Threshold Signatures in the Head

Thibauld Feneuil, Matthieu Rivain, Damien Vergnaud, Auguste Warmé-Janville

NIST Workshop on Multi-Party Threshold Schemes, 3^{rd} session – October 2026





Presented on 2026-10-06 @ NIST TCPT3

Threshold signatures in the Head – context and contribution

PQ threshold signatures

- Mostly based on lattices.
- Few alternatives (UOV, group actions), still under development.
- One MPCitH construction using GGM trees [CC24].

[1] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

Threshold signatures in the Head – context and contribution

PQ threshold signatures

- Mostly based on lattices.
- Few alternatives (UOV, group actions), still under development.
- One MPCitH construction using GGM trees [CC24].

MPCitH signatures: ZKPoK via PIOP + PCS[1]

- GGM commitments: shortest signatures ($\sim$ 2.5-4 kB), used by all NIST candidates.
- Merkle commitments: larger signatures ($\sim$ 6-15 kB), better suited to advanced signatures.

[1] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

Threshold signatures in the Head – context and contribution

PQ threshold signatures

- Mostly based on lattices.
- Few alternatives (UOV, group actions), still under development.
- One MPCitH construction using GGM trees [CC24].

MPCitH signatures: ZKPoK via PIOP + PCS[1]

- GGM commitments: shortest signatures ($\sim$ 2.5-4 kB), used by all NIST candidates.
- Merkle commitments: larger signatures ($\sim$ 6-15 kB), better suited to advanced signatures.

Impossibility result

Threshold MPCitH requires costly threshold symmetric primitives or signatures that grow with the number of signers [DKR24].

[1] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

Threshold signatures in the Head – context and contribution

PQ threshold signatures

- Mostly based on lattices.
- Few alternatives (UOV, group actions), still under development.
- One MPCitH construction using GGM trees [CC24].

MPCitH signatures: ZKPoK via PIOP + PCS[1]

- GGM commitments: shortest signatures ($\sim$ 2.5-4 kB), used by all NIST candidates.
- Merkle commitments: larger signatures ($\sim$ 6-15 kB), better suited to advanced signatures.

Impossibility result

Threshold MPCitH requires costly threshold symmetric primitives or signatures that grow with the number of signers [DKR24].

Our contribution – TSitH (eprint 2026/1125)

In [FRVWJ26] we reduce the linear signature growth per signer from $\sim$ 2.4kB [CC24] to a few hundred bytes.

[1] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

PIOP + PCS[2] signature

Base signature

Non-interactive zero-knowledge proof of knowledge protocol (uses Fiat-Shamir). Supports any OWF statement x with witness w. Ψ encodes the OWF instance verification.

$\mathsf{Sign}(\mathsf{pk} : x, \mathsf{sk} : w, \mathsf{msg})$:

1. $\boldsymbol{P}(X) \in \mathbb{F}[X]^{\leq d} \leftarrow \mathsf{Encode}(w, \$)$
2. $\mathsf{com} \leftarrow \mathsf{PCS.Commit}(\boldsymbol{P})$
3. $c \leftarrow \mathsf{Hash}_1(x, \mathsf{com})$
4. $Q \leftarrow \Psi(\boldsymbol{P}, x, c)$
5. $E \subset \mathbb{E} \leftarrow \mathsf{Hash}_2(x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$
6. $(\{v_e\}_{e \in E}, \pi) \leftarrow \mathsf{PCS.Open}(\mathsf{com}, \boldsymbol{P}, E)$
7. $\mathsf{sig} = (\mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e \in E})$

- Linear, private input
- Non-linear, private input
- Symmetric, private input
- Public input

[2] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

PIOP + PCS[2] signature

Base signature

Non-interactive zero-knowledge proof of knowledge protocol (uses Fiat-Shamir). Supports any OWF statement x with witness w. Ψ encodes the OWF instance verification.

Sign(pk : x, sk : w, msg) :

1. $\boldsymbol{P}(X) \in \mathbb{F}[X]^{\leq d} \leftarrow \mathsf{Encode}(w, \$)$
2. com $\leftarrow$ PCS.Commit($\boldsymbol{P}$)
3. $c \leftarrow \mathsf{Hash}_1(x, \mathsf{com})$
4. $Q \leftarrow \Psi(\boldsymbol{P}, x, c)$
5. $E \subset \mathbb{E} \leftarrow \mathsf{Hash}_2(x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$
6. $(\{v_e\}_{e \in E}, \pi) \leftarrow$ PCS.Open(com, $\boldsymbol{P}, E$)
7. sig $= (\mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e \in E})$

- Linear, private input
- Non-linear, private input
- Symmetric, private input
- Public input

[2] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

PIOP + PCS[2] signature

Base signature

Non-interactive zero-knowledge proof of knowledge protocol (uses Fiat-Shamir). Supports any OWF statement x with witness w. Ψ encodes the OWF instance verification.

$\mathsf{Sign}(\mathsf{pk} : x, \mathsf{sk} : w, \mathsf{msg})$:

1. $\boldsymbol{P}(X) \in \mathbb{F}[X]^{\leq d} \leftarrow \mathsf{Encode}(w, \$)$
2. $\mathsf{com} \leftarrow \mathsf{PCS.Commit}(\boldsymbol{P})$
3. $c \leftarrow \mathsf{Hash}_1(x, \mathsf{com})$
4. $Q \leftarrow \Psi(\boldsymbol{P}, x, c)$
5. $E \subset \mathbb{E} \leftarrow \mathsf{Hash}_2(x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$
6. $(\{v_e\}_{e \in E}, \pi) \leftarrow \mathsf{PCS.Open}(\mathsf{com}, \boldsymbol{P}, E)$
7. $\mathsf{sig} = (\mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e \in E})$

- Linear, private input
- Non-linear, private input
- Symmetric, private input
- Public input

[2] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

PIOP + PCS[2] signature

Base signature

Non-interactive zero-knowledge proof of knowledge protocol (uses Fiat-Shamir). Supports any OWF statement x with witness w. Ψ encodes the OWF instance verification.

$\mathsf{Sign}(\mathsf{pk} : x, \mathsf{sk} : w, \mathsf{msg})$:

1. $\boldsymbol{P}(X) \in \mathbb{F}[X]^{\leq d} \leftarrow \mathsf{Encode}(w, \$)$
2. $\mathsf{com} \leftarrow \mathsf{PCS.Commit}(\boldsymbol{P})$
3. $c \leftarrow \mathsf{Hash}_1(x, \mathsf{com})$
4. $Q \leftarrow \Psi(\boldsymbol{P}, x, c)$
5. $E \subset \mathbb{E} \leftarrow \mathsf{Hash}_2(x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$
6. $(\{v_e\}_{e \in E}, \pi) \leftarrow \mathsf{PCS.Open}(\mathsf{com}, \boldsymbol{P}, E)$
7. $\mathsf{sig} = (\mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e \in E})$

- Linear, private input
- Non-linear, private input
- Symmetric, private input
- Public input

[2] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

PIOP + PCS[2] signature

Base signature

Non-interactive zero-knowledge proof of knowledge protocol (uses Fiat-Shamir). Supports any OWF statement x with witness w. Ψ encodes the OWF instance verification.

$\mathsf{Sign}(\mathsf{pk} : x, \mathsf{sk} : w, \mathsf{msg})$:

1. $\boldsymbol{P}(X) \in \mathbb{F}[X]^{\leq d} \leftarrow \mathsf{Encode}(w, \$)$
2. $\mathsf{com} \leftarrow \mathsf{PCS.Commit}(\boldsymbol{P})$
3. $c \leftarrow \mathsf{Hash}_1(x, \mathsf{com})$
4. $Q \leftarrow \Psi(\boldsymbol{P}, x, c)$
5. $E \subset \mathbb{E} \leftarrow \mathsf{Hash}_2(x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$
6. $(\{v_e\}_{e \in E}, \pi) \leftarrow \mathsf{PCS.Open}(\mathsf{com}, \boldsymbol{P}, E)$
7. $\mathsf{sig} = (\mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e \in E})$

- Linear, private input
- Non-linear, private input
- Symmetric, private input
- Public input

[2] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

PIOP + PCS[2] signature

Base signature

Non-interactive zero-knowledge proof of knowledge protocol (uses Fiat-Shamir). Supports any OWF statement x with witness w. Ψ encodes the OWF instance verification.

$\mathsf{Sign}(\mathsf{pk} : x, \mathsf{sk} : w, \mathsf{msg})$:

1. $\boldsymbol{P}(X) \in \mathbb{F}[X]^{\leq d} \leftarrow \mathsf{Encode}(w, \$)$
2. $\mathsf{com} \leftarrow \mathsf{PCS.Commit}(\boldsymbol{P})$
3. $c \leftarrow \mathsf{Hash}_1(x, \mathsf{com})$
4. $Q \leftarrow \Psi(\boldsymbol{P}, x, c)$
5. $E \subset \mathbb{E} \leftarrow \mathsf{Hash}_2(x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$
6. $(\{v_e\}_{e \in E}, \pi) \leftarrow \mathsf{PCS.Open}(\mathsf{com}, \boldsymbol{P}, E)$
7. $\mathsf{sig} = (\mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e \in E})$

- Linear, private input
- Non-linear, private input
- Symmetric, private input
- Public input

Thresholdization

Only the PCS and Ψ are sensitive and nonlinear. In our instances, Ψ is typically quadratic.

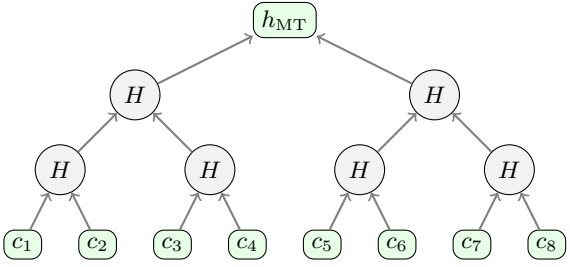
[2] PIOP: Polynomial Interactive Oracle Proof; PCS: Polynomial Commitment Scheme

Merkle-tree commitment and opening

PCS instantiation

We instantiate the PCS with DECS[3] and derive a threshold variant.

Figure: Merkle tree $|\mathbb{E}| = 8$ leaves



DECS.Commit($\boldsymbol{P}$):

1. $\boldsymbol{M}(X) \xleftarrow{\$} \mathbb{F}[X]^{\eta}$
2. $\forall e \in \mathbb{E},\ c_e = \mathsf{Hash}(\boldsymbol{P}(e), \boldsymbol{M}(e))$
3. $h_{\mathsf{MT}} := \mathsf{MerkleRoot}(\{c_e\}_{e \in \mathbb{E}})$
4. $\Gamma \in \mathbb{F}^{n \cdot \eta} := \mathsf{Hash}(h_{\mathsf{MT}})$
5. $\boldsymbol{R}(X) := \Gamma \cdot \boldsymbol{P}(X) + \boldsymbol{M}(X)$

Output com $:= (h_{\mathsf{MT}}, \boldsymbol{R})$.

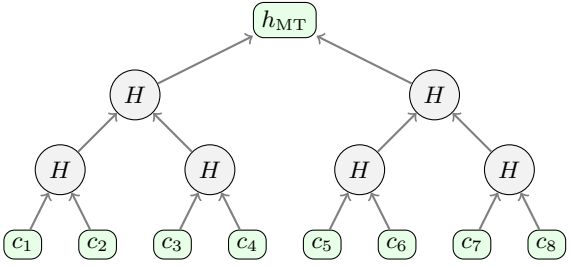
[3] Degree-enforcing commitment scheme [FR25]

Merkle-tree commitment and opening

PCS instantiation

We instantiate the PCS with DECS[3] and derive a threshold variant.

Figure: Merkle tree $|\mathbb{E}| = 8$ leaves



DECS.Commit($\boldsymbol{P}$):

1. $\boldsymbol{M}(X) \xleftarrow{\$} \mathbb{F}[X]^{\eta}$
2. $\forall e \in \mathbb{E},\ c_e = \mathsf{Hash}(\boldsymbol{P}(e), \boldsymbol{M}(e))$
3. $h_{\mathsf{MT}} := \mathsf{MerkleRoot}(\{c_e\}_{e \in \mathbb{E}})$
4. $\Gamma \in \mathbb{F}^{n \cdot \eta} := \mathsf{Hash}(h_{\mathsf{MT}})$
5. $\boldsymbol{R}(X) := \Gamma \cdot \boldsymbol{P}(X) + \boldsymbol{M}(X)$

Output com $:= (h_{\mathsf{MT}}, \boldsymbol{R})$.

[3] Degree-enforcing commitment scheme [FR25]

How to achieve efficient distributed hash-based commitments?

Hash commitments with masked inputs

Efficient distributed commitments that allow T parties to securely commit to a shared secret $[\![w]\!]_1, \ldots, [\![w]\!]_T$.

How to achieve efficient distributed hash-based commitments?

Hash commitments with masked inputs

Efficient distributed commitments that allow T parties to securely commit to a shared secret $[\![w]\!]_1, \ldots, [\![w]\!]_T$.

Commit($[\![w]\!]_1, \ldots, [\![w]\!]_T$) : For session sid, each $\mathcal{P}_i, i \in [T]$:

1. Sample and commit to a secret seed:
$$\rho^{(i)} \xleftarrow{\$} \{0,1\}^\lambda, \quad \mathsf{com}^{(i)} \leftarrow \mathsf{Hash}(\mathsf{sid}, i, \rho^{(i)})$$
2. Expand the seed into a mask:
$$\mathsf{mask}^{(i)} \leftarrow \mathsf{XOF}(\mathsf{sid}, i, \rho^{(i)})$$
3. Mask the share:
$$[\![\hat{w}]\!]_i \leftarrow [\![w]\!]_i + \mathsf{mask}^{(i)}$$
Broadcast $(\mathsf{com}^{(i)}, [\![\hat{w}]\!]_i)$. Store every broadcast.
4. Reconstruct the masked secret:
$$\hat{w} = \sum_{i\in[T]} [\![\hat{w}]\!]_i = \sum_{i\in[T]} [\![w]\!]_i + \mathsf{mask}^{(i)} = w + \sum_{i\in[T]} \mathsf{mask}^{(i)}$$
5. Hash to obtain the commitment:
$$\mathsf{com} := \mathsf{Hash}(\mathsf{sid}, \hat{w}, \{\mathsf{com}^{(i)}\}_{i\in[T]}) \ .$$

Output com.

How to achieve efficient distributed hash-based commitments?

Hash commitments with masked inputs

Efficient distributed commitments that allow T parties to securely commit to a shared secret $[\![w]\!]_1, \ldots, [\![w]\!]_T$.

Commit($[\![w]\!]_1, \ldots, [\![w]\!]_T$) : For session sid, each $\mathcal{P}_i, i \in [T]$:

1. Sample and commit to a secret seed:
$$\rho^{(i)} \xleftarrow{\$} \{0,1\}^\lambda, \quad \mathsf{com}^{(i)} \leftarrow \mathsf{Hash}(\mathsf{sid}, i, \rho^{(i)})$$
2. Expand the seed into a mask:
$$\mathsf{mask}^{(i)} \leftarrow \mathsf{XOF}(\mathsf{sid}, i, \rho^{(i)})$$
3. Mask the share:
$$[\![\hat{w}]\!]_i \leftarrow [\![w]\!]_i + \mathsf{mask}^{(i)}$$
Broadcast $(\mathsf{com}^{(i)}, [\![\hat{w}]\!]_i)$. Store every broadcast.
4. Reconstruct the masked secret:
$$\hat{w} = \sum_{i\in[T]} [\![\hat{w}]\!]_i = \sum_{i\in[T]} [\![w]\!]_i + \mathsf{mask}^{(i)} = w + \sum_{i\in[T]} \mathsf{mask}^{(i)}$$
5. Hash to obtain the commitment:
$$\mathsf{com} := \mathsf{Hash}(\mathsf{sid}, \hat{w}, \{\mathsf{com}^{(i)}\}_{i\in[T]}) \ .$$

Output com.

How to achieve efficient distributed hash-based commitments?

Hash commitments with masked inputs

Efficient distributed commitments that allow T parties to securely commit to a shared secret $[\![w]\!]_1, \ldots, [\![w]\!]_T$.

Commit($[\![w]\!]_1, \ldots, [\![w]\!]_T$) : For session sid, each $\mathcal{P}_i, i \in [T]$:

1. Sample and commit to a secret seed:
$$\rho^{(i)} \xleftarrow{\$} \{0,1\}^\lambda, \quad \mathsf{com}^{(i)} \leftarrow \mathsf{Hash}(\mathsf{sid}, i, \rho^{(i)})$$
2. Expand the seed into a mask:
$$\mathsf{mask}^{(i)} \leftarrow \mathsf{XOF}(\mathsf{sid}, i, \rho^{(i)})$$
3. Mask the share:
$$[\![\hat{w}]\!]_i \leftarrow [\![w]\!]_i + \mathsf{mask}^{(i)}$$
Broadcast $(\mathsf{com}^{(i)}, [\![\hat{w}]\!]_i)$. Store every broadcast.
4. Reconstruct the masked secret:
$$\hat{w} = \sum_{i \in [T]} [\![\hat{w}]\!]_i = \sum_{i \in [T]} [\![w]\!]_i + \mathsf{mask}^{(i)} = w + \sum_{i \in [T]} \mathsf{mask}^{(i)}$$
5. Hash to obtain the commitment:
$$\mathsf{com} := \mathsf{Hash}(\mathsf{sid}, \hat{w}, \{\mathsf{com}^{(i)}\}_{i \in [T]}) \ .$$

Output com.

How to achieve efficient distributed hash-based commitments?

Hash commitments with masked inputs

Efficient distributed commitments that allow T parties to securely commit to a shared secret $[\![w]\!]_1, \ldots, [\![w]\!]_T$.

Commit($[\![w]\!]_1, \ldots, [\![w]\!]_T$) : For session sid, each $\mathcal{P}_i, i \in [T]$:

1. Sample and commit to a secret seed:
$$\rho^{(i)} \xleftarrow{\$} \{0,1\}^\lambda, \quad \mathsf{com}^{(i)} \leftarrow \mathsf{Hash}(\mathsf{sid}, i, \rho^{(i)})$$
2. Expand the seed into a mask:
$$\mathsf{mask}^{(i)} \leftarrow \mathsf{XOF}(\mathsf{sid}, i, \rho^{(i)})$$
3. Mask the share:
$$[\![\hat{w}]\!]_i \leftarrow [\![w]\!]_i + \mathsf{mask}^{(i)}$$
Broadcast $(\mathsf{com}^{(i)}, [\![\hat{w}]\!]_i)$. Store every broadcast.
4. Reconstruct the masked secret:
$$\hat{w} = \sum_{i\in[T]} [\![\hat{w}]\!]_i = \sum_{i\in[T]} [\![w]\!]_i + \mathsf{mask}^{(i)} = w + \sum_{i\in[T]} \mathsf{mask}^{(i)}$$
5. Hash to obtain the commitment:
$$\mathsf{com} := \mathsf{Hash}(\mathsf{sid}, \hat{w}, \{\mathsf{com}^{(i)}\}_{i\in[T]}) \ .$$

Output com.

Open(com) : Each $\mathcal{P}_i, i \in [T]$:

1. Open $\mathsf{com}^{(i)}$ and broadcast $\rho^{(i)}$.
2. Recompute the masks for all $i \in [T]$:
$$\mathsf{mask}^{(i)} \leftarrow \mathsf{XOF}(\mathsf{sid}, i, \rho^{(i)})$$
3. Recover the secret from commitment broadcasts:
$$w = \hat{w} + \sum_{i\in[T]} \mathsf{mask}^{(i)}$$

Output $w, \{\rho_{(i)}\}_{i\in[T]}$.

How to achieve efficient distributed hash-based commitments?

Hash commitments with masked inputs

Efficient distributed commitments that allow T parties to securely commit to a shared secret $[\![w]\!]_1, \ldots, [\![w]\!]_T$.

Commit($[\![w]\!]_1, \ldots, [\![w]\!]_T$) : For session sid, each $\mathcal{P}_i, i \in [T]$:

1. Sample and commit to a secret seed:
$$\rho^{(i)} \xleftarrow{\$} \{0,1\}^\lambda, \quad \mathsf{com}^{(i)} \leftarrow \mathsf{Hash}(\mathsf{sid}, i, \rho^{(i)})$$
2. Expand the seed into a mask:
$$\mathsf{mask}^{(i)} \leftarrow \mathsf{XOF}(\mathsf{sid}, i, \rho^{(i)})$$
3. Mask the share:
$$[\![\hat{w}]\!]_i \leftarrow [\![w]\!]_i + \mathsf{mask}^{(i)}$$
Broadcast $(\mathsf{com}^{(i)}, [\![\hat{w}]\!]_i)$. Store every broadcast.
4. Reconstruct the masked secret:
$$\hat{w} = \sum_{i\in[T]} [\![\hat{w}]\!]_i = \sum_{i\in[T]} [\![w]\!]_i + \mathsf{mask}^{(i)} = w + \sum_{i\in[T]} \mathsf{mask}^{(i)}$$
5. Hash to obtain the commitment:
$$\mathsf{com} := \mathsf{Hash}(\mathsf{sid}, \hat{w}, \{\mathsf{com}^{(i)}\}_{i\in[T]}) \ .$$

Output com.

Open(com) : Each $\mathcal{P}_i, i \in [T]$:

1. Open $\mathsf{com}^{(i)}$ and broadcast $\rho^{(i)}$.
2. Recompute the masks for all $i \in [T]$:
$$\mathsf{mask}^{(i)} \leftarrow \mathsf{XOF}(\mathsf{sid}, i, \rho^{(i)})$$
3. Recover the secret from commitment broadcasts:
$$w = \hat{w} + \sum_{i\in[T]} \mathsf{mask}^{(i)}$$

Output $w, \{\rho_{(i)}\}_{i\in[T]}$.

Opening cost

Opening w costs $|w| + T\lambda$ bits.

How to achieve efficient distributed hash-based commitments?

Hash commitments with masked inputs

Efficient distributed commitments that allow T parties to securely commit to a shared secret $[\![w]\!]_1, \ldots, [\![w]\!]_T$.

Commit($[\![w]\!]_1, \ldots, [\![w]\!]_T$) : For session sid, each $\mathcal{P}_i, i \in [T]$:

1. Sample and commit to a secret seed:
$$\rho^{(i)} \xleftarrow{\$} \{0,1\}^\lambda, \quad \mathsf{com}^{(i)} \leftarrow \mathsf{Hash}(\mathsf{sid}, i, \rho^{(i)})$$
2. Expand the seed into a mask:
$$\mathsf{mask}^{(i)} \leftarrow \mathsf{XOF}(\mathsf{sid}, i, \rho^{(i)})$$
3. Mask the share:
$$[\![\hat{w}]\!]_i \leftarrow [\![w]\!]_i + \mathsf{mask}^{(i)}$$
Broadcast $(\mathsf{com}^{(i)}, [\![\hat{w}]\!]_i)$. Store every broadcast.
4. Reconstruct the masked secret:
$$\hat{w} = \sum_{i\in[T]} [\![\hat{w}]\!]_i = \sum_{i\in[T]} [\![w]\!]_i + \mathsf{mask}^{(i)} = w + \sum_{i\in[T]} \mathsf{mask}^{(i)}$$
5. Hash to obtain the commitment:
$$\mathsf{com} := \mathsf{Hash}(\mathsf{sid}, \hat{w}, \{\mathsf{com}^{(i)}\}_{i\in[T]}) \ .$$

Output com.

Open(com) : Each $\mathcal{P}_i, i \in [T]$:

1. Open $\mathsf{com}^{(i)}$ and broadcast $\rho^{(i)}$.
2. Recompute the masks for all $i \in [T]$:
$$\mathsf{mask}^{(i)} \leftarrow \mathsf{XOF}(\mathsf{sid}, i, \rho^{(i)})$$
3. Recover the secret from commitment broadcasts:
$$w = \hat{w} + \sum_{i\in[T]} \mathsf{mask}^{(i)}$$

Output $w, \{\rho_{(i)}\}_{i\in[T]}$.

Opening cost

Opening w costs $|w| + T\lambda$ bits.

Threshold DECS

Straightforward to efficiently thresholdize the DECS using this distributed hash commitment.

Complete threshold signature (simplified)

Threshold sig.

Using the threshold DECS and arithmetic black box (ABB) abstraction, we define a threshold MPCitH signature.

Sign(sid, pk : x, sk : $[\![w]\!]$, msg): For a session id sid, each party $\mathcal{P}_i$ runs:

1. $[\![\boldsymbol{P}]\!]_i(X) \in \mathbb{F}[X]^{\leq d} \leftarrow \mathsf{Encode}([\![w]\!]_i, \$)$
2. $\mathsf{com} \leftarrow \mathsf{TDECS.Commit}(\mathsf{sid}, [\![\boldsymbol{P}]\!])$
3. $c \leftarrow \mathsf{Hash}_1(\mathsf{sid}, x, \mathsf{com})$
4. $[\![\boldsymbol{Q}]\!] \leftarrow \mathsf{ABB.Eval}(\mathsf{sid}, \Psi(\cdot, x, c), [\![\boldsymbol{P}]\!])$
5. $\boldsymbol{Q} \leftarrow \mathsf{ABB.Reveal}(\mathsf{sid}, [\![\boldsymbol{Q}]\!])$
6. $E \leftarrow \mathsf{Hash}_2(\mathsf{sid}, x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$
7. $\left(\{v_e\}_{e\in E}, \pi\right) \leftarrow \mathsf{TDECS.Open}(\mathsf{sid}, \mathsf{com}, E)$
8. Output the signature:

$$\mathsf{sig} = (\mathsf{sid}, \mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e\in E})$$

Complete threshold signature (simplified)

Threshold sig.

Using the threshold DECS and arithmetic black box (ABB) abstraction, we define a threshold MPCitH signature.

Sign(sid, pk : x, sk : $[\![w]\!]$, msg): For a session id sid, each party $\mathcal{P}_i$ runs:

1. $[\![\boldsymbol{P}]\!]_i(X) \in \mathbb{F}[X]^{\leq d} \leftarrow \mathsf{Encode}([\![w]\!]_i, \$)$
2. $\mathsf{com} \leftarrow \mathsf{TDECS.Commit}(\mathsf{sid}, [\![\boldsymbol{P}]\!])$
3. $c \leftarrow \mathsf{Hash}_1(\mathsf{sid}, x, \mathsf{com})$
4. $[\![\boldsymbol{Q}]\!] \leftarrow \mathsf{ABB.Eval}(\mathsf{sid}, \Psi(\cdot, x, c), [\![\boldsymbol{P}]\!])$
5. $\boldsymbol{Q} \leftarrow \mathsf{ABB.Reveal}(\mathsf{sid}, [\![\boldsymbol{Q}]\!])$
6. $E \leftarrow \mathsf{Hash}_2(\mathsf{sid}, x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$
7. $(\{v_e\}_{e\in E}, \pi) \leftarrow \mathsf{TDECS.Open}(\mathsf{sid}, \mathsf{com}, E)$
8. Output the signature:

$$\mathsf{sig} = (\mathsf{sid}, \mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e\in E})$$

Complete threshold signature (simplified)

Threshold sig.

Using the threshold DECS and arithmetic black box (ABB) abstraction, we define a threshold MPCitH signature.

Sign(sid, pk : x, sk : $[\![w]\!]$, msg): For a session id sid, each party $\mathcal{P}_i$ runs:

1. $[\![\boldsymbol{P}]\!]_i(X) \in \mathbb{F}[X]^{\leq d} \leftarrow \mathsf{Encode}([\![w]\!]_i, \$)$
2. $\mathsf{com} \leftarrow \mathsf{TDECS.Commit}(\mathsf{sid}, [\![\boldsymbol{P}]\!])$
3. $c \leftarrow \mathsf{Hash}_1(\mathsf{sid}, x, \mathsf{com})$
4. $[\![\boldsymbol{Q}]\!] \leftarrow \mathsf{ABB.Eval}(\mathsf{sid}, \Psi(\cdot, x, c), [\![\boldsymbol{P}]\!])$
5. $\boldsymbol{Q} \leftarrow \mathsf{ABB.Reveal}(\mathsf{sid}, [\![\boldsymbol{Q}]\!])$
6. $E \leftarrow \mathsf{Hash}_2(\mathsf{sid}, x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$
7. $\left(\{v_e\}_{e\in E}, \pi\right) \leftarrow \mathsf{TDECS.Open}(\mathsf{sid}, \mathsf{com}, E)$
8. Output the signature:

$$\mathsf{sig} = (\mathsf{sid}, \mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e\in E})$$

TSitH instances

Table: For $\lambda = 128$ security parameter. Signature size $|\sigma|$ (in kB), number of multiplications $N_{\text{mult}}^{\text{ABB}}$ (over $\mathbb{K}$), presignature/signature communication per party (in kB).

	Merkle tree size	$\lvert\sigma\rvert$	$N_{\text{mult}}^{\text{ABB}}$	Presign com.	Sign com.
AES	2^{13}	$13.72 + 0.27 \cdot T$	1625	3565	11
MQ-$\mathbb{F}_{256}$	2^{13}	$7.06 + 0.19 \cdot T$	10752	7842	4
SD	2^{13}	$9.07 + 0.256 \cdot T$	47	2181	6

On communication

The presign communication is mainly composed of the TDECS commitment broadcast.

TSitH instances

Table: For $\lambda = 128$ security parameter. Signature size $|\sigma|$ (in kB), number of multiplications $N_{\text{mult}}^{\text{ABB}}$ (over $\mathbb{K}$), presignature/signature communication per party (in kB).

	Merkle tree size	$\lvert\sigma\rvert$	$N_{\text{mult}}^{\text{ABB}}$	Presign com.	Sign com.
AES	2^{13}	$13.72 + 0.27 \cdot T$	1625	3565	11
MQ-$\mathbb{F}_{256}$	2^{13}	$7.06 + 0.19 \cdot T$	10752	7842	4
SD	2^{13}	$9.07 + 0.256 \cdot T$	47	2181	6
AES	$\sim 2^8$	$22.54 + 0.64 \cdot T$	7750	483	24
MQ-$\mathbb{F}_{256}$	$\sim 2^8$	$7.09 + 0.32 \cdot T$	36000	447	3

On communication

The presign communication is mainly composed of the TDECS commitment broadcast.

TSitH instances

Table: For $\lambda = 128$ security parameter. Signature size $|\sigma|$ (in kB), number of multiplications $N_{\text{mult}}^{\text{ABB}}$ (over $\mathbb{K}$), presignature/signature communication per party (in kB).

	Merkle tree size	$\lvert\sigma\rvert$	$N_{\text{mult}}^{\text{ABB}}$	Presign com.	Sign com.
AES	2^{13}	$13.72 + 0.27 \cdot T$	1625	3565	11
MQ-$\mathbb{F}_{256}$	2^{13}	$7.06 + 0.19 \cdot T$	10752	7842	4
SD	2^{13}	$9.07 + 0.256 \cdot T$	47	2181	6
AES	$\sim 2^{8}$	$22.54 + 0.64 \cdot T$	7750	483	24
MQ-$\mathbb{F}_{256}$	$\sim 2^{8}$	$7.09 + 0.32 \cdot T$	36000	447	3
AES	$\sim 2^{16}$	$12.59 + 0.21 \cdot T$	1425	28838	8
MQ-$\mathbb{F}_{256}$	$\sim 2^{16}$	$7.48 + 0.16 \cdot T$	8064	24281	4

On communication

The presign communication is mainly composed of the TDECS commitment broadcast.

Conclusion

Submission contents

- From [FRVWJ26], algorithmic descriptions and security analysis of the $\mathcal{MQ}^4$ and AES-based threshold signatures,
- Some improvements to the OWF arithmetizations (for AES, SD[5]),
- A C implementation + benchmarks (execution times, memory footprint)
- Estimations of the costs of the external building blocks (e.g. preprocessing phase for generating authenticated Beaver triples),

[4] Multivariate Quadratic
[5] Syndrome Decoding

Conclusion

Submission contents

- From [FRVWJ26], algorithmic descriptions and security analysis of the $\mathcal{MQ}^4$ and AES-based threshold signatures,
- Some improvements to the OWF arithmetizations (for AES, SD[5]),
- A C implementation + benchmarks (execution times, memory footprint)
- Estimations of the costs of the external building blocks (e.g. preprocessing phase for generating authenticated Beaver triples),

Signature upsides

- Generic: can use any OWF,
- Can use unstructured problems,
- Using AES: threshold sig. with Minicrypt assumptions.

[4] Multivariate Quadratic
[5] Syndrome Decoding

References I


Eliana Carozza and Geoffroy Couteau.
On threshold signatures from MPC-in-the-head.
Cryptology ePrint Archive, Report 2024/1897, 2024.


Jack Doerner, Yashvanth Kondi, and Leah Namisa Rosenbloom.
Sometimes you can't distribute random-oracle-based proofs.
In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part V*, volume 14924 of *LNCS*, pages 323–358. Springer, Cham, August 2024.


Thibauld Feneuil and Matthieu Rivain.
Threshold computation in the head: Improved framework for post-quantum signatures and zero-knowledge arguments.
Journal of Cryptology, 38(3):28, July 2025.


Thibauld Feneuil, Matthieu Rivain, Damien Vergnaud, and Auguste Warmé-Janville.
Threshold signatures in the head.
Cryptology ePrint Archive, Paper 2026/1125, 2026.