Beyond the Threshold

Detecting Aborts in Lattice-based Threshold Signature Schemes

Guilhem Niot, joint works with *Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Thomas Prest, Michael Reichle, Kaoru Takemure*

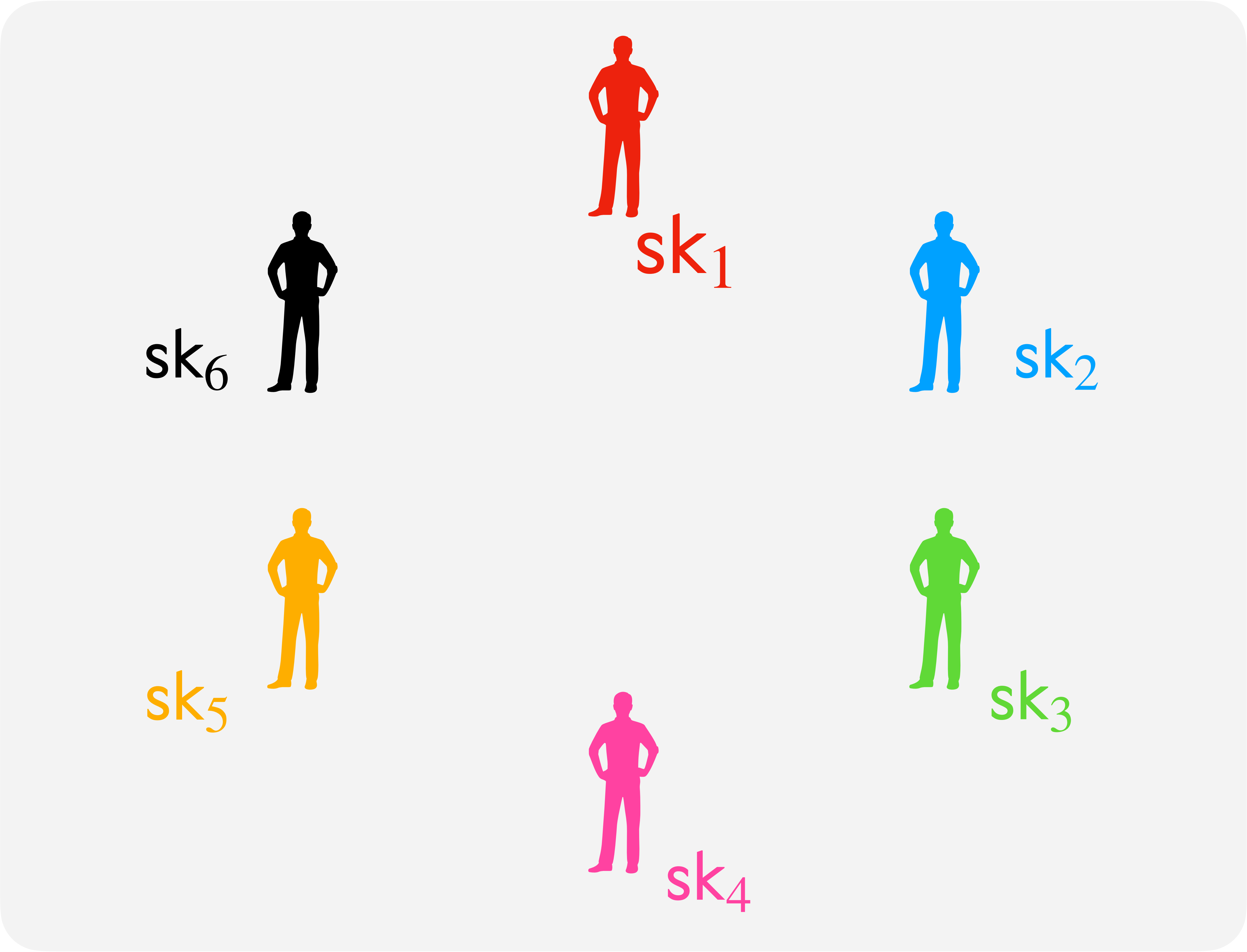
Visit Cryptography and Privacy Lab, Seoul National University - Dec. 2024

PQ SHIELD

1. Background

(T-out-of-N) threshold signatures
What are they?

An interactive protocol to distribute signature generation.

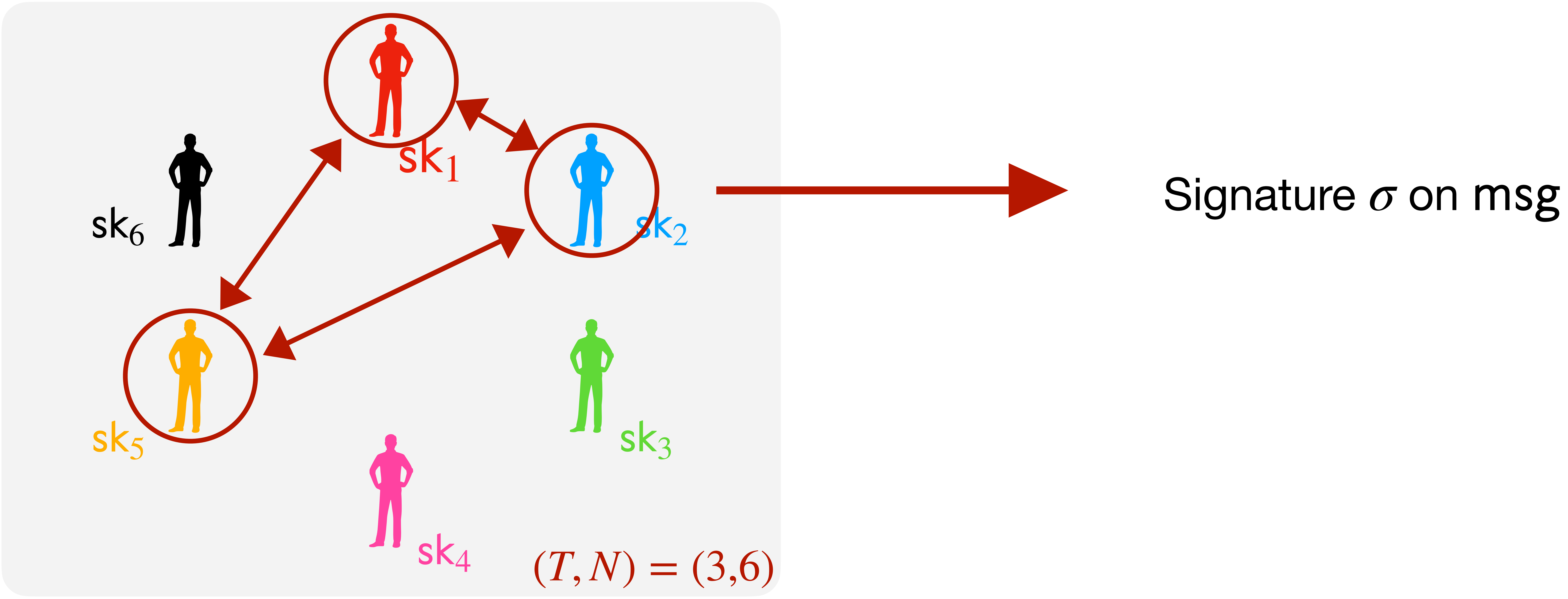


- 1 verification key vk
- 1 partial signing key sk_i per party
- Given at least T-out-of-N partial signing keys, we can sign.

(T-out-of-N) threshold signatures

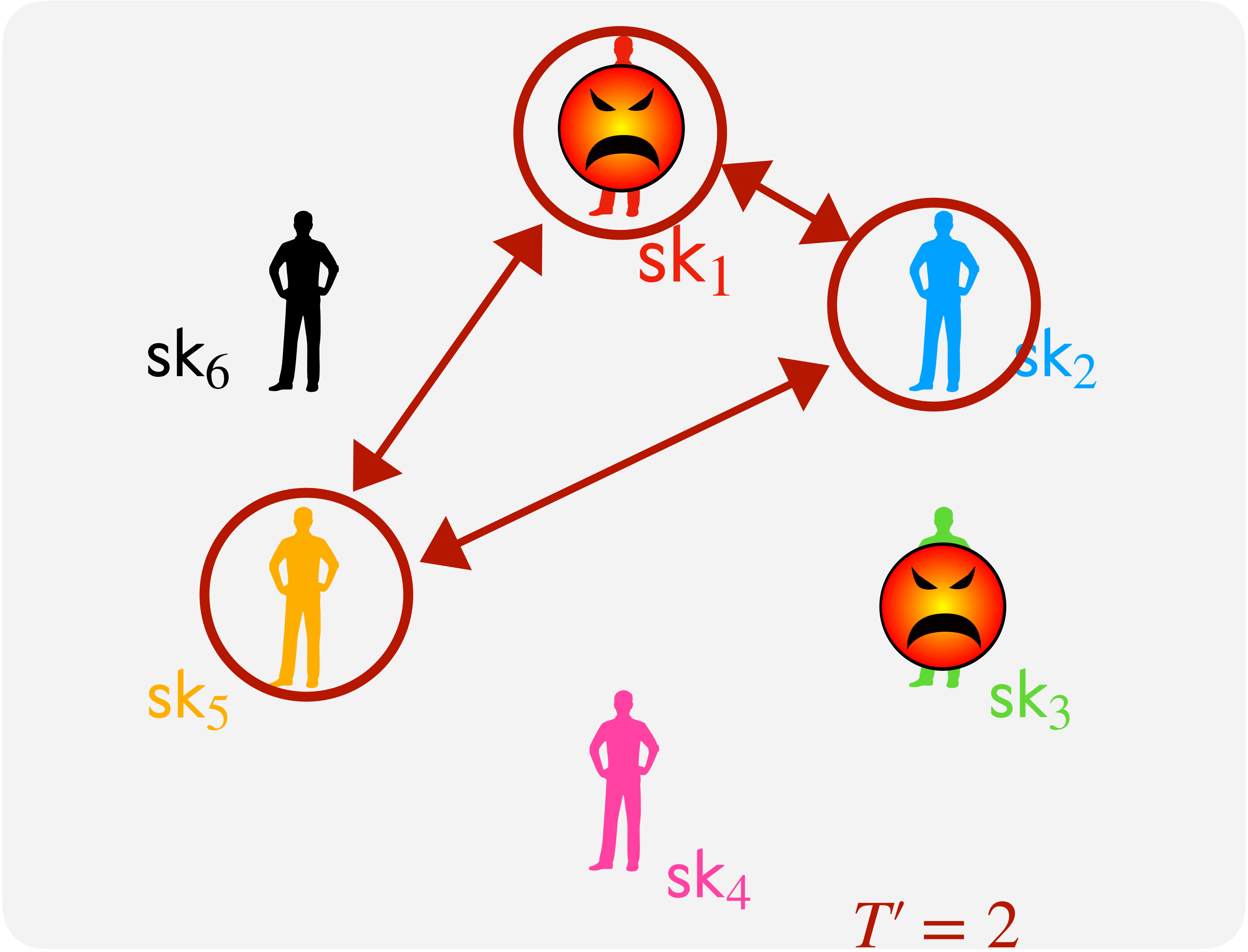
What are they?

An interactive protocol to distribute signature generation.



Core security properties

- **Correctness:** Given at least T-out-of-N partial signing keys, we can sign.
- **Unforgeability:** The signature scheme remains unforgeable even if up to $T-1$ parties are corrupted.



It's not possible to forge a new signature, even by taking part in the signing protocol.

More desirable properties

- **Distributed Key Generation:** Protocol allowing to distributively sample key material.
- **Abort identification (or robustness):** In the presence of malicious users, the signature protocol can identify misbehaving users (or guarantee a valid output).
- **Small round complexity:** Ideally can be as low as one round.
- **Backward compatibility:** Threshold schemes should ideally be compatible with existing primitives.

Threshold Signatures based on Lattices

- MPC-based solutions [CS19], [TPCZ24]
- 2-round TS via FHE: [BGG+18], [ASY22], [GKS23]
- TS with noise flooding (based on Raccoon): 3-round [dPKM+24], many follow-ups in 2024

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]

Threshold Signatures based on Lattices

- MPC-based solutions [CS19], [TPCZ24]
- 2-round TS via FHE: [BGG+18], [ASY22], [GKS23]
- TS with noise flooding (based on Raccoon): 3-round [dPKM+24], many follow-ups in 2024

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]

Threshold Raccoon, a practical 3-round threshold signature

κ	Number Signers	| vk |	| sig |	Total communication
128	≤ 1024	4 kB	13 kB	40 kB

… but only considers core security properties: correctness and unforgeability.

Advanced properties for ThRaccoon

Small round complexity

2-round [EKT24], [BKLM+24]

Distributed Key Generation (DKG) + Robustness

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

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

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*, *MPI-SP, Germany*
Akira Takahashi, *JPMorgan AI Research & AlgoCRYPT CoE, USA*
Mehdi Tibouchi, *NTT, Japan*

***Flood and Submerse*: Distributed Key Generation and Robust Threshold Signature from Lattices**

Thomas Espitau[1], Guilhem Niot[1,2], and Thomas Prest[1]

Advanced properties for ThRaccoon

Distributed Key Generation (DKG) + Robustness

***Flood and Submerse*: Distributed Key Generation and Robust Threshold Signature from Lattices**

Thomas Espitau[1], Guilhem Niot[1,2], and Thomas Prest[1]

κ	# rounds	Signers per session	| vk |	| sig |	Total communication
128	4	3T	4 kB	13 kB	56T kB

- Question: can we avoid the cost of robustness when parties behave honestly?
  - Only identify aborts instead of correcting them?

Focus of this presentation

- **Efficient Abort Identification**
  - Separate signing protocol and (costly) abort identification protocol
  - Signing protocol in 3 rounds + small communication

- Overview of 3 techniques to achieve Abort Identification
  - Based on Non-Interactive ZK proofs (NIZK)
  - Based on Verifiable Secret Sharing (VSS) [ENP24]
  - Novel Short Secret Sharing technique (for small thresholds)

2. Signing with (Threshold) Raccoon

Raccoon signature scheme

Raccoon . Keygen() $\rightarrow$ sk, vk

- vk $= [\mathbf{A} \quad \mathbf{I}] \cdot$ sk, for sk short

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

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

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

- $\mathbf{w} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{z} - c \cdot \text{vk}$
- Assert $c = H(\mathbf{w}, \text{msg})$
- Assert $\mathbf{z}$ short

Raccoon signature scheme

$\mathsf{Raccoon.Keygen}() \to \mathsf{sk}, \mathsf{vk}$

- $\mathsf{vk} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathsf{sk}$, for sk short

$\mathsf{Raccoon.Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

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

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

- $\mathbf{w} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{z} - c \cdot \mathsf{vk}$
- Assert $c = H(\mathbf{w}, \mathsf{msg})$
- Assert $\mathbf{z}$ short

Unforgeable assuming

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

Hint-MLWE assumption [KLSS23].

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

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

As hard as MLWE_σ if

$$\sigma_{\mathbf{r}} \geq \sqrt{Q} \cdot s_1(c) \cdot \sigma$$

Threshold Raccoon

$\mathsf{Raccoon}.\mathsf{Keygen}() \to \mathsf{sk}, \mathsf{vk}$

- $\mathsf{vk} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathsf{sk}$, for sk short

$\mathsf{Raccoon}.\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

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

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

- $\mathbf{w} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{z} - c \cdot \mathsf{vk}$
- Assert $c = H(\mathbf{w}, \mathsf{msg})$
- Assert $\mathbf{z}$ short

Shamir sharing on secret

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

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

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 $= [\mathbf{A} \quad \mathbf{I}] \cdot$ sk, for sk short

Raccoon . Sign(sk, msg) → sig

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

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

- $\mathbf{w} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{z} - c \cdot \mathsf{vk}$
- Assert $c = H(\mathbf{w}, \mathsf{msg})$
- Assert $\mathbf{z}$ short

First (insecure) attempt

ThRaccoon . Sign(sk, msg) → sig

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- 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

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

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 $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- 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

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

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

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

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- 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

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

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:
  - Any set of $< T$ values Δ_i is uniformly random
  - $\sum_{i \in S} \Delta_i = 0$

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

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- 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

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

Building a zero-share

	1	2	3	4
1	0	$\mathbf{m}_{1,2}$	$\mathbf{m}_{1,3}$	$\mathbf{m}_{1,4}$
2	$\mathbf{m}_{2,1}$	0	$\mathbf{m}_{2,3}$	$\mathbf{m}_{2,4}$
3	$\mathbf{m}_{3,1}$	$\mathbf{m}_{3,2}$	0	$\mathbf{m}_{3,4}$
4	$\mathbf{m}_{4,1}$	$\mathbf{m}_{4,2}$	$\mathbf{m}_{4,3}$	0

- Users i and j share a symmetric key $K_{i,j}$ and generate a fresh $\mathbf{m}_{i,j} = \mathsf{PRF}(K_{i,j}, \mathsf{sid})$ during each session
- User i knows all the $\mathbf{m}_{i,j}$ in its row and column

Building a zero-share

	1	2	3	4
1	0	$\mathbf{m}_{1,2}$	$\mathbf{m}_{1,3}$	$\mathbf{m}_{1,4}$
2	$\mathbf{m}_{2,1}$	0	$\mathbf{m}_{2,3}$	$\mathbf{m}_{2,4}$
3	$\mathbf{m}_{3,1}$	$\mathbf{m}_{3,2}$	0	$\mathbf{m}_{3,4}$
4	$\mathbf{m}_{4,1}$	$\mathbf{m}_{4,2}$	$\mathbf{m}_{4,3}$	0

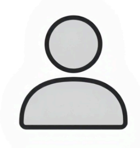
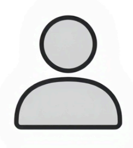
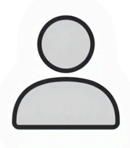
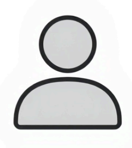
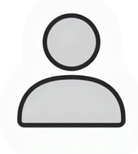
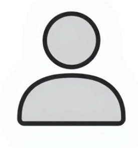
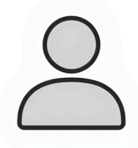
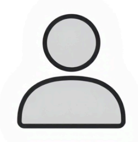
- Users i and j share a symmetric key $K_{i,j}$ and generate a fresh $\mathbf{m}_{i,j} = \text{PRF}(K_{i,j}, \text{sid})$ during each session
- User i knows all the $\mathbf{m}_{i,j}$ in its row and column

Building a zero-share

	1	2	3	4
1	0	$\mathbf{m}_{1,2}$	$\mathbf{m}_{1,3}$	$\mathbf{m}_{1,4}$
2	$\mathbf{m}_{2,1}$	0	$\mathbf{m}_{2,3}$	$\mathbf{m}_{2,4}$
3	$\mathbf{m}_{3,1}$	$\mathbf{m}_{3,2}$	0	$\mathbf{m}_{3,4}$
4	$\mathbf{m}_{4,1}$	$\mathbf{m}_{4,2}$	$\mathbf{m}_{4,3}$	0

- Users i and j share a symmetric key $K_{i,j}$ and generate a fresh $\mathbf{m}_{i,j} = \text{PRF}(K_{i,j}, \text{sid})$ during each session
- User i knows all the $\mathbf{m}_{i,j}$ in its row and column

Building a zero-share

	 $_1$	 $_2$	 $_3$	 $_4$
 $_1$	0	$\mathbf{m}_{1,2}$	$\mathbf{m}_{1,3}$	$\mathbf{m}_{1,4}$
 $_2$	$\mathbf{m}_{2,1}$	0	$\mathbf{m}_{2,3}$	$\mathbf{m}_{2,4}$
 $_3$	$\mathbf{m}_{3,1}$	$\mathbf{m}_{3,2}$	0	$\mathbf{m}_{3,4}$
 $_4$	$\mathbf{m}_{4,1}$	$\mathbf{m}_{4,2}$	$\mathbf{m}_{4,3}$	0

- Users i and j share a symmetric key $K_{i,j}$ and generate a fresh $\mathbf{m}_{i,j} = \text{PRF}(K_{i,j}, \text{sid})$ during each session
- User i knows all the $\mathbf{m}_{i,j}$ in its row and column
- We take $\Delta_i = \sum_{j \neq i} \mathbf{m}_{i,j} - \mathbf{m}_{j,i} \bmod q$

$\rightarrow$ valid sharing of 0

Building a zero-share

	1	2	3	4
1	0	$\mathbf{m}_{1,2}$	$\mathbf{m}_{1,3}$	$\mathbf{m}_{1,4}$
2	$\mathbf{m}_{2,1}$	0	$\mathbf{m}_{2,3}$	$\mathbf{m}_{2,4}$
3	$\mathbf{m}_{3,1}$	$\mathbf{m}_{3,2}$	0	$\mathbf{m}_{3,4}$
4	$\mathbf{m}_{4,1}$	$\mathbf{m}_{4,2}$	$\mathbf{m}_{4,3}$	0

- Users i and j share a symmetric key $K_{i,j}$ and generate a fresh $\mathbf{m}_{i,j} = \text{PRF}(K_{i,j}, \text{sid})$ during each session
- User i knows all the $\mathbf{m}_{i,j}$ in its row and column
- We take $\Delta_i = \sum_{j \neq i} \mathbf{m}_{i,j} - \mathbf{m}_{j,i} \bmod q$

  → valid sharing of 0
- If $< T$ users are corrupted, nothing more than the zero-sum with the remaining shares leaks

3. Abort identification

Identify aborts via NIZK

ThRaccoon . Sign(sk, msg) → sig

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

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

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$
- $\mathbf{r}_i$ is not consistent with $\mathbf{w}_i$
- $\mathbf{z}_i$ is incorrectly computed
  - Δ_i is not the correct one
  - or incorrect computation of $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![\mathsf{s}k]\!]_i + \mathbf{r}_i + \Delta_i$

Identify aborts via NIZK

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

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

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

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$
- $\mathbf{r}_i$ is not consistent with $\mathbf{w}_i$
- $\mathbf{z}_i$ is incorrectly computed
  - Δ_i is not the correct one
  - or incorrect computation of $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![sk]\!]_i + \mathbf{r}_i + \Delta_i$

Identify aborts via NIZK

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

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

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

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$
- $\mathbf{r}_i$ is not consistent with $\mathbf{w}_i$
- $\mathbf{z}_i$ is incorrectly computed
  - Δ_i is not the correct one
  - or incorrect computation of $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![sk]\!]_i + \mathbf{r}_i + \Delta_i$

Identify aborts via NIZK

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

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

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

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$
- $\mathbf{r}_i$ is not consistent with $\mathbf{w}_i$
- $\mathbf{z}_i$ is incorrectly computed
  - Δ_i is not the correct one
  - or incorrect computation of $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![\mathsf{s}k]\!]_i + \mathbf{r}_i + \Delta_i$

Identify aborts via NIZK

$\mathsf{ThRaccoon.Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

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

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$
- $\mathbf{r}_i$ is not consistent with $\mathbf{w}_i$
- $\mathbf{z}_i$ is incorrectly computed
  - Δ_i is not the correct one
  - or incorrect computation of $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![sk]\!]_i + \mathbf{r}_i + \Delta_i$

- The scheme is mostly linear: let's try proving shortness of $\mathbf{r}_i$ and correct computation of $\mathbf{z}_i$ via NIZK!
  - Issue: Δ_i is secretly sampled with a PRF… **Costly** to prove.
  - Instead: Ensure that user i and j agree on $\mathbf{m}_{i,j}$

Identify aborts via NIZK

ThRaccoon . Sign(sk, msg) → sig

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

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

ThRaccoon . IdAbort()

Round 1:

- Broadcast commitments on values $\mathbf{r}_i, (\mathbf{m}_{i,j}, \mathbf{m}_{j,i})_j$
- Broadcast Π_i proving that:
  - $\mathbf{r}_i$ is small and $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
  - $\mathbf{z}_i = L_{S,i} \cdot c \cdot [\![sk]\!]_i + \mathbf{r}_i + \Delta_i$ where $\Delta_i = \sum_j \mathbf{m}_{i,j} - \mathbf{m}_{j,i}$

Round 2:

- Check consistency of others' commitment on $\mathbf{m}_{i,j}, \mathbf{m}_{j,i}$
  - If inconsistent, broadcast complaint against j and reveal $K_{i,j}$
- Check proofs Π_i

Round 3:

- Review complaints: recompute $\mathbf{m}_{i,j}$ from $K_{i,j}$ and determine cheating user
- Mark users with invalid proofs as malicious

Identify aborts via NIZK

Instantiating this scheme aiming for compactness.

- Use Ajtai commitments for the T polynomials committed by each user: size does not increase with the size of the witness.
- Perform the proof with the exact proof system LNP.
- Finally, compress proof with the SNARK Labrador .

<table>
<tr><th>Phase</th><th># rounds</th><th>Signers per session</th><th>| vk |</th><th>| sig |</th><th>Total communication</th></tr>
<tr><td>Signing</td><td>3</td><td>T</td><td rowspan="2">4 kB</td><td rowspan="2">13 kB</td><td>30 kB</td></tr>
<tr><td>Abort Identification</td><td>3</td><td>T</td><td>60 + 6T kB</td></tr>
</table>

Identify aborts via NIZK

Instantiating this scheme aiming for compactness.

- Additional contributions
  - First description and security analysis of NIZK based on Labrador
  - Extraction from $n = \text{poly}(\lambda)$ proofs at once without an exponential loss

4. Abort identification without NIZK

Abort identification without NIZK

$\mathsf{ThRaccoon.Sign(sk, msg)} \to \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

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

Start over!

Why is it challenging to avoid a NIZK for aborts in ThRaccoon?

- Incompatibility of the sharings of sk and $\mathbf{r}_i$, that prevent a simple verification of computations.
- Additional non-linearity introduced by Δ_i

Abort identification without NIZK

$\mathsf{ThRaccoon.Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

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

Start over!

Why is it challenging to avoid a NIZK for aborts in ThRaccoon?

- Incompatibility of the sharings of sk and $\mathbf{r}_i$, that prevent a simple verification of computations.
- Additional non-linearity introduced by Δ_i

Let's use compatible sharings for sk and $\mathbf{r}_i$!

- Shamir sharing [ENP24]
- Novel short secret sharing

Abort identification by Shamir-Sharing $\mathbf{r}_i$

$\mathsf{ThRaccoon}.\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

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

$[\mathsf{ENP24}].\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$, and Shamir sharing $[\![\mathbf{r}_i]\!]$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$
- Privately send $[\![\mathbf{r}_i]\!]_j$ to user j

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

$$(c, \textstyle\sum_{i \in S} L_{s,i} \cdot [\![\mathbf{z}]\!]_i)$$

Abort identification by Shamir-Sharing $\mathbf{r}_i$

[ENP24] . $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$, and Shamir sharing $[\![\mathbf{r}_i]\!]$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$
- Privately send $[\![\mathbf{r}_i]\!]_j$ to user j

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

$$(c, \sum_{i \in S} L_{s,i} \cdot [\![\mathbf{z}]\!]_i)$$

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$, inconsistent with $\mathbf{w}_i$
- $[\![\mathbf{r}_i]\!]$ is invalid
- $\mathbf{z}_i$ is incorrectly computed
  - incorrect computation of $[\![\mathbf{z}]\!]_i = c \cdot [\![\mathsf{sk}]\!]_i + \sum_j [\![\mathbf{r}_j]\!]_i$

Abort identification by Shamir-Sharing $\mathbf{r}_i$

[ENP24] . $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$, and Shamir sharing $[\![\mathbf{r}_i]\!]$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$
- Privately send $[\![\mathbf{r}_i]\!]_j$ to user j

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

$$(c, \sum_{i \in S} L_{s,i} \cdot [\![\mathbf{z}]\!]_i)$$

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$, inconsistent with $\mathbf{w}_i$
- $[\![\mathbf{r}_i]\!]$ is invalid
- $\mathbf{z}_i$ is incorrectly computed
  - incorrect computation of $[\![\mathbf{z}]\!]_i = c \cdot [\![\mathsf{sk}]\!]_i + \sum_j [\![\mathbf{r}_j]\!]_i$

Abort identification by Shamir-Sharing $\mathbf{r}_i$

[ENP24] . $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$, and Shamir sharing $[\![\mathbf{r}_i]\!]$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$
- Privately send $[\![\mathbf{r}_i]\!]_j$ to user j

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

$$(c, \sum_{i \in S} L_{s,i} \cdot [\![\mathbf{z}]\!]_i)$$

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$, inconsistent with $\mathbf{w}_i$
- $[\![\mathbf{r}_i]\!]$ is invalid
- $\mathbf{z}_i$ is incorrectly computed
  - incorrect computation of $[\![\mathbf{z}]\!]_i = c \cdot [\![\mathsf{sk}]\!]_i + \sum_j [\![\mathbf{r}_j]\!]_i$

Abort identification by Shamir-Sharing $\mathbf{r}_i$

[ENP24] . $\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \rightarrow \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$, and Shamir sharing $[\![\mathbf{r}_i]\!]$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$
- Privately send $[\![\mathbf{r}_i]\!]_j$ to user j

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

$$(c, \sum_{i \in S} L_{s,i} \cdot [\![\mathbf{z}]\!]_i)$$

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$, inconsistent with $\mathbf{w}_i$
- $[\![\mathbf{r}_i]\!]$ is invalid
- $\mathbf{z}_i$ is incorrectly computed
  - incorrect computation of $[\![\mathbf{z}]\!]_i = c \cdot [\![\mathsf{sk}]\!]_i + \sum_j [\![\mathbf{r}_j]\!]_i$

Abort identification by Shamir-Sharing $\mathbf{r}_i$

[ENP24] . Sign(sk, msg) $\rightarrow$ sig

Round 1:

- Sample a short $\mathbf{r}_i$, and Shamir sharing $[\![\mathbf{r}_i]\!]$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$
- Privately send $[\![\mathbf{r}_i]\!]_j$ to user j

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

$$(c, \sum_{i \in S} L_{s,i} \cdot [\![\mathbf{z}]\!]_i)$$

What can go wrong?

- A malicious user uses a large $\mathbf{r}_i$, inconsistent with $\mathbf{w}_i$
- $[\![\mathbf{r}_i]\!]$ is invalid
- $\mathbf{z}_i$ is incorrectly computed
  - incorrect computation of $[\![\mathbf{z}]\!]_i = c \cdot [\![\mathsf{sk}]\!]_i + \sum_j [\![\mathbf{r}_j]\!]_i$

- [ENP24] introduced a Verifiable Secret Sharing (VSS) allowing to prove the (approximate) shortness of $\mathbf{r}_i$ and consistency of the sharing $[\![\mathbf{r}_i]\!]$
- Assuming the presence of $3T$ users during abort identification, Shamir-sharing allows error correction, and re-computation of $[\![\mathbf{z}]\!]$ to detect malicious users

Abort identification by Shamir-Sharing $\mathbf{r}_i$

[ENP24] . Sign(sk, msg) → sig

Round 1:

- Sample a short $\mathbf{r}_i$, and Shamir sharing $[\![\mathbf{r}_i]\!]$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$
- Privately send $[\![\mathbf{r}_i]\!]_j$ to user j

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

$$(c, \sum_{i \in S} L_{s,i} \cdot [\![\mathbf{z}]\!]_i)$$

Verifiable Secret Sharing:

- $\mathsf{VSS}.\mathsf{Prove}([\![\mathbf{r}]\!]) \to \pi, (\pi_j)_j$
- For user i, $\mathsf{VSS}.\mathsf{Verify}([\![\mathbf{r}]\!]_i, \pi, \pi_i) \to 0 \,|\, 1$

Guarantee: if T honest users verify VSS proofs, then $\mathbf{r}$ is small and consistently shared.

Abort identification by Shamir-Sharing $\mathbf{r}_i$

[ENP24] . Sign(sk, msg) → sig **with T users**

Round 1:

- Sample a short $\mathbf{r}_i$, and Shamir sharing $[\![\mathbf{r}_i]\!]$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- Broadcast $\mathsf{cmt}_i = H_{\mathsf{cmt}}(\mathbf{w}_i)$
- Privately send $[\![\mathbf{r}_i]\!]_j$ to user j

Round 2:

- Broadcast $\mathbf{w}_i$

Round 3:

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

Combine: the final signature is

$$(c, \textstyle\sum_{i \in S} L_{s,i} \cdot [\![\mathbf{z}]\!]_i)$$

IdAbort() **with 3T users**

Round 1:

- Run $\pi, \pi_i^j = \mathsf{VSS}.\mathsf{Prove}([\![\mathbf{r}_i]\!])$
- Privately send $[\![\mathbf{r}_i]\!]_j, \pi_i^j$ to user j
- Broadcast $\pi, [\![\mathbf{w}_i]\!] = [\mathbf{A} \quad \mathbf{I}] \cdot [\![\mathbf{r}_i]\!]$

Round 2:

- Check $\mathsf{VSS}.\mathsf{Verify}([\![\mathbf{r}_j]\!]_i, \pi, \pi_j^i)$ and $[\![\mathbf{w}_j]\!]_i = [\mathbf{A} \quad \mathbf{I}] \cdot [\![\mathbf{r}_j]\!]_i$ for $j \neq i$
  - If invalid, broadcast complaint and reveal $[\![\mathbf{r}_j]\!]_i$ and π_j^i.
- Broadcast $[\![\mathbf{z}]\!]_i = c \cdot [\![sk]\!]_i + \sum_j [\![\mathbf{r}_j]\!]_i$

Round 3:

- Mark as malicious users that sent invalid proofs or inconsistent $[\![\mathbf{w}_i]\!]$
- Mark as malicious users that sent $\mathsf{Reconstruct}([\![\mathbf{w}_i]\!])$ different from $\mathbf{w}_i$ used during signing
- Recover $[\![\mathbf{z}]\!]$ from the $[\![\mathbf{z}]\!]_i$ using Reed-Solomon error-correction
  - Mark as malicious users that sent a different $[\![\mathbf{z}]\!]_i$ during signing

Abort identification by Shamir-Sharing $\mathbf{r}_i$

Instantiating this scheme.

- We can use the VSS from [ENP24] to instantiate this scheme, that relies on Hint-MLWE to prove security.
- Additional optimizations:
  - Adaptive variant of Hint-MLWE to leverage that only $\ll Q$ VSS proofs are produced in this scheme.
  - Compress proof of correct computation of $\mathbf{w}_i$

Phase	# rounds	Signers per session	| vk |	| sig |	Total communication
Signing	3	T	4 kB	13 kB	30 + 0.032T kB
Abort Identification	3	3T			13 + 70T kB

- Successfully defers all the expensive parts of [ENP24] to the abort identification protocol (more users, larger communication)

Another approach with a novel short sharing

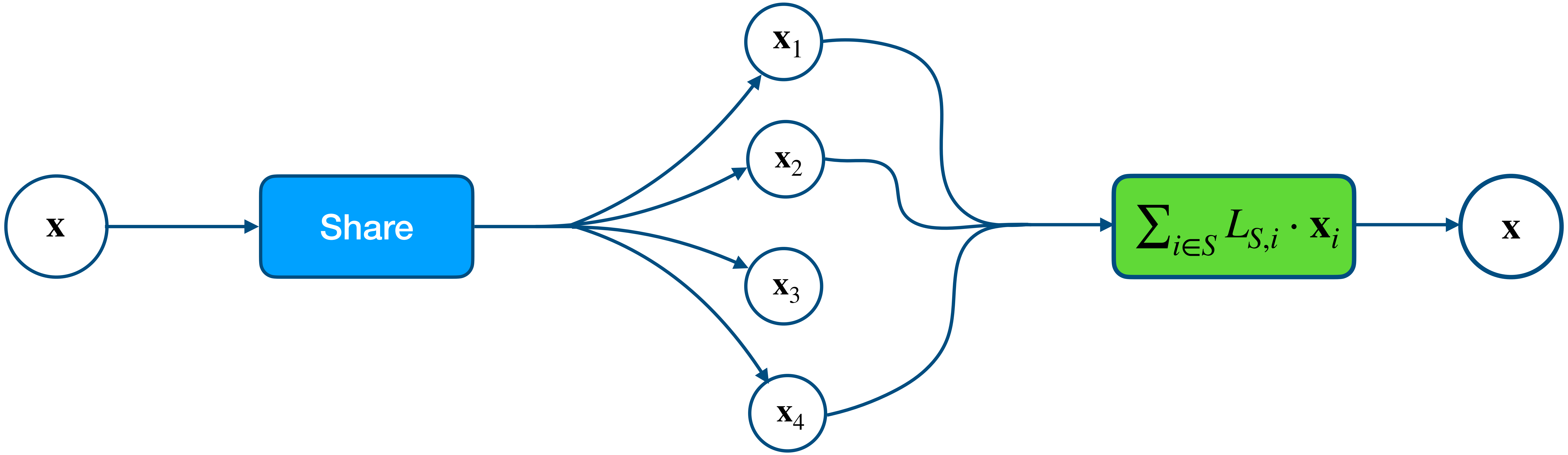
- How about using another sharing for sk instead?

  $\rightarrow$ The core issue in ThRaccoon was that the reconstruction coefficients and shares of sk were large, and $\mathbf{r}_i$ could not hide them: let's make them small!

Another approach with a novel short sharing

- How about using another sharing for sk instead?

  → The core issue in ThRaccoon was that the reconstruction coefficients and shares of sk were large, and $\mathbf{r}_i$ could not hide them: let's make them small!



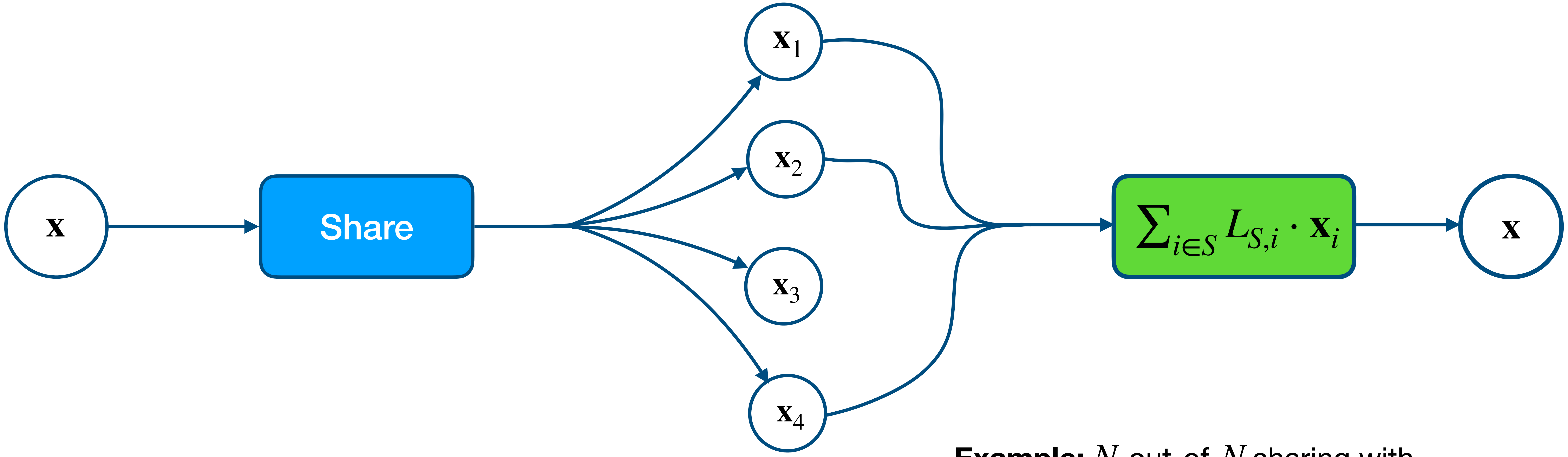
Short sharing requires:

- Short shares $\mathbf{x}_i$
- Small reconstruction coefficients $L_{S,i}$

Another approach with a novel short sharing

- How about using another sharing for sk instead?

$\rightarrow$ The core issue in ThRaccoon was that the reconstruction coefficients and shares of sk were large, and $\mathbf{r}_i$ could not hide them: let's make them small!



Short sharing requires:

- Short shares $\mathbf{x}_i$
- Small reconstruction coefficients $L_{S,i}$

Example: N-out-of-N sharing with

- $\mathbf{x}_1, \ldots, \mathbf{x}_{N-1} \leftarrow \mathscr{D}_\sigma^{N-1}$ and $\mathbf{x}_n = \mathbf{x} - \sum_{j<N} \mathbf{x}_i$
- $L_{S,i} = 1$

Extends to T-out-of-N with replicated secret sharing and $\binom{N}{T-1}$ shares per party.

Another approach with a novel short sharing

$\mathsf{ShortSS}.\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- 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 $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i$

Combine: the final signature is

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

For simplicity, we consider $T = N$ and $L_{S,i} = 1$.

Security.

- Everything is short in $\mathbf{z}_i$ and $\mathbf{r}_i$ hides $c \cdot \mathsf{sk}_i$.
  - Prove security with Hint-MLWE

Another approach with a novel short sharing

ShortSS . Sign(sk, msg) → sig

Round 1:

- Sample a short $\mathbf{r}_i$
- $\mathbf{w}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}_i$
- 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 $\mathbf{z}_i = c \cdot \mathsf{sk}_i + \mathbf{r}_i$

Combine: the final signature is

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

For simplicity, we consider $T = N$ and $L_{S,i} = 1$.

Security.

- Everything is short in $\mathbf{z}_i$ and $\mathbf{r}_i$ hides $c \cdot \mathsf{sk}_i$.
  - Prove security with Hint-MLWE

Identifiable aborts.

- Each $\mathsf{vk}_i = [\mathbf{A} \quad \mathbf{I}] \cdot \mathsf{sk}_i$ is a valid public key (sk_i is short)

  → Each $(c, \mathbf{z}_i)$ is a valid signature for vk_i
- Identifiable abort is as easy as verifying partial signatures!

Another approach with a novel short sharing

Instantiating this scheme.

- In the T-out-of-N setting, the number of shares grows with $\binom{N}{T-1}$, this scheme thus only supports a small number of parties.

For $N \leq 16$,

Phase	# rounds	Signers per session	| vk |	| sig |	Total communication
Signing	3	T	4 kB	11 kB	25 kB
Abort Identification	0	T			

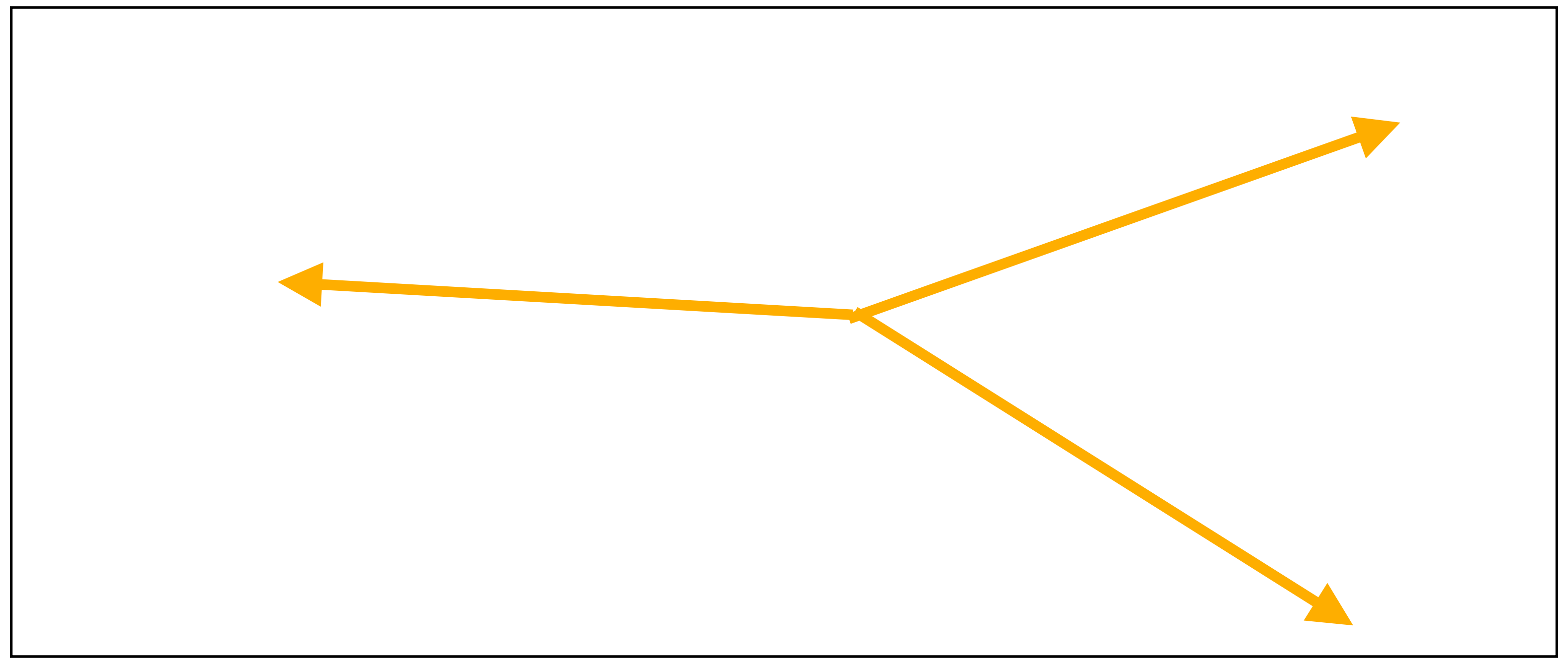
4. How large is the sum of T vectors?

How large is the sum of T vectors?

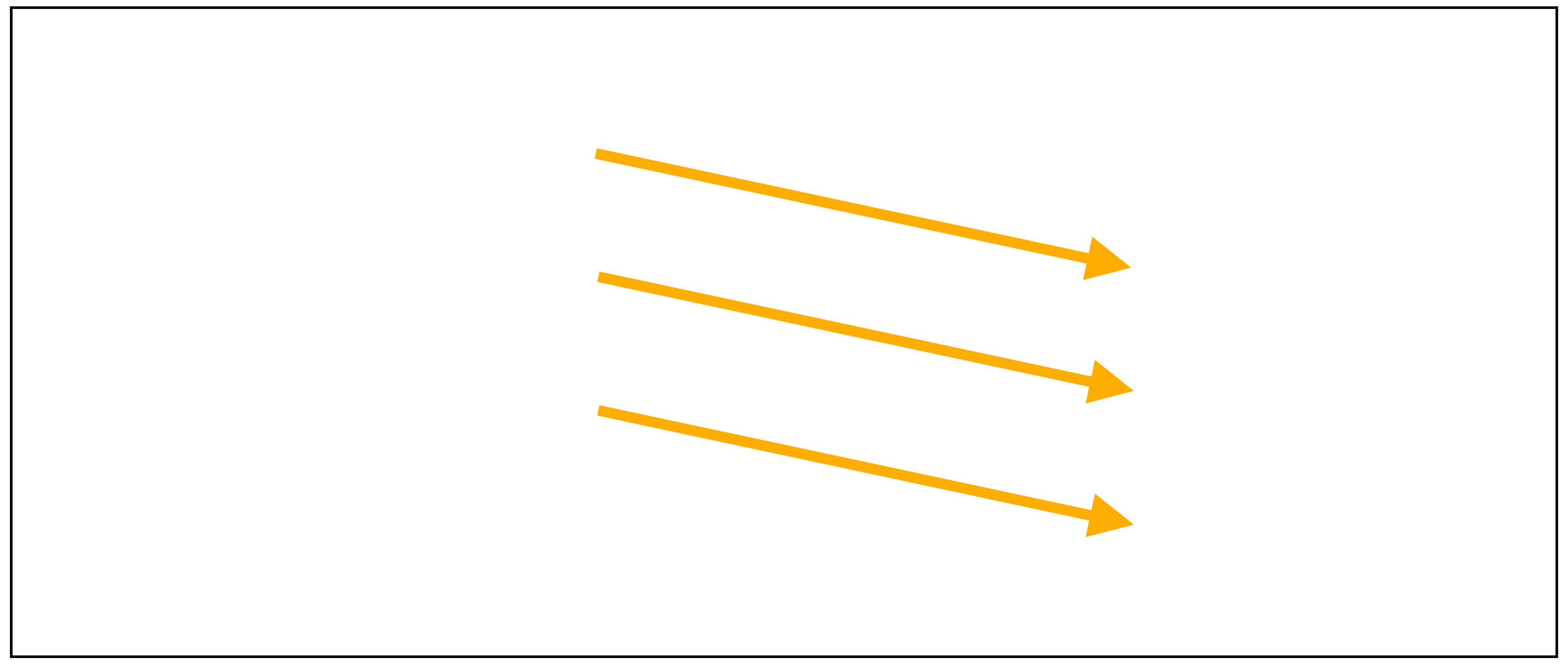
Taking a step back, all the presented schemes prove the shortness of $\mathbf{r}_i$ and deduce the shortness of $\sum_i \mathbf{r}_i$.

Consider vectors $\mathbf{r}_i \leftarrow \mathscr{D}_\sigma$.

What can we say about the norm of their sum?



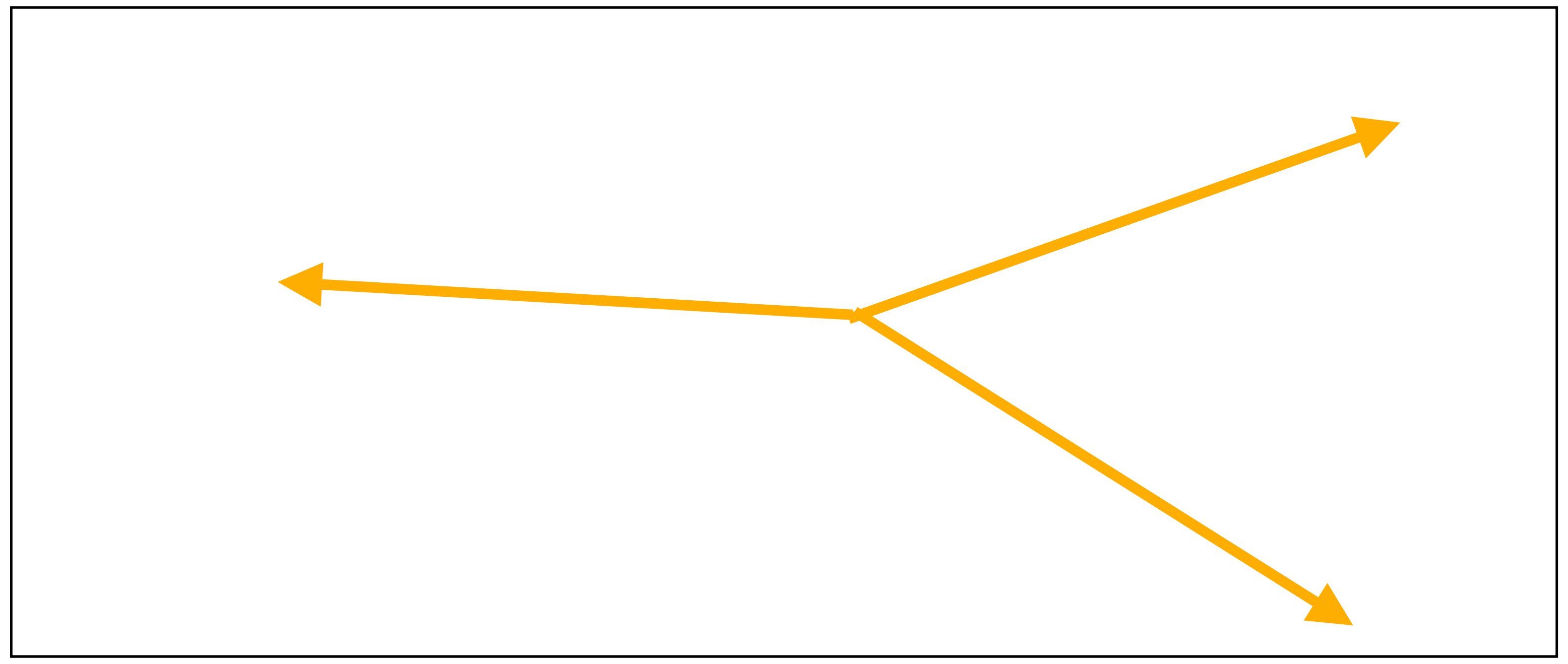
Average-case: $O(\sqrt{T})$



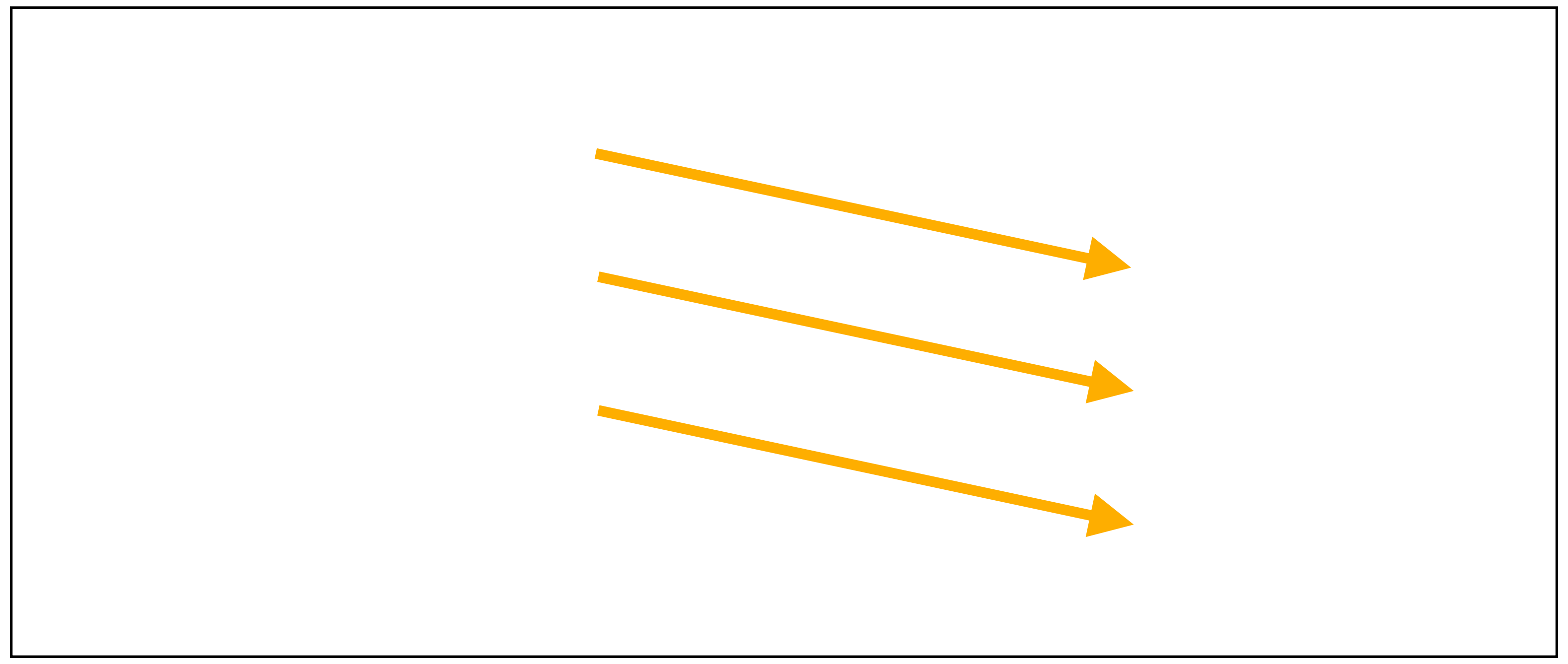
Worst-case: $O(T)$

- When users are honest: average-case.
- Colliding malicious users can force worst-case.

How large is the sum of T vectors?



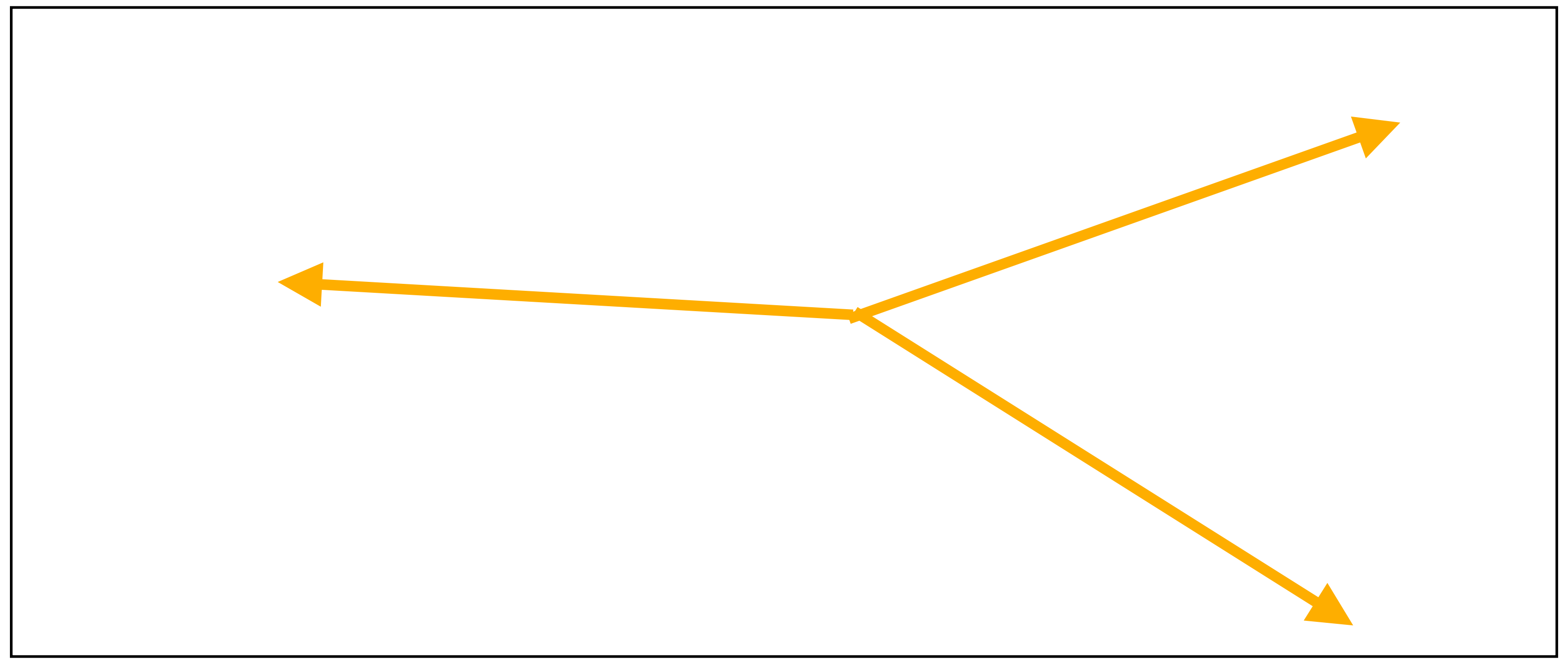
Average-case: $O(\sqrt{T})$



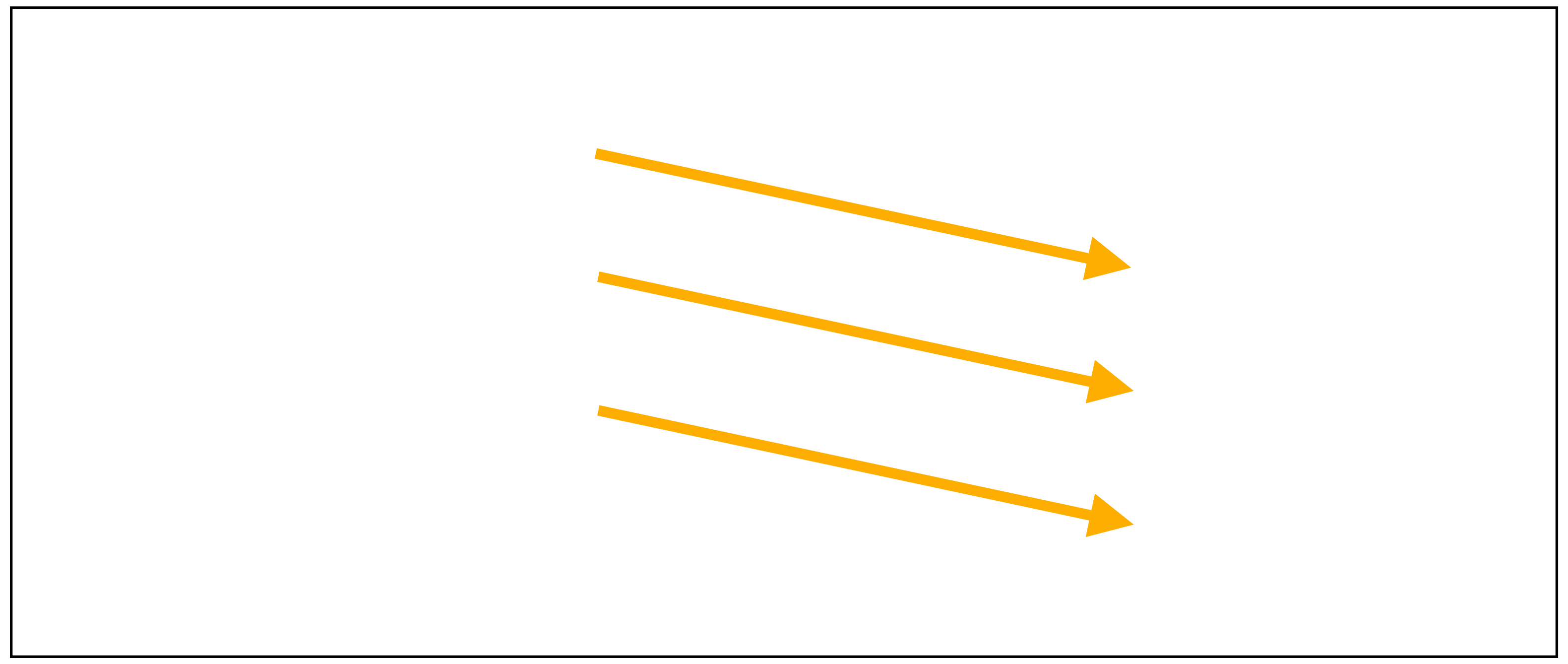
Worst-case: $O(T)$

In our two first schemes, no direct access to $\mathbf{r}_i$ (use of uniform-looking sharings) $\rightarrow$ bound in $O(T)$ that reduces security 😞

How large is the sum of T vectors?



Average-case: $O(\sqrt{T})$

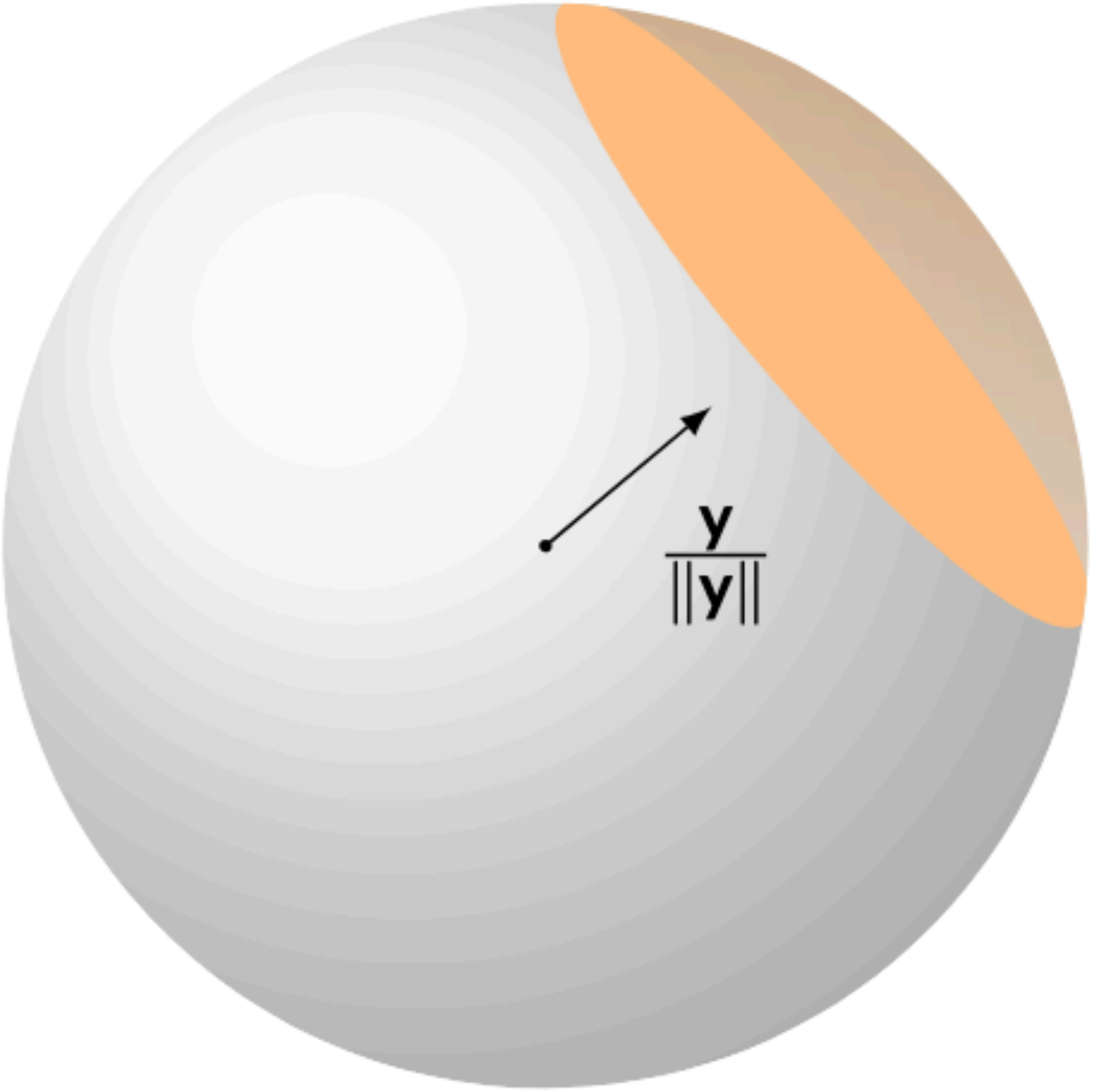


Worst-case: $O(T)$

In our two first schemes, no direct access to $\mathbf{r}_i$ (use of uniform-looking sharings) $\rightarrow$ bound in $O(T)$ that reduces security 😞

Can we do better with short secret sharing?

The Death Star Algorithm



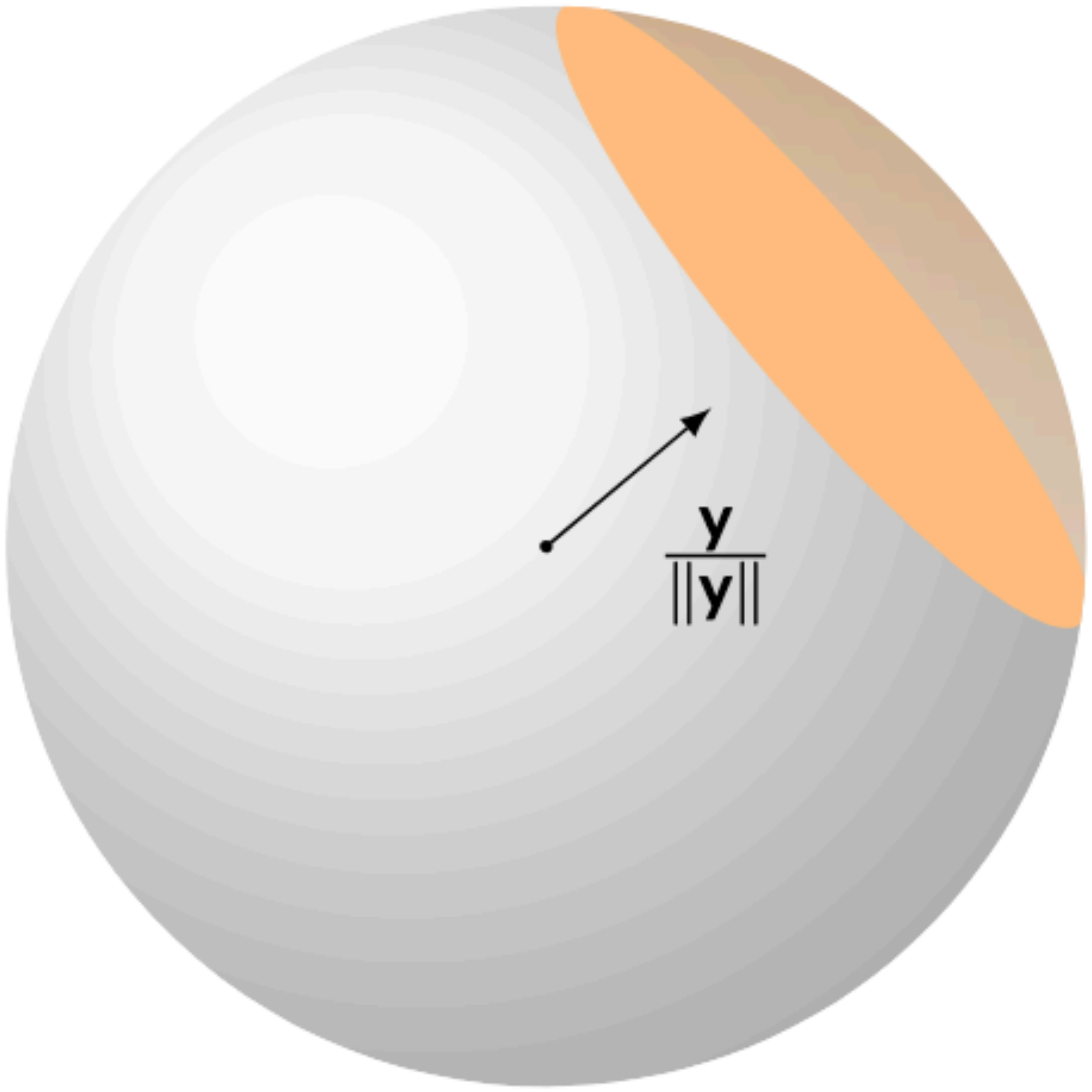
If $\mathbf{x} \leftarrow \mathscr{D}_{\sigma}$,

- $||\mathbf{x}||$ is concentrated around its expected value $\sqrt{n}\sigma$
- For any vector $\mathbf{y}$,

$$\langle \mathbf{x}, \mathbf{y} \rangle < \sigma\sqrt{O(\lambda)} \cdot ||\mathbf{y}||$$

except with probability $2^{-\lambda}$.

The Death Star Algorithm



The Death Star Algorithm

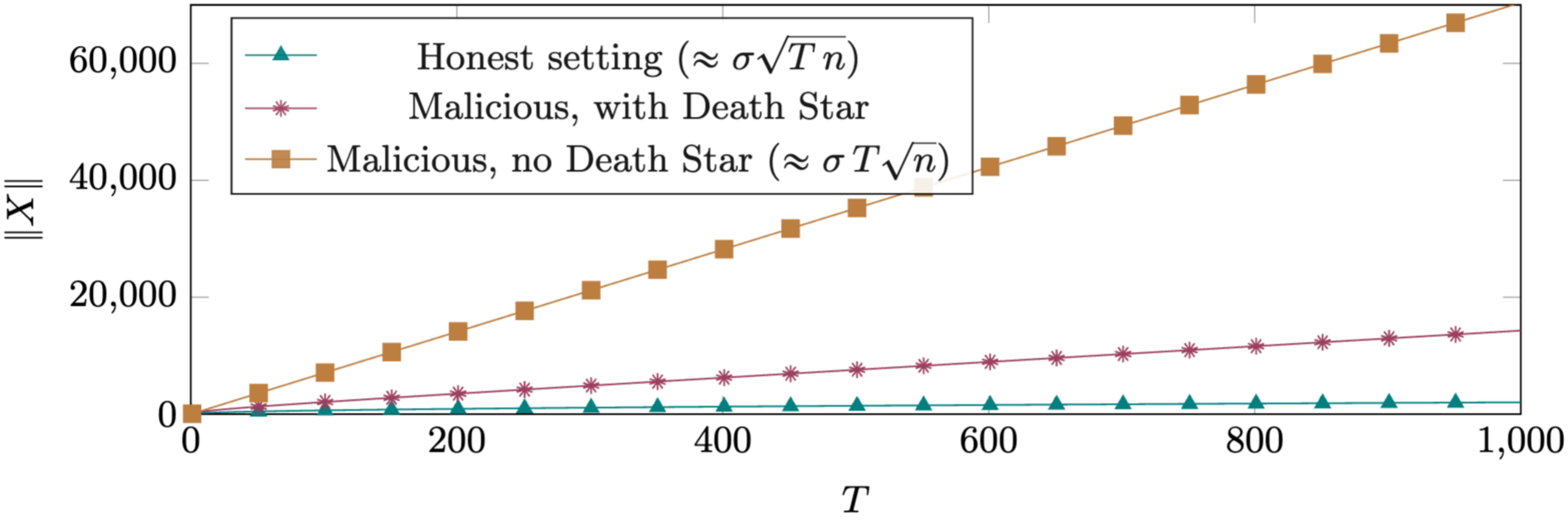
For each signer i,

- If $\|\mathbf{x}_i\| \geq (1 + o(1))\sqrt{n}\sigma$, reject i
- If $\langle \mathbf{x}_i, \mathbf{y}_i \rangle \geq \sigma\sqrt{O(\lambda)}\|\mathbf{y}_i\|$, where $\mathbf{y}_i = \sum_{j \neq i} \mathbf{x}_j$, reject i

When no signer is rejected, the sum $\mathbf{x} = \sum_i \mathbf{x}_i$ verifies

$$\|\mathbf{x}\| \leq \sigma \cdot T \cdot \sqrt{2 \log 2 \cdot \lambda} + \sigma \cdot \sqrt{T \cdot n} \cdot (1 + \varepsilon)$$

The Death Star Algorithm



Norm of $\mathbf{x} = \sum_i \mathbf{x}_i$ for $\sigma = 1$, $n = 4096$, 128 bits of security, and $T \leq 1000$

Conclusion

Conclusion

- **We proposed 3 lattice-based threshold signature schemes with efficient identifiable abort.**

- Fundamental difference in the secret sharings used for $(\mathsf{sk}, \mathbf{r}_i)$
  - (Shamir, Additive) → NIZK scheme
  - (Shamir, Shamir) → VSS scheme
  - (Short, Short) → Partial verifications + Death Star Algorithm

- Other contributions
  - Death Star algorithm
  - Security analysis of NIZK based on Labrador
  - Adaptive Hint-MLWE

Conclusion

Scheme	Signing	Abort Identification		max N
	Communication	# parties	Communication	
NIZK-based	30 kB	T	60 + 6T kB	1024
VSS-based	30 kB	3T	13 + 70T kB	1024
Short SS + partial verifications	25kB			16

Questions?