Hermine

An Efficient Raccoon-Style Non-Interactive Threshold Signature with Advanced Properties

Giacomo Borin, Sofía Celi, Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Guilhem Niot, Thomas Prest, Kaoru Takemure

IBM, University of Zurich, brave, University of Bristol, PQShield, AIST, Université de Rennes

MPTS 2026: NIST Workshop on Multi-Party Threshold Schemes 2026

Executive Summary

- **Hermine** = **Raccoon** (lattice-based) + **FROST** (two-round)
- Good scalability ($N \lesssim 64$)
- Advanced features:
  - Distributed Key Generation
  - Key Refresh
  - Identifiable Aborts

Starting Point: Raccoon

Raccoon: Schnorr over lattices

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

1. $\mathsf{vk} = \begin{bmatrix} \mathbf{A} & 1 \end{bmatrix} \cdot \mathsf{sk}$, for sk short.

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

1. Sample a short $\mathbf{r}$
2. $\mathbf{w} = \begin{bmatrix} \mathbf{A} & 1 \end{bmatrix} \cdot \mathbf{r}$
3. $c = H(\mathbf{w}, \mathsf{msg})$
4. $\mathbf{z} = \mathbf{r} + c \cdot \mathsf{sk}$
5. Output $\mathsf{sig} = (c, \mathbf{z})$

Raccoon.Verify$(\mathsf{vk}, \mathsf{msg}, \mathsf{sig}) \to \top/\bot$

1. $\mathbf{w}' = \begin{bmatrix} \mathbf{A} & 1 \end{bmatrix} \cdot \mathbf{z} - c \cdot \mathsf{vk}$
2. Assert $H(\mathbf{w}', \mathsf{msg}) = c$
3. Assert $\mathbf{z}$ is short

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

1. $\mathsf{vk} = g^{\mathsf{sk}}$, for sk uniform.

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

1. Sample r
2. $w = g^r$
3. $c = H(w, \mathsf{msg})$
4. $z = r + c \cdot \mathsf{sk}$
5. Output $\mathsf{sig} = (c, z)$

Schnorr.Verify$(\mathsf{vk}, \mathsf{msg}, \mathsf{sig}) \to \top/\bot$

1. $w' = g^z \cdot \mathsf{vk}^{-c}$
2. Assert $H(w', \mathsf{msg}) = c$

Security of Raccoon

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

1. $\mathsf{vk} = [\mathbf{A} \quad 1] \cdot \mathsf{sk}$, for sk short.

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

1. Sample a short $\mathbf{r}$
2. $\mathbf{w} = [\mathbf{A} \quad 1] \cdot \mathbf{r}$
3. $c = H(\mathbf{w}, \mathtt{msg})$
4. $\mathbf{z} = \mathbf{r} + c \cdot \mathsf{sk}$
5. Output $\mathtt{sig} = (c, \mathbf{z})$

Raccoon.Verify$(\mathsf{vk}, \mathtt{msg}, \mathtt{sig}) \rightarrow \top / \bot$

1. $\mathbf{w}' = [\mathbf{A} \quad 1] \cdot \mathbf{z} - c \cdot \mathsf{vk}$
2. Assert $H(\mathbf{w}', \mathtt{msg}) = c$
3. Assert $\mathbf{z}$ is short

Pseudorandomness of vk

vk is pseudorandom under ***Hint*-MLWE** [KLSS23]:

- → $(\mathbf{A}, \mathsf{vk})$ is a (usual) MLWE sample
- → The signatures are noisy multiples of sk.

As secure as MLWE_σ with $\sigma = O(1)$ if:

$$\frac{1}{\sigma_{\mathsf{sk}}^2} + \frac{\#\text{Queries} \cdot \|c\|}{\sigma_r^2} = O(1) \qquad (1)$$

Unforgeability

Self-target MSIS: same as ML-DSA.

Security of Raccoon

Raccoon.Keygen() → sk, vk

1. $\mathsf{vk} = 2 \cdot [\mathbf{A} \quad 1] \cdot \mathsf{sk}$, for sk short.

Raccoon.Sign(sk, msg) → sig

1. Sample a short $\mathbf{r}$
2. $\mathbf{w} = [\mathbf{A} \quad 1] \cdot \mathbf{r}$
3. $c = H(\mathbf{w}, \mathsf{msg})$
4. $\mathbf{z} = \mathbf{r} + 2 \cdot c \cdot \mathsf{sk}$
5. Output $\mathsf{sig} = (c, \mathbf{z})$

Raccoon.Verify(vk, msg, sig) → ⊤/⊥

1. $\mathbf{w}' = [\mathbf{A} \quad 1] \cdot \mathbf{z} - c \cdot \mathsf{vk}$
2. Assert $H(\mathbf{w}', \mathsf{msg}) = c$
3. Assert $\mathbf{z}$ is short

Pseudorandomness of vk

vk is pseudorandom under ***Hint*-MLWE** [KLSS23]:

→ ($\mathbf{A}$, vk) is a (usual) MLWE sample

→ The signatures are noisy multiples of sk.

As secure as MLWE_σ with $\sigma = O(1)$ if:

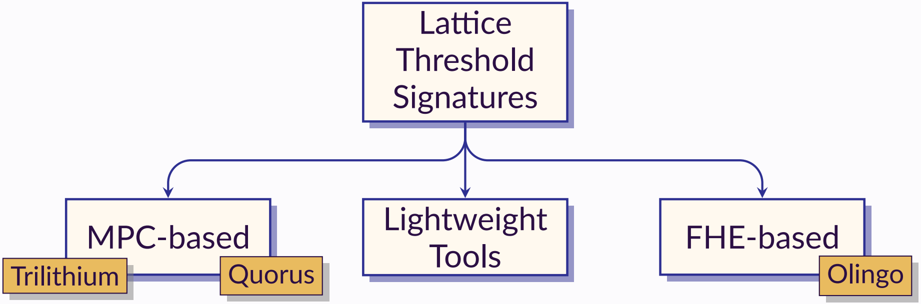
$$\frac{1}{\sigma_{\mathsf{sk}}^2} + \frac{\#\mathrm{Queries} \cdot \|2 \cdot c\|}{\sigma_r^2} = O(1) \quad (1)$$

Unforgeability

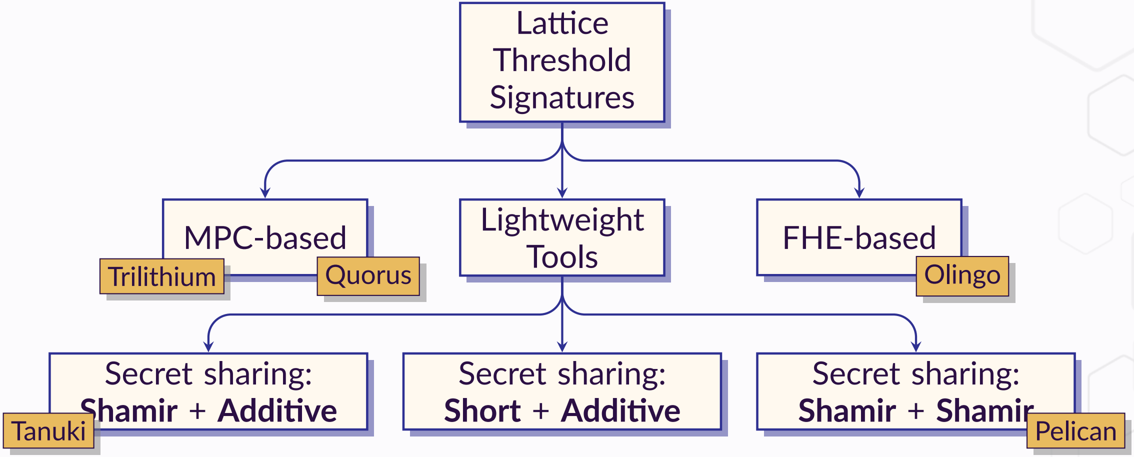
Self-target MSIS: same as ML-DSA.

Thresholdizing Lattices

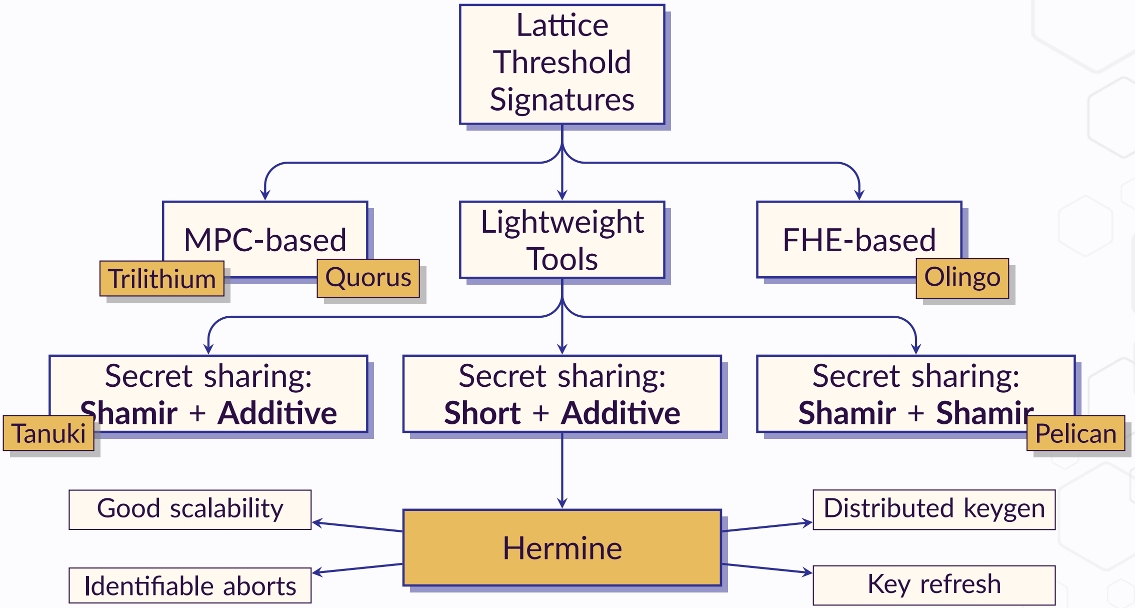
Design choices



Design choices



Design choices



Hermine = Raccoon + FROST

FROST

FROST.Preprocess(...)

1. $r_i, s_i \leftarrow \mathbb{Z}_q^2$
2. $R_i, S_i \leftarrow g^{r_i}, g^{s_i}$
3. Output (R_i, S_i)

FROST.Sign(...)

1. $\forall j, \rho_j = H(j, \mathsf{vk}, \mathsf{msg}, (j, R_j, S_j)_j)$
2. $R = \prod_j R_j \cdot S_j^{\rho_j}$
3. $c = H(R, \mathsf{vk}, \mathsf{msg})$
4. Output $z_i = r_i + s_i \cdot \rho_j + c \cdot \mathsf{sk}_i \cdot \lambda_{i,\mathsf{act}}$

FROST.Combine(...)

1. Compute c as in Round 2
2. Output $(c, z = \sum_j z_j)$

Security

TS-UF-3 in the ROM under AOM-DL (*algebraic one-more discrete logarithm*)

ROS attacks

Unlike Schnorr, FROST uses *two* nonces r_i, s_i and a randomizer ρ_j in order to resist ROS (*Random Overdetermined System*) attacks.

Hermine

Hermine.Preprocess(...)

1. Sample short $r_{i,1}, \ldots, r_{i,\mathsf{rep}}$
2. $\forall b \in \{1, \ldots, \mathsf{rep}\}, \mathbf{w}_{i,b} = [\mathbf{A} \quad \mathbf{I}] \cdot r_{i,b}$
3. Output $(\mathbf{w}_{i,b})_b$

Hermine.Sign(...)

1. $(\beta_2 \| \ldots \| \beta_{\mathsf{rep}}) = H(\mathsf{vk}, \mathtt{msg}, \mathtt{act}, (\mathbf{w}_{j,b})_{j,b})$
2. $\forall j \in \mathtt{act}, \mathbf{w}_j = \mathbf{w}_{j,1} + \sum_{b>1} \beta_b \cdot \mathbf{w}_{i,b}$
3. $\mathbf{w} = \sum_{j \in \mathtt{act}} \mathbf{w}_j$
4. $c = H(\mathbf{w}, \mathsf{vk}, \mathtt{msg})$
5. $\mathbf{z}_i = \mathbf{r}_{i,1} + \sum_{b>1} \beta_b \cdot \mathbf{r}_{i,b} + 2 \cdot c \cdot \mathsf{sk}_{i,\mathtt{act}}$
6. Output $\mathbf{z}_i$

Hermine.Combine(...)

1. Compute c as in Sign()
2. Output $(c, \mathbf{z} = \sum_j \mathbf{z}_j)$

ROS attacks

We require $\mathsf{rep} \approx 10$ commitments per party instead of 2 for FROST.

Hermine

Hermine.Preprocess(...)

1. Sample short $r_{i,1}, \dots, r_{i,\text{rep}}$
2. $\forall b \in \{1, \dots, \text{rep}\}, \mathbf{w}_{i,b} = [\mathbf{A} \quad \mathbf{I}] \cdot r_{i,b}$
3. Output $(\mathbf{w}_{i,b})_b$

Hermine.Sign(...)

1. $(\beta_2 \| \dots \| \beta_{\text{rep}}) = H(\mathtt{vk}, \mathtt{msg}, \mathtt{act}, (\mathbf{w}_{j,b})_{j,b})$
2. $\forall j \in \mathtt{act}, \mathbf{w}_j = \mathbf{w}_{j,1} + \sum_{b>1} \beta_b \cdot \mathbf{w}_{i,b}$
3. $\mathbf{w} = \sum_{j \in \mathtt{act}} \mathbf{w}_j$
4. $c = H(\mathbf{w}, \mathtt{vk}, \mathtt{msg})$
5. $\mathbf{z}_i = \mathbf{r}_{i,1} + \sum_{b>1} \beta_b \cdot \mathbf{r}_{i,b} + 2 \cdot c \cdot \mathtt{sk}_{i,\mathtt{act}}$
6. Output $\mathbf{z}_i$

Hermine.Combine(...)

1. Compute c as in Sign()
2. Output $(c, \mathbf{z} = \sum_j \mathbf{z}_j)$

ROS attacks

We require rep $\approx$ 10 commitments per party instead of 2 for FROST.

Security

Under AOM-MSIS [ZT25], Hermine is TS-sUF-2 in the ROM.

→ AOM-MSIS = "*Algebraic One-More Module Short Integer Solution*"

→ MSIS + MLWE $\Rightarrow$ AOM-MSIS

Hermine

Hermine.Preprocess(...)

1. Sample short $r_{i,1}, \ldots, r_{i,\mathrm{rep}}$
2. $\forall b \in \{1, \ldots, \mathrm{rep}\}, \mathbf{w}_{i,b} = [\mathbf{A} \quad \mathbf{I}] \cdot r_{i,b}$
3. Output $(\mathbf{w}_{i,b})_b$

Hermine.Sign(...)

1. $(\beta_2 \| \ldots \| \beta_{\mathrm{rep}}) = H(\mathtt{vk}, \mathtt{msg}, \mathtt{act}, (\mathbf{w}_{j,b})_{j,b})$
2. $\forall j \in \mathtt{act}, \mathbf{w}_j = \mathbf{w}_{j,1} + \sum_{b>1} \beta_b \cdot \mathbf{w}_{i,b}$
3. $\mathbf{w} = \sum_{j \in \mathtt{act}} \mathbf{w}_j$
4. $c = H(\mathbf{w}, \mathtt{vk}, \mathtt{msg})$
5. $\mathbf{z}_i = \mathbf{r}_{i,1} + \sum_{b>1} \beta_b \cdot \mathbf{r}_{i,b} + 2 \cdot c \cdot \mathtt{sk}_{i,\mathtt{act}}$
6. Output $\mathbf{z}_i$

Hermine.Combine(...)

1. Compute c as in Sign()
2. Output $(c, \mathbf{z} = \sum_j \mathbf{z}_j)$

ROS attacks

We require rep $\approx$ 10 commitments per party instead of 2 for FROST.

Security

Under AOM-MSIS [ZT25], Hermine is TS-sUF-2 in the ROM.

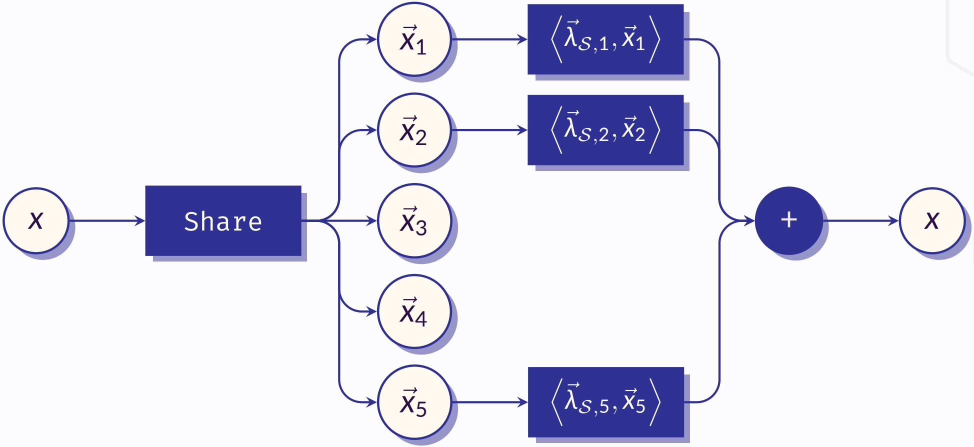
- → AOM-MSIS = "*Algebraic One-More Module Short Integer Solution*"
- → MSIS + MLWE $\Rightarrow$ AOM-MSIS

Secret sharing

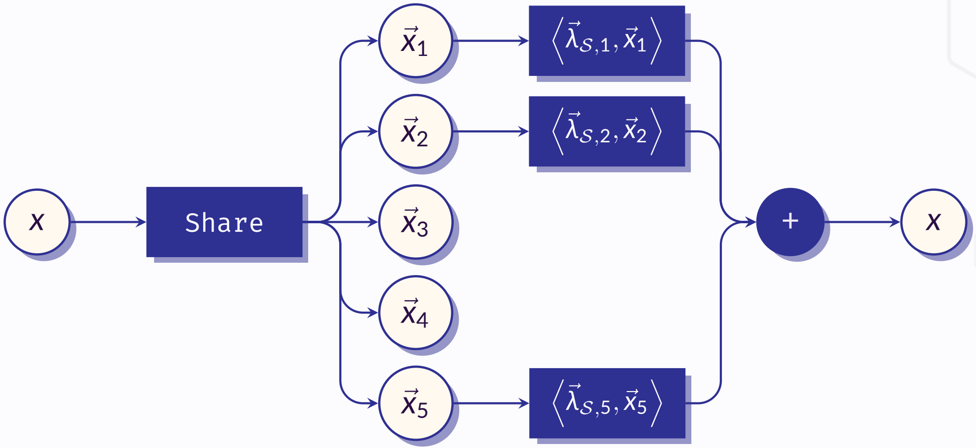
- → **Shamir SSS (e.g. Tanuki):** *zero-shares* [DKM+24] required for security.
- → **Short Secret Sharing (Hermine):** $\mathtt{sk}_{i,\mathtt{act}}$ is guaranteed to be short. Security follows from Hint-MLWE.

Short Secret Sharings?

(Short) secret sharings



(Short) secret sharings



"Short" secret sharing: we require that:

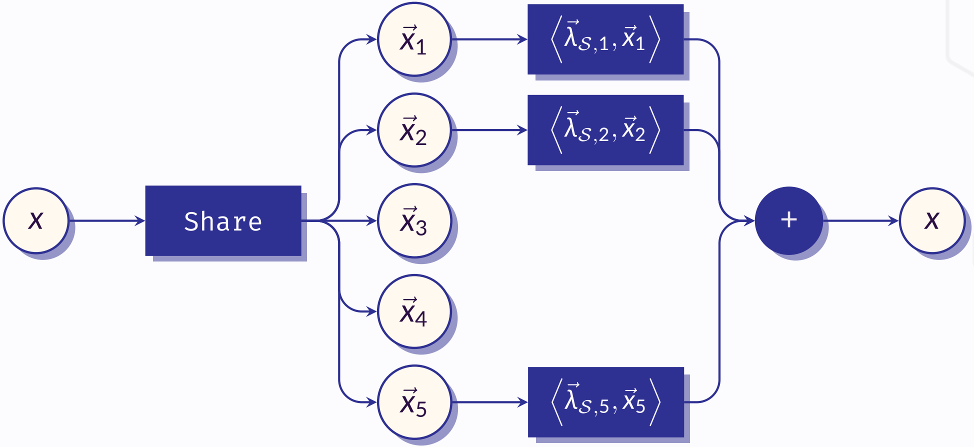
1. If x is short, the shares x_i are short
2. Reconstruction vectors $\vec{\lambda}_{\mathcal{S},i}$ are short

Example: N-out-of-N sharing where:

→ $(x_i)_{1 \le i < N} \leftarrow D_\sigma^{N-1}$ and $x_N = x - \sum_{i<N} x_i$

→ $\lambda_{\mathcal{S},i} = 1$

(Short) secret sharings



"Short" secret sharing: we require that:

1. If x is short, the shares x_i are short
2. Reconstruction vectors $\vec{\lambda}_{\mathcal{S},i}$ are short

Example: N-out-of-N sharing where:

→ $(x_i)_{1 \leq i < N} \leftarrow D_\sigma^{N-1}$ and $x_N = x - \sum_{i<N} x_i$

→ $\lambda_{\mathcal{S},i} = 1$

What about $T < N$?

Replicated secret sharing

Replicated secret sharing

- We create one share $\mathbf{s}_J$ for each subset of $\{1,...,N\}$ of size $N - T + 1$
  - A user $u \in \{1,...,N\}$ is given $\mathbf{s}_J$ if and only if $u \in J$
  - The secret is $\mathbf{s} = \sum_J \mathbf{s}_J$
- **T-correctness:** for each share $\mathbf{s}_J$, exactly $T - 1$ users do not have it
- **$(T - 1)$-privacy:** for any set $\mathtt{act}$ of size $T - 1$, no member of $\mathtt{act}$ has $\mathbf{s}_{\{1,...,N\}\setminus\mathtt{act}}$
- **Short secret sharing:** If the $\mathbf{s}_J$ are short, this is a short secret sharing.
- **Exponential growth:** The number of shares/party is $\binom{N-1}{T-1} = O(2^N)$

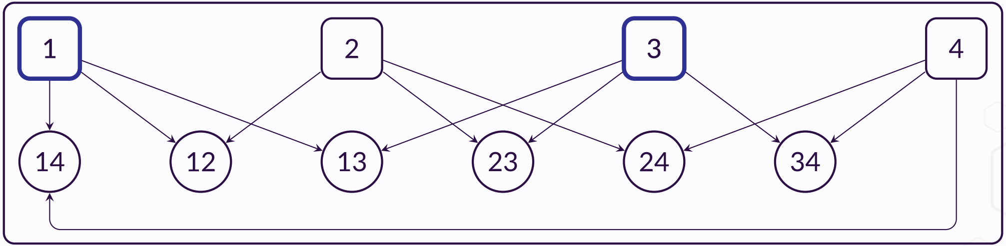


Figure 1: Illustration with $(N, T) = (4, 3)$.

Replicated secret sharing

Replicated secret sharing

- We create one share $\mathbf{s}_J$ for each subset of $\{1, \dots, N\}$ of size $N - T + 1$
  - A user $u \in \{1, \dots, N\}$ is given $\mathbf{s}_J$ if and only if $u \in J$
  - The secret is $\mathbf{s} = \sum_J \mathbf{s}_J$
- ***T*-correctness:** for each share $\mathbf{s}_J$, exactly $T - 1$ users do not have it
- **$(T-1)$-privacy:** for any set $\mathtt{act}$ of size $T - 1$, no member of $\mathtt{act}$ has $\mathbf{s}_{\{1,\dots,N\}\setminus \mathtt{act}}$
- **Short secret sharing:** If the $\mathbf{s}_J$ are short, this is a short secret sharing.
- **Exponential growth:** The number of shares/party is $\binom{N-1}{T-1} = O(2^N)$

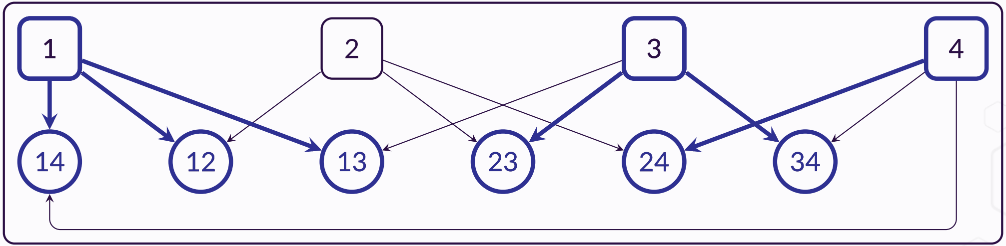


Figure 1: Illustration with $(N, T) = (4, 3)$.

Replicated secret sharing

Replicated secret sharing

- We create one share $\mathbf{s}_J$ for each subset of $\{1, \ldots, N\}$ of size $N - T + 1$
  - A user $u \in \{1, \ldots, N\}$ is given $\mathbf{s}_J$ if and only if $u \in J$
  - The secret is $\mathbf{s} = \sum_J \mathbf{s}_J$
- **T-correctness:** for each share $\mathbf{s}_J$, exactly $T - 1$ users do not have it
- **$(T - 1)$-privacy:** for any set $\mathtt{act}$ of size $T - 1$, no member of $\mathtt{act}$ has $\mathbf{s}_{\{1,\ldots,N\}\backslash \mathtt{act}}$
- **Short secret sharing:** If the $\mathbf{s}_J$ are short, this is a short secret sharing.
- **Exponential growth:** The number of shares/party is $\binom{N-1}{T-1} = O(2^N)$

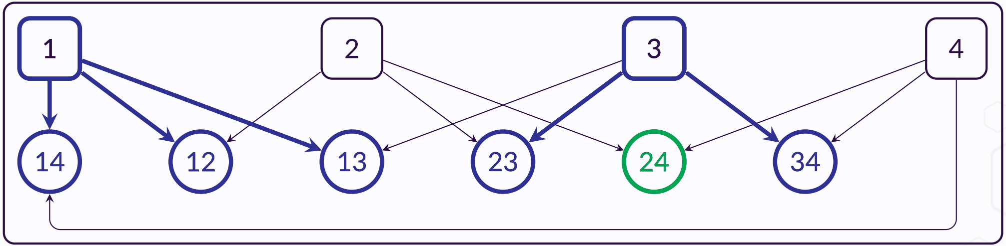


Figure 1: Illustration with $(N, T) = (4, 3)$.

Recursive/Vandermonde secret sharing

Vandermonde's identity

For $0 \leq T \leq N$:

$$\binom{N}{T} = \sum_{k=0}^{T} \binom{\lfloor N/2 \rfloor}{k} \cdot \binom{\lceil N/2 \rceil}{T-k} \tag{2}$$

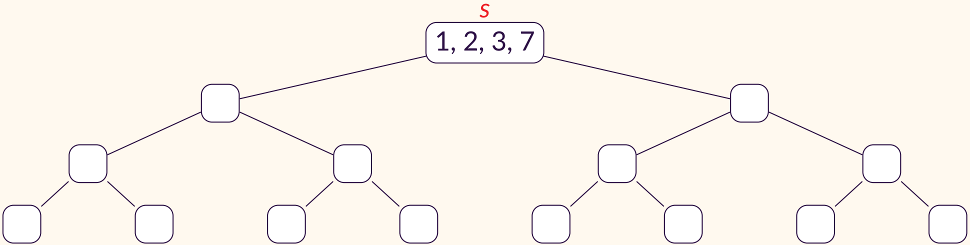
Recursive secret sharing (Desmedt-Di Crescenzo-Burmester'94)

Turn Eq. (2) into a secret sharing:

1. Enumerating all the possible disjunctions of the form in Eq. (2)
2. For each disjunction, share the secret s in two: $s = s_0 + (s - s_0)$
   1. Recursively share s_0 across members of $\texttt{act}_{<N/2}$
   2. Recursively share $s - s_0$ across members of $\texttt{act}_{\geq N/2}$

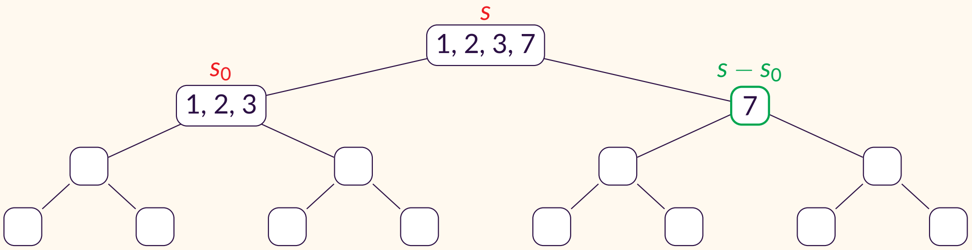
Example: 4-out-of-8

Recover with $\texttt{act} = \{1, 2, 3, 7\}$



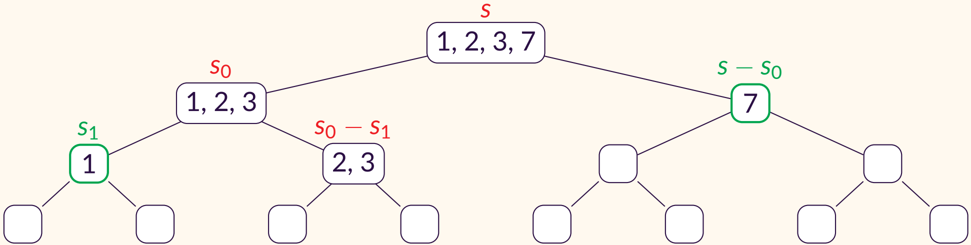
Example: 4-out-of-8

Recover with $\texttt{act} = \{1, 2, 3, 7\}$



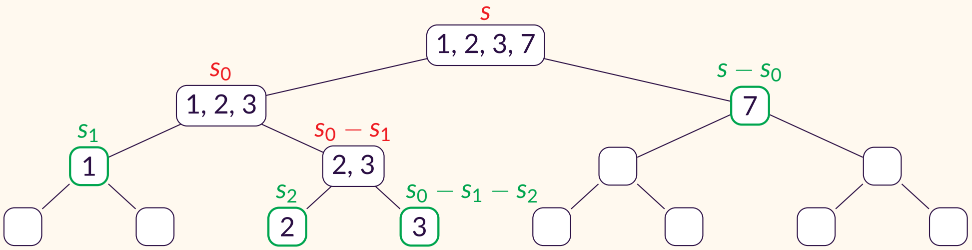
Example: 4-out-of-8

Recover with $\texttt{act} = \{1, 2, 3, 7\}$



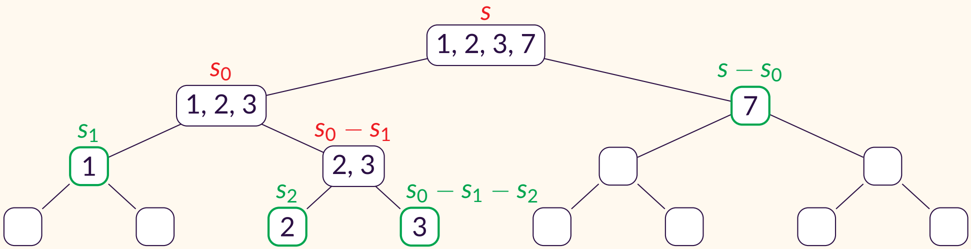
Example: 4-out-of-8

Recover with $\texttt{act} = \{1, 2, 3, 7\}$



Example: 4-out-of-8

Recover with $\texttt{act} = \{1, 2, 3, 7\}$



Fun-yet-useful facts:

- → This is a $\{-1, 0, 1\}$-LSSS (*Linear Integer Secret Sharing Scheme*).
  - If all s_i are sampled form a short distribution, this is also a **short secret sharing**.
- → The Share procedure needs to enumerate disjunctions, but this is rather efficient (more than replicated secret sharing).

Efficiency comparison

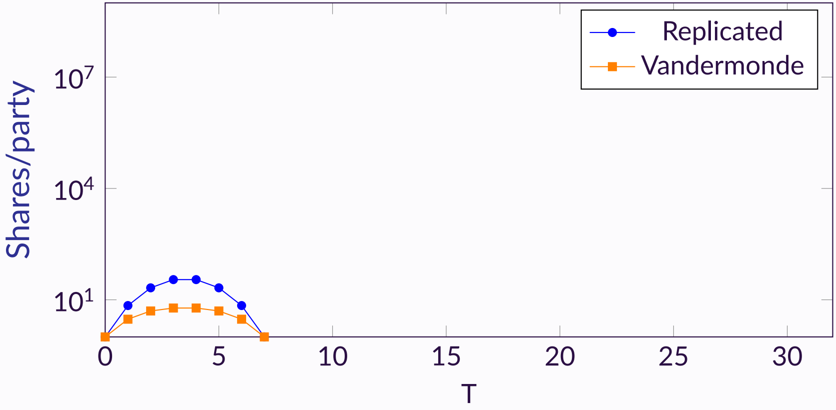


Figure 2: Number of shares/party as a function of T (N = 8)

Efficiency comparison

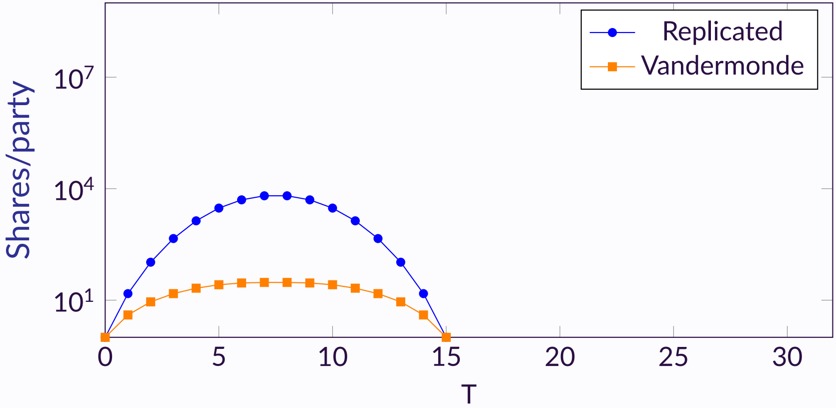


Figure 2: Number of shares/party as a function of T (N = 16)

Efficiency comparison

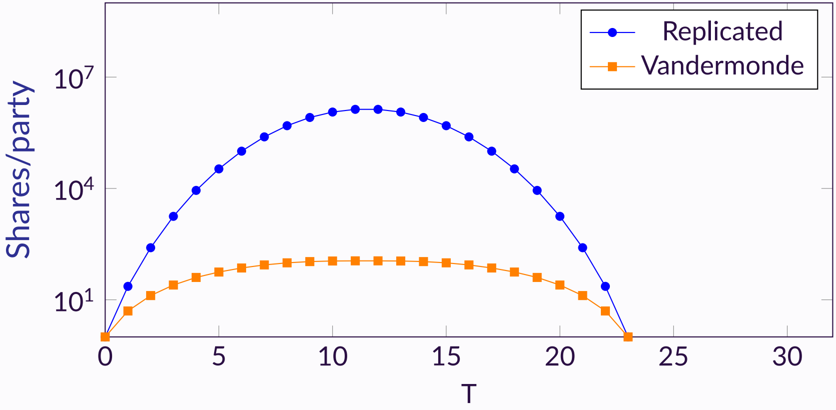


Figure 2: Number of shares/party as a function of T (N = 24)

Efficiency comparison

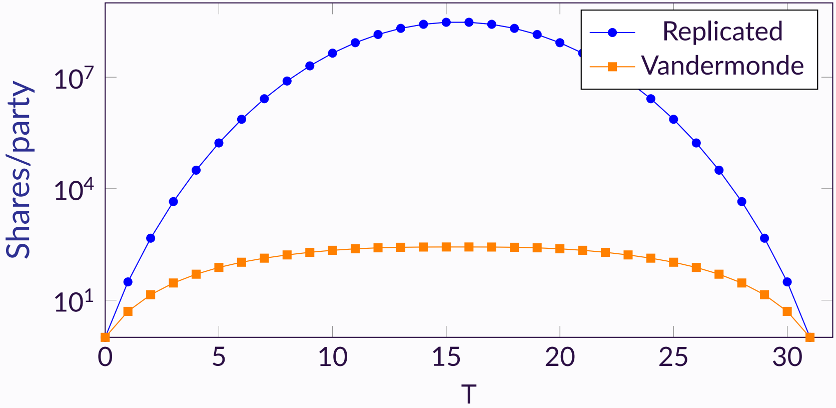


Figure 2: Number of shares/party as a function of T (N = 32)

Storage cost:

→ **Vandermonde:** up to ≈ 1 MB

→ **Replicated:** up to ≈ 1 TB

Advanced Properties

Back to Hermine: Identifiable aborts

Hermine.Preprocess(...)

1. Sample short $r_{i,1}, \ldots, r_{i,\mathsf{rep}}$
2. $\forall b \in \{1, \ldots, \mathsf{rep}\}, \mathbf{w}_{i,b} = [\mathbf{A} \quad \mathbf{I}] \cdot r_{i,b}$
3. Output $(\mathbf{w}_{i,b})_b$

Hermine.Sign(...)

1. $(\beta_2 \| \ldots \| \beta_{\mathsf{rep}}) = H(\mathtt{vk}, \mathtt{msg}, \mathtt{act}, (\mathbf{w}_{j,b})_{j,b})$
2. $\forall j \in \mathtt{act}, \mathbf{w}_j = \mathbf{w}_{j,1} + \sum_{b>1} \beta_b \cdot \mathbf{w}_{i,b}$
3. $\mathbf{w} = \sum_{j \in \mathtt{act}} \mathbf{w}_j$
4. $c = H(\mathbf{w}, \mathtt{vk}, \mathtt{msg})$
5. $\mathbf{z}_i = \mathbf{r}_{i,1} + \sum_{b>1} \beta_b \cdot \mathbf{r}_{i,b} + 2 \cdot c \cdot \mathtt{sk}_{i,\mathtt{act}}$
6. Output $\mathbf{z}_i$

Hermine.Combine(...)

1. Compute c as in Sign()
2. Output $(c, \mathbf{z} = \sum_j \mathbf{z}_j)$

Identifiable aborts

Let $\mathtt{vk}_{i,\mathtt{act}} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathtt{sk}_{i,\mathtt{act}}$.

→ $(\mathtt{sk}_{i,\mathtt{act}}, \mathtt{vk}_{i,\mathtt{act}})$ is a valid keypair

→ $(c, \mathbf{z}_i)$ is a valid "partial signature"

1. $\mathbf{z}_i$ is short
2. $[\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{z}_i = \mathbf{w}_i + 2 \cdot c \cdot \mathtt{vk}_{i,\mathtt{act}}$

We exploit this observation to identify misbehaving parties.

DKG and Key Refresh

Key Refresh (KR)

1. Compute and distribute a short secret sharing $(\mathtt{rk}_{j,\mathtt{act}})_{j,\mathtt{act}}$ of 0.
   1. Privately send to each party i their private shares $\mathtt{sk}_{i,\mathtt{act}}$
   2. Broadcast the partial verification keys $\mathtt{vk}_{i,\mathtt{act}} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathtt{sk}_{i,\mathtt{act}}$
2. Each party updates their known partial keys accordingly.

Distributed Key Generation (DKG)

1. Each dealer j generates a keypair $\mathtt{vk}_j, \mathtt{sk}_j$ and shares them among parties.
   1. Public (partial) keys are broadcast.
   2. Private (partial) keys are sent over a private channel.
2. Each party checks the validity of their own keypairs, and aggregates the keys.

Next steps

Implementation and experiments

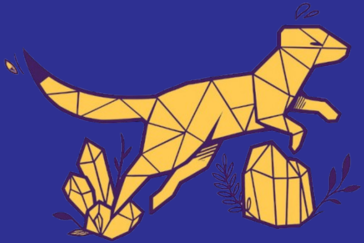
Formalization

Your feedback:

- Constraints?
  - Number of parties N
  - Threshold T
  - Sizes/communication/computation/storage
  - etc.
- Do you need DKG?
- Do you need Identifiable aborts?
- Do you need Key Refresh?
- Do you need other properties?

Thank you!

https://hermine-th.org/



Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen.
Threshold raccoon: Practical threshold signatures from standard lattice assumptions.
In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 219–248. Springer, Cham, May 2024.

Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song.
Toward practical lattice-based proof of knowledge from hint-MLWE.
In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 549–580. Springer, Cham, August 2023.

Chenzhi Zhu and Stefano Tessaro.
The algebraic one-more MISIS problem and applications to threshold signatures.
In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part I*, volume 16000 of *LNCS*, pages 548–581. Springer, Cham, August 2025.