavolta, *Bocconi University, Italy*; Akira Takahashi, *JPMorgan AI Research & AlgoCRYPT CoE, USA*; Mehdi Tibouchi, *NTT Social Informatics Laboratories, Japan*

Threshold Raccoon: Practical Threshold Signatures from Standard Lattice Assumptions

Rafael del Pino[1], Shuichi Katsumata[1,2], Mary Maller[1,3], Fabrice Mouhartem[4], Thomas Prest[1], Markku-Juhani Saarinen[1,5]

Two-Round Threshold Signature from Algebraic One-More Learning with Errors

Thomas Espitau[1], Shuichi Katsumata[1,2], Kaoru Takemure* [1,2]

In 2023, NIST selected 3 post-quantum signature schemes for standardization.

ML-DSA

FN-DSA

Based on lattices

SLH-DSA

Based on hash functions

ML-DSA signatures

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

- vk $= \mathbf{A} \cdot \mathsf{sk} + \mathbf{e}$, for sk, $\mathbf{e}$ short

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

Signing $\rightarrow$ Fiat-Shamir transform applied to protocol proving knowledge of $(\mathsf{sk}, \mathsf{e})$

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

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

ML-DSA signatures

ML-DSA . Keygen() → sk, vk

- vk $= \mathbf{A} \cdot$ 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, e)

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:
   - 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.

Distributing ML-DSA

MPC + UC framework

- Protocol simulatable from a trusted execution

Quorum

Trilithium

Tailored + Game-based

- Focus on specific properties, *unforgeability* and *correctness*

Mithril

Key properties

Security:

- Dishonest Majority (up to $T-1$ corruptions)
- Active security
- Arguably adaptive security

Mithril

Compatibility: Valid FIPS 204 signatures.

Distributing ML-DSA

ML-DSA . Keygen() → sk, vk

- vk $= \mathbf{A} \cdot$ sk $+ \mathbf{e}$, for sk, $\mathbf{e}$ short

MLWE **assumption:** vk appears uniformly distributed for $\mathbf{A}$ wide enough (more inputs than outputs)

Signing → prove knowledge of (sk, e)

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 $\mathbf{z} = c \cdot$ sk $+ \mathbf{r}$. Rejection sample:
   - If $\mathbf{z} \notin S$ → **restart**
   - If $\mathbf{z} \in S$ → **output**

Distributing ML-DSA: *Mithril at a high level*

Centralized

Sample short $\mathbf{r}$

Distributed

Sample short $\mathbf{r}_i$

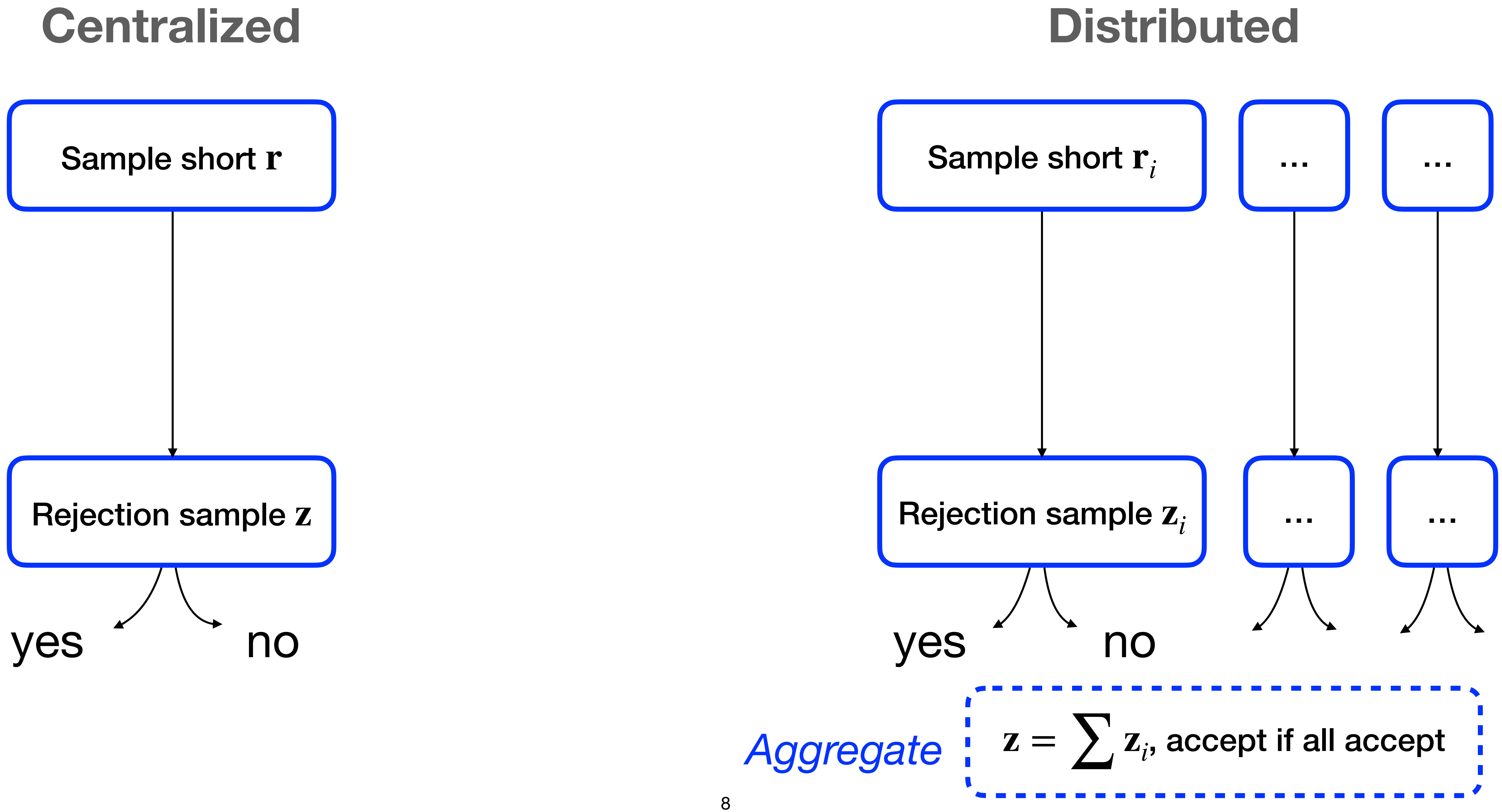
...

...

Aggregate

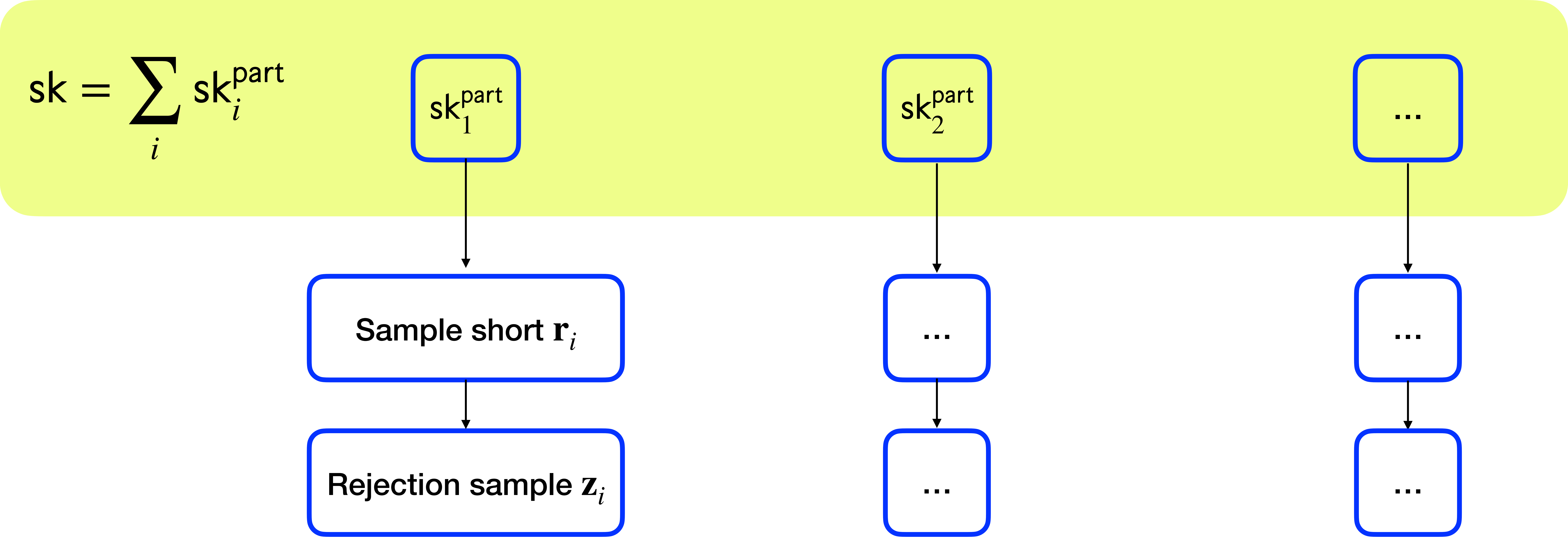
$$\mathbf{r} = \sum_i \mathbf{r}_i$$

Distributing ML-DSA: *Mithril at a high level*



Technique 1: Replicated Secret Sharing

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



Technique 1: Replicated Secret Sharing

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

Use Replicated Secret Sharing as in [dPN25].

ML-DSA*. Keygen() $\rightarrow$ 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
- $\mathsf{vk} = \sum_i \mathsf{vk}_I$

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

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

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 $c = H(\mathbf{w}, \mathsf{msg})$.

3. **Response:** Compute $\mathbf{z} = c \cdot \mathsf{sk} + \mathbf{r}$. Rejection sample:

Our protocol

$\mathsf{Mithril.Sign}(\mathsf{msg}) \rightarrow \mathsf{sig}$

Round 1:

- Sample short $\mathbf{r}_i, \mathbf{e}'_i$
- $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- Broadcast $\mathsf{commit}_i = H(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

- $\mathbf{w} = \sum_i \mathbf{w}_i$ + abort if inconsistent commit_i
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- $\mathbf{z}_i = c \cdot \sum_{I \in m_i} \mathsf{sk}_I + \mathbf{r}_i, \mathbf{y}_i = c \cdot \sum_{I \in m_i} \mathsf{e}_I + \mathbf{e}'_i$
- If $(\mathbf{z}_i, \mathbf{y}_i)$ in S, **broadcast $\mathbf{z}_i$, else abort**

Combine:

- $\mathsf{sig} = (\sum_i \mathbf{z}_i, \lfloor \mathbf{w} \rceil)$
- If sig not in S', **restart**
- **return** sig

Technique 2: Optimized rejection sampling

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

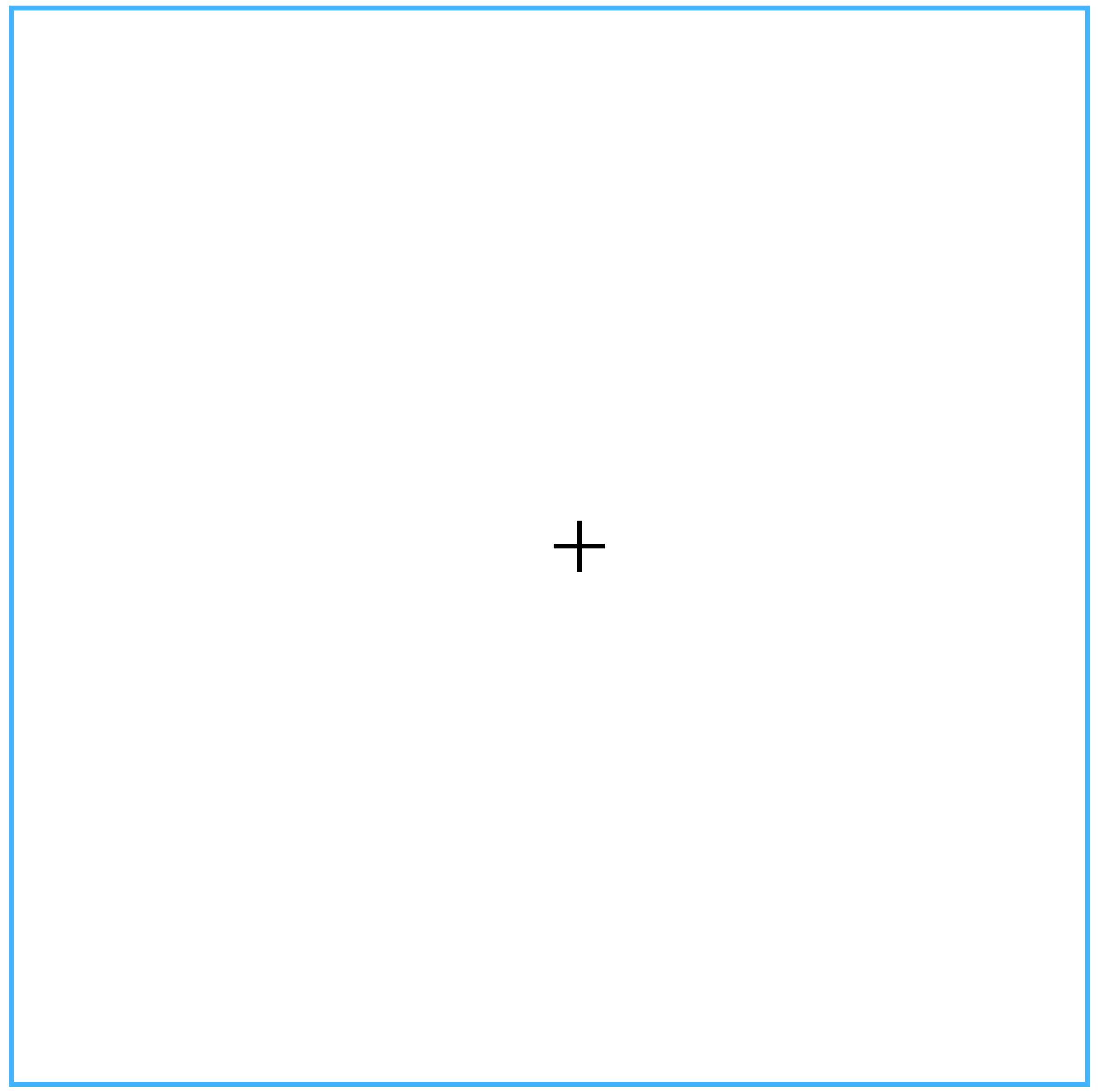
Exponential degradation over centralized setting.

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**.



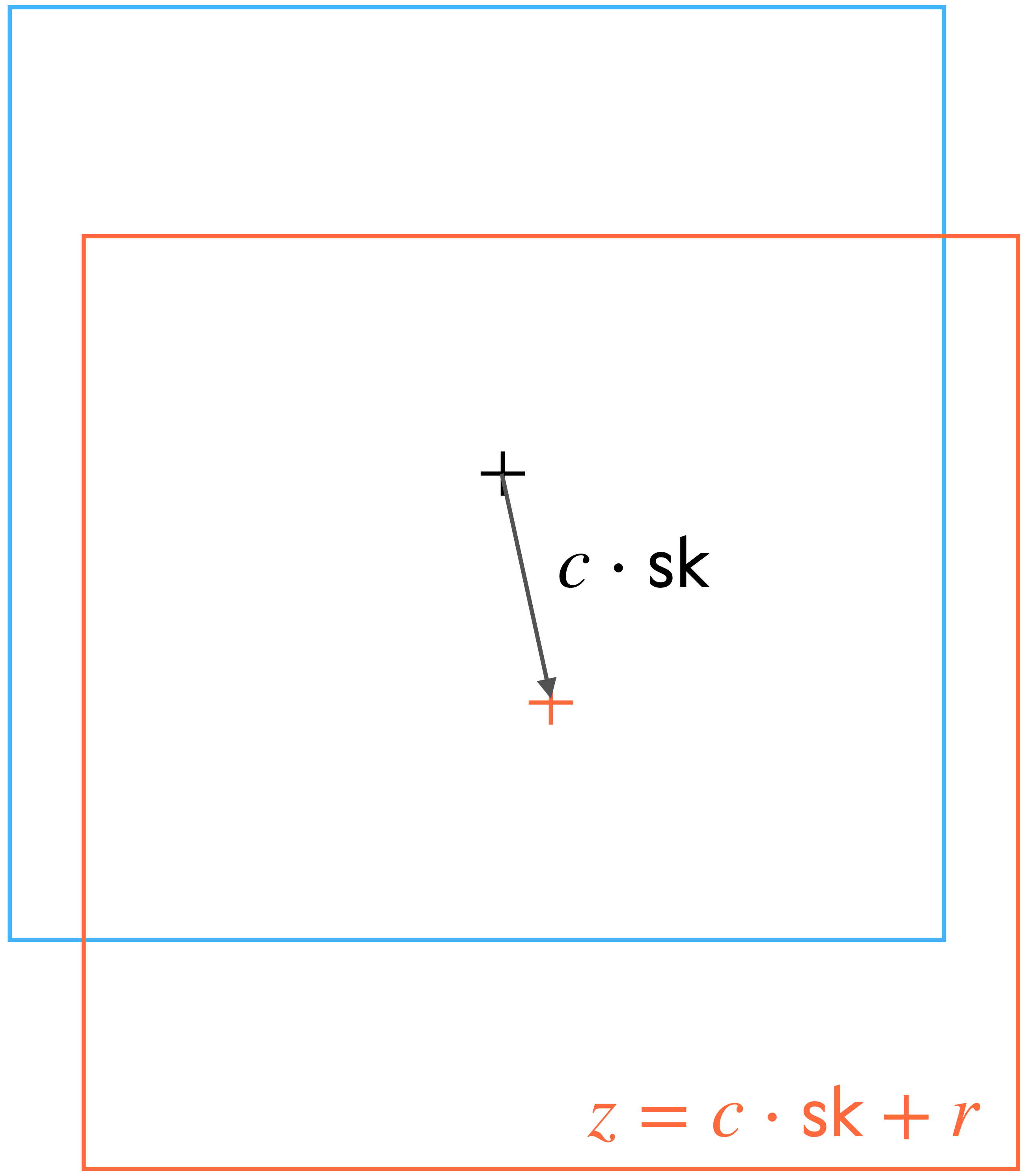
Technique 2: Optimized rejection sampling

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

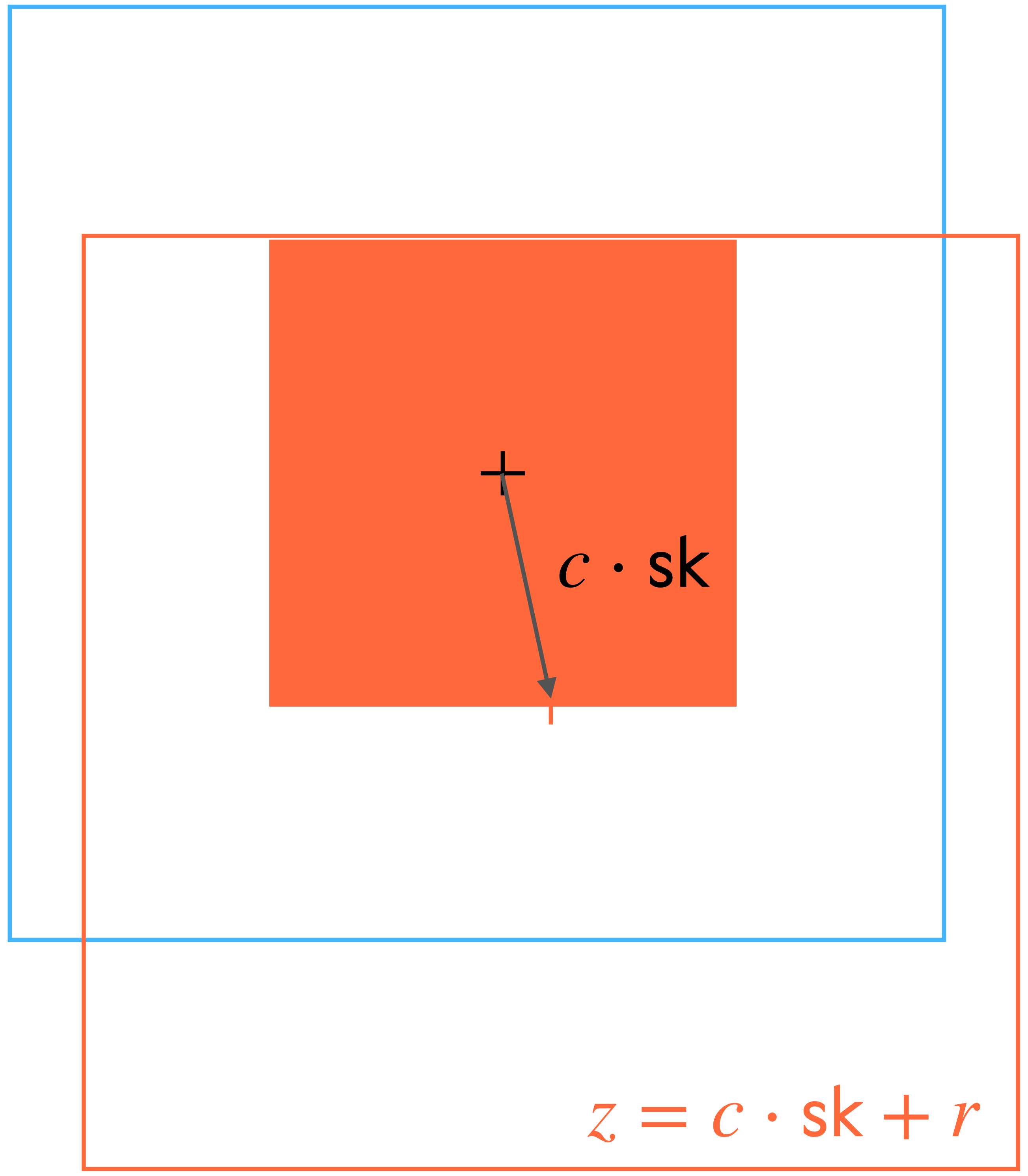
When T users sign → proba that all parties pass rejection sampling is p^T.

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.



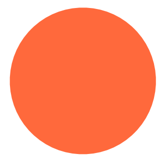
Technique 2: Optimized rejection sampling

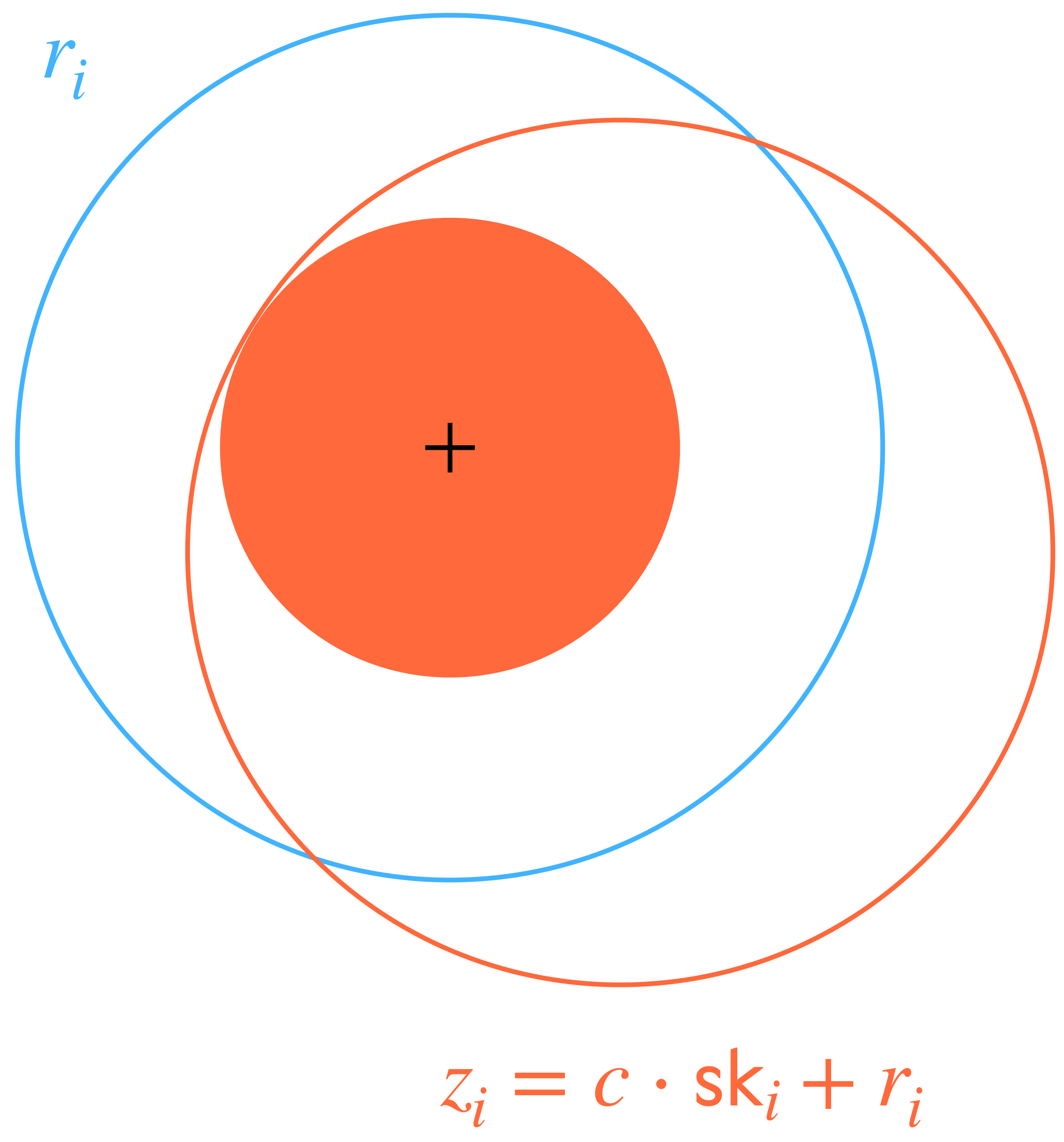
When T users sign → proba that all parties pass rejection sampling is p^T.

Exponential degradation over centralized setting.

Sample r in a centered **hyperball**.

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

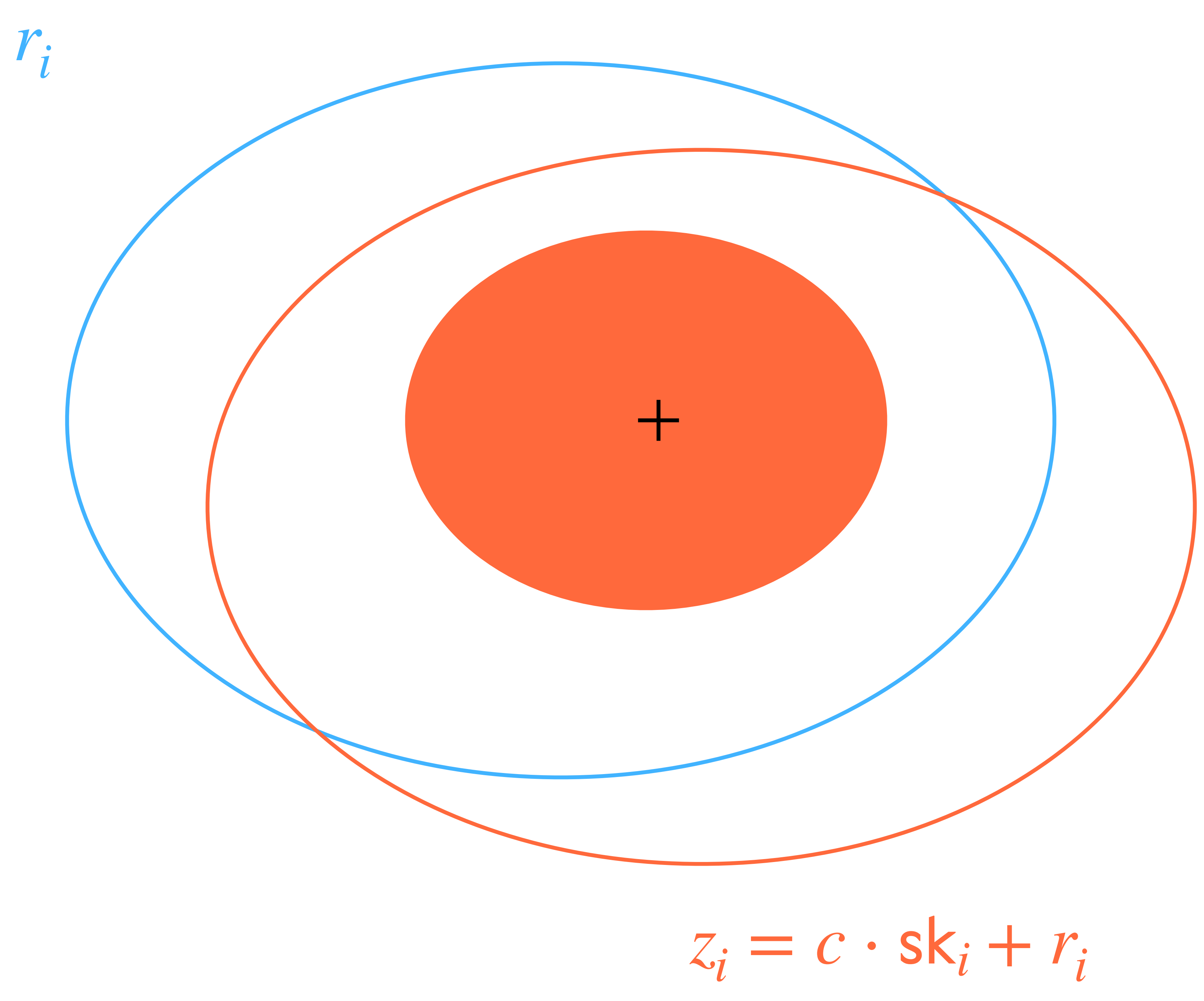
When T users sign → proba that all parties pass rejection sampling is p^T.

Exponential degradation over centralized setting.

Sample r in a centered **hyperball**.

Then, the distribution of z depends on the secret.

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



Key properties

Target: Small number of parties ($N \leq 8$)

Security:

- Dishonest Majority (up to $T-1$ corruptions)
- Active security
- Arguably adaptive security

Mithril

Compatibility: Valid FIPS 204 signatures.

Real-world efficiency: Few rounds, low communication/ computation

Key management

Distributed key generation

4 rounds

High efficiency

A posteriori key distribution

Max 7-12 bits security loss in case of corruptions

Distributed key generation

ML-DSA*. Keygen() $\to$ 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
- $\mathsf{vk} = \sum_i \mathsf{vk}_I$

Distributed key generation

ML-DSA*. Keygen() $\rightarrow$ 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
- $\mathsf{vk} = \sum_i \mathsf{vk}_I$

Rounds 1-2:

- Exchange shared secret K_I for each group I of $N - T + 1$ parties.
- Collaboratively sample coin.

Distributed key generation

ML-DSA* . Keygen() $\rightarrow$ 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
- $\mathsf{vk} = \sum_i \mathsf{vk}_I$

Rounds 1-2:

- Exchange shared secret K_I for each group I of $N - T + 1$ parties.
- Collaboratively sample coin.

Rounds 3-4:

- Derive secrets $\mathsf{sk}_I = H(\text{coin}, K_I)$.
- Commit-and-reveal $\mathsf{vk}_I = [\mathbf{A} \quad \mathbf{I}] \cdot \mathsf{sk}_I$.
- Define $\mathsf{vk} = \sum_I \mathsf{vk}_I$.

A posteriori 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
- $\mathsf{vk} = \sum_i \mathsf{vk}_I$

A posteriori key generation

$\text{ML-DSA}^*.\text{Keygen}() \rightarrow \text{sk}, \text{vk}$

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

Given an ML-DSA secret key sk:

- Sample Gaussians $(\text{sk}_I)_I$ such that $\sum_I \text{sk}_I = \text{sk}$
- Corrupting all but one share can be seen as obtaining a hint on sk.

Implementation

PoC in Go

Already open-source

Reference

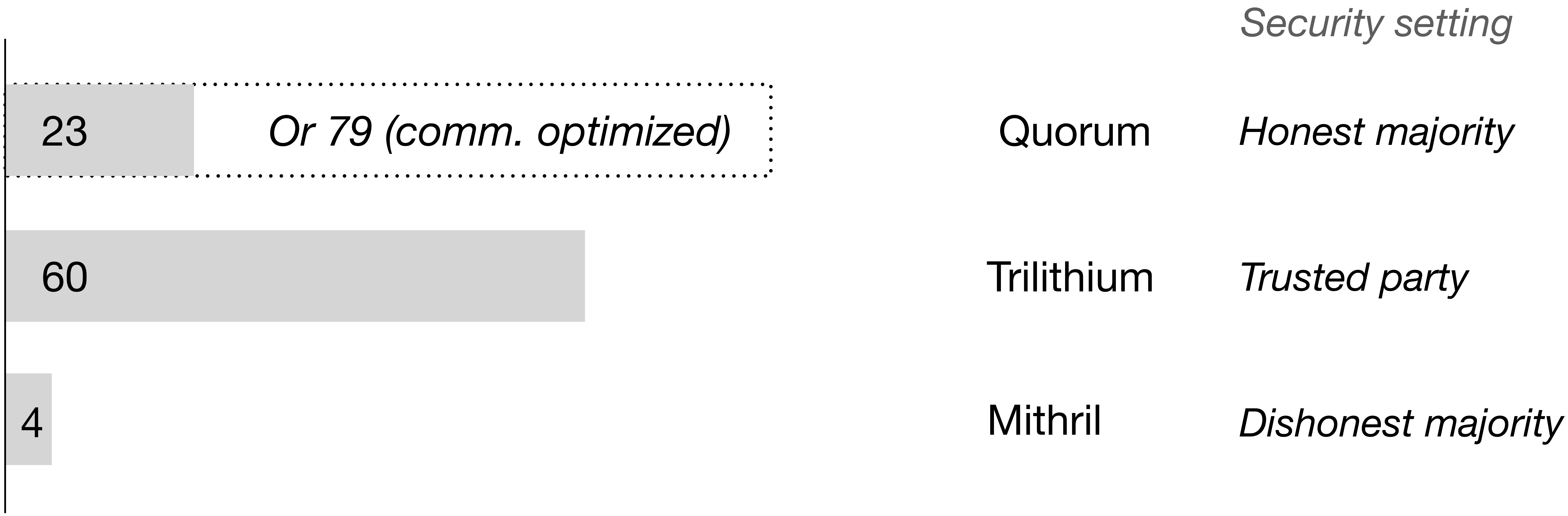
Ongoing work

NIST Submission

Full spec

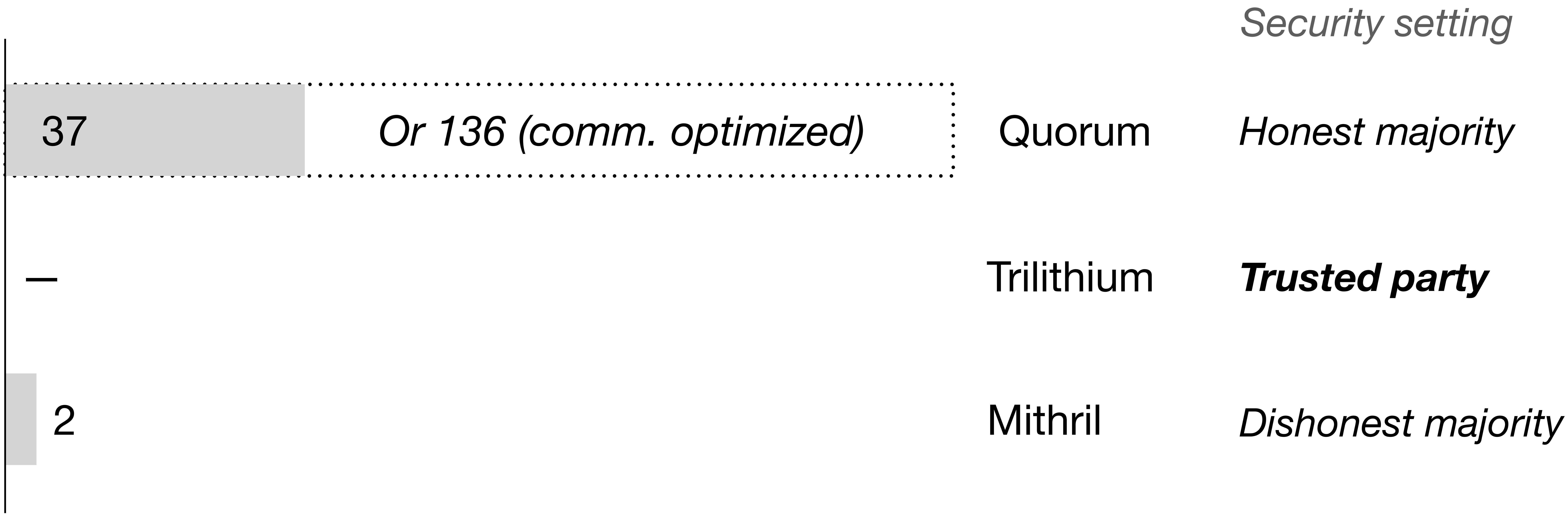
Performance: Number of rounds

Online efficient



Performance: Number of rounds

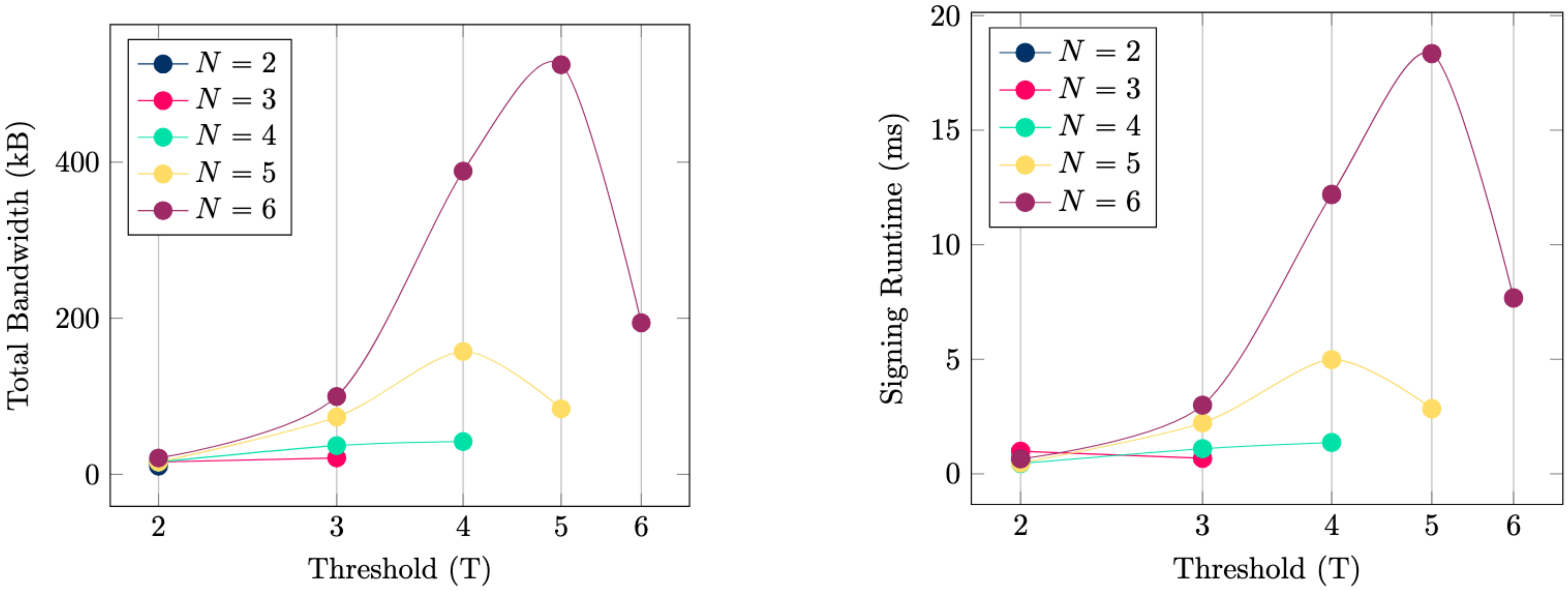
Offline efficient



Performance: Bandwidth and local latency

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

Efficient up to 6 parties.



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.

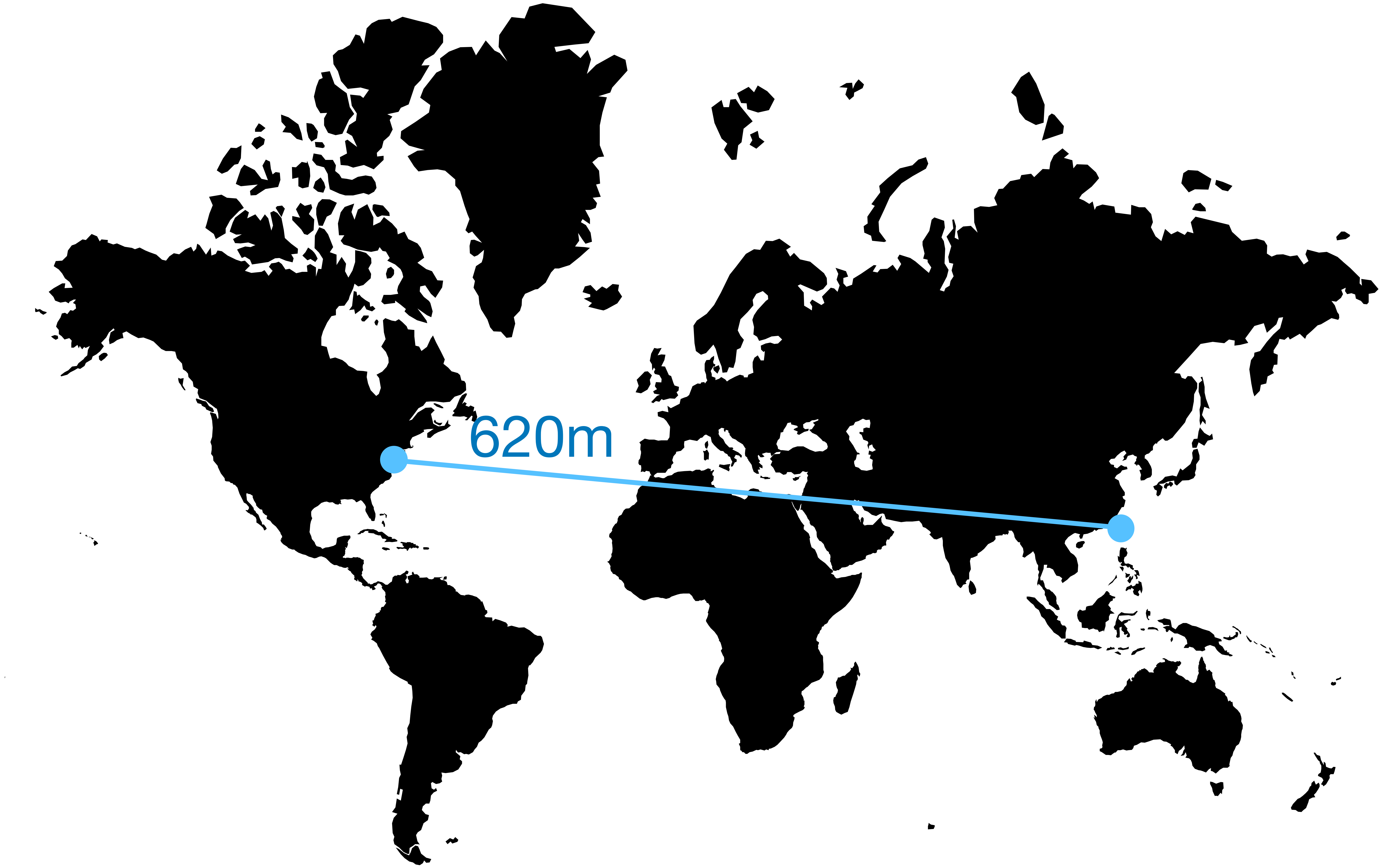
Performance: WAN latency

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

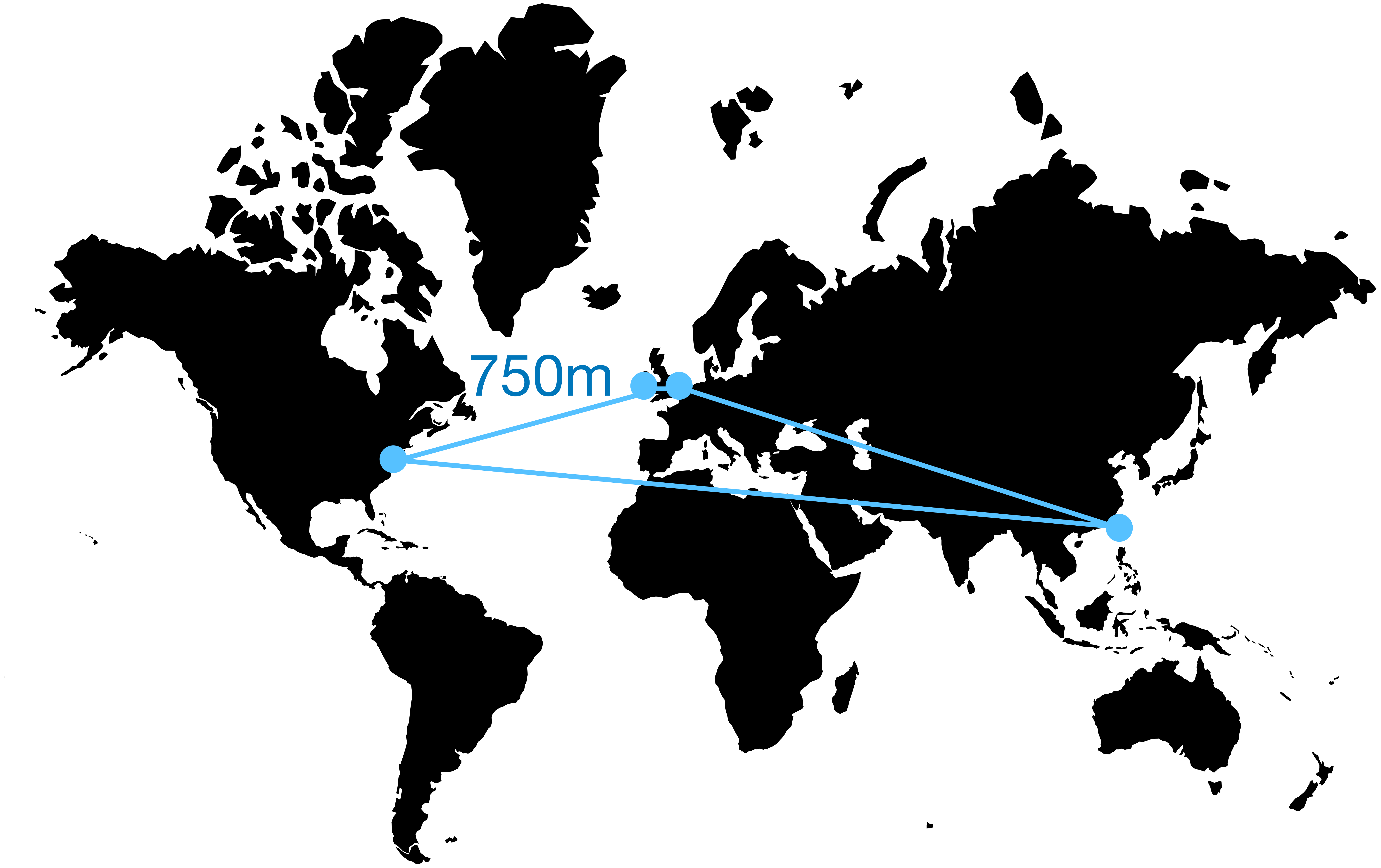
T=2, N=6



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.

Questions?

“Efficient Threshold ML-DSA”

By Rafael del Pino, Sofía Celi, Thomas Espitau, Guilhem Niot, Thomas Prest

USENIX Security 2026

eprint.iacr.org/2026/013

Evaluation

Other ML-DSA parameter sets