PQC + Threshold

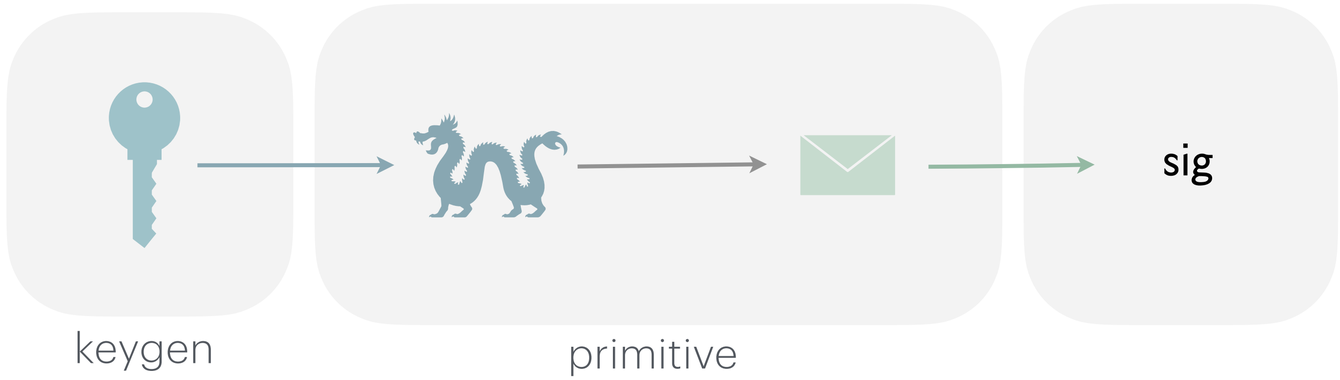
State of the Art in Threshold Quantum-Resistant Signatures

Guilhem Niot

CryptoDay @ Télécom Paris - 18/09/2025

Threshold Signatures

Centralized setting



Threshold Signatures

What if the party is corrupted or becomes unresponsive…

Question: can we split the trust among several parties?

Threshold Signatures

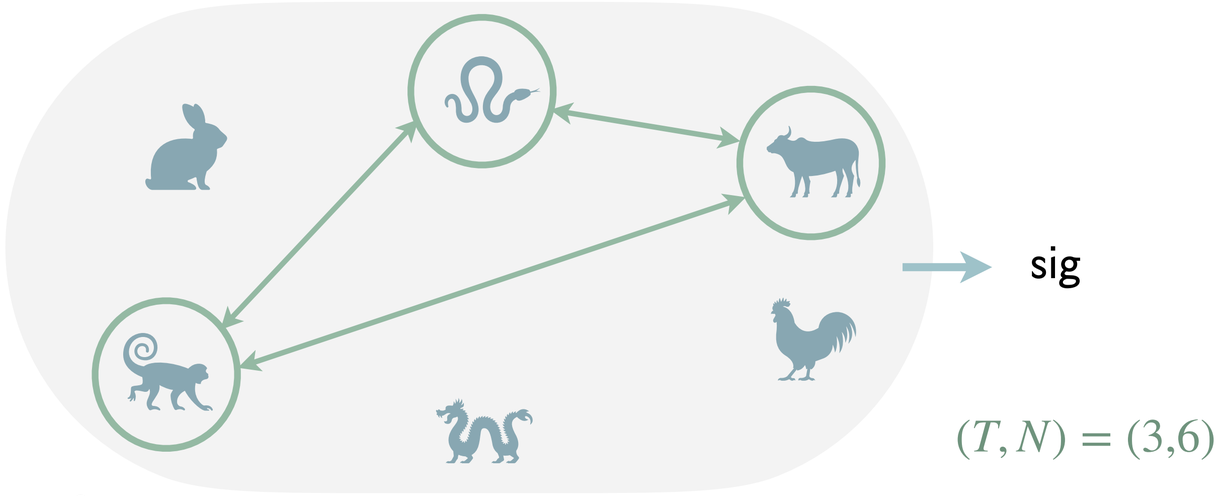
What if the party is corrupted or becomes unresponsive…

Question: can we split the trust among several parties?

Interactive protocol to distribute the scheme:

T-out-of-N parties can collaborate to sign and

$T-1$ parties cannot.



Applications of Threshold Signatures

Cryptocurrency wallets & DeFi

Distributed signing for CDNs

Distributed consensus in Tor

NIST Call for Threshold Schemes

PUBLICATIONS

NIST IR 8214C (2nd Public Draft)

NIST First Call for Multi-Party Threshold Schemes

Date Published: March 27, 2025
Comments Due: April 30, 2025
Email Comments to: nistir-8214C-comments@nist.gov

Author(s)

Luís T. A. N. Brandão (NIST, Strativia), Rene Peralta (NIST)

Announcement

This is a second public draft. Threshold schemes should NOT be submitted until the final version of this report is published. However, the present draft can be used as a baseline to prepare for future submissions.

The scope of the call is organized into categories related to signing (Sign), public-key encryption (PKE), symmetric-key cryptography and hashing (Symm), key generation (KeyGen), fully homomorphic encryption

Goal of this talk

- Provide an overview of the most practical PQ threshold signatures
- Explain their technical and practical differences

The trade-offs of threshold schemes

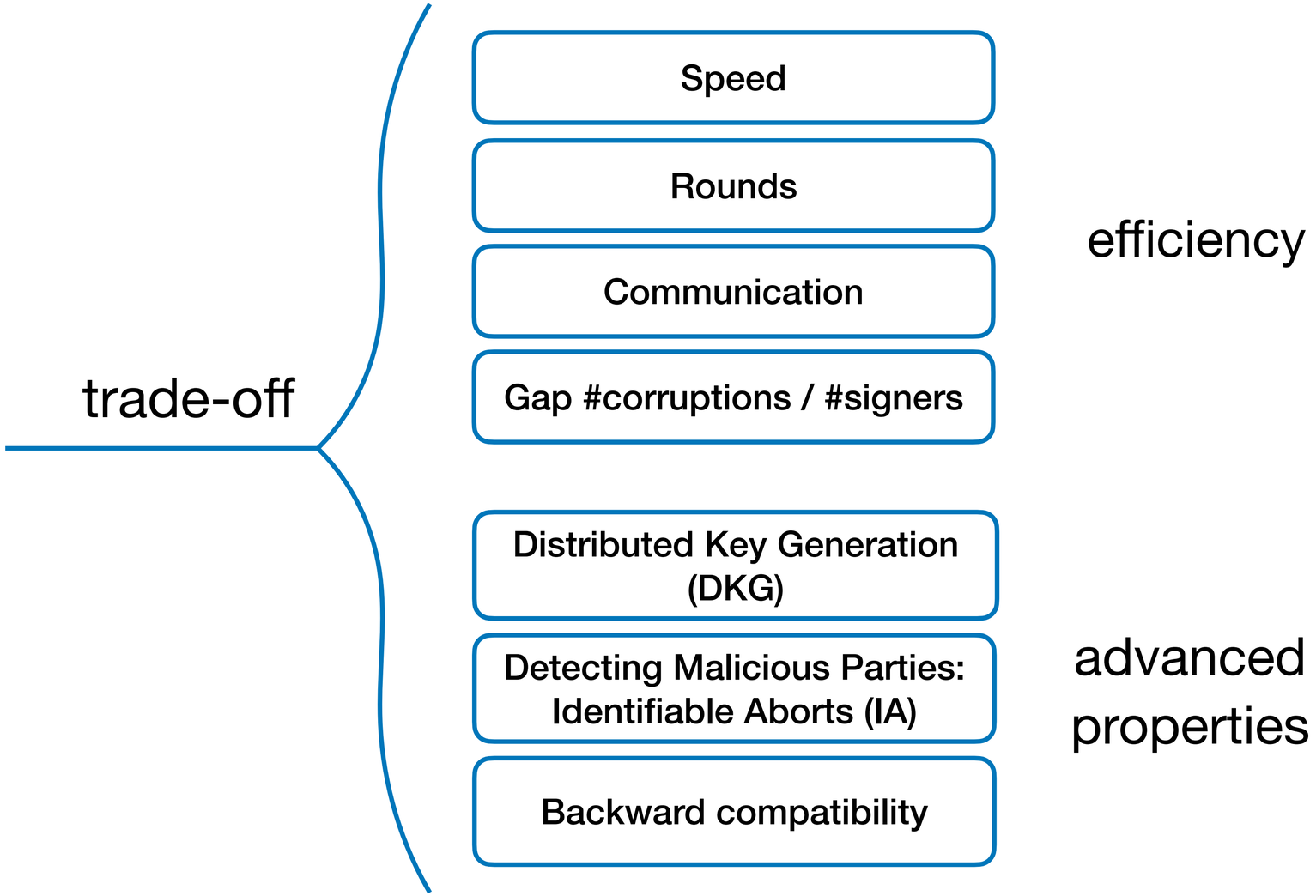
Select a base scheme

- Lattice-based
- Hash-based
- Multivariate-based
- Isogeny-based

The trade-offs of threshold schemes

Select a base scheme

- Lattice-based
- Hash-based
- Multivariate-based
- Isogeny-based



Post-Quantum Threshold Signatures?

Lattice-based

Threshold ML-DSA

Threshold Signatures Reloaded
ML-DSA and Enhanced Raccoon with Identifiable Aborts

Giacomo Borin[1], Sofia Celi[2], Rafael del Pino[3], Thomas Espitau[3], Guilhem Niot[3,4], Thomas Prest[3]

Efficient, Scalable Threshold ML-DSA Signatures: An MPC Approach

Alexander Bienstock*‡, Leo de Castro*†✉, Daniel Escudero*‡, Antigoni Polychroniadou*‡, Akira Takahashi*‡
JPMorgan Chase & Co.
**JPMC AlgoCRYPT CoE, †JPMC CTC Cryptography, ‡JPMC AI Research*
{firstname}.{lastname}@jpmchase.com

\+ Standard

\+ DKG

\- Limited scalability

Post-Quantum Threshold Signatures?

Lattice-based

Threshold ML-DSA

Threshold Signatures Reloaded
ML-DSA and Enhanced Raccoon with Identifiable Aborts

Giacomo Borin[1], Sofia Celi[2], Rafael del Pino[3], Thomas Espitau[3], Guilhem Niot[3,4], Thomas Prest[3]

Efficient, Scalable Threshold ML-DSA Signatures: An MPC Approach

Alexander Bienstock*‡, Leo de Castro*†✉, Daniel Escudero*‡, Antigoni Polychroniadou*‡, Akira Takahashi*‡
JPMorgan Chase & Co.
**JPMC AlgoCRYPT CoE, †JPMC CTC Cryptography, ‡JPMC AI Research*
{firstname}.{lastname}@jpmchase.com

\+ Standard

\+ DKG

\- Limited scalability

FN-DSA based?

Post-Quantum Threshold Signatures?

Lattice-based

Threshold ML-DSA

Threshold Signatures Reloaded
ML-DSA and Enhanced Raccoon with Identifiable Aborts

Giacomo Borin[1], Sofia Celi[2], Rafael del Pino[3], Thomas Espitau[3], Guilhem Niot[3,4], Thomas Prest[3]

Efficient, Scalable Threshold ML-DSA Signatures: An MPC Approach

Alexander Bienstock*‡, Leo de Castro*†✉, Daniel Escudero*‡, Antigoni Polychroniadou*‡, Akira Takahashi*‡
JPMorgan Chase & Co.
**JPMC AlgoCRYPT CoE, †JPMC CTC Cryptography, ‡JPMC AI Research*
{firstname}.{lastname}@jpmchase.com

+ Standard
+ DKG
- Limited scalability

FN-DSA based?

Raccoon based

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]

Simple and Efficient Lattice Threshold Signatures with Identifiable Aborts

Rafael del Pino[1], Thomas Espitau[1], Guilhem Niot[1,2], and Thomas Prest[1]

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

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

+ Efficient and scalable
+ DKG + Abort identification
- Non standard (~10KB sig)

Post-Quantum Threshold Signatures?

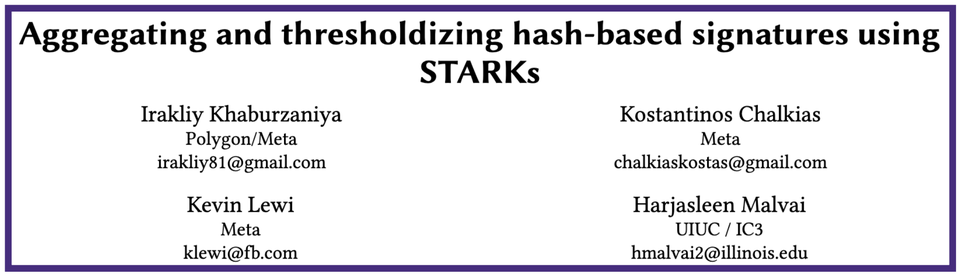
Hash-based

Threshold SPHINCS+?

Post-Quantum Threshold Signatures?

Hash-based

Threshold SPHINCS+?



Aggregating and thresholdizing hash-based signatures using STARKs

Irakliy Khaburzaniya
Polygon/Meta
irakliy81@gmail.com

Kostantinos Chalkias
Meta
chalkiaskostas@gmail.com

Kevin Lewi
Meta
klewi@fb.com

Harjasleen Malvai
UIUC / IC3
hmalvai2@illinois.edu

- Non-standard
- Large >100KB signatures

Post-Quantum Threshold Signatures?

Hash-based

Threshold SPHINCS+?

Aggregating and thresholdizing hash-based signatures using STARKs

Irakliy Khaburzaniya
Polygon/Meta
irakliy81@gmail.com

Kostantinos Chalkias
Meta
chalkiaskostas@gmail.com

Kevin Lewi
Meta
klewi@fb.com

Harjasleen Malvai
UIUC / IC3
hmalvai2@illinois.edu

- Non-standard
- Large >100KB signatures

Turning Hash-Based Signatures into Distributed Signatures and Threshold Signatures

Delegate Your Signing Capability, and Distribute it Among Trustees

John Kelsey[1,2], Nathalie Lang[3] and Stefan Lucks[3]

- Non-standard
- Stateful
- Small number of parties

Post-Quantum Threshold Signatures?

Multivariate

UOV and MAYO

Share the MAYO: thresholdizing MAYO

Sofia Celi[1], Daniel Escudero[2], and Guilhem Niot[3]

+ NIST candidates (UOV and MAYO)

Post-Quantum Threshold Signatures?

Isogeny based

CSI-FiSh based

Threshold Schemes from Isogeny Assumptions

Luca De Feo[1 [0000−0002−9321−0773]] and Michael Meyer[2,3⋆]

+ Efficient
- Non-standard
- Less trusted assumption

Post-Quantum Threshold Signatures?

Focus:

- NIST standard or candidate: **Threshold ML-DSA, UOV, MAYO**
- **Raccoon**: scalable + efficient advanced properties

Threshold MAYO and UOV

MAYO and UOV

UOV

- **Small signatures:** as small as 96 bytes.

MAYO

Parameter set	MAYO_one	MAYO_two	MAYO_three	MAYO_five
security level	1	1	3	5
secret key size	24 B	24 B	32 B	40 B
public key size	1420 B	4912 B	2986 B	5554 B
signature size	454 B	186 B	681 B	964 B

MAYO and UOV

Multivariate Quadratic (MQ) cryptography is based on the assumed hardness of finding a solution to **a system of multivariate quadratic equations** (over a finite field).

The current record $\mathrm{mod}\ 31$ is solving a system of 22 equations in 22 variables.

$$x + 5x^2 + 3xy = 4 \bmod 7$$
$$x^2 + 5xy + 5y^2 = 1 \bmod 7$$

MAYO and UOV

Multivariate Quadratic (MQ) cryptography is based on the assumed hardness of finding a solution to **a system of multivariate quadratic equations** (over a finite field).

The current record $\mathrm{mod}\ 31$ is solving a system of 22 equations in 22 variables.

$$x + 5x^2 + 3xy = 4 \bmod 7$$
$$x^2 + 5xy + 5y^2 = 1 \bmod 7$$

Define a multivariate map $\mathscr{P}$

$$\mathscr{P}(\mathbf{x}) = \mathbf{t}$$

MAYO and UOV

How to design a signature scheme from MQ?

Add some structure to $\mathscr{P}$: define a secret subspace O such that $\mathscr{P}(O) = 0$.

Signing process:

1. Derive a target $\mathbf{t} = H(\text{msg})$. Goal: find $\mathbf{x}$ such that $\mathscr{P}(\mathbf{x}) = \mathbf{t}$.
2. Sample a random vector $\mathbf{v}$.
3. Search for $\mathbf{o} \in O$ such that $\mathscr{P}(\mathbf{v} + \mathbf{o}) = \mathbf{t}$

MAYO and UOV

How to design a signature scheme from MQ?

Add some structure to $\mathcal{P}$: define a secret subspace O such that $\mathcal{P}(O) = 0$.

Signing process:

1. Derive a target $\mathbf{t} = H(\text{msg})$. Goal: find $\mathbf{x}$ such that $\mathcal{P}(\mathbf{x}) = \mathbf{t}$.
2. Sample a random vector $\mathbf{v}$.
3. Search for $\mathbf{o} \in O$ such that $\mathcal{P}(\mathbf{v} + \mathbf{o}) = \mathbf{t}$

$$= \mathcal{P}(\mathbf{v}) + \underbrace{\mathcal{P}(\mathbf{o})}_{=0} + \underbrace{\Delta_{\mathbf{v}}(\mathbf{o})}_{\text{linear}}$$

Threshold MAYO and UOV

High-level signing process

Compute target $\mathbf{t} = H(\mathsf{msg})$

Solve $\mathbf{A} \cdot \mathbf{x} = \mathbf{y}$, for

- $\mathbf{A}$ a randomized function of the secret
- $\mathbf{y}$ a function of $\mathbf{t}$

Threshold MAYO and UOV

High-level signing process

UOV . Sign(sk, msg) → sig

- Compute $\mathbf{t} = H(\text{msg})$
- Sample uniform matrix $\mathbf{V}$
- Derive system $\mathbf{A} = \text{ComputeA}(\text{sk}, \mathbf{V})$
- Derive $\mathbf{y} = \text{ComputeY}(\mathbf{t}, \mathbf{V})$
- Return $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{y}$

Compute target $\mathbf{t} = H(\text{msg})$

Solve $\mathbf{A} \cdot \mathbf{x} = \mathbf{y}$, for

- $\mathbf{A}$ a randomized function of the secret
- $\mathbf{y}$ a function of $\mathbf{t}$

Threshold MAYO and UOV

High-level signing process

UOV . Sign(sk, msg) $\rightarrow$ sig

- Compute $\mathbf{t} = H(\text{msg})$
- Sample uniform matrix $\mathbf{V}$
- Derive system $\mathbf{A} = \text{ComputeA}(\text{sk}, \mathbf{V})$
- Derive $\mathbf{y} = \text{ComputeY}(\mathbf{t}, \mathbf{V})$
- Return $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{y}$

- ComputeA and ComputeY are composed of secret addition and multiplications

Threshold MAYO and UOV

High-level signing process

UOV . Sign(sk, msg) $\rightarrow$ sig

- Compute $\mathbf{t} = H(\text{msg})$
- Sample uniform matrix $\mathbf{V}$
- Derive system $\mathbf{A} = \text{ComputeA}(\text{sk}, \mathbf{V})$
- Derive $\mathbf{y} = \text{ComputeY}(\mathbf{t}, \mathbf{V})$
- Return $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{y}$

- ComputeA and ComputeY are composed of secret addition and multiplications

Addition: share all secret values

$$\mathbf{a} = \mathbf{a}_1 + \ldots + \mathbf{a}_T$$

Threshold MAYO and UOV

High-level signing process

UOV . Sign(sk, msg) → sig

- Compute $\mathbf{t} = H(\text{msg})$
- Sample uniform matrix $\mathbf{V}$
- Derive system $\mathbf{A} = \text{ComputeA}(\text{sk}, \mathbf{V})$
- Derive $\mathbf{y} = \text{ComputeY}(\mathbf{t}, \mathbf{V})$
- Return $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{y}$

- ComputeA and ComputeY are composed of secret addition and multiplications

Addition: share all secret values

$$\mathbf{a} = \mathbf{a}_1 + \ldots + \mathbf{a}_T$$

$$\mathbf{a} + \mathbf{b} = (\mathbf{a}_1 + \mathbf{b}_1) + \ldots + (\mathbf{a}_T + \mathbf{b}_T)$$

Threshold MAYO and UOV

High-level signing process

UOV . Sign(sk, msg) → sig

- Compute $\mathbf{t} = H(\text{msg})$
- Sample uniform matrix $\mathbf{V}$
- Derive system $\mathbf{A} = \text{ComputeA}(\text{sk}, \mathbf{V})$
- Derive $\mathbf{y} = \text{ComputeY}(\mathbf{t}, \mathbf{V})$
- Return $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{y}$

- ComputeA and ComputeY are composed of secret addition and multiplications

Addition: share all secret values

$$\mathbf{a} = \mathbf{a}_1 + \ldots + \mathbf{a}_T$$

$$\mathbf{a} + \mathbf{b} = (\mathbf{a}_1 + \mathbf{b}_1) + \ldots + (\mathbf{a}_T + \mathbf{b}_T)$$

Multiplication

Possible with pre-shared triples (a, b, ab)

Threshold MAYO and UOV

High-level signing process

UOV . Sign(sk, msg) → sig

- Compute $\mathbf{t} = H(\text{msg})$
- Sample uniform matrix $\mathbf{V}$
- Derive system $\mathbf{A} = \text{ComputeA}(\text{sk}, \mathbf{V})$
- Derive $\mathbf{y} = \text{ComputeY}(\mathbf{t}, \mathbf{V})$
- Return $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{y}$

- ComputeA and ComputeY are composed of secret addition and multiplications
- Inversion algorithm can be implemented efficiently by revealing a masked matrix A (multiplied on both sides by random matrices)

Threshold MAYO and UOV

High-level signing process

UOV . Sign(sk, msg) → sig

- Compute $\mathbf{t} = H(\text{msg})$
- Sample uniform matrix $\mathbf{V}$
- Derive system $\mathbf{A} = \text{ComputeA}(\text{sk}, \mathbf{V})$
- Derive $\mathbf{y} = \text{ComputeY}(\mathbf{t}, \mathbf{V})$
- Return $\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{y}$

- ComputeA and ComputeY are composed of secret addition and multiplications

- Inversion algorithm can be implemented efficiently by revealing a masked matrix A (multiplied on both sides by random matrices)

1. Compute and reveal $\mathbf{R} \cdot \mathbf{A} \cdot \mathbf{T}$ and
2. Compute $(\mathbf{R} \cdot \mathbf{A} \cdot \mathbf{T})^{-1}$
3. Compute $\mathbf{x} = \mathbf{T} \cdot (\mathbf{R} \cdot \mathbf{A} \cdot \mathbf{T})^{-1} \cdot \mathbf{R} \cdot \mathbf{y}$

Lattice-based Threshold Signatures

ML-DSA signatures

ML-DSA . Keygen() → sk, vk

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

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

ML-DSA signatures

ML-DSA . Keygen() → sk, vk

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

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

To sign: prove knowledge of sk, $\mathbf{e}$, without revealing sk, $\mathbf{e}$. *(Fiat-Shamir type signature)*

Prover **Challenger**

1 Sample short $\mathbf{r}$

$\mathbf{w} = \mathbf{A} \cdot \mathbf{r}$

$\mathbf{w}$ →

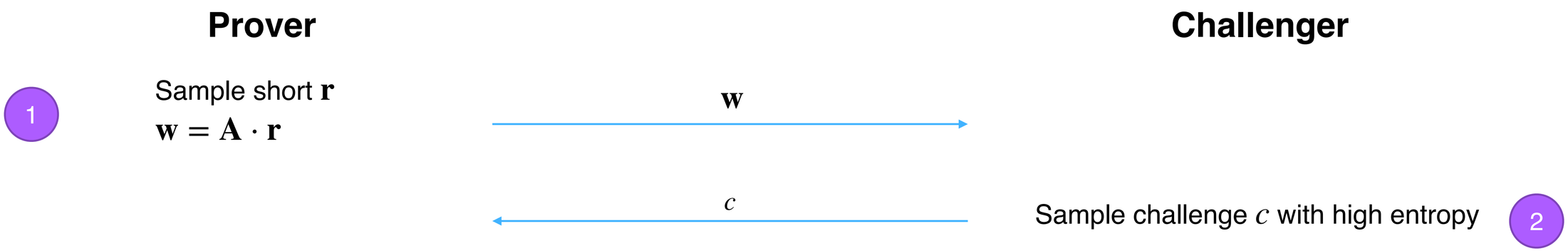
ML-DSA signatures

ML-DSA . Keygen() → sk, vk

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

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

To sign: prove knowledge of sk, $\mathbf{e}$, without revealing sk, $\mathbf{e}$. *(Fiat-Shamir type signature)*



ML-DSA signatures

ML-DSA . Keygen() → sk, vk

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

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

To sign: prove knowledge of sk, $\mathbf{e}$, without revealing sk, $\mathbf{e}$. *(Fiat-Shamir type signature)*

	Prover		**Challenger**
1	Sample short $\mathbf{r}$ $\mathbf{w} = \mathbf{A} \cdot \mathbf{r}$	$\mathbf{w}$ →	
		c ←	Sample challenge c with high entropy 2
3	Compute response $\mathbf{z} = c \cdot \mathrm{sk} + \mathbf{r}$	$\mathbf{z}$ →	Verify $\mathbf{w} - (\mathbf{A} \cdot \mathbf{z} - c \cdot \mathrm{vk})$ is short Verify that $\mathbf{z}$ is short

ML-DSA signatures

ML-DSA . Keygen() → sk, vk

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

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

To sign: prove knowledge of sk, $\mathbf{e}$, without revealing sk, $\mathbf{e}$. *(Fiat-Shamir type signature)*

	Prover		**Challenger**
1	Sample short $\mathbf{r}$ $\mathbf{w} = \mathbf{A} \cdot \mathbf{r}$	$\mathbf{w}$ →	
		c ←	Sample challenge c with high entropy 2
3	Compute response $\mathbf{z} = c \cdot \mathrm{sk} + \mathbf{r}$ If $\mathbf{z} \notin S$, restart	$\mathbf{z}$ →	Verify $\mathbf{w} - (\mathbf{A} \cdot \mathbf{z} - c \cdot \mathrm{vk})$ is short Verify that $\mathbf{z}$ is short

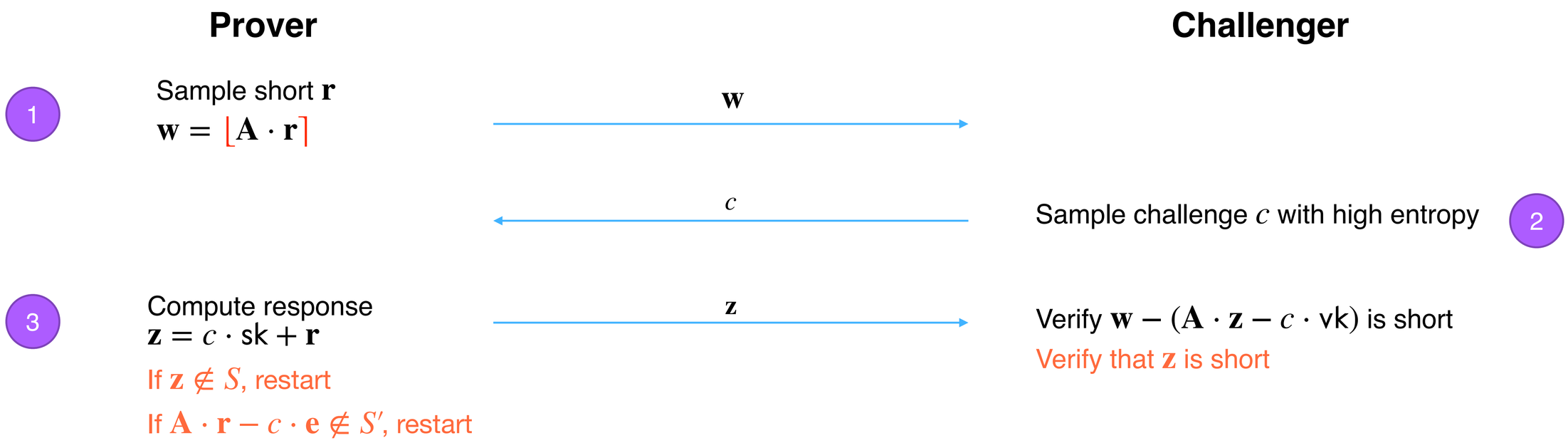
ML-DSA signatures

ML-DSA . Keygen() → sk, vk

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

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

To sign: prove knowledge of sk, $\mathbf{e}$, without revealing sk, $\mathbf{e}$. *(Fiat-Shamir type signature)*



ML-DSA signatures

ML-DSA . Keygen() → sk, vk

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

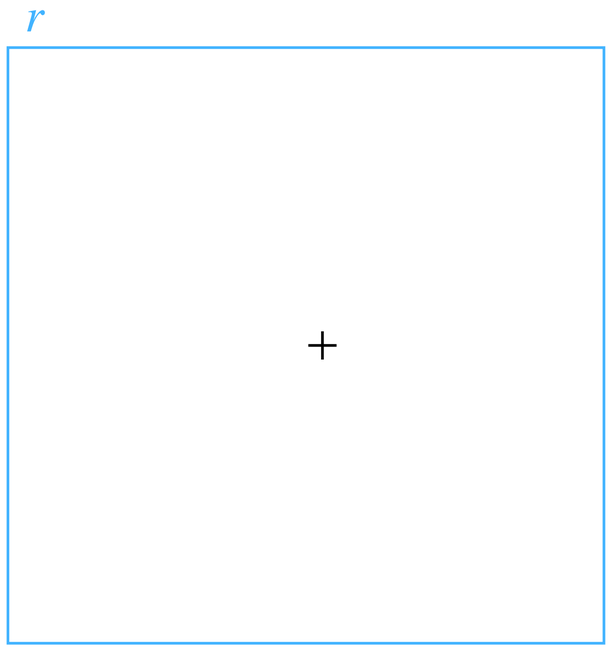
MLWE **assumption:** vk appears uniformly distributed for $\mathbf{A}$ wide enough (more inputs than outputs)

To sign: prove knowledge of sk, $\mathbf{e}$, without revealing sk, $\mathbf{e}$. *(Fiat-Shamir type signature)*

	Prover		**Challenger**
1	Sample short $\mathbf{r}$ $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} \rceil$		
2	$c = H(\mathbf{w}, \mathsf{msg})$		
3	Compute response $\mathbf{z} = c \cdot \mathsf{sk} + \mathbf{r}$ If $\mathbf{z} \notin S$, restart If $\mathbf{A} \cdot \mathbf{r} - c \cdot \mathbf{e} \notin S'$, restart	$\mathbf{z}$ →	Verify $\mathbf{w} - (\mathbf{A} \cdot \mathbf{z} - c \cdot \mathsf{vk})$ is short Verify that $\mathbf{z}$ is short

Rejection sampling

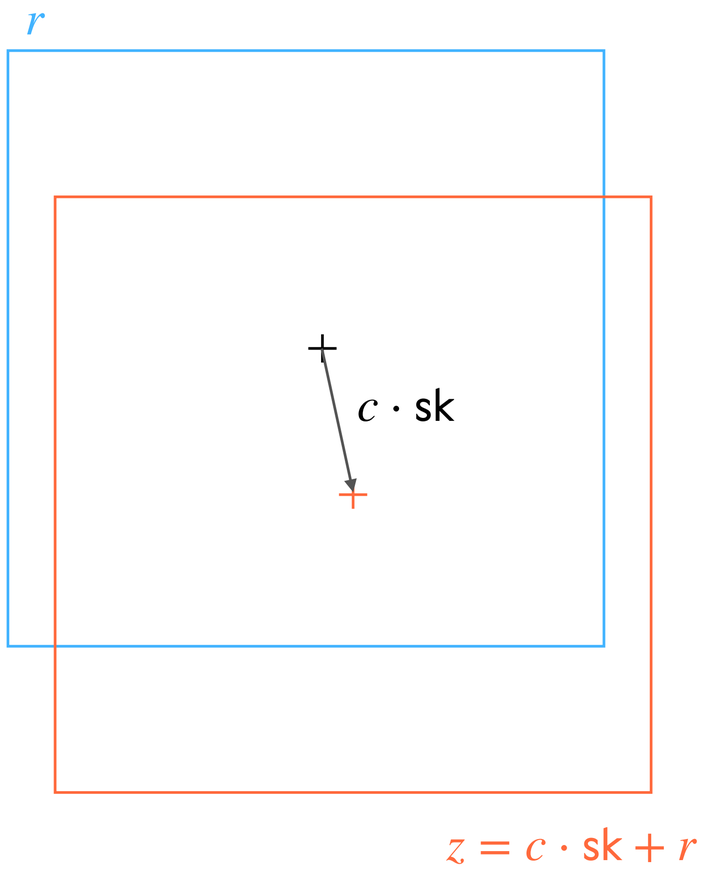
Sample r in a centered hypercube.



Rejection sampling

Sample r in a centered hypercube.

Then, the distribution of z depends on the secret.

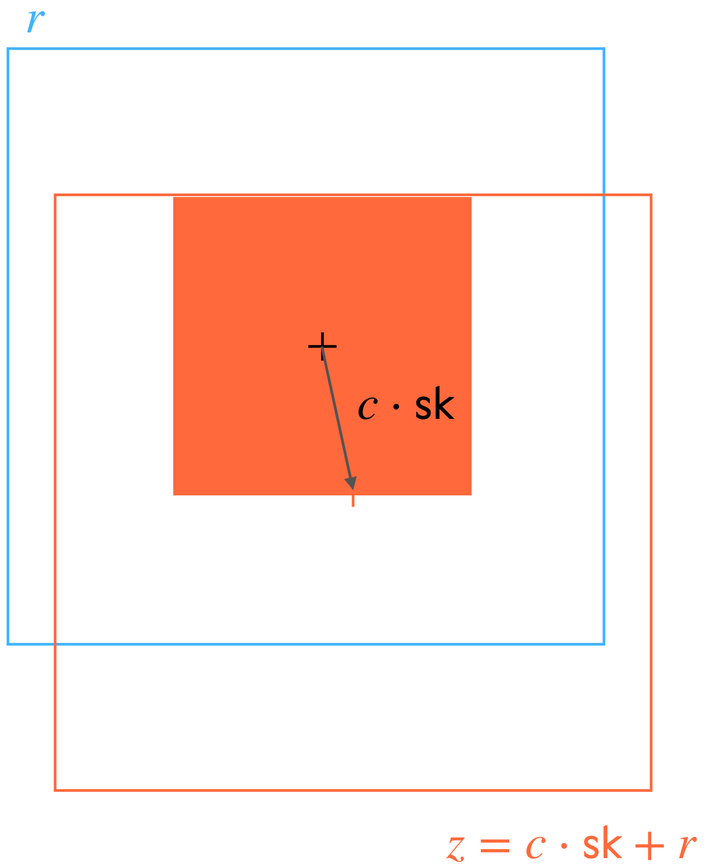


Rejection sampling

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.



ML-DSA signatures

ML-DSA . Keygen() → sk, vk

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

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

ML-DSA . Sign(sk, msg) → sig

- Sample short $\mathbf{r}$
- $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} \rceil$
- $c = H(\mathbf{w}, \text{msg})$
- $\mathbf{z} = c \cdot \text{sk} + \mathbf{r}$
- If $\mathbf{z}$ not in S, **restart**
- If $\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{e}$ not in S', **restart**
- Output $\text{sig} = (\mathbf{z}, \mathbf{w})$

ML-DSA . Verify(vk, msg, sig $= (\mathbf{z}, \mathbf{w})$)

- $c = H(\mathbf{w}, \text{msg})$
- $\mathbf{w} - (\mathbf{A} \cdot \mathbf{z} - c \cdot \text{vk})$ is short
- Assert $\mathbf{z}$ is small

ML-DSA signatures

ML-DSA . Keygen() → sk, vk

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

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

ML-DSA . Sign(sk, msg) → sig

- Sample short $\mathbf{r}$
- $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} \rceil$
- $c = H(\mathbf{w}, \text{msg})$
- $\mathbf{z} = c \cdot \text{sk} + \mathbf{r}$
- If $\mathbf{z}$ not in S, **restart**
- If $\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{e}$ not in S', **restart**
- Output sig $= (\mathbf{z}, \mathbf{w})$

ML-DSA . Verify(vk, msg, sig $= (\mathbf{z}, \mathbf{w})$)

- $c = H(\mathbf{w}, \text{msg})$
- $\mathbf{w} - (\mathbf{A} \cdot \mathbf{z} - c \cdot \text{vk})$ is short
- Assert $\mathbf{z}$ is small

Short vector sampling and rejection sampling are hard to thresholdize

Threshold ML-DSA with generic MPC

ML-DSA . Keygen() → sk, vk

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

ML-DSA . Sign(sk, msg) → sig

- Sample short $\mathbf{r}$
- $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} \rceil$
- $c = H(\mathbf{w}, \mathsf{msg})$
- $\mathbf{z} = c \cdot \mathsf{sk} + \mathbf{r}$
- If $\mathbf{z}$ not in S, **restart**
- If $\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{e}$ not in S', **restart**
- Output $\mathsf{sig} = (\mathbf{z}, \mathbf{w})$

First solution: Implement the signing algorithm with Generic MPC

Efficient, Scalable Threshold ML-DSA Signatures: An MPC Approach

Alexander Bienstock*‡, Leo de Castro*†✉, Daniel Escudero*‡, Antigoni Polychroniadou*‡, Akira Takahashi*‡
JPMorgan Chase & Co.
**JPMC AlgoCRYPT CoE, †JPMC CTC Cryptography, ‡JPMC AI Research*
{firstname}.{lastname}@jpmchase.com

Threshold ML-DSA with generic MPC

ML-DSA . Keygen() → sk, vk

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

ML-DSA . Sign(sk, msg) → sig

- Sample short $\mathbf{r}$ ∗∗∗∗
- $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} + \mathbf{e} \rceil$ ∗∗∗∗
- $c = H(\mathbf{w}, \mathsf{msg})$ 
- $\mathbf{z} = c \cdot \mathsf{sk} + \mathbf{r}$ ∗∗∗∗
- If $\mathbf{z}$ not in S, **restart** ∗∗∗∗
- If $\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{e}$ not in S', **restart** ∗∗∗∗
- Output $\mathsf{sig} = (\mathbf{z}, \mathbf{w})$

First solution: Implement the signing algorithm with Generic MPC

Efficient, Scalable Threshold ML-DSA Signatures: An MPC Approach

Alexander Bienstock*‡, Leo de Castro*†✉, Daniel Escudero*‡, Antigoni Polychroniadou*‡, Akira Takahashi*‡
JPMorgan Chase & Co.
**JPMC AlgoCRYPT CoE, †JPMC CTC Cryptography, ‡JPMC AI Research*
{firstname}.{lastname}@jpmchase.com

- Small modification in ML-DSA to safely reveal $\mathbf{w}$ and compute c in clear, relies on [dPN25]

Threshold ML-DSA with generic MPC

ML-DSA . Keygen() → sk, vk

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

ML-DSA . Sign(sk, msg) → sig

- Sample short $\mathbf{r}$ ∗∗∗∗
- $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} + \mathbf{e} \rceil$ ∗∗∗∗
- $c = H(\mathbf{w}, \mathsf{msg})$ 
- $\mathbf{z} = c \cdot \mathsf{sk} + \mathbf{r}$ ∗∗∗∗
- If $\mathbf{z}$ not in S, **restart** ∗∗∗∗
- If $\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{e}$ not in S', **restart** ∗∗∗∗
- Output sig $= (\mathbf{z}, \mathbf{w})$

First solution: Implement the signing algorithm with Generic MPC

Efficient, Scalable Threshold ML-DSA Signatures: An MPC Approach

Alexander Bienstock*‡, Leo de Castro*†✉, Daniel Escudero*‡, Antigoni Polychroniadou*‡, Akira Takahashi*‡
JPMorgan Chase & Co.
**JPMC AlgoCRYPT CoE, †JPMC CTC Cryptography, ‡JPMC AI Research*
{firstname}.{lastname}@jpmchase.com

- Small modification in ML-DSA to safely reveal $\mathbf{w}$ and compute c in clear, relies on [dPN25]
- Batch comparisons for rejection sampling

Threshold ML-DSA with generic MPC

ML-DSA . Keygen() → sk, vk

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

ML-DSA . Sign(sk, msg) → sig

- Sample short $\mathbf{r}$ ∗∗∗∗
- $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} + \mathbf{e} \rceil$ ∗∗∗∗
- $c = H(\mathbf{w}, \mathsf{msg})$ 
- $\mathbf{z} = c \cdot \mathsf{sk} + \mathbf{r}$ ∗∗∗∗
- If $\mathbf{z}$ not in S, **restart** ∗∗∗∗
- If $\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{e}$ not in S', **restart** ∗∗∗∗
- Output sig $= (\mathbf{z}, \mathbf{w})$

First solution: Implement the signing algorithm with Generic MPC

Efficient, Scalable Threshold ML-DSA Signatures: An MPC Approach

Alexander Bienstock*‡, Leo de Castro*†✉, Daniel Escudero*‡, Antigoni Polychroniadou*‡, Akira Takahashi*‡
JPMorgan Chase & Co.
**JPMC AlgoCRYPT CoE, †JPMC CTC Cryptography, ‡JPMC AI Research*
{firstname}.{lastname}@jpmchase.com

Concretely,

- Online:
  - 92 rounds w/ 1.2 MB comm
  - 24 rounds w/ 2.3 MB comm
- **Honest-majority** for better efficiency (can only tolerate $T/2$ corruptions for T signers)
- Offline: ?

Threshold ML-DSA for N parties ($T = N$)

ML-DSA* . Keygen() → sk, vk

- For $1 \le i \le N$, $\mathsf{vk}_i = \mathbf{A} \cdot \mathsf{sk}_i + \mathbf{e_i}$, where sk, $\mathbf{e}_i$ short
- $\mathsf{vk} = \sum_i \mathsf{vk}_i$

Second solution: More tailored / using lattice properties
Sample N secrets, and aggregate the knowledge proofs.

ML-DSA . Verify(vk, msg, sig $= (\mathbf{z}, \lfloor \mathbf{w} \rceil)$)

- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- $\lfloor \mathbf{w} \rceil - (\mathbf{A} \cdot \mathbf{z} - c \cdot \mathsf{vk})$ is short
- Assert $\mathbf{z}$ is small

Threshold ML-DSA for N parties ($T = N$)

ML-DSA* . Keygen() → sk, vk

- For $1 \le i \le N$, $\mathsf{vk}_i = \mathbf{A} \cdot \mathsf{sk}_i + \mathbf{e_i}$, where sk, $\mathbf{e}_i$ short
- $\mathsf{vk} = \sum_i \mathsf{vk}_i$

ML-DSA* . Sign(sk, msg) → sig

- For $1 \le i \le N$
  - Sample short $\mathbf{r}_i, \mathbf{e}'_i$
  - $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- $\mathbf{w} = \sum_i \mathbf{w}_i$

Second solution: More tailored / using lattice properties
Sample N secrets, and aggregate the knowledge proofs.

ML-DSA . Verify(vk, msg, sig $= (\mathbf{z}, \lfloor \mathbf{w} \rceil)$)

- $c = H(\lfloor \mathbf{w} \rceil, \mathrm{msg})$
- $\lfloor \mathbf{w} \rceil - (\mathbf{A} \cdot \mathbf{z} - c \cdot \mathsf{vk})$ is short
- Assert $\mathbf{z}$ is small

Sample a $\mathbf{w}_i$ for each secret, and do not rely on rounding for security:
reintroduce error in $\mathbf{w}_i$ for rejection sampling on $\mathbf{e}$

Threshold ML-DSA for N parties ($T = N$)

ML-DSA* . Keygen() → sk, vk

- For $1 \le i \le N$, $\mathsf{vk}_i = \mathbf{A} \cdot \mathsf{sk}_i + \mathbf{e_i}$, where sk, $\mathbf{e}_i$ short
- $\mathsf{vk} = \sum_i \mathsf{vk}_i$

ML-DSA* . Sign(sk, msg) → sig

- For $1 \le i \le N$
  - Sample short $\mathbf{r}_i, \mathbf{e}'_i$
  - $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- For $1 \le i \le N$,
  $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i = c \cdot \mathbf{e}_i + \mathbf{e}'_i$
- If any $(\mathbf{z}_i, \mathbf{y}_i)$ not in S, **restart**

Second solution: More tailored / using lattice properties
Sample N secrets, and aggregate the knowledge proofs.

ML-DSA . Verify(vk, msg, sig $= (\mathbf{z}, \lfloor \mathbf{w} \rceil)$)

- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- $\lfloor \mathbf{w} \rceil - (\mathbf{A} \cdot \mathbf{z} - c \cdot \mathsf{vk})$ is short
- Assert $\mathbf{z}$ is small

Sample a $\mathbf{w}_i$ for each secret, and do not rely on rounding for security:
reintroduce error in $\mathbf{w}_i$ for rejection sampling on $\mathbf{e}$

Threshold ML-DSA for N parties ($T = N$)

ML-DSA* . Keygen() → sk, vk

- For $1 \le i \le N$, $\mathsf{vk}_i = \mathbf{A} \cdot \mathsf{sk}_i + \mathbf{e_i}$, where sk, $\mathbf{e}_i$ short
- $\mathsf{vk} = \sum_i \mathsf{vk}_i$

ML-DSA* . Sign(sk, msg) → sig

- For $1 \le i \le N$
  - Sample short $\mathbf{r}_i, \mathbf{e}'_i$
  - $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- For $1 \le i \le N$,
  $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i = c \cdot \mathbf{e}_i + \mathbf{e}'_i$
- If any $(\mathbf{z}_i, \mathbf{y}_i)$ not in S, **restart**
- $\mathsf{sig} = (\sum_i \mathbf{z}_i, \lfloor \mathbf{w} \rceil)$
- If sig not in S', **restart**
- **return** sig

Second solution: More tailored / using lattice properties
Sample N secrets, and aggregate the knowledge proofs.

ML-DSA . Verify(vk, msg, sig $= (\mathbf{z}, \lfloor \mathbf{w} \rceil)$)

- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- $\lfloor \mathbf{w} \rceil - (\mathbf{A} \cdot \mathbf{z} - c \cdot \mathsf{vk})$ is short
- Assert $\mathbf{z}$ is small

Sample a $\mathbf{w}_i$ for each secret, and do not rely on rounding for security:
reintroduce error in $\mathbf{w}_i$ for rejection sampling on $\mathbf{e}$

Threshold ML-DSA for N parties ($T = N$)

ML-DSA* . Keygen() → sk, vk

- For $1 \le i \le N$, $\mathsf{vk}_i = \mathbf{A} \cdot \mathsf{sk}_i + \mathbf{e_i}$, where sk, $\mathbf{e}_i$ short
- $\mathsf{vk} = \sum_i \mathsf{vk}_i$

ML-DSA* . Sign(sk, msg) → sig

- For $1 \le i \le N$
  - Sample short $\mathbf{r}_i, \mathbf{e}'_i$
  - $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- For $1 \le i \le N$,
  $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i = c \cdot \mathbf{e}_i + \mathbf{e}'_i$
- If any $(\mathbf{z}_i, \mathbf{y}_i)$ not in S, **restart**
- $\mathsf{sig} = (\sum_i \mathbf{z}_i, \lfloor \mathbf{w} \rceil)$
- If sig not in S', **restart**
- **return** sig

Second solution: More tailored / using lattice properties
Sample N secrets, and aggregate the knowledge proofs.

ML-DSA . Verify(vk, msg, sig $= (\mathbf{z}, \lfloor \mathbf{w} \rceil)$)

- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- $\lfloor \mathbf{w} \rceil - (\mathbf{A} \cdot \mathbf{z} - c \cdot \mathsf{vk})$ is short
- Assert $\mathbf{z}$ is small

We use more compact distributions than ML-DSA to still pass verification
→ supports up to 6 parties

Threshold ML-DSA for N parties ($T = N$)

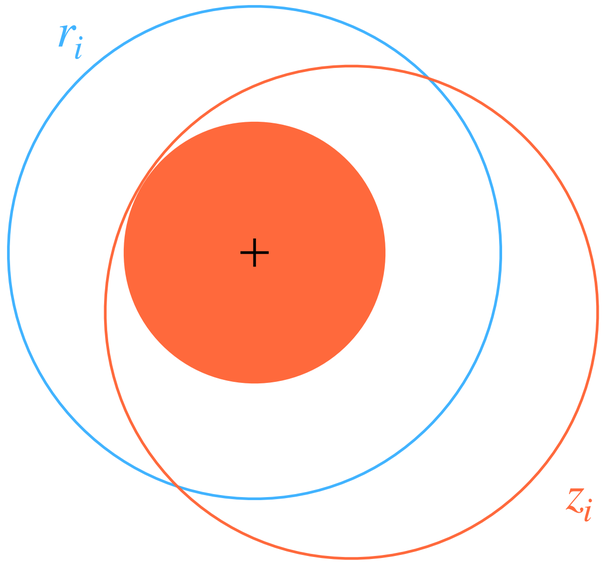
ML-DSA* . Keygen() →

- For $1 \le i \le N$, vk
- $\mathsf{vk} = \sum_i \mathsf{vk}_i$

ML-DSA* . Sign(sk, msg

- For $1 \le i \le N$
  - Sample short
  - $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i +$
- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\lfloor \mathbf{w} \rceil, \mathrm{ms}$
- For $1 \le i \le N$,
  $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i =$
- If any $(\mathbf{z}_i, \mathbf{y}_i)$ not in S, **restart**
- $\mathsf{sig} = (\sum_i \mathbf{z}_i, \lfloor \mathbf{w} \rceil)$
- If sig not in S', **restart**
- **return** sig

Rejection sampling with hyperballs



g lattice properties
e knowledge proofs.

We use more compact distributions than ML-DSA to still pass verification → supports up to 6 parties

Threshold ML-DSA for N parties ($T = N$)

ML-DSA* . Keygen() → sk, vk

- For $1 \le i \le N$, $\mathsf{vk}_i = \mathbf{A} \cdot \mathsf{sk}_i + \mathbf{e_i}$, where sk, $\mathbf{e}_i$ short
- $\mathsf{vk} = \sum_i \mathsf{vk}_i$

ML-DSA* . Sign(sk, msg) → sig

- For $1 \le i \le N$
  - Sample short $\mathbf{r}_i, \mathbf{e}'_i$
  - $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- For $1 \le i \le N$,
  $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i = c \cdot \mathbf{e}_i + \mathbf{e}'_i$
- If any $(\mathbf{z}_i, \mathbf{y}_i)$ not in S, **restart**
- $\mathsf{sig} = (\sum_i \mathbf{z}_i, \lfloor \mathbf{w} \rceil)$
- If sig not in S', **restart**
- **return** sig

Th-ML-DSA . Sign(sk, msg) → sig

Round 1:

- Sample short $\mathbf{r}_i, \mathbf{e}'_i$
- Broadcast $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$

Round 2:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i = c \cdot \mathsf{e}_i + \mathbf{e}'_i$
- 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

Threshold ML-DSA for N parties ($T = N$)

ML-DSA* . Keygen() → sk, vk

- For $1 \le i \le N$, $\mathsf{vk}_i = \mathbf{A} \cdot \mathsf{sk}_i + \mathbf{e_i}$, where $\mathsf{sk}, \mathbf{e}_i$ short
- $\mathsf{vk} = \sum_i \mathsf{vk}_i$

ML-DSA* . Sign(sk, msg) → sig

- For $1 \le i \le N$
  - Sample short $\mathbf{r}_i, \mathbf{e}'_i$
  - $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- For $1 \le i \le N$,
  $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i = c \cdot \mathbf{e}_i + \mathbf{e}'_i$
- If any $(\mathbf{z}_i, \mathbf{y}_i)$ not in S, **restart**
- $\mathsf{sig} = (\sum_i \mathbf{z}_i, \lfloor \mathbf{w} \rceil)$
- If sig not in S', **restart**
- **return** sig

But, the scheme is only secure if corrupted parties do not bias $\mathbf{w}$

Th-ML-DSA . Sign(sk, msg) → sig

Round 1:

- Sample short $\mathbf{r}_i, \mathbf{e}'_i$
- Broadcast $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$

Round 2:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i = c \cdot \mathsf{e}_i + \mathbf{e}'_i$
- 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

Threshold ML-DSA for N parties ($T = N$)

ML-DSA* . Keygen() → sk, vk

- For $1 \le i \le N$, $\mathsf{vk}_i = \mathbf{A} \cdot \mathsf{sk}_i + \mathbf{e_i}$, where sk, $\mathbf{e}_i$ short
- $\mathsf{vk} = \sum_i \mathsf{vk}_i$

ML-DSA* . Sign(sk, msg) → sig

- For $1 \le i \le N$
  - Sample short $\mathbf{r}_i, \mathbf{e}'_i$
  - $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- For $1 \le i \le N$,
  $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i = c \cdot \mathbf{e}_i + \mathbf{e}'_i$
- If any $(\mathbf{z}_i, \mathbf{y}_i)$ not in S, **restart**
- $\mathsf{sig} = (\sum_i \mathbf{z}_i, \lfloor \mathbf{w} \rceil)$
- If sig not in S', **restart**
- **return** sig

Th-ML-DSA . Sign(sk, msg) → 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
- $c = H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$
- $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i, \mathbf{y}_i = c \cdot \mathsf{e}_i + \mathbf{e}'_i$
- 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

Threshold ML-DSA for $T \neq N$ parties

Use Replicated Secret Sharing [dPN25]

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

1. When at most $T - 1$ parties are corrupted, at least one of these secrets remains hidden.

Threshold ML-DSA for $T \neq N$ parties

Use Replicated Secret Sharing [dPN25]

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

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

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

Threshold ML-DSA for $T \neq N$ parties

Use Replicated Secret Sharing [dPN25]

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

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

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

Th-ML-DSA . Sign(sk, msg) → 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
- $c = H(\lfloor \mathbf{w} \rceil, \mathrm{msg})$
- $\mathbf{z}_i = c \cdot \sum_{I \in m_i} \mathsf{sk}_I + \mathbf{r}_i, \mathbf{y}_i = c \cdot \sum_{I \in m_i} \mathsf{e}_I + \mathbf{e}'_i$
- 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

Techniques from [dPN25].

Threshold ML-DSA for $T \neq N$ parties

Use Replicated Secret Sharing [dPN25]

ML-DSA* . Keygen() → sk, vk

- For every possible set I of $N - T + 1$ pa
  - $\mathsf{vk}_I = \mathbf{A} \cdot \mathsf{sk}_I + \mathbf{e}_I$, where $\mathsf{sk}_I, \mathbf{e}_I$ sho
  - 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
at least one of these secrets remai

2. T parties can collaboratively reconstruct the full secret.

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

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

… plus some other optimizations to make parameters as tight as possible

Th-ML-DSA . Sign(sk, msg) → sig

Round 1:

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

:

cast $\mathbf{w}_i$

:

$\sum_i \mathbf{w}_i$ + abort if inconsistent commit_i

$H(\lfloor \mathbf{w} \rceil, \mathsf{msg})$

$c \cdot \sum_{I \in m_i} \mathsf{sk}_I + \mathbf{r}_i, \mathbf{y}_i = c \cdot \sum_{I \in m_i} \mathbf{e}_I + \mathbf{e}'_i$

- If $(\mathbf{z}_i, \mathbf{y}_i)$ in S, **broadcast $\mathbf{z}_i$, else abort**

Combine:

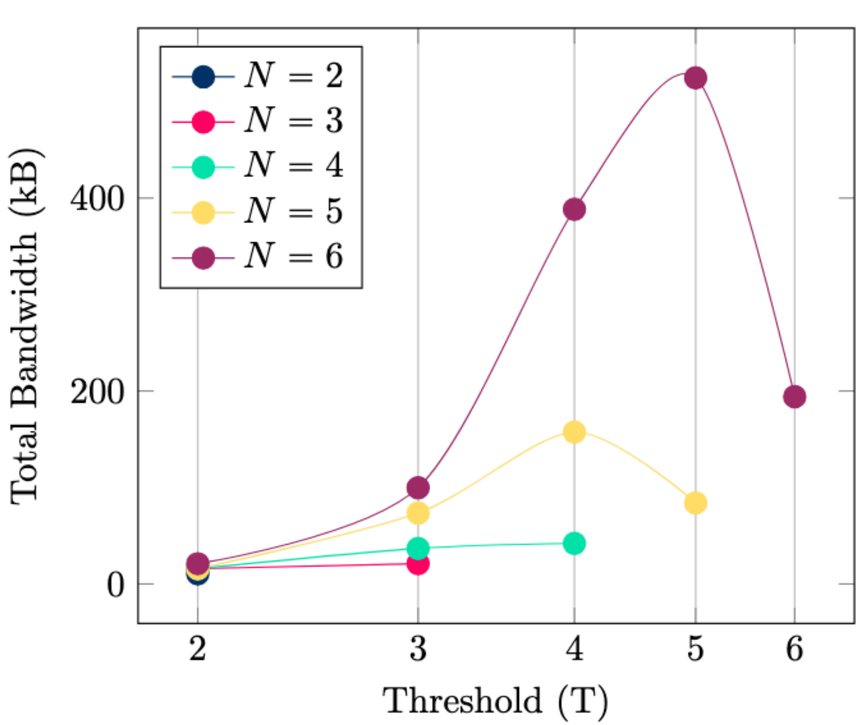
- $\mathsf{sig} = (\sum_i \mathbf{z}_i, \lfloor \mathbf{w} \rceil)$
- If sig not in S', **restart**
- **return** sig

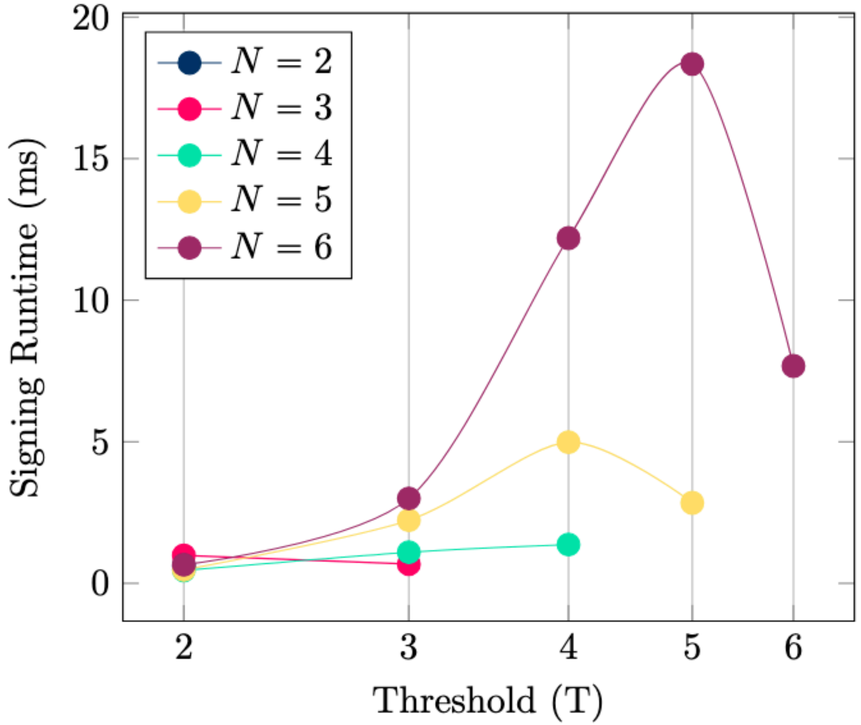
Techniques from [dPN25].

Evaluation

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)

Threshold ML-DSA

Scheme	# Parties	# Rounds	Comm (MB)	Computation	Paradigm	Security
Our work	6	6	0.021 to 1.05	Lightweight	Game-based	Standard
Bienstock et al.	Unlimited	96	>1.2*	Online lightweight*	UC	Honest Majority
		24	>2.3*			

Average # rounds, and communication per party to obtain a valid signature

** Communication and computation exclude cost of offline correlated randomness generation.*

Threshold ML-DSA

Scheme	# Parties	# Rounds	Comm (MB)	Computation	Paradigm	Security
Our work	6	6	0.021 to 1.05	Lightweight	Game-based	Standard
Bienstock et al.	Unlimited	96	>1.2*	Online lightweight*	UC	Honest Majority
		24	>2.3*			

Advanced properties?

DKG ✓

Threshold ML-DSA

Scheme	# Parties	# Rounds	Comm (MB)	Computation	Paradigm	Security
Our work	6	6	0.021 to 1.05	Lightweight	Game-based	Standard
Bienstock et al.	Unlimited	96	>1.2*	Online lightweight*	UC	Honest Majority
		24	>2.3*			

Advanced properties?

DKG ✓

Identifiable Aborts ✘

Single online round ✘

Efficient for many parties ✘

Raccoon: ML-DSA without aborts

Raccoon . Keygen() → sk, vk

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

Raccoon . Sign(sk, msg) → sig

- Sample a short $\mathbf{r}, \mathbf{e}'$
- $\mathbf{w} = \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$
- $c = H(\mathbf{w}, \text{msg})$
- $\mathbf{z} = c \cdot \text{sk} + \mathbf{r}$
- ~~If $\mathbf{z}$ not in S, **restart**~~
- Output sig $= (\mathbf{w}, \mathbf{z})$

Let's remove the rejection sampling!

Raccoon: ML-DSA without aborts

Raccoon . Keygen() → sk, vk

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

Raccoon . Sign(sk, msg) → sig

- Sample a short $\mathbf{r}, \mathbf{e}'$
- $\mathbf{w} = \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$
- $c = H(\mathbf{w}, \text{msg})$
- $\mathbf{z} = c \cdot \text{sk} + \mathbf{r}$
- ~~If $\mathbf{z}$ not in S,~~ **~~restart~~**
- Output sig $= (\mathbf{w}, \mathbf{z})$

Unforgeable under

- **Hint-MLWE**
- **SelfTargetMSIS**

| vk |	| sig |
2.3 kB	11.5 kB

Hint-MLWE assumption [KLSS23].

$(\mathbf{A}, \text{vk})$ is pseudorandom even if given Q "hints":

$$(c_i, \mathbf{z}_i := c_i \cdot \text{sk} + \mathbf{r}_i) \text{ for } i \in [Q]$$

As hard as MLWE_σ if

$$\sigma_{\mathbf{r}} \geq \sqrt{Q} \cdot \|c\| \cdot \sigma$$

Threshold Raccoon

Raccoon . Keygen() → sk, vk

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

Raccoon . Sign(sk, msg) → sig

- Sample a short $\mathbf{r}, \mathbf{e}'$
- $\mathbf{w} = \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$
- $c = H(\mathbf{w}, \mathsf{msg})$
- $\mathbf{z} = c \cdot \mathsf{sk} + \mathbf{r}$
- Output $\mathsf{sig} = (\mathbf{w}, \mathbf{z})$

Raccoon . Verify(vk, msg, sig $= (\mathbf{w}, \mathbf{z})$)

- $c = H(\mathbf{w}, \mathsf{msg})$
- $\mathbf{y} = \mathbf{w} - \mathbf{A} \cdot \mathbf{z} + c \cdot \mathsf{vk}$
- Assert $(\mathbf{y}, \mathbf{z})$ short

Shamir sharing on secret $\mathsf{sk} \in \mathscr{R}_q^\ell$

Sample polynomial $f \in \mathscr{R}_q^\ell[X]$ s.t.

- $f(0) = \mathsf{sk}$ and $\deg f \leq T - 1$
- Partial signing keys $\mathsf{sk}_i := [\![\mathsf{sk}]\!]_i = f(i)$

Properties:

- with $< T$ shares, sk is perfectly hidden
- with a set S of $\geq T$ shares, reconstruct sk via Lagrange interpolation

$$\mathsf{sk} = \sum_{i \in S} L_{S,i} \cdot [\![\mathsf{sk}]\!]_i$$

Threshold Raccoon

Raccoon . Keygen() → sk, vk

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

Raccoon . Sign(sk, msg) → sig

- Sample a short $\mathbf{r}, \mathbf{e}'$
- $\mathbf{w} = \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$
- $c = H(\mathbf{w}, \text{msg})$
- $\mathbf{z} = c \cdot \text{sk} + \mathbf{r}$
- Output sig $= (\mathbf{w}, \mathbf{z})$

Raccoon . Verify(vk, msg, sig $= (\mathbf{w}, \mathbf{z})$)

- $c = H(\mathbf{w}, \text{msg})$
- $\mathbf{y} = \mathbf{w} - \mathbf{A} \cdot \mathbf{z} + c \cdot \text{vk}$
- Assert $(\mathbf{y}, \mathbf{z})$ short

First (insecure) attempt

ThRaccoon . Sign(sk, msg) → sig

Round 1:

- Sample a short $\mathbf{r}_i, \mathbf{e}'_i$
- $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- Broadcast $\text{cmt}_i = H_{\text{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\mathbf{w}, \text{msg})$
- Broadcast $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![\text{s}k]\!]_i + \mathbf{r}_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

Threshold Raccoon

- Prevent ROS attack with commit-reveal of $\mathbf{w}_i$

First (insecure) attempt

ThRaccoon . Sign(sk, msg) → sig

Round 1:

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

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\mathbf{w}, \mathsf{msg})$
- Broadcast $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![\mathsf{s}k]\!]_i + \mathbf{r}_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

Threshold Raccoon

- Prevent ROS attack with commit-reveal of $\mathbf{w}_i$
- But, $\mathbf{r}_i$ is small vs $L_{S,i} \cdot c \cdot [\![sk]\!]_i$ is large

  $\rightarrow$ Leaks $[\![sk]\!]_i$

First (insecure) attempt

ThRaccoon . Sign(sk, msg) $\rightarrow$ sig

Round 1:

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

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\mathbf{w}, \mathsf{msg})$
- Broadcast $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![sk]\!]_i + \mathbf{r}_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

Threshold Raccoon

- Prevent ROS attack with commit-reveal of $\mathbf{w}_i$
- But, $\mathbf{r}_i$ is small vs $L_{S,i} \cdot c \cdot [\![sk]\!]_i$ is large

  $\rightarrow$ Leaks $[\![sk]\!]_i$

- Solution: add a zero-share Δ_i:
  - Derived with a PRF, using pre-shared pairwise keys
  - Any set of $< T$ values Δ_i is uniformly random
  - $\sum_{i \in S} \Delta_i = 0$

ThRaccoon . Sign(sk, msg) $\rightarrow$ sig

Round 1:

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

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\mathbf{w}, \mathsf{msg})$
- Broadcast $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![sk]\!]_i + \mathbf{r}_i + \Delta_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

Threshold Raccoon

max N	Speed	Rounds	| vk |	| sig |	Total communication
1024	Fast	3	4 kB	13 kB	40 kB

Two-round ThRaccoon [EKT24]

2Rnd-ThRaccoon . Sign(sk, msg) → sig

Round 1:

- Sample short $\mathbf{r}_{i,j}, \mathbf{e}'_{i,j}$ for $j \in [\text{rep}]$
- $\mathbf{w}_{i,j} = \mathbf{A} \cdot \mathbf{r}_{i,j} + \mathbf{e}'_{i,j}$ for $j \in [\text{rep}]$
- Broadcast $(\mathbf{w}_{i,j})_j$

Round 2:

- Derive coefficients $(\beta_{i,j})_{i,j} = H((\mathbf{w}_{i,j})_{i,j})$
- $\mathbf{w} = \sum_{i,j} \beta_{i,j} \mathbf{w}_{i,j}$
- $c = H(\mathbf{w}, \text{msg})$
- Broadcast $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![\text{s}k]\!]_i + \sum_j \beta_{i,j} \mathbf{r}_{i,j} + \Delta_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

ThRaccoon . Sign(sk, msg) → sig

Round 1:

- Sample a short $\mathbf{r}_i, \mathbf{e}'_i$
- $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$
- Broadcast $\text{cmt}_i = H_{\text{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\mathbf{w}, \text{msg})$
- Broadcast $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![\text{s}k]\!]_i + \mathbf{r}_i + \Delta_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

Two-round ThRaccoon [EKT24]

2Rnd-ThRaccoon . Sign(sk, msg) → sig

Round 1:

- Sample short $\mathbf{r}_{i,j}, \mathbf{e}'_{i,j}$ for $j \in [\text{rep}]$
- $\mathbf{w}_{i,j} = \mathbf{A} \cdot \mathbf{r}_{i,j} + \mathbf{e}'_{i,j}$ for $j \in [\text{rep}]$
- Broadcast $(\mathbf{w}_{i,j})_j$

Round 2:

- Derive coefficients $(\beta_{i,j})_{i,j} = H((\mathbf{w}_{i,j})_{i,j})$
- $\mathbf{w} = \sum_{i,j} \beta_{i,j} \mathbf{w}_{i,j}$
- $c = H(\mathbf{w}, \text{msg})$
- Broadcast $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![\text{sk}]\!]_i + \sum_j \beta_{i,j} \mathbf{r}_{i,j} + \Delta_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

Unforgeable under

- **AOMLWE**
- **SelfTargetMSIS**

| vk |	| sig |
5.5 kB	10.8 kB

~270 kB communication (offline + online): 5x TRaccoon

AOMLWE assumption.

$(\mathbf{A}, \text{vk})$ is pseudorandom even if given many $(\mathbf{w}_j)$, and adaptively choosing coefficients β_j to obtain:

$$(c_i, \mathbf{z}_i := c_i \cdot \text{sk} + \sum_j \beta_j \cdot \mathbf{r}_j) \text{ for } i \in [Q]$$

Detecting signing failures origins

Identify aborts in ThRaccoon with NIZK

- We can identify aborts using NIZK.
- Hard to prove to prove correct computation of Δ_i, but we can prove all other computations with the NIZK
- At a high-level, $\Delta_i = \sum_j m_{i,j}$ where $m_{i,j}$ is the output of a PRF known by i and j.
  - We can simply check that all i, j agree on $m_{i,j}$, if not one cheated and we can reveal the corresponding PRF key to check computations.

NIZK-ThRaccoon . Sign(sk, msg) → sig

Round 1:

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

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\mathbf{w}, \mathsf{msg})$
- Broadcast $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![\mathsf{s}k]\!]_i + \mathbf{r}_i + \Delta_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

DKG + Detecting signing failures origins

Identify aborts in ThRaccoon with sDKG

- We can also leverage a new class of secret sharings: “short secret sharing”
  - Secret sharing with shares of small norms: partial leak, but ok for lattices
  - In this case, we can remove the zero-share Δ_i!

sDKG-ThRaccoon . Sign(sk, msg) → sig

Round 1:

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

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\mathbf{w}, \mathsf{msg})$
- Broadcast $\mathbf{z}_i = c \cdot \langle L_{S,i} \cdot \mathsf{sk}_i \rangle + \mathbf{r}_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

DKG + Detecting signing failures origins

Identify aborts in ThRaccoon with sDKG

- We can also leverage a new class of secret sharings: “short secret sharing”
  - Secret sharing with shares of small norms: partial leak, but ok for lattices
  - In this case, we can remove the zero-share Δ_i!
- Identifiable abort for free, but larger private key
- Scales up to 64 parties

sDKG-ThRaccoon . Sign(sk, msg) → sig

Round 1:

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

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

- $\mathbf{w} = \sum_i \mathbf{w}_i$
- $c = H(\mathbf{w}, \mathsf{msg})$
- Broadcast $\mathbf{z}_i = c \cdot \langle L_{S,i} \cdot \mathsf{sk}_i \rangle + \mathbf{r}_i$

Combine: the final signature is

$$(\mathbf{w}, \sum_{i \in S} \mathbf{z}_i)$$

Conclusion

Conclusion

- Diverse proposal for Threshold Signatures: lattices, multivariate, hash-based, isogenies
- Practical schemes compatible with **ML-DSA, UOV and MAYO**
- Possible fallback to Threshold Raccoon for large thresholds / advanced properties

Questions?

Evaluation

Other ML-DSA parameter sets