Achieving Practical Post-Quantum Threshold Signatures from Lattices

Guilhem Niot

PhD defense - 19/05/2026

Pierre-Alain Fouque (director)

Thomas Prest (co-director)

Cryptography

Science of securing communications in the presence of adversarial behavior.

Digital Signatures: Authenticating Documents

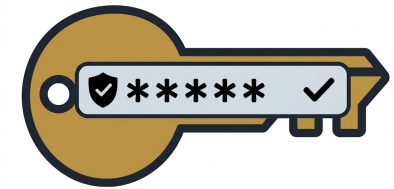
Real-life analogy:

- Only you know how to write your signature.
- Anyone can verify it by comparing against another sample.

1. Introduction

Digital Signatures: Authenticating Documents

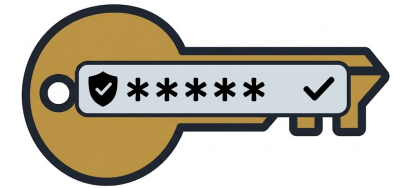
Private Key, kept confidential.
Used for signature creation.



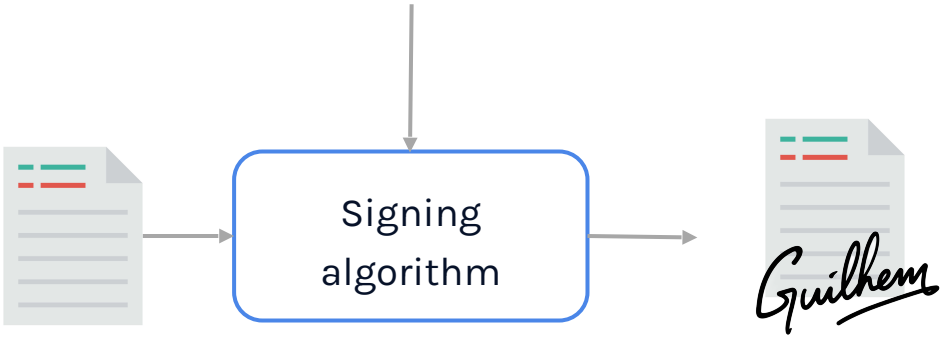
Public Key, available to the public.
Used for signature verification.

Digital Signatures: Authenticating Documents

Private Key, kept confidential.
Used for signature creation.

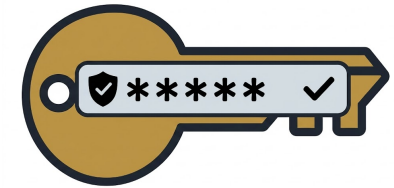


Public Key, available to the public.
Used for signature verification.



Digital Signatures: Authenticating Documents

Private Key, kept confidential.
Used for signature creation.



Public Key, available to the public.
Used for signature verification.

The Need For Post-Quantum Cryptography

Classical Cryptography

RSA: based on the factorization problem. Given $n = p \cdot q$, hard to find p and q.

Elliptic curves: based on the discrete logarithm problem. Given g^a, hard to find a.

The Need For Post-Quantum Cryptography

Classical Cryptography

RSA: based on the factorization problem. Given $n = p \cdot q$, hard to find p and q.

Elliptic curves: based on the discrete logarithm problem. Given g^a, hard to find a.

The Threat of Quantum Computing

Classical bit

Quantum bit (qubit)

Entanglement: Correlate the probabilities of several qubits.

Allows parallel processing and exponential speedups for some problems.

The Need For Post-Quantum Cryptography

Classical Cryptography

RSA: based on the factorization problem. Given $n = p \cdot q$, hard to find p and q.

Elliptic curves: based on the discrete logarithm problem. Given g^a, hard to find a.

The Threat of Quantum Computing

Classical bit

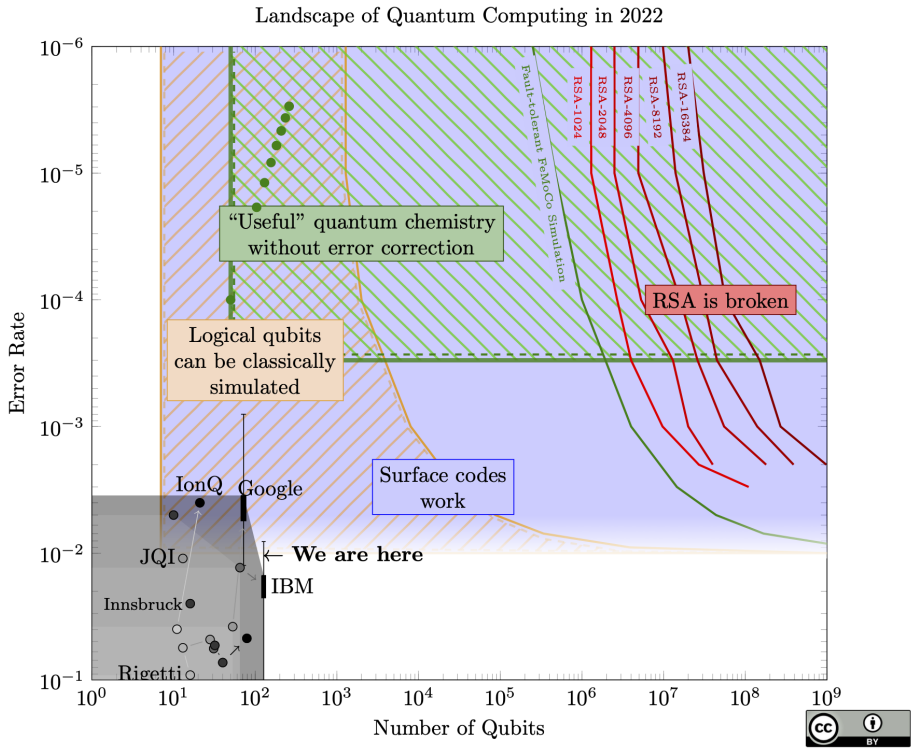
Quantum bit (qubit)

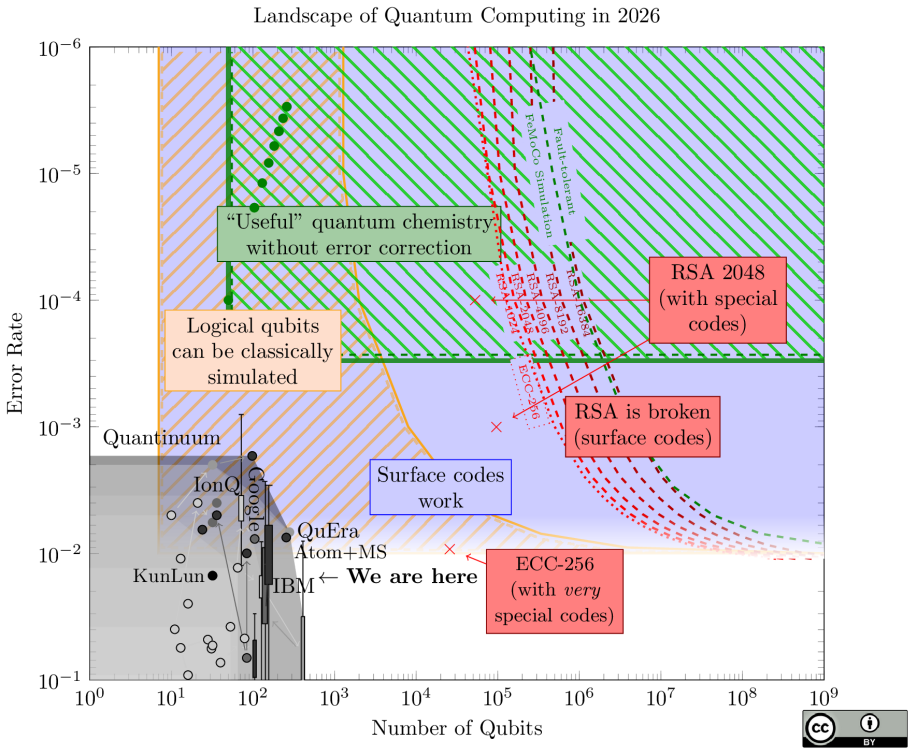
Entanglement: Correlate the probabilities of several qubits.

Allows parallel processing and exponential speedups for some problems.

Shor's algorithm (1994)

Efficiently breaks factorization and discrete logarithm problems with qubits…





State of Quantum Computing, by *Samuel Jaques*

1. Introduction

NIST Standardization Process

2017

Initial call for post-quantum constructions proposals

2022

Algorithms selected

2024

First FIPS standards published

Standardized PQC Algorithms

Encryption (KEM)

- **Kyber** (ML-KEM)
- **HQC** (HQC-KEM)

Digital Signatures

- **Dilithium** (ML-DSA)
- **Falcon** (FN-DSA)
- **Sphincs+** (SLH-DSA)

Ongoing Deployment

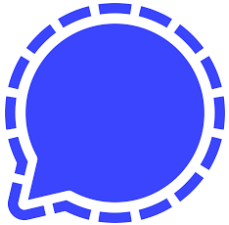
Browsers

Google Chrome

Firefox

...

Secure Messaging



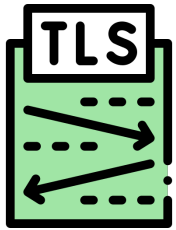
Signal

Apple iMessage

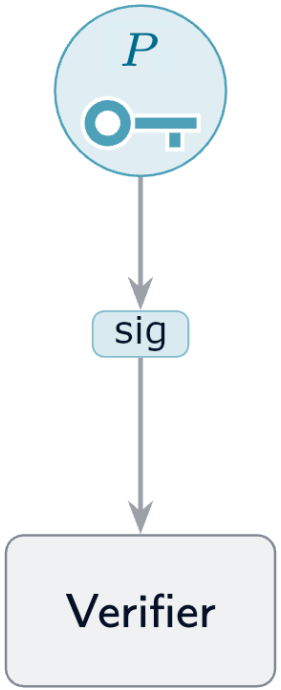
Cloud Providers

What Else Is Needed?

- **Integration Problems:** Integration in remaining existing protocols, within constraints.
  - Remaining challenges in Signal, VPNs (Wireguard), TLS authentication, etc.
- **Need For Advanced Properties:** Some applications are not satisfied with traditional signature schemes.
  - Anonymity, Group settings, Aggregation, etc ...
  - ***In this thesis:*** **High-value settings need higher key protection.**

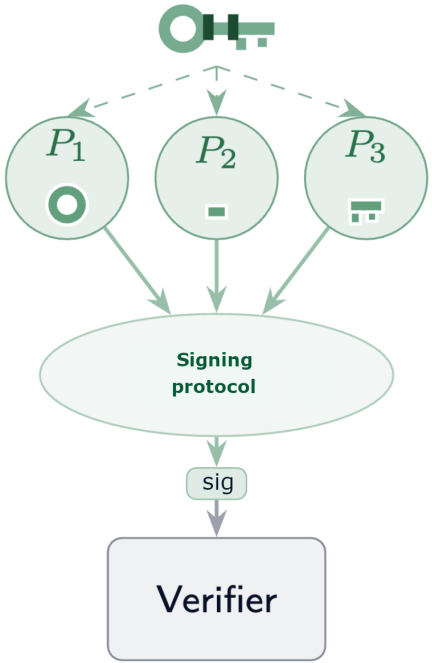


Threshold Signatures



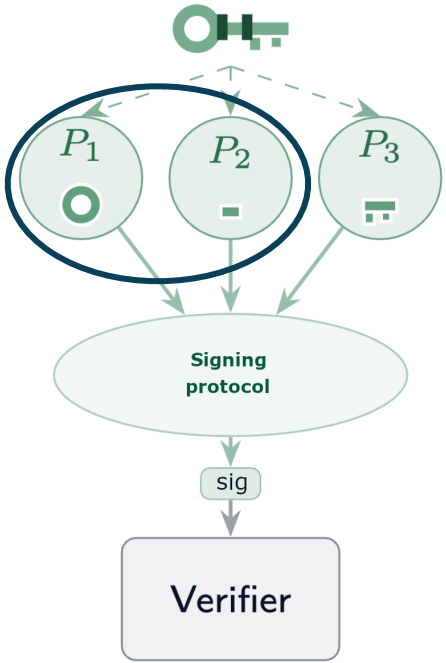
- Generally, the signing key is stored and used from a centralized location.
  - What if *P* becomes unavailable?
  - What if *P* gets corrupted?

Threshold Signatures



- Distribute the signing key to mitigate single point of failures.

Threshold Signatures



- Distribute the signing key to mitigate single point of failures.
- T-out-of-N threshold signing.
  - **Correctness:** Any set of T parties can sign.
  - **Security:** Any set of fewer than T parties cannot.

Threshold Signatures: The Security Model

- **Correctness:** Just check that the signature is valid when any T signers execute the protocol honestly!
- **Security:** *"Any set of fewer than T parties cannot sign."*

Corruption model of up to $T - 1$ parties?

Threshold Signatures: The Security Model

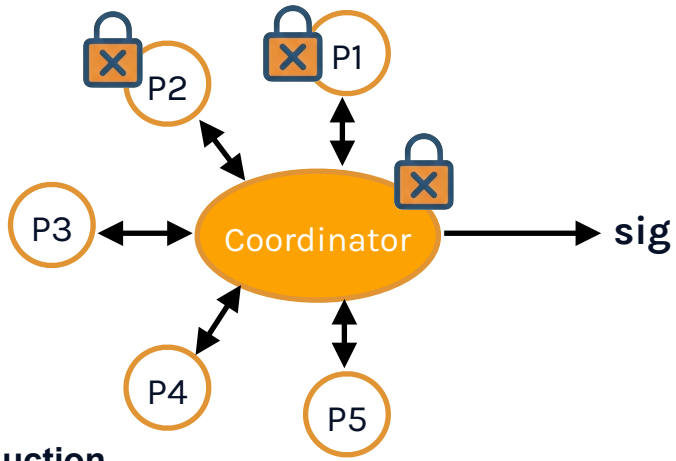
- **Correctness:** Just check that the signature is valid when any T signers execute the protocol honestly!
- **Security:** *"Any set of fewer than T parties cannot sign."*

Corruption model of up to $T - 1$ parties?

Threshold Signatures: The Security Model

- **Correctness:** Just check that the signature is valid when any T signers execute the protocol honestly!
- **Security:** *"Any set of fewer than T parties cannot sign."*

What does it mean to be unable to produce a signature?



- We consider active attackers and a malicious coordinator.
- We count a message as signed as soon as one honest signer accepts it.

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]

Threshold Signatures: The PQ Standards?

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

ML-DSA

(Dilithium)

FN-DSA

(Falcon)

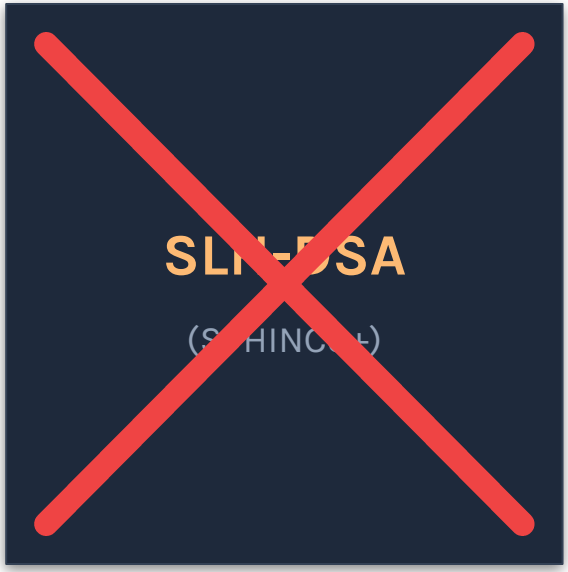
SLH-DSA

(SPHINCS+)

Threshold Signatures: The PQ Standards?

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

Operations too complex to distribute efficiently (e.g. floating point)

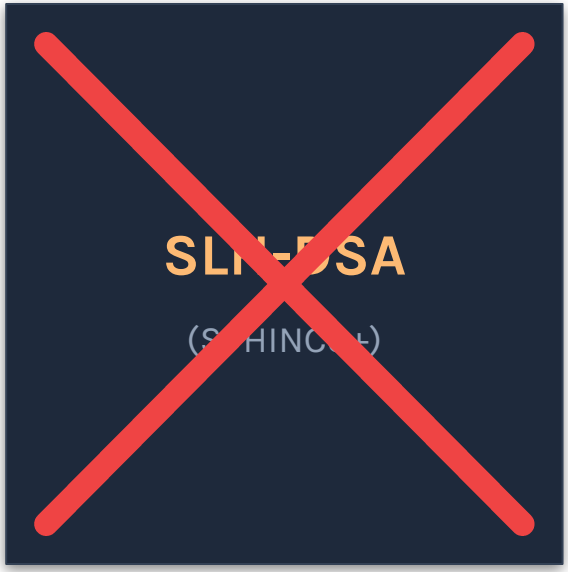


Theoretical solution exists, but impractical due to high storage

Threshold Signatures: The PQ Standards?

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

Operations too complex to distribute efficiently (e.g. floating point)



Theoretical solution exists, but impractical due to high storage

Threshold Signatures: The PQ Standards?

ML-DSA

(Dilithium)

Possible with heavy techniques:

- From **MPC**: [BCEPT25], but high communication
- From **FHE**: [PPS26], but high complexity / computation cost

Threshold Signatures: The PQ Standards?

ML-DSA

(Dilithium)

Possible with heavy techniques:

- From **MPC**: [BCEPT25], but high communication
- From **FHE**: [PPS26], but high complexity / computation cost

The main contribution of this thesis:

Design of an efficient Threshold ML-DSA for a small number of parties (e.g. <10), without heavy techniques.

Approaches For Threshold ML-DSA

Threshold ML-DSA constructions still come with important trade-offs.

Paradigm	# parties supported	Comm./party	Computation	Detect Misbehavior
MPC	∞	**> 1 MB (> 20 rounds)**	Medium	✗
FHE	∞	25 kB (2 rounds)	**High**	✗
This thesis	**< 10**	20 kB - 1MB (3 rounds)	Low	✗

Threshold Signatures: Advanced properties?

Raccoon

Threshold-friendly signature scheme.

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

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]

# parties supported	Comm./party	Computation	Detect Misbehavior
1024	40 kB (3 rounds)	Low	✗

Threshold Signatures: Advanced properties?

Raccoon

Threshold-friendly signature scheme.

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

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]

Second part of this talk:

Threshold Raccoon with protection against misbehavior.

This Thesis & Presentation Roadmap

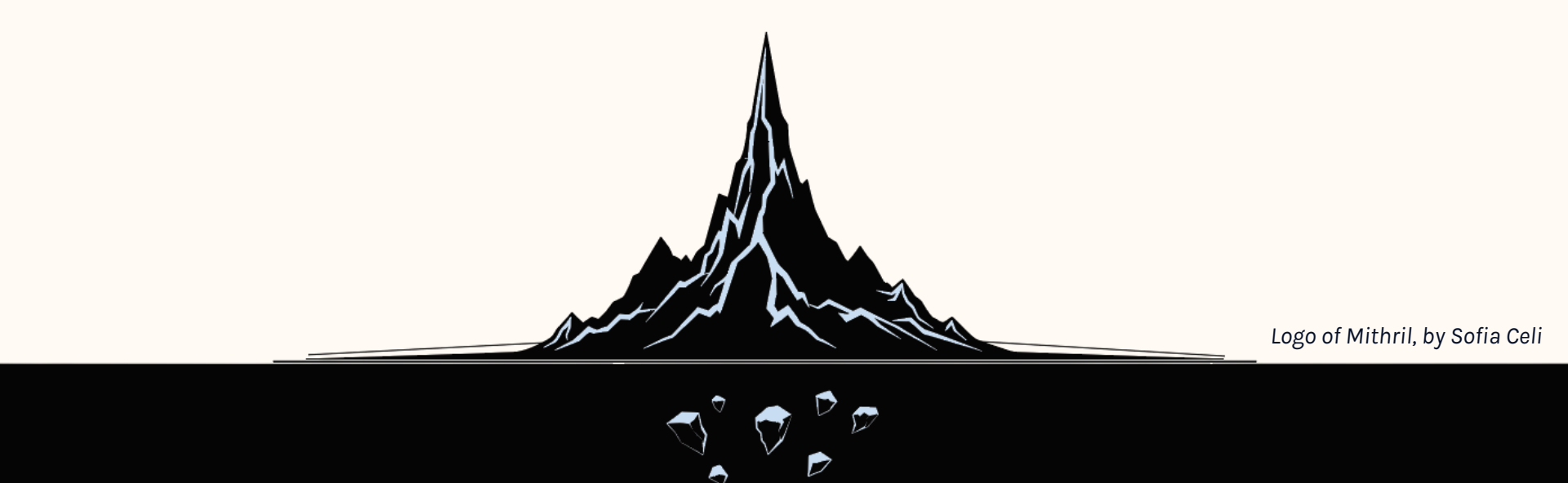
This thesis introduces three new threshold constructions answering open questions.

Scheme	Feature
Mithril	Standard compatibility
IA-Raccoon	Identifies bad behavior
RB-Raccoon	Corrects bad behavior

This talk*

* Focus on the high-level constructions, omitting proof techniques.

Part 1. 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

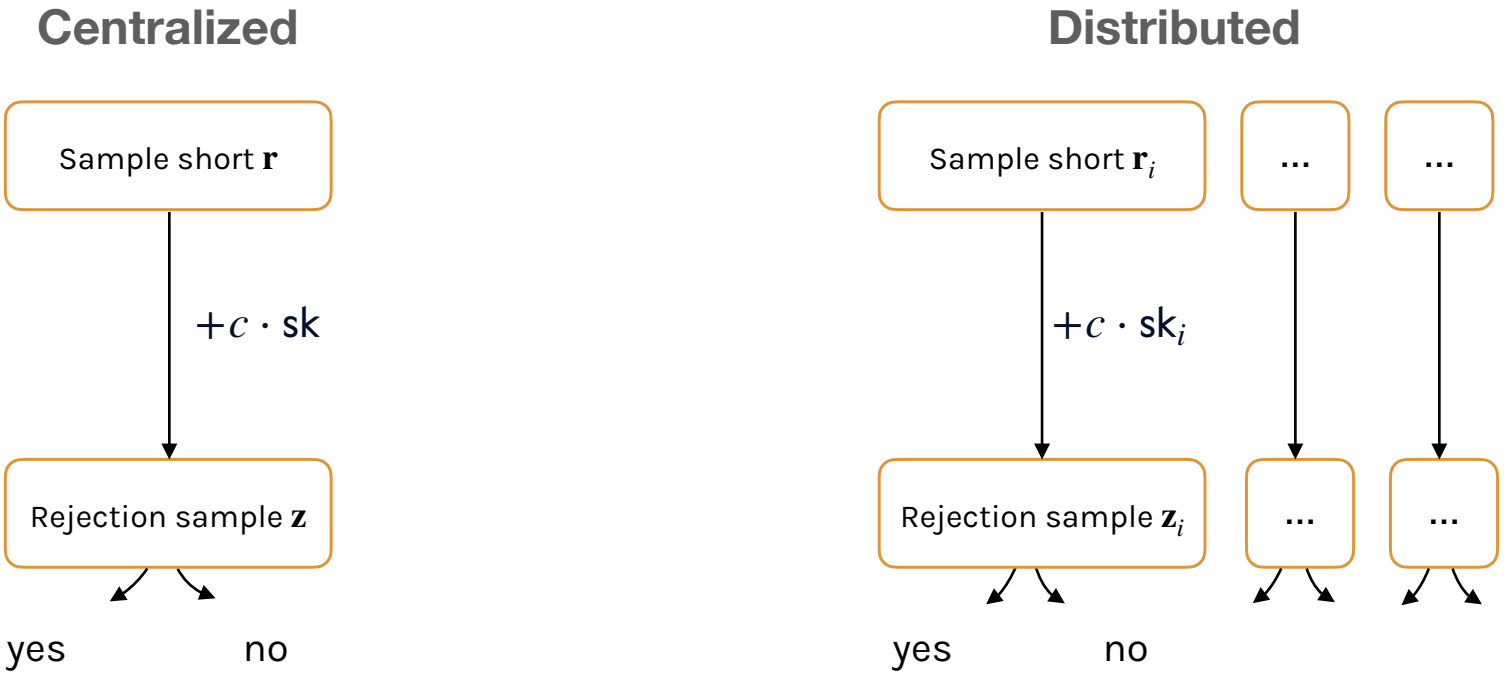
Sample short $\mathbf{r}$

Distributed

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

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

Distributing ML-DSA: Mithril At A High Level

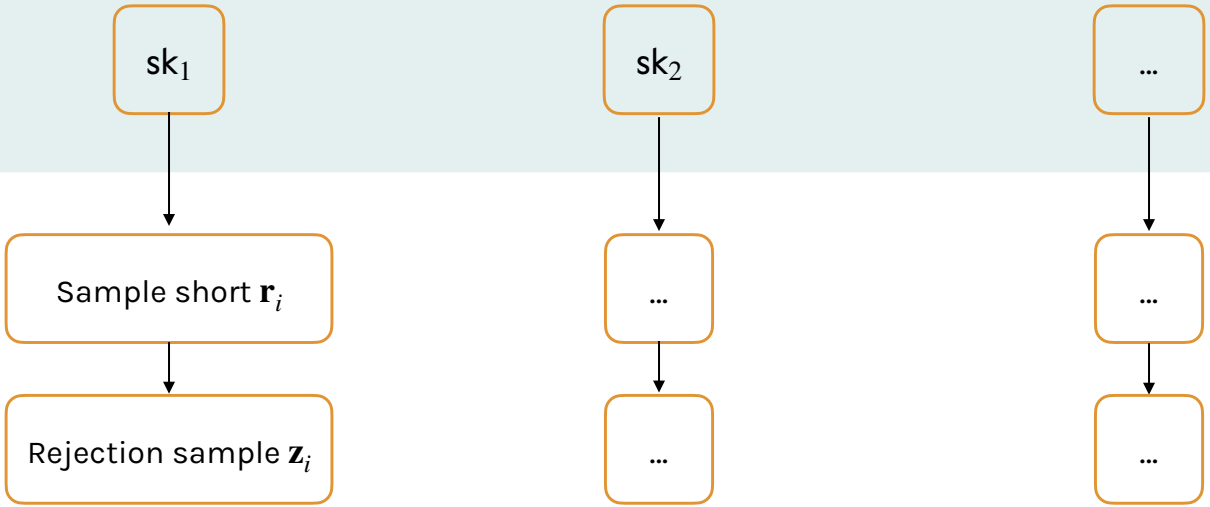


If all parties accepted, aggregate $\mathbf{z} = \sum \mathbf{z}_i$

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.

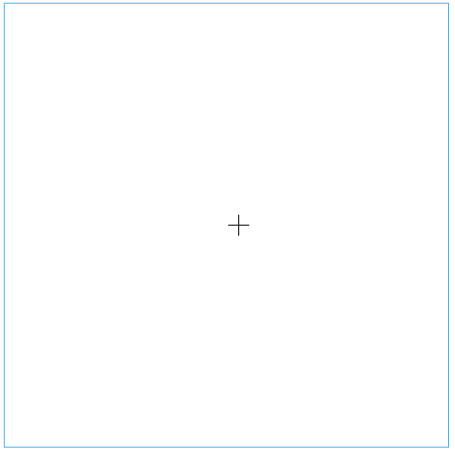
Exponential degradation over centralized setting.

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**.



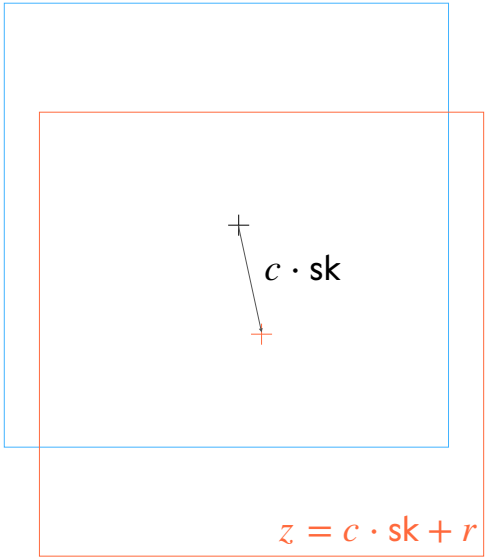
Technique 2: Optimized Rejection Sampling

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

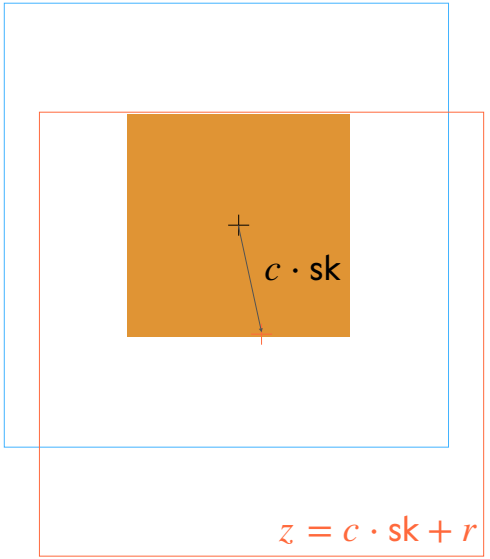
When T users sign → proba that all parties pass rejection sampling is p^T.

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

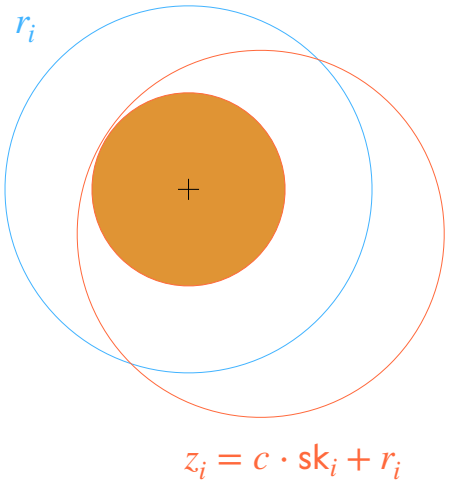
When T users sign → proba that all parties pass rejection sampling is p^T.

Exponential degradation over centralized setting.

Sample r in a centered **hyperball**.

Then, the distribution of z depends on the secret.

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



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

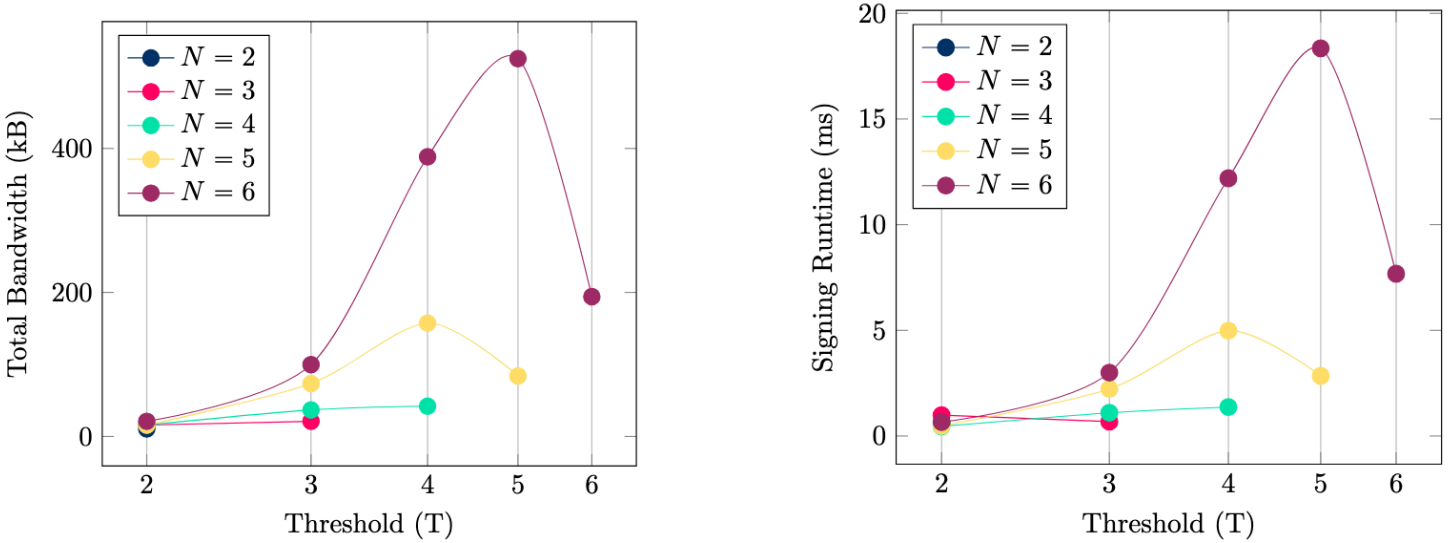
In the threshold setting, $\mathbf{w}_i$ for discarded values is still revealed.

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.

Instantiation

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

... but no mechanism to detect misbehavior making the protocol abort.

Part 2. Advanced Properties With Threshold Raccoon

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**

3. Threshold Raccoon with Misbehavior Detection

Raccoon 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.

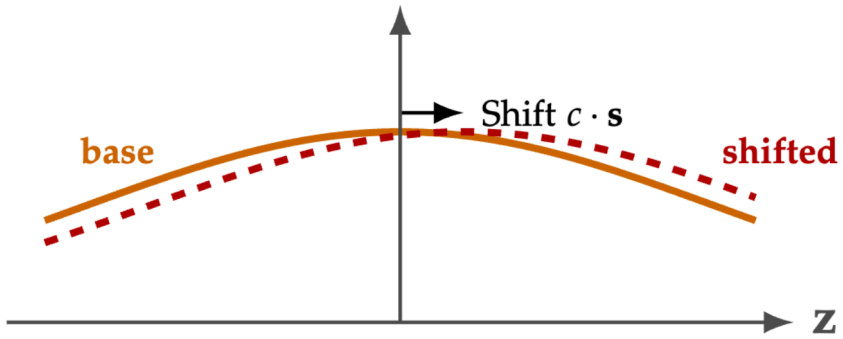
Signing → Fiat-Shamir transform applied to protocol proving knowledge of sk.

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**

3. Threshold Raccoon with Misbehavior Detection

Raccoon Signatures

If $\mathbf{r}$ sufficiently large, the secret shift is hard to detect.



→ Raccoon chooses larger randomness and gets bigger signatures (10 kB vs 2.5 kB for ML-DSA).

Prior Work: Threshold Raccoon Without Misbehavior Detection

Raccoon signing

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

Shamir shares

Threshold Raccoon

ThRaccoon . 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. $\mathbf{z}_i = c \cdot \lambda_i \cdot \text{sk}_i + \mathbf{r}_i + \Delta_i$, where $\Delta_i = \text{ZeroShare(transcript)}$
4. ~~If $(\mathbf{z}_i, \mathbf{y}_i)$ 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

3. Threshold Raccoon with Misbehavior Detection

Adding Misbehavior Detection

Raccoon signing

1. **Randomness + Commitment:** Sample short $\mathbf{r}, \mathbf{e}$, and commit $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} + \mathbf{e} \rceil$.
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

ThRaccoon . 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 $(\mathbf{z}_i, \mathbf{y}_i)$ 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

3. Threshold Raccoon with Misbehavior Detection

Adding Misbehavior Detection

Can we identify malicious behavior in this protocol?

When using short shares as in Mithril, we can simply check that $\mathbf{z}_i$ is short, $\mathbf{w}_i \approx \mathbf{A} \cdot \mathbf{z}_i - c \cdot \mathsf{vk}_i$.

Our protocol

ThRaccoon . 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 $\mathsf{commit}_i = H(\mathbf{w}_i)$

Round 2:

1. Broadcast $\mathbf{w}_i$

Round 3:

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

Combine:

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

3. Threshold Raccoon with Misbehavior Detection

Supporting More Parties?

Sadly, replicated secret sharing has $\binom{N}{T-1}$ shares. Realistically supports up to ~16 parties.

Supporting More Parties?

Sadly, replicated secret sharing has $\binom{N}{T-1}$ shares. Realistically supports up to ~16 parties.

We can adapt a recursive secret sharing from [DDB95] to produce short shares.

[DDB95] Yvo Desmedt, Giovanni Di Crescenzo, and Mike Burmester. Multiplicative non-abelian sharing schemes and their application to threshold cryptography. ASIACRYPT' 94.

Supporting More Parties?

Sadly, replicated secret sharing has $\binom{N}{T-1}$ shares. Realistically supports up to ~16 parties.

We can adapt a recursive secret sharing from [DDB95] to produce short shares.

Sample x_L.

$$x = x_L + \underbrace{(x - x_L)}_{x_R}$$

[DDB95] Yvo Desmedt, Giovanni Di Crescenzo, and Mike Burmester. Multiplicative non-abelian sharing schemes and their application to threshold cryptography. ASIACRYPT' 94.

Supporting More Parties?

Sadly, replicated secret sharing has $\binom{N}{T-1}$ shares. Realistically supports up to ~16 parties.

We can adapt a recursive secret sharing from [DDB95] to produce short shares.

Sample x_L.

$x = x_L + \underbrace{(x - x_L)}_{x_R}$

k-out-of-N/2 (T-k)-out-of-N/2

For any set of T among N users, there will be k users on the left, $T - k$ on the right.

[DDB95] Yvo Desmedt, Giovanni Di Crescenzo, and Mike Burmester. Multiplicative non-abelian sharing schemes and their application to threshold cryptography. ASIACRYPT' 94.

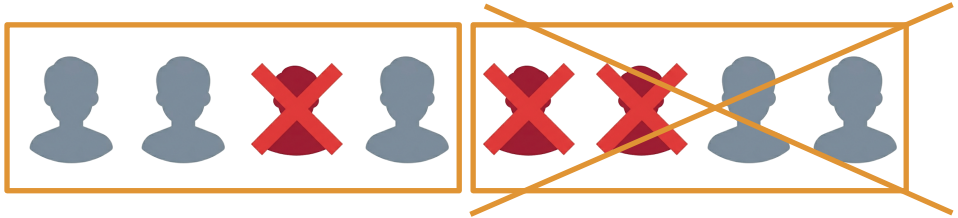
Supporting More Parties?

Sadly, replicated secret sharing has $\binom{N}{T-1}$ shares. Realistically supports up to ~16 parties.

We can adapt a recursive secret sharing from [DDB95] to produce short shares.

Sample x_L.

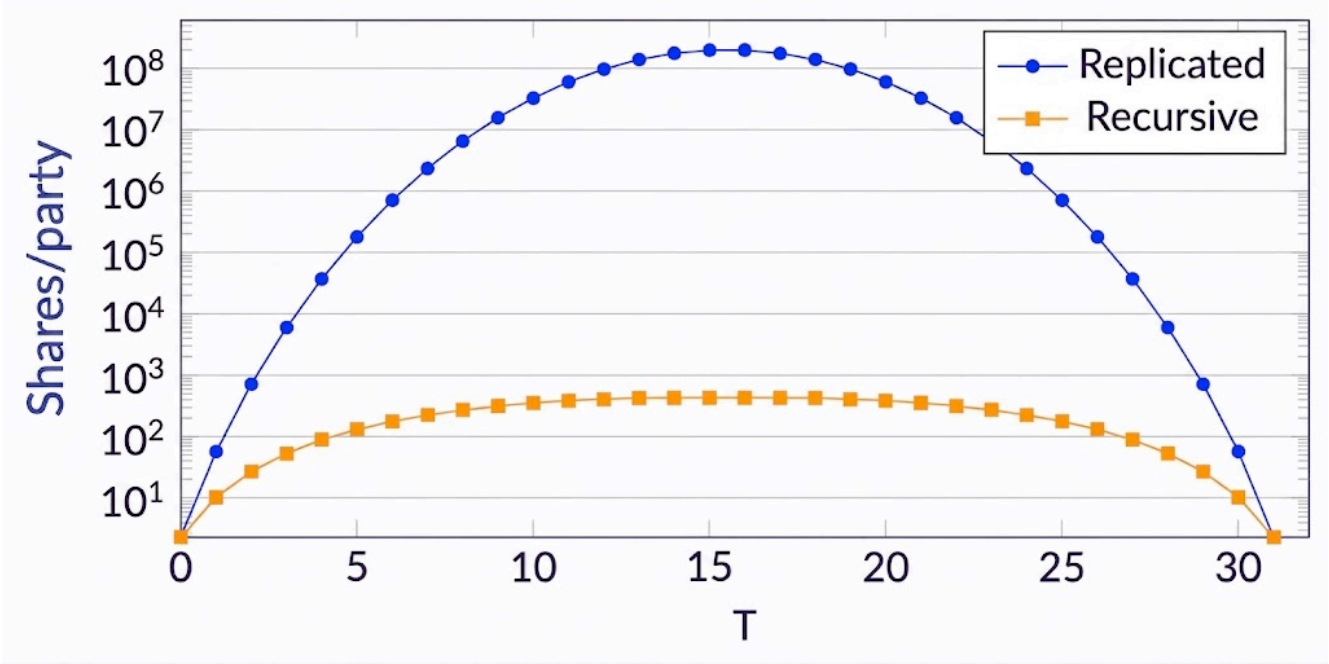
$x = x_L + \underbrace{(x - x_L)}_{x_R}$



As there are at most $T-1$ corrupted parties, less than k on the left OR less than $T-k$ on the right.

[DDB95] Yvo Desmedt, Giovanni Di Crescenzo, and Mike Burmester. Multiplicative non-abelian sharing schemes and their application to threshold cryptography. ASIACRYPT' 94.

Supporting More Parties?



Number of shares for $N = 32$ parties.

3. Threshold Raccoon with Misbehavior Detection

Instantiation

Scheme	Feature	Signature Size
Mithril	Standard compatibility	2.5 kB (ML-DSA)
IA-Raccoon	Identifies bad behavior	12.3 kB

3. Threshold Raccoon with Misbehavior Detection

Threshold Raccoon: Preventing malicious behavior?

In this presentation, we developed techniques to **detect** malicious behavior.

In my thesis, also techniques to **correct** it.

- Use of Shamir Sharing + Distributed proofs of shortness for $\mathbf{r}$.
- Only one share per party → no memory blow-up.
- But,
  - This requires a gap between the security and signing threshold.
  - Larger communication, and slightly larger signature size.

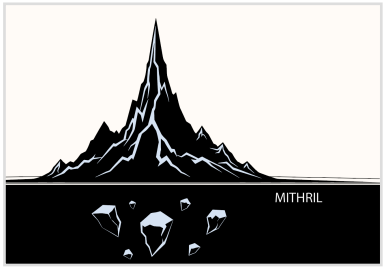
Conclusion

This thesis aimed to increase the practicality of lattice-based threshold signatures.

Scheme	Feature	Signature Size
Mithril	Standard compatibility	2.5 kB (ML-DSA)
IA-Raccoon	Identifies bad behavior	12.3 kB
RB-Raccoon	Corrects bad behavior	15.1 kB

NIST Call For Multi-Party Threshold Cryptography

Since 2019, NIST organized a few workshops to discuss threshold cryptography. In 2026, formal call for proposals: will consolidate knowledge and potentially lead to standards.



Mithril

Threshold ML-DSA

Hermine

2-round Threshold Raccoon

- *Key refresh*
- *Identifiable Aborts*

Vinaigrette

Threshold MAYO

Techniques not discussed in the manuscript.

Open Problems

- For the schemes presented here specifically:
  - Scalability to more parties of Mithril, with better secret sharing
  - 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?