Threshold Signatures in the Head

Thibauld Feneuil[1], Matthieu Rivain[1], Damien Vergnaud[2], and Auguste Warmé-Janville[1,2]

[1] CryptoExperts, Paris, France
thibauld.feneuil@cryptoexperts.com
matthieu.rivain@cryptoexperts.com
auguste.warme-janville@cryptoexperts.com

[2] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
damien.vergnaud@lip6.fr

Abstract. Threshold cryptography distributes trust among multiple parties by enabling joint cryptographic operations without reconstructing secret keys. While post-quantum signature schemes based on the MPC-in-the-Head (MPCitH) paradigm are highly generic, recent impossibility results show that their thresholdization either incurs prohibitive distributed symmetric computations or leads to signature sizes growing with the number of signers. Achieving practical tradeoffs in this setting remains challenging. In this paper, we propose a generic framework for threshold MPCitH signatures based on Merkle-tree commitments. Our approach adapts the PIOP+PCS paradigm to the distributed setting by introducing and instantiating the notion of threshold polynomial commitment schemes (TPCS). We present a generic compiler combining a PIOP, a TPCS, and an arithmetic black box into a threshold signature scheme, and prove its unforgeability from the security of its components. We further provide a concrete Merkle-tree-based TPCS achieving moderate signature-size overhead, as low as 200 bytes per signer at the 128-bit security level. This is to be compared with an overhead of roughly 2 kB per signer for the previously suggested approach to thresholdize MPC-in-the-Head based on GGM trees. By compiling this TPCS with a standard PIOP, we obtain a generic threshold signature scheme from any hard problem or one-way function, which we showcase MQ-based and AES-based instantiations.

Keywords: Threshold Signatures · MPC-in-the-Head · Post-Quantum Cryptography · Polynomial Commitments · Hash-based Proof Systems

1 Introduction

Multi-party computation (MPC) is the cryptography field studying distributed protocols where a group of parties computes a joint function on private inputs. Threshold cryptography is a subfield of MPC where the goal is to design distributed protocols for specific cryptographic primitives, such as encryption or digital signatures. Threshold cryptography – and the design of threshold signature schemes in particular – is a very active research area, as it allows to protect cryptographic keys without relying on costly hardware, or to distribute trust in applications such as blockchains.

While a lot of pre-quantum threshold signature schemes have been proposed in the literature, notably based on the discrete logarithm, the literature is much sparser when it comes to post-quantum threshold signatures, although several candidates based on lattices have recently been proposed [GKS24, DKM+24, EKT24, KRT24, CATZ24, BKL+25, ZT25, PN25, PKN+25] which often mimic their discrete logarithm analogs. However, there are much fewer post-quantum threshold signatures based on other assumptions. In [CS19], the authors study the thresholdization of signature candidates to the first NIST PQC process and show that UOV-like schemes are good candidates for this purpose. This work was extended in [CEN25], notably covering the MAYO signature scheme. A few other candidates have been proposed based on isogenies [DM20, CS20], code-equivalence group action [BBMP24, BBD+25] and stateful hash-based signatures [KLL25].

On the other hand, MPC-in-the-Head (MPCitH) is a generic framework to build post-quantum signatures for any hardness assumption using symmetric cryptography primitives and which has recently received a lot of

attention. Unfortunately, a recent impossibility result [DKR24] has shown that threshold MPCitH signatures must either rely on the thresholdization of the underlying symmetric primitives (inducing huge computation and communication overheads) or have a signature size that increases with the number of signing parties (which is not desirable in practice). Nevertheless, in the latter case, one might hope for design with a low impact of the number of signers on the signature size, thus reaching interesting tradeoffs in practice. This is the direction we explore in this work.

A first stone in this direction has been laid by Carozza and Couteau in [CC24]. The authors bypass the need for threshold computation of the GGM trees (involving the costly use of symmetric primitives) by letting each active party computes its own tree, and including the associated opening paths in the signature. This results in a signature where only the symmetric part grows with the number of parties, with an overhead of around 2 kB per party for a 128-bit security level, which makes this tweak quickly prohibitive in practice as the number of parties grows. An alternative to GGM trees in MPCitH signatures is to rely on Merkle trees as put forward in the Threshold-Computation-in-the-Head (TCitH) framework [FR23, FR25b]. Although this approach does not achieve signatures as short as with GGM trees, it reveals more convenient for contexts requiring computation on secret-shared data.

Our Contributions. In this work, we present a generic framework to build threshold signatures in an MPC-in-the-Head fashion, with Merkle trees as the underlying commitment scheme. For this purpose, we translate the standard PIOP+PCS[3] paradigm to the distributed setting. Specifically, our contributions are fourfold:

1. We introduce the notion of *threshold polynomial commitment scheme* (TPCS), a scheme allowing parties to jointly commit to a polynomial secret-shared between them, and to jointly open evaluations of the committed polynomial (Section 3).
2. We formalize the notion of *hybrid sigma PIOP* (HS-PIOP), which encompasses the PIOP protocols used in modern MPCitH constructions, and we present a generic construction of threshold signature scheme compiling an HS-PIOP with a TPCS, further relying on an *arithmetic black-box* (ABB) to distribute the arithmetic part of the signature (Section 4). We prove the unforgeability of this threshold signature scheme based on the HS-PIOP and TPCS security properties (Section 5).
3. We provide a concrete construction of hash-based TPCS. We first introduce a technique to efficiently translate any Merkle-tree-based PCS in a distributed setting, and then apply this technique to obtain a concrete TPCS based on the degree-enforcing commitment scheme (DECS) from [FR25b, FR25a] (Section 6). As with the multiple GGM tree approach, our Merkle-tree-based TPCS incurs an opening proof size that grows with the number of parties, but this growth is much more moderate and can be as low as 200 bytes per party for a 128-bit security level.
4. We instantiate our framework with the threshold DECS and the LPPC PIOP from [FR25b] (a generalization of Ligero [AHIV17, AHIV23]). This provides a generic threshold signature scheme from any hard problem or one-way function (arithmetized as an LPPC instance). We showcase concrete instances based on the $\mathcal{MQ}$ problem and the AES block cipher. We provide parameters for different instances that allow for tradeoffs between signature size and MPC communication. For instance, for a number of $T = 8$ active signers, we obtain a $\mathcal{MQ}$ signature of 8.6 kB and an AES signature of size 15 kB.

Links with masking-friendly signatures. Masked signatures in the context of side-channel attacks and threshold signatures present similarities: both reduce to computing over shared data. A similar approach to the one in [CC24] has been taken in [FRWJ25] to design masking-friendly MPCitH signatures (*i.e.* signatures that are simple and efficient to protect with masking, the typical countermeasure against side-channel attacks), but it also suffers from a linear size overhead that grows with the number of secret shares. In [FRWJ25], the authors also explore designing masking-friendly signatures from TCitH with Merkle trees, and conclude that this variant of TCitH is much more amenable in this context than its GGM-tree counterpart. Although these masked signatures *are not* threshold signatures, they are a good starting point for exploring the design of such signatures. The key difference between masking and MPC lies in the security model. While the masked

[3] PIOP stands for Polynomial Interactive Oracle Proof and PCS stands for Polynomial Commitment Scheme.

circuit is guaranteed to be executed correctly, each participant in an MPC protocol is prone to cheating. So while a masking-friendly signature can be a good starting point for designing a threshold signature, the latter is usually more tricky. This can be illustrated by the Raccoon signature scheme, which was initially proposed as a masking-friendly signature [dPKPR24] and then later adapted into a threshold signature [DKM+24].

2 Preliminaries

In this work, we denote $\boldsymbol{P} = (P_1, \ldots, P_n) \in (\mathbb{F}[X])^n$ a vector polynomial. We use the notation $(\mathbb{F}[X]^{(\leq d)})^n$ to denote the set of vector polynomials of degree at most d and of dimension n. The evaluations of $\boldsymbol{P}$ over a set E are given by $\{\boldsymbol{P}(e)\}_{e\in E} = \{(P_1(e), \ldots, P_n(e))\}_{e\in E}$. Given such a set of evaluations we define the interpolation function $\boldsymbol{P} = \mathsf{Interpol}(E, \{\boldsymbol{P}(e)\}_{e\in E})$ that returns the lowest-degree vector polynomial matching the input evaluations.

Secret Sharing. A T-out-of-N secret sharing scheme (SSS) is a pair of algorithms $(\mathsf{Share}, \mathsf{Rec})$. Running $[\![w]\!] \leftarrow \mathsf{Share}_{(T,N)}(w)$ produces a random sharing $[\![w]\!] = ([\![w]\!]_1, \ldots, [\![w]\!]_N)$ of w such that any set of $T-1$ shares is random and independent of w while any set of T shares allow for the full recovery of w by $w = \mathsf{Rec}_{\mathcal{T}}([\![w]\!]_{\mathcal{T}})$ for $\mathcal{T} \subseteq [N]$ of size T and where $[\![w]\!]_{\mathcal{T}} = ([\![w]\!]_i)_{i\in\mathcal{T}}$.

The *additive secret sharing scheme* is a (T,T)-SSS for which the Share algorithm picks $T-1$ random shares $[\![w]\!]_1, \ldots, [\![w]\!]_{T-1}$ and defines the last share as $[\![w]\!]_T = w - (\sum_{i=1}^{T-1} [\![w]\!]_i)$. We then simply have $\mathsf{Rec}([\![w]\!]) = \sum_{i=1}^{T} [\![w]\!]_i$.

The *Shamir's secret sharing scheme* [Sha79] is a (T,N)-SSS for any $T \leq N \leq |\mathbb{F}|$. Given N public evaluation points $\mathbb{E} = \{e_1, \ldots, e_N\} \subseteq \mathbb{F} \setminus \{0\}$, the Share algorithm is defined as $[\![w]\!] := (\boldsymbol{P}(e_1), \ldots, \boldsymbol{P}(e_N)) \leftarrow \mathsf{Share}_{(T,N)}(w)$ with $\boldsymbol{P}(X)$ a random polynomial from $(\mathbb{F}[X])^n$ of degree $\leq T-1$ such that $P(0) = w$. The reconstruct algorithm $\mathsf{Rec}_{\mathcal{T}}([\![w]\!]_{\mathcal{T}})$ recovers $\boldsymbol{P}(X)$ by interpolation from the shares $[\![w]\!]_{\mathcal{T}}$ for any set $\mathcal{T} \subseteq [N]$ of size T, and outputs $w = \boldsymbol{P}(0)$.

Threshold Signatures. A (T,N)-threshold signature scheme TSIG is defined by a 4-tuple of algorithms and interactive protocols $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$. The setup and key generation protocols distribute secret shares among N parties such that any subset of size at least T can interactively generate a signature valid under a global verification key. We refer to Appendix A for the formal syntax and correctness property.

Security. We assume a static corruption of $T-1$ parties and rely on two standard notions [BCK+22, TZ23] formalized in Appendix A:

- **TS-EUF-KOA (Key-Only Attack):** This ensures unforgeability against a passive adversary who holds $T-1$ key shares but cannot interact with honest parties nor get message/signature pairs.
- **TS-OMUF-CMA (One-More Unforgeability):** This ensures unforgeability against an active adversary capable of interleaving multiple signing sessions. The adversary wins if they can produce more valid signatures than the number of successfully completed sessions with honest parties.

3 Threshold Polynomial Commitment Scheme

In this section, we formally introduce the notion of *threshold polynomial commitment scheme* (threshold PCS, or TPCS). In such a scheme, several parties jointly compute a polynomial commitment for which they can later jointly open evaluations, namely, jointly compute the evaluations along with their opening proofs. A public verification algorithm allows checking the consistency of the opened evaluations and proofs with the commitment.

Definition 1 (Threshold Polynomial Commitment Scheme). *Let $T, n, d \in \mathbb{N}$. Let $\mathsf{H}(\cdot)$ be a hash function modeled as a random oracle. Let $\mathbb{F}$ be a finite field and $\mathbb{E} \subseteq \mathbb{F}$ (the evaluation domain). A* threshold polynomial commitment scheme *(threshold PCS) with parameters $\mathbb{F}, \mathbb{E}, n, d, T$ and random oracle $\mathsf{H}(\cdot)$ is composed of two T-party protocols $\mathsf{Commit}^{\mathsf{H}(\cdot)}$ and $\mathsf{Open}^{\mathsf{H}(\cdot)}$ and one PPT algorithm $\mathsf{Verif}^{\mathsf{H}(\cdot)}$ defined as follows:*

- $\mathsf{Commit}^{\mathsf{H}(\cdot)} : (\mathsf{sid}, [\![\boldsymbol{P}]\!]) \to (\mathsf{com}, \{\mathsf{key}_i\}_{i\in[T]})$. *In the $\mathsf{Commit}^{\mathsf{H}(\cdot)}$ protocol, T parties initially hold a common session identifier sid and a sharing $[\![\boldsymbol{P}]\!]$ of a vector polynomial $\boldsymbol{P} \in (\mathbb{F}[X]^{(\leq d)})^n$. The protocol outputs a commitment com of the polynomial $\boldsymbol{P}$ and each party further outputs a private opening key key_i.*
- $\mathsf{Open}^{\mathsf{H}(\cdot)}(\mathsf{sid}, \mathsf{com}, E, \{\mathsf{key}_i\}_{i\in[T]}) \to (\{\boldsymbol{v}_e\}_{e\in E}, \pi)$. *In the $\mathsf{Open}^{\mathsf{H}(\cdot)}$ protocol, T parties initially hold a common session identifier sid, a commitment com, an evaluation set $E \subseteq \mathbb{E}$ and their private opening keys $\{\mathsf{key}_i\}_{i\in[T]}$. The protocol returns a set of evaluation vectors $\{\boldsymbol{v}_e\}_{e\in E}$, with $\boldsymbol{v}_e \in \mathbb{F}^n$, and an evaluation proof π.*
- $\mathsf{Verif}^{\mathsf{H}(\cdot)} : (\mathsf{sid}, \mathsf{com}, E, \{\boldsymbol{v}_e\}_{e\in E}, \pi) \to$ Accept/Reject. *The $\mathsf{Verif}^{\mathsf{H}(\cdot)}$ algorithm takes as input a session identifier sid, a commitment com, an evaluation set $E \subseteq \mathbb{E}$, a set of evaluation vectors $\{\boldsymbol{v}_e\}_{e\in E}$ and an evaluation proof π, and returns* Accept *or* Reject.

We now introduce security definitions for a TPCS. In a nutshell, we say that a TPCS is extractable binding if, given a commitment com, there exists a single polynomial that can be opened to com, and there is an efficient algorithm allowing to extract the committed polynomial using only com and the random oracle queries.

Definition 2 (Extractable Binding). *A threshold PCS with parameters $\mathbb{F}, \mathbb{E}$, n, d, T is* (t, ε)-extractable binding *if there exists a PPT algorithm Ext which for any adversary $\mathcal{A}$ running in time t, verifies:*

$$\Pr\left[\begin{array}{l|l} \mathsf{Verif}^{\mathsf{H}(\cdot)}(\mathsf{sid}, \mathsf{com}, E, \{\boldsymbol{v}_e\}_{e\in E}, \pi) & (\mathsf{sid}, \mathsf{com}, E, \{\boldsymbol{v}_e\}_{e\in E}, \pi) \leftarrow \mathcal{A}^{\mathsf{H}(\cdot)}() \\ \quad = \textsc{Accept} \ \wedge \ \boldsymbol{P}|_E \neq \{\boldsymbol{v}_e\}_{e\in E} & \boldsymbol{P} \leftarrow \mathsf{Ext}(\mathcal{Q}_{\mathsf{H}}, \mathsf{sid}, \mathsf{com}) \end{array}\right] \ \leq \ \varepsilon \ ,$$

where $\boldsymbol{P}$ returned by Ext lies in $(\mathbb{F}[X]^{(\leq d)})^n$ and $\mathcal{Q}_{\mathsf{H}}$ denotes the set of query-response pairs made by $\mathcal{A}$ to $\mathsf{H}(\cdot)$.

We say that a TPCS satisfies the zero-knowledge property if it is possible to efficiently produce a transcript of the protocol with no knowledge of the secret inputs; and if it is impossible (except with some negligible probability) to distinguish between the simulated transcript and a real execution of the protocol.

Definition 3 (Zero Knowledge). *A threshold PCS with parameters $(\mathbb{F}, \mathbb{E}, n, d, T)$ is* (t, ε)-zero knowledge *if there exists a PPT algorithm Sim (the simulator) such that for any stateful adversary $\mathcal{A}$ running in time t, we have:*

$$\Pr\left[\hat{b} = b \ \middle| \ (b, \hat{b}) \leftarrow \mathit{ZKExp}^{\mathcal{A},\mathsf{Sim}}\right] \ \leq \ \frac{1}{2} + \varepsilon,$$

where the experiment $\mathit{ZKExp}^{\mathcal{A},\mathsf{Sim}}$ is the experiment described in Figure 1, with Sim replying to $\mathcal{A}$'s random oracle queries (in a uniformly random way).

In a nutshell the experiment in Figure 1 challenges the adversary to distinguish between a real execution of the threshold PCS protocol and an execution generated by the zero-knowledge simulator. The adversary wins if it can distinguish the two executions with non-negligible probability. Conversely, we say the TPCS is zero-knowledge if the adversary's advantage over random guessing is negligible (in practice, we require $\varepsilon \leq 2^{-\lambda}$).

In this zero-knowledge notion, we consider static corruptions, meaning that the adversary selects the corrupted parties at the beginning of the experiment. Moreover, the adversary controls the evaluation points that are opened. During the commitment phase, the simulator receives no input other than the session identifier, whereas the adversary obtains the inputs of all corrupted parties. During the opening phase, the simulator is given the shares of the evaluations for which it is expected to produce opening proofs.

1. Call the adversary: $(\boldsymbol{P}, T, \mathcal{C}) \leftarrow \mathcal{A}^{\mathsf{H}()}()$ where $\boldsymbol{P} \in (\mathbb{F}[X]^{(\leq d)})^n$, $T \in \mathbb{N}$ and $\mathcal{C} \subseteq [1, T]$ (*the set identifying corrupted parties*) is of cardinality $|\mathcal{C}| \leq T - 1$.
2. Let $\mathcal{H} = [T] \setminus \mathcal{C}$ (*the set identifying honest parties*).
3. Sample a random sharing $[\![\boldsymbol{P}]\!] \leftarrow \mathsf{Share}(\boldsymbol{P})$.
4. Sample a random bit $b \leftarrow \{0, 1\}$.
5. If $b = 0$ (*real experiment*) then:
   - Run the Commit protocol with session identifier sid, parties in $\mathcal{H}$ receiving their shares from $[\![\boldsymbol{P}]\!]$ as input and behaving honestly and parties in $\mathcal{C}$ controlled by $\mathcal{A}^{\mathsf{H}()}$, where $\mathcal{A}^{\mathsf{H}()}$ receives $\{[\![\boldsymbol{P}]\!]_i\}_{i \in \mathcal{C}}$ as input. If the protocol does not abort, it outputs com and each honest party further outputs a private opening key key_i.

   If $b = 1$ (*simulated experiment*) then:
   - Run the Commit protocol with session identifier sid, parties in $\mathcal{H}$ simulated by Sim and parties in $\mathcal{C}$ controlled by $\mathcal{A}^{\mathsf{H}()}$, where $\mathcal{A}^{\mathsf{H}()}$ receives $\{[\![\boldsymbol{P}]\!]_i\}_{i \in \mathcal{C}}$ as input. If the protocol does not abort, it outputs com.
6. Call the adversary: $E \leftarrow \mathcal{A}^{\mathsf{H}()}()$, where $E \subseteq \mathbb{E}$.
7. If $b = 0$ (*real experiment*) then:
   - Run the Open protocol with session identifier sid, commitment com, evaluation set E, parties in $\mathcal{H}$ using their private opening keys $\{\mathsf{key}_i\}_{i \in \mathcal{H}}$ and behaving honestly, and parties in $\mathcal{C}$ controlled by $\mathcal{A}^{\mathsf{H}()}$. If the protocol does not abort, it outputs a set of evaluation vectors $\{\boldsymbol{v}_e\}_{e \in E}$ and an evaluation proof π.

   If $b = 1$ (*simulated experiment*) then:
   - Compute $[\![\boldsymbol{v}_e]\!]_i = [\![\boldsymbol{P}]\!]_i(e)$ for each $e \in E$ and $i \in \mathcal{H}$.
   - Run the Open protocol with session identifier sid, commitment com, evaluation set E, parties in $\mathcal{H}$ simulated by Sim on input E and $\{[\![\boldsymbol{v}_e]\!]_i\}_{e \in E, i \in \mathcal{H}}$ and parties in $\mathcal{C}$ controlled by $\mathcal{A}^{\mathsf{H}()}$. If the protocol does not abort, it outputs a set of evaluation vectors $\{\boldsymbol{v}_e\}_{e \in E}$ and an evaluation proof π.
8. Call the adversary: $\hat{b} \leftarrow \mathcal{A}^{\mathsf{H}()}()$, where $\hat{b} \in \{0, 1\}$.
9. Return $(b, \hat{b})$.

Fig. 1. Zero-knowledge experiment for threshold PCS.

4 Threshold Signatures from Threshold PCS and Polynomial IOP

4.1 Polynomial IOP Model

An interactive oracle proof (IOP) is a special type of proof systems. While the prover's message in basic proof systems are field element strings that the verifier can entirely read, the prover's messages in an IOP are *oracles* to field element strings that the verifier may query at a prescribed number of positions. In such proofs, the verifier does not have access to the responses for all the possible oracle queries, but only to a small set of them. A *polynomial* IOP (PIOP) is an IOP where the prover's oracles messages encode polynomials of *known* degree. Querying an oracle consists in asking the evaluation of the encoded polynomials in some points. In some (P)IOP constructions, it may be useful to allow the prover to send non-oracles messages. This is the case for PIOPs involved in signature schemes build from the MPC-in-the-Head paradigm (see, e.g.,

[FR25b]). In the present work, we consider such hybrid PIOP where the first prover's message is a polynomial oracle and the next messages are non-oracle ones. We further assume that the PIOP is three-round and has a single non-oracle message from the prover. We formalize such a protocol in the following definition.

Definition 4 (Hybrid Sigma PIOP). *Let $\mathbb{F}$ a finite field. A hybrid sigma polynomial interactive oracle proof (HS-PIOP) for a relation $\mathcal{R} \subseteq \mathbb{F}^{|x|} \times \mathbb{F}^{|w|}$ is an interactive protocol between a prover $\mathcal{P}$ on input (x, w) and a verifier $\mathcal{V}$ on input x, where $(x, w) \in \mathcal{R}$. It is defined with respect to an evaluation domain $\mathbb{E} \subseteq \mathbb{F}$, a query complexity parameter $\ell \in \mathbb{N}$ and three functions Φ, Ψ, Υ, where Φ and Υ are $\mathbb{F}$-linear functions and Ψ is an $\mathbb{F}$-algebraic function (i.e. can be expressed as an algebraic circuit over $\mathbb{F}$). The interactions between $\mathcal{P}$ and $\mathcal{V}$ are as follows:*

- *$\mathcal{P}$ samples some randomness r, encodes the witness as a vector polynomial $\boldsymbol{P} := \Phi(w, r)$. We require Φ to be invertible, i.e. one can extract (w, r) from $\boldsymbol{P}$. $\mathcal{P}$ then sends a polynomial oracle $[\boldsymbol{P}]$ to $\mathcal{V}$;*
- *$\mathcal{V}$ samples and sends randomness c to $\mathcal{P}$;*
- *$\mathcal{P}$ answers a vector polynomial $\boldsymbol{Q} := \Psi(\boldsymbol{P}, x, c)$.*

At the end of the interaction, the verifier $\mathcal{V}$ samples a random set of points $E \subseteq \mathbb{E}$ of size $|E| = \ell$, queries the polynomial oracle $[\boldsymbol{P}]$ at these points, and receives the evaluations $v_e := \boldsymbol{P}(e)$ for all $e \in E$. The verifier accepts if and only if the following conditions hold:

- ***Linear predicate on $\boldsymbol{Q}$:** The vector polynomial $\boldsymbol{Q}$ satisfies the linear constraint $\Upsilon(\boldsymbol{Q}) = \boldsymbol{0}$, where $\boldsymbol{0}$ denotes the zero vector of appropriate dimension.*
- ***Consistency of evaluations:** For all $e \in E$, the oracle response is consistent with $\boldsymbol{Q}$, i.e., $\boldsymbol{Q}(e) = \Psi(v_e, x, c)$.*

An HS-PIOP is said to be *complete* if, whenever the prover follows the protocol honestly with a valid witness w, the verifier always accepts. It is *sound* if, for any (possibly malicious) prover that does not possess a valid witness w, the probability that the verifier accepts is negligible. Finally, it is *honest-verifier zero-knowledge* if there exists a simulator that, given only the public statement x, can efficiently generate a transcript $(c, \boldsymbol{Q}, E, \{v_e\}_{e \in E})$ that is indistinguishable from one produced by an honest execution of the protocol, without access to the witness w. We formally define the two latter properties hereafter.

Definition 5 (Round-by-Round Soundness of an HS-PIOP). *An HS-PIOP for a relation $\mathcal{R}$ is said* $(\varepsilon_1, \varepsilon_2)$-round-by-round sound *if for any $(x, w) \notin \mathcal{R}$, any randomness r, and any (possibly malicious) prover $\mathcal{A}$ that, on input x, sends the first oracle message $[\boldsymbol{P}]$ with $\boldsymbol{P} = \Phi(w, r)$, the following holds:*

1. *The probability over the verifier's random challenge c that the linear constraint $\Upsilon(\Psi(\boldsymbol{P}, x, c)) = \boldsymbol{0}$ is satisfied is at most ε_1.*
2. *For any (non-oracle) message $\boldsymbol{Q} \neq \Psi(\boldsymbol{P}, x, c)$ from the prover, the probability over the verifier's random choice of E that $\boldsymbol{Q}(e) = \Psi(\boldsymbol{P}(e), x, c)$ for all $e \in E$ is at most ε_2.*

Consequently, the overall probability that the verifier accepts is at most $\varepsilon_1 + \varepsilon_2$.

Definition 6 (Honest-Verifier Zero-Knowledge of an HS-PIOP). *An HS-PIOP for a relation $\mathcal{R}$ is* honest-verifier zero-knowledge (HVZK) *if accepting transcripts $(c, \boldsymbol{Q}, E, \{v_e\}_{e \in E})$ are uniformly distributed over the transcript space $\mathbb{F}^{|c|} \times (\mathbb{F}[X]^{(\leq d_{\boldsymbol{Q}})})^{|\boldsymbol{Q}|} \times \binom{\mathbb{E}}{\ell} \times \mathbb{F}^{\ell}$, conditioned to $\Upsilon(\boldsymbol{Q}) = \boldsymbol{0}$ and $\boldsymbol{Q}(e) = \Psi(v_e, x, c)$ for all $e \in E$, where $d_{\boldsymbol{Q}}$ denotes the maximum degree of the polynomials in $\boldsymbol{Q}$ and $|\boldsymbol{Q}|$ denotes its dimension.*

Random part of $\boldsymbol{Q}$. To lead to a valid transcript, $\boldsymbol{Q}$ should be in $\ker(\Upsilon)$, the kernel of the linear constraint Υ. There exists a linear bijective "compression" function $\mathsf{Compr}_{\ker(\Upsilon)} : \ker(\Upsilon) \to \mathbb{F}^{(d_{\boldsymbol{Q}}+1)\cdot|\boldsymbol{Q}|-|\Upsilon|}$ mapping an element of the kernel into a vector of $(d_{\boldsymbol{Q}} + 1) \cdot |\boldsymbol{Q}| - |\Upsilon|$ field elements, where $|\Upsilon|$ denotes the dimension of the image of Υ. We denote $\bar{\boldsymbol{Q}} := \mathsf{Compr}_{\ker(\Upsilon)}(\boldsymbol{Q})$ the *random part* of a $\boldsymbol{Q} \in \ker(\Upsilon)$, and we further denote $\bar{\Psi} := \mathsf{Compr}_{\ker(\Upsilon)} \circ \Psi$. From this definition, the HVZK property can be restated as follows: an HS-PIOP is HVZK if for any fixed x, w, c and E, the distribution of $\left(\bar{\boldsymbol{Q}}, \{v_e\}_{e \in E}\right)$ is uniform over $\mathbb{F}^{(d_{\boldsymbol{Q}}+1)\cdot|\boldsymbol{Q}|-|\Upsilon|} \times \mathbb{F}^{\ell}$.

Masking HS-PIOP. We shall consider HS-PIOP with slightly more structure in the following. In particular, we consider such protocols where, by definition, the polynomial $\boldsymbol{Q}$ is additively masked by a random polynomial part of $\boldsymbol{P}$ derived from the prover's randomness r. This is formally introduced in the next definition.

Definition 7 (Masking HS-PIOP). *An HS-PIOP for a relation $\mathcal{R}$ is said* masking *if there exists a decomposition of the vector polynomial $\boldsymbol{P}$ into two parts $\boldsymbol{P} = (\boldsymbol{P}^{(\mathsf{wit})}, \boldsymbol{P}^{(\mathsf{mask})})$ such that the vector polynomial $\boldsymbol{Q}$ can be expressed as*

$$\boldsymbol{Q} = \Psi'(\boldsymbol{P}^{(\mathsf{wit})}, x, c) + \boldsymbol{P}^{(\mathsf{mask})} \ ,$$

where Ψ' is an algebraic function. Moreover, if $\boldsymbol{P} = (\boldsymbol{P}^{(\mathsf{wit})}, \boldsymbol{P}^{(\mathsf{mask})})$ is genuinely derived from a correct witness w and uniform randomness r, then we have $\Upsilon(\Psi'(\boldsymbol{P}^{(\mathsf{wit})}, x, c)) = \Upsilon(\boldsymbol{P}^{(\mathsf{mask})}) = \boldsymbol{0}$ for any challenge c and the distribution of $\boldsymbol{P}^{(\mathsf{mask})}$ is uniform over the set of vector polynomials satisfying $\Upsilon(\boldsymbol{P}^{(\mathsf{mask})}) = \boldsymbol{0}$.

Remark 1. In a masking HS-PIOP, for any fixed x, w, c, and E, the uniform randomness of $(\bar{\boldsymbol{Q}}, \{v_e\}_{e\in E})$ is entirely induced by the prover randomness r. More precisely, and without loss of generality, we can assume that $r \in \mathbb{F}^{(d_{\boldsymbol{Q}}+1)\cdot|\boldsymbol{Q}|-|\Upsilon|+\ell\cdot|\boldsymbol{P}^{(\mathsf{wit})}|}$ and that the map $r \mapsto (\bar{\boldsymbol{Q}}, \{v_e\}_{e\in E})$ is a bijection. In particular, given x, w, c, and E, the prover randomness r can be uniquely recovered from $(\bar{\boldsymbol{Q}}, \{v_e\}_{e\in E})$.

From HS-PIOP and PCS to Signature Scheme. From such an HS-PIOP and a (hiding and binding) polynomial commitment scheme (PCS), one can build a signature scheme using the Fiat-Shamir transformation. In such a context, the verification key is the statement x and the secret key is the statement-witness pair (x, w). The signing algorithm has the following structure: given a secret key (x, w) and a message to sign msg,

- The signer samples some randomness r, builds the vector polynomial $\boldsymbol{P} := \Phi(w, r)$ and commits it using the PCS. Let us denote com the obtained commitment.
- The signer computes the challenge $c := \mathsf{Hash}_1(x, \mathsf{com})$ and computes the vector polynomial $\boldsymbol{Q} := \Psi(\boldsymbol{P}, x, c)$.
- The signer computes the oracle-query challenge $E := \mathsf{Hash}_2(x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$, computes the evaluations $v_e := \boldsymbol{P}(e)$ for all $e \in E$ and a PCS opening proof π that the evaluations $\{v_e\}_{e\in E}$ are consistent with the commitment com.
- The signer outputs the signatures $\mathsf{sig} := (\mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e\in E})$.

The verification algorithm takes as input a verification key x, a message msg and a signature sig and performs the following steps. It recomputes the challenges c and E as above, verifies that $\Upsilon(\boldsymbol{Q}) = \boldsymbol{0}$ and for all $e \in E$, $\boldsymbol{Q}(e) = \Psi(v_e, x, c)$, and finally runs the PCS verification of the proof π to check that the evaluations $\{v_e\}_{e\in E}$ are consistent with the commitment com.

Remark 2. In the above blueprint, the message is injected only into the final Fiat-Shamir hash used to derive the query set E. As a result, all preceding data $(\mathsf{com}, \{\boldsymbol{Q}_j\}_j)$ can be precomputed once as a reusable, message-independent pre-signature, and later "completed" for any message through a single additional hash and opening step. Security is preserved because the message still influences the final challenge, while the earlier challenge remain unpredictably bound to the committed $\boldsymbol{P}$. This approach has already been employed in several prior signature schemes obtained via the Fiat-Shamir transformation of multi-round protocols, including NIST signature candidates such as SDitH [ABB+24] and MQOM [BBFR24].

In the next section, we will show how to thresholdize such a signature scheme using a threshold PCS and a threshold computation of the functions Ψ.

4.2 Arithmetic Black Box Model

To thresholdize the above signature scheme, we need to enable the parties to jointly compute the function Ψ on shared inputs. Since Ψ is an algebraic function, it can be represented as an arithmetic circuit over

$\mathbb{F}$ using addition, subtraction, and multiplication gates. This computation can be securely performed in a distributed manner using the *arithmetic black box* (ABB) model, introduced by Damgård and Nielsen in [DN03]. This ideal functionality serves as an abstraction in the design of modular secure MPC protocols, particularly within the Universal Composability framework due to Canetti [Can01].

$\mathcal{F}_{\mathsf{ABB}}$ interacts with a set of parties $\boldsymbol{\mathcal{U}} = \{\mathcal{U}_1, \ldots, \mathcal{U}_N\}$. It maintains a memory $\mathcal{M}$ where each entry is of the form $(v, \boldsymbol{\mathcal{T}})$, with $v \in \mathbb{F}$ and $\boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{U}}$. It is defined with respect to a (T, N)-threshold-linear secret sharing $(\mathsf{Share}, \mathsf{Rec})$ over a finite field $\mathbb{F}$.

- **Input:** On input $(\texttt{Input}, id_x, x, \boldsymbol{\mathcal{T}})$ from party $\mathcal{U}_i \in \boldsymbol{\mathcal{U}}$:
  1. Check: If $\boldsymbol{\mathcal{T}} \not\subseteq \boldsymbol{\mathcal{U}}$ or $|\boldsymbol{\mathcal{T}}| < T$, ignore the request.
  2. Store $\mathcal{M}[id_x] \leftarrow (x, \boldsymbol{\mathcal{T}})$.
  3. Output $(\texttt{InputStored}, id_x)$ to all $\mathcal{U}_i \in \boldsymbol{\mathcal{T}}$.
- **InputSharing:** On input $(\texttt{InputSharing}, id_x, \boldsymbol{\mathcal{T}})$ from all $\mathcal{U}_i \in \boldsymbol{\mathcal{T}}$ with $\boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{U}}$ and $|\boldsymbol{\mathcal{T}}| \geq T$:
  1. Wait for shares $[\![x]\!]_i$ from all parties in $\boldsymbol{\mathcal{T}}$.
  2. Check: Reconstruct the value $x = \mathsf{Rec}(([\![x]\!]_i)_{i \in \boldsymbol{\mathcal{T}}})$. If $x = \perp$ ($([\![x]\!]_i)_{i \in \boldsymbol{\mathcal{T}}}$ is not a valid sharing), ignore the request.
  3. Store $\mathcal{M}[id_x] \leftarrow (x, \boldsymbol{\mathcal{T}})$.
  4. Output $(\texttt{InputStored}, id_x)$ to all $\mathcal{U}_i \in \boldsymbol{\mathcal{T}}$.
- **Compute:** On input $(\texttt{Compute}, \mathsf{op}, id_x, id_y, id_z, \boldsymbol{\mathcal{T}})$ from all $\mathcal{U}_i \in \boldsymbol{\mathcal{T}}$ with $\boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{U}}$ and $|\boldsymbol{\mathcal{T}}| \geq T$:
  1. Retrieve $(x, \boldsymbol{\mathcal{T}}_x) = \mathcal{M}[id_x]$ and $(y, \boldsymbol{\mathcal{T}}_y) = \mathcal{M}[id_y]$.
  2. Check: If $\boldsymbol{\mathcal{T}} \not\subseteq \boldsymbol{\mathcal{T}}_x$ or $\boldsymbol{\mathcal{T}} \not\subseteq \boldsymbol{\mathcal{T}}_y$, ignore the request.
  3. Compute $z = x \ \mathsf{op} \ y$ where $\mathsf{op} \in \{+_{\mathbb{F}}, -_{\mathbb{F}}, \times_{\mathbb{F}}\}$.
  4. Store $\mathcal{M}[id_z] \leftarrow (z, \boldsymbol{\mathcal{T}})$.
  5. Output $(\texttt{ComputationDone}, id_z)$ to all $\mathcal{U}_i \in \boldsymbol{\mathcal{T}}$.
- **Reveal:** On input $(\texttt{Reveal}, id_x, \boldsymbol{\mathcal{T}})$ from all $\mathcal{U}_i \in \boldsymbol{\mathcal{T}}$ with $\boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{U}}$ and $|\boldsymbol{\mathcal{T}}| \geq T$:
  1. Retrieve $(x, \boldsymbol{\mathcal{T}}_x) = \mathcal{M}[id_x]$.
  2. Check: If $\boldsymbol{\mathcal{T}} \not\subseteq \boldsymbol{\mathcal{T}}_x$, ignore the request.
  3. Output $(\texttt{Revealed}, id_x, x)$ to all $\mathcal{U}_i \in \boldsymbol{\mathcal{T}}$.

Fig. 2. Ideal functionality $\mathcal{F}_{\mathsf{ABB}}$ for the Arithmetic Black Box.

The ABB model is a standard approach for describing secret-sharing-based protocols in modern protocols such as SPDZ [DPSZ12] and its variants. It is usually instantiated *via* a preprocessing phase to enable fast, unconditionally secure MPC (with dishonest majority). We extend the standard ABB functionality to allow the parties to jointly input a sharing $[\![x]\!]$ to the functionality, thereby providing a framework for threshold signature schemes. This extended functionality $\mathcal{F}_{\mathsf{ABB}}$ is described in Figure 2.

The extra "input sharing" query allows the parties to inject a shared key into the ABB functionality, which can subsequently be used for multiple threshold signing sessions with varying subsets of T signers. For a given signing session, this query also allows the signers to locally generate their shares of the randomness r which are input to the ABB for the authenticated computation of $\boldsymbol{Q}$ and $\{v_e\}_{e \in E}$ while being used in the plain model (outside the ABB) for the threshold PCS.

Notations. For the sake of simplicity, in the description below, we shall denote by $[\![x]\!]^\star$ a sharing *living* in the ideal functionality $\mathcal{F}_{\mathsf{ABB}}$. Namely, $[\![x]\!]^\star$ refers to a record x in the functionality. Moreover, we shall denote:

– $[\![x]\!]^\star \leftarrow \mathsf{ABB.InputSharing}((([\![x]\!]_i)_{i\in\boldsymbol{\mathcal{T}}}, \boldsymbol{\mathcal{T}})$ the protocol running the **InputSharing** query of $\mathcal{F}_{\mathsf{ABB}}$ on input a sharing $([\![x]\!]_i)_{i\in\boldsymbol{\mathcal{T}}}$ for a set of parties $\boldsymbol{\mathcal{T}}$, resulting in a recorded sharing $[\![x]\!]^\star$ of x in the functionality.

– $[\![y]\!]^\star \leftarrow \mathsf{ABB.Eval}(f, [\![x]\!]^\star)$ the protocol running a sequence of **Compute** queries to evaluate an arithmetic function f on a recorded input x, resulting in a recorded output y.

– $x \leftarrow \mathsf{ABB.Reveal}([\![x]\!]^\star)$ the protocol running the **Reveal** query of $\mathcal{F}_{\mathsf{ABB}}$ on a recorded value x.

For a typical instantiation of the ABB functionality using SPDZ's style MPC protocol, $[\![x]\!]^\star$ corresponds to an authenticated sharing of x under a global MAC key.

4.3 Threshold Signature from TPCS and PIOP

We now present our generic construction of a T-out-of-N threshold signature scheme from a TPCS and a HS-PIOP. The construction is generic in the sense that it can be instantiated with any TPCS and PIOP satisfying the properties defined earlier, where the PIOP is defined with respect to some *hard problem* relation $\mathcal{R}$, for which an instance $(x, w) \in \mathcal{R}$ defines a verification key x and a secret key w for the underlying signature scheme. In the distributed setting w is shared among the parties using a T-out-of-N secret sharing scheme which is pre-recorded in the ABB functionality.

1. Sample a random instance $(x, w) \in \mathcal{R}$.
2. Run $[\![w]\!] \leftarrow \mathsf{Share}_{(T,N)}(w)$.
3. Let $\mathsf{sk}_i := [\![w]\!]_i$ for $i \in [1, N]$ and $\mathsf{vk} := x$.
4. **Return** $(\mathsf{sk}_1, \ldots, \mathsf{sk}_N)$ and vk.

Fig. 3. Key generation algorithm.

The key generation algorithm is described in Figure 3. It simply consists in sampling a random instance $(x, w) \in \mathcal{R}$, sharing the secret witness w using the underlying T-out-of-N secret sharing scheme, and defining each party's secret key as its share of w and the verification key as the statement x. The key generation algorithm may be run by a trusted dealer or by the parties themselves using a distributed key generation protocol (which we do not specify here). In addition, we assume that upon receiving their initial signing keys sk_i, the parties have pre-recorded the corresponding authenticated sharing $[\![w]\!]^\star$ of w in the ABB functionality, which will be used as a common reference in all the signing protocol executions.

The signing protocol is described in Figure 4. In a nutshell, this protocol follows the HS-PIOP+PCS paradigm in the distributed setting. Namely, the parties rely on a TPCS to commit to the polynomial $\boldsymbol{P} = \Phi(w, r)$ and compute the associated polynomial $\boldsymbol{Q} = \bar{\Psi}(\boldsymbol{P}, x, c)$ by running a secure computation using the arithmetic black box. Then the parties open the evaluations of $\boldsymbol{Q}$ through the TPCS opening protocol. We recall that $\bar{\boldsymbol{Q}} = \bar{\Psi}(\boldsymbol{P}, x, c)$ is the random part of $\boldsymbol{Q}$ as defined in Section 4.1. The function $\mathsf{PolyEval}_e$ is the evaluation function in the point e. We note that, for the sake of efficiency, the protocol avoids computing redundant information. On one hand, it computes $\bar{\boldsymbol{Q}}$ from which $\boldsymbol{Q}$ can be locally recovered. On the other hand, it computes $\{v_e^{(\mathsf{wit})}\}_{e\in E}$ and then locally derives $\{v_e\}_{e\in E}$ from the $\boldsymbol{Q}(e)$ and $v_e^{(\mathsf{wit})}$.

Let us stress that the first phase (polynomial commitment) does not need to rely on the ABB model, while the second phase (PIOP computation) does. The use of the ABB model to derive $\boldsymbol{Q}$ (and $\bar{\boldsymbol{Q}}$) ensures that the opened evaluations $\{v_e^{(\mathsf{wit})}\}_{e\in E}$ at the beginning of the third phase are consistent with $\boldsymbol{Q}$. Moreover, we further stress that this computation of $\{v_e\}_{e\in E}$ through the ABB is necessary for the security of the scheme. This enables the simulation of the honest parties in the TS-OMUF-CMA security proof, without requiring knowledge of the secret witness w.

Setup. The parties $\boldsymbol{\mathcal{U}}$ own a T-out-of-N secret sharing $(\mathsf{sk}_i)_{i\in\boldsymbol{\mathcal{U}}}$ of a secret witness w, pre-recorded in $\mathcal{F}_{\mathsf{ABB}}$, referenced as $[\![w]\!]^\star$. The parties $\boldsymbol{\mathcal{U}}$ also own the verification key $\mathsf{vk} = x$ where $(x, w) \in \mathcal{R}$. The signing protocol is run by a set $\boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{U}}$ of T active parties sharing a common session identifier sid and message msg to be signed. In the following description, we slightly abuse notations and let $[\![w]\!]$ be the T-out-of-T secret sharing of w obtained from $(\mathsf{sk}_i)_{i\in\boldsymbol{\mathcal{T}}}$.

Phase 1: Polynomial Commitment (in plain model).

1. The parties locally sample random shares $[\![r]\!]_i$. Let $[\![r]\!] = ([\![r]\!]_i)_{i\in\boldsymbol{\mathcal{T}}}$ the associated T-out-of-T secret sharing.
2. The parties locally compute $[\![\boldsymbol{P}]\!] = \Phi([\![w]\!], [\![r]\!])$.
3. The parties run $\mathsf{com} \leftarrow \mathsf{TPCS.Commit}(\mathsf{sid}, [\![\boldsymbol{P}]\!])$. Each party additionally gets a private opening key key_i.

Phase 2: PIOP Polynomial (in the ABB model).

1. The parties locally compute $c = \mathsf{Hash}_1(\mathsf{sid}, x, \mathsf{com})$.
2. The parties run $[\![r]\!]^\star \leftarrow \mathsf{ABB.InputSharing}([\![r]\!], \boldsymbol{\mathcal{T}})$.
3. The parties run $[\![\boldsymbol{P}]\!]^\star \leftarrow \mathsf{ABB.Eval}(\Phi(\cdot,\cdot), ([\![w]\!]^\star, [\![r]\!]^\star))$.
4. The parties run $[\![\bar{\boldsymbol{Q}}]\!]^\star \leftarrow \mathsf{ABB.Eval}(\bar{\Psi}(\cdot, x, c), [\![\boldsymbol{P}]\!]^\star)$.
5. The parties run $\bar{\boldsymbol{Q}} \leftarrow \mathsf{ABB.Reveal}([\![\bar{\boldsymbol{Q}}]\!]^\star)$. They locally compute $\boldsymbol{Q} := \mathsf{Compr}^{-1}_{\ker(\Upsilon)}(\bar{\boldsymbol{Q}})$.

Phase 3: Evaluation Queries.

1. The parties locally compute $E = \mathsf{Hash}_2(\mathsf{sid}, x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$.
2. The parties run $[\![v_e^{(\mathsf{wit})}]\!]^\star \leftarrow \mathsf{ABB.Eval}(\mathsf{PolyEval}_e(\cdot), [\![\boldsymbol{P}^{(\mathsf{wit})}]\!]^\star)$ for all $e \in E$.
3. The parties run $v_e^{(\mathsf{wit})} \leftarrow \mathsf{ABB.Reveal}([\![v_e^{(\mathsf{wit})}]\!]^\star)$ for all $e \in E$.
4. The parties locally compute $v_e^{(\mathsf{mask})} = \boldsymbol{Q}(e) - \Psi'(v_e^{(\mathsf{wit})}, x, c)$ and let $v_e = (v_e^{(\mathsf{wit})}, v_e^{(\mathsf{mask})})$ for all $e \in E$.
5. The parties run $\big(\{v'_e\}_{e\in E}, \pi\big) \leftarrow \mathsf{TPCS.Open}(\mathsf{sid}, \mathsf{com}, E, \{\mathsf{key}_i\}_i)$.
6. The parties locally check that $v_e = v'_e$ for all $e \in E$. If the check fails, they abort (and set $\mathsf{comp} = 0$). Otherwise, they set $\mathsf{comp} = 1$.

Signature. The signature is defined as:

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

Fig. 4. Threshold signing protocol.

The verification algorithm is described in Figure 5. It simply verifies the TPCS opening proof and checks that the evaluations of the polynomial $\boldsymbol{Q}$ at the points in E are consistent with the $\Psi(\cdot, x, c)$ applied to the open evaluations.

In the next section, we prove the security of our signature scheme using the notions we defined in the previous sections.

5 Unforgeability of the Threshold Signature Scheme

To prove the TS-OMUF-CMA security of the threshold signature scheme described above, we use the standard approach of first proving the TS-EUF-KOA security (i.e., where the adversary only has access to the verification key and cannot interact with honest parties) and then using a series of game transitions, we show

Input. The verification algorithm takes as input a verification key x and message msg and a signature sig.

Output. The verification algorithm outputs ACCEPT if the signature is valid under vk, and REJECT otherwise.

1. Parse sig as $\mathsf{sig} = (\mathsf{sid}, \mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e \in E})$.
2. Compute $E = \mathsf{Hash}_2(\mathsf{sid}, x, \mathsf{com}, \boldsymbol{Q}, \mathsf{msg})$ and run the TPCS verification algorithm:
$$out \leftarrow \mathsf{TPCS.Verif}(\mathsf{sid}, \mathsf{com}, E, \{\boldsymbol{v}_e\}_{e \in E}, \pi) \ .$$
If $out = \text{REJECT}$, return REJECT.
3. Compute $c = \mathsf{Hash}_1(\mathsf{sid}, x, \mathsf{com})$ and, for all $e \in E$:
$$\hat{\boldsymbol{Q}}(e) = \Psi(\boldsymbol{v}_e, x, c) \ .$$
4. Check that $\Upsilon(\boldsymbol{Q}) = \boldsymbol{0}$ and $\hat{\boldsymbol{Q}}(e) = \boldsymbol{Q}(e)$ for all $e \in E$. If any check fails, return REJECT.
5. Return ACCEPT.

Fig. 5. Verification algorithm.

how to simulate the honest parties in signing protocols initiated by the adversary to extend the result to the full TS-OMUF-CMA security.

5.1 Key-Only Unforgeability

The main idea of the TS-EUF-KOA (Definition 9) proof is to use the extractable binding property of the TPCS to extract a committed polynomial $\boldsymbol{P}^*$ from a forged signature sig^*. From $\boldsymbol{P}^*$, we can then extract a candidate witness w^* and show that if w^* is not a valid witness for the statement x (i.e., if $(x, w^*) \notin \mathcal{R}$), then the probability that the forged signature sig^* passes verification is negligible. This is achieved by leveraging the round-by-round soundness of the HS-PIOP to argue that the challenges and evaluations in the forged signature cannot be consistent with an invalid witness.

Theorem 1. *Let $\mathcal{R}$ be an NP-relation. Let TSIG be the above threshold signature scheme constructed from a $(\varepsilon_{\text{PIOP-SND},1}, \varepsilon_{\text{PIOP-SND},2})$-round-by-round sound masking HS-PIOP and a $\varepsilon_{\text{TPCS-EB}}$-extractable binding TPCS. For any adversary $\mathcal{A}$ against the TS-EUF-KOA security of TSIG running in time τ and making at most q_{RO} queries to the random oracles, there exists an algorithm $\mathcal{B}$ that solves the search problem for $\mathcal{R}$ in time $\tau' \approx \tau + \tau_{\mathsf{ext}}$ such that:*

$$\mathbf{Adv}^{\mathsf{euf\text{-}koa}}_{\mathsf{TSIG}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{search}}_{\mathcal{R}}(\mathcal{B}) + \varepsilon_{\text{TPCS-EB}} + q_{\mathsf{RO}} \cdot (\varepsilon_{\text{PIOP-SND},1} + \varepsilon_{\text{PIOP-SND},2})$$

where τ_{ext} is the running time of the TPCS extractor.

Proof. Let $\mathcal{A}$ be an adversary in the random experiment $\mathsf{Exp}^{\text{uf-koa}}_{\mathsf{TSIG},\mathcal{A}}(\lambda)$. We construct a reduction $\mathcal{B}$ that converts any successful such adversary into an algorithm solving the search problem of the underlying relation $\mathcal{R}$. Given a statement x in the language $\mathcal{L}$ defined by the relation $\mathcal{R}$ as input, $\mathcal{B}$ receives x as input and interacts with the adversary by simulating the key generation algorithm and controlling the random oracles RO, RO_1 and RO_2 idealizing the hash functions H (used in the TPCS), Hash_1 and Hash_2 (respectively). If the adversary produces a valid forgery, $\mathcal{B}$ extracts from it a witness w satisfying $(x, w) \in \mathcal{R}$.

The reduction algorithm $\mathcal{B}$ simulates the random oracles RO, RO_1 and RO_2 via lazy sampling: it maintains a table $\mathcal{Q}_{\mathsf{RO}}$, $\mathcal{Q}_{\mathsf{RO}_1}$ and $\mathcal{Q}_{\mathsf{RO}_2}$ of query-response pairs, initially empty, and answers new queries by sampling a fresh random value in their respective domains and storing the tuple. When $\mathcal{A}$ outputs its forgery $(\mathsf{msg}^*, \sigma^*)$,

the reduction $\mathcal{B}$ checks whether $\mathsf{Verif}(\mathsf{vk}, \mathsf{msg}^*, \sigma^*) = 1$. This verification may involve random oracle queries to RO, RO_1 and RO_2 that were not made by $\mathcal{A}$. In this case, $\mathcal{B}$ makes the corresponding query on $\mathcal{A}$'s behalf by sampling a fresh random responses and recording them.

The reduction $\mathcal{B}$ then runs as follows:

1. $\mathcal{B}$ receives as input a statement $x \in \mathcal{L}$ and sets the verification key $\mathsf{vk} := x$.
2. $\mathcal{B}$ calls $\mathcal{A}$ and receives $\mathcal{C}$, such that $|\mathcal{C}| = T - 1$, where $\mathcal{C}$ is the set of corrupted parties.
3. $\mathcal{B}$ simulates the KeyGen algorithm for corrupted parties by letting $\mathsf{sk}_i := (\llbracket w \rrbracket_i^\star, \llbracket \Delta \rrbracket_i)$, for $i \in \mathcal{C}$, for a random w and Δ.
4. $\mathcal{B}$ calls $\mathcal{A}$ on input vk and $(\mathsf{sk}_i)_{i \in \mathcal{C}}$. The privacy property of the secret sharing ensures that the view of $\mathcal{A}$ is indistinguishable from a real execution.
5. Upon receiving the output $(\mathsf{msg}^*, \mathsf{sig}^*)$ from $\mathcal{A}$, $\mathcal{B}$ checks if sig^* is a valid signature on msg^* using the verification key vk. If it is valid, $\mathcal{B}$ parses sig^* as $(\mathsf{sid}, \mathsf{com}, \boldsymbol{Q}, \pi, \{v_e\}_{e \in E})$ and calls the extractor Ext of the TPCS extractable binding property on input $(\mathcal{Q}_{\mathsf{RO}}, \mathsf{sid}, \mathsf{com})$ to get a polynomial $\boldsymbol{P}^*$. From $\boldsymbol{P}^*$, $\mathcal{B}$ extracts and outputs the candidate witness w^* such that $\boldsymbol{P}^* = \Phi(w^*, r^*)$ for some r^*.

If w^* is such that $(x, w^*) \notin \mathcal{R}$, then by the round-by-round soundness of the HS-PIOP, the probability that the RO_1 request to Hash_1 returns c such that $\boldsymbol{Q}^* = \Psi(\boldsymbol{P}^*, c)$ satisfies $\Upsilon(\boldsymbol{Q}^*) = \boldsymbol{0}$ is at most $\varepsilon_{\text{PIOP-SND},1}$. Then, if the committed $\bar{\boldsymbol{Q}}^*$ is such that $\boldsymbol{Q}^* \neq \Psi(\boldsymbol{P}^*, c)$, the probability that the RO_2 request to Hash_2 returns E such that for all $e \in E$, $\boldsymbol{Q}^*(e) = \Psi(\boldsymbol{P}^*(e), c)$ is at most $\varepsilon_{\text{PIOP-SND},2}$. Finally, by the extractable binding of the TPCS, the probability that the opened evaluations $\{v_e\}_{e \in E}$ are not consistent with com and $\boldsymbol{P}^*$ is upper bounded by $\varepsilon_{\text{TPCS-EB}}$. We deduce:

$$\Pr[\mathsf{TSIG.Verif}(\mathsf{vk}, \mathsf{msg}^*, \mathsf{sig}^*) = \textsc{Accept} \mid (x, w^*) \notin \mathcal{R}] \leq q_{\mathsf{RO}} \cdot (\varepsilon_{\text{PIOP-SND},1} + \varepsilon_{\text{PIOP-SND},2}) + \varepsilon_{\text{TPCS-EB}} \ .$$

□

5.2 Simulating Honest Parties

The game transitions for extending the TS-EUF-KO security to full TS-OMUF-CMA security involve simulating the honest parties' behavior in signing protocols initiated by the adversary. The goal is to reach an ultimate game in which the secret witness w involved in the key generation can be replaced by a random witness w' without the adversary being able to distinguish this change.

Theorem 2. *Let $\mathcal{R}$ be an NP-relation. Let TSIG be the above threshold signature scheme constructed from a HVZK masking HS-PIOP and a $\varepsilon_{\text{TPCS-ZK}}$-zero-knowledge TPCS. For any adversary $\mathcal{A}$ against the TS-OMUF-CMA security of TSIG running in time τ and making at most q_{sign} queries to the signing oracle, there exists an algorithm $\mathcal{B}$ against the TS-EUF-KOA security of TSIG running in time $\tau' \approx \tau$, such that:*

$$\mathbf{Adv}^{\mathsf{omuf\text{-}cma}}_{\mathsf{TSIG}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{euf\text{-}koa}}_{\mathsf{TSIG}}(\mathcal{B}) + q_{\mathsf{sign}} \cdot \varepsilon_{\text{TPCS-ZK}} \ .$$

Proof. In the following description, we assume that for each session identifier sid for which the adversary starts a signing protocol, the game keeps track of all the broadcasts, publicly reconstructed values, states of the honest parties, state of the zero-knowledge simulator and random oracle queries made during the execution of that signing protocol. For the sake of clarity, we shall keep these details implicit in the description of the games below.

Starting from Game 0, which is the original TS-OMUF-CMA game, we gradually modify the game to reach Game 3 where the honest parties are simulated without knowledge of the secret witness w. The adversary's advantage in this final game is hence upper bounded by the advantage of the adversary against the TS-EUF-KO security of the scheme.

Game 0: This is the original TS-OMUF-CMA game depicted in Figure 12. The adversary's advantage in this game is denoted by ε_0.

Game 1: In this game, we modify the honest parties' computations within the signing oracle $\mathcal{O}_{\mathsf{sign}}$. Instead of running the real $\mathsf{TPCS.Commit}$ and $\mathsf{TPCS.Open}$ protocols, the oracle uses the zero-knowledge simulator $\mathsf{Sim}_{\text{TPCS}}$. More precisely, the honest parties are simulated as follows:

- Whenever the adversary starts a signing protocol with a new session identifier sid, the game initializes a new instance of the TPCS zero-knowledge simulator $\mathsf{Sim}^{(\mathsf{sid})}$ (with its own state).
- Whenever the adversary runs a signing protocol up to the execution of

$$\mathsf{com} \leftarrow \mathsf{TPCS.Commit}(\mathsf{sid}, [\![\boldsymbol{P}]\!]) \ ,$$

  in Phase 1, the game replaces the honest parties' execution of the $\mathsf{TPCS.Commit}$ protocol by calling the simulator $\mathsf{Sim}^{(\mathsf{sid})}$.
- Whenever the adversary runs a signing protocol up to the execution of

$$\big(\{v'_e\}_{e\in E}, \pi\big) \leftarrow \mathsf{TPCS.Open}(\mathsf{sid}, \mathsf{com}, E, \{\mathsf{key}_i\}_i) \ ,$$

  in Phase 3, the game replaces the honest parties' execution of the $\mathsf{TPCS.Open}$ protocol by calling the simulator $\mathsf{Sim}^{(\mathsf{sid})}$ on input the shares $\{[\![v'_e]\!]_i\}_{e\in E} = [\![\boldsymbol{P}]\!]_i(e)$.

The different zero-knowledge simulators $\mathsf{Sim}^{(\mathsf{sid})}$ are called on inputs that respect the distribution of the TPCS zero-knowledge experiment. Hence, by a hybrid argument, the adversary's advantage ε_1 in Game 1 satisfies:

$$|\varepsilon_1 - \varepsilon_0| \leq q_{\mathsf{sign}} \cdot \varepsilon_{\text{TPCS-ZK}} \ .$$

Game 2: In this game, we modify how the honest parties' shares $\{[\![v_e]\!]_i\}_{i\in\mathcal{H}}$ (passed to the TPCS simulator) are generated for the TPCS opening step in Phase 3 as described below.

Whenever a signing session initiated by the adversary reaches Step 4 of Phase 3, the game first reconstructs the value r from w, $\bar{\boldsymbol{Q}}$ and $\{v_e^{(\mathsf{wit})}\}_{e\in E}$ based on Remark 1. The game then retrieves the sum $\sum_{i\in\mathcal{C}}[\![r]\!]_i$ by:

$$\sum_{i\in\mathcal{C}}[\![r]\!]_i = r - \sum_{i\in\mathcal{H}}[\![r]\!]_i \ .$$

The shares $\{[\![v'_e]\!]_i\}_{e\in E, i\in\mathcal{H}}$ in input of the zero-knowledge simulator are then randomly sampled subject to the following linear constraint:

$$\sum_{i\in\mathcal{H}}[\![v'_e]\!]_i = v_e - \Phi\Big(\sum_{i\in\mathcal{C}}[\![w]\!]_i \, , \sum_{i\in\mathcal{C}}[\![r]\!]_i\Big)(e) \ , \tag{1}$$

for all $e \in E$. Note that the right-hand side of the above equation is well defined since the game has access to the shares $([\![w]\!]_i)_{i\in\mathcal{C}}$ of the corrupted parties and can reconstruct r from $\bar{\boldsymbol{Q}}$ and $\{v_e^{(\mathsf{wit})}\}_{e\in E}$.

One can check that the distribution of the shares $\{[\![v'_e]\!]_i\}_{e\in E, i\in\mathcal{H}}$ defined above is identical to their distribution in Game 1 from the adversary's point of view. Indeed, these shares are uniformly random subject to the linear constraint of Equation 1 in both games. We hence deduce that the adversary's advantage ε_2 in Game 2 satisfies:

$$\varepsilon_2 = \varepsilon_1 \ .$$

Game 3: In this game, we modify the KeyGen algorithm: the secret witness w used to generate the parties secret keys during the KeyGen algorithm is replaced by a witness w' sampled at random from the witness space of the statement x. This implies that the parties executing the signing protocol use shares of w' instead of shares of w. By the zero-knowledge property of the HS-PIOP, the joint distribution of $\bar{\boldsymbol{Q}}$ and $\{v_e^{(\mathsf{wit})}\}_{e\in E}$ is independent of the witness used by the honest parties. Moreover, the definition of the honest parties' shares $\{[\![v'_e]\!]_i\}_{e\in E, i\in\mathcal{H}}$ used in input of the zero-knowledge simulator during the TPCS opening step in Phase 3 depends only on the reconstructed values $\bar{\boldsymbol{Q}}$ and $\{v_e^{(\mathsf{wit})}\}_{e\in E}$ (see Game 2). Hence, the distribution of

these shares is also identical in Game 3 and in Game 2. We deduce that the joint distribution of all the values known to the adversary in Game 3 is identical to their distribution in Game 2. Hence, the adversary's advantage ε_3 in Game 3 is identical to its advantage in Game 2:

$$\varepsilon_3 = \varepsilon_2 \ .$$

Finally, in Game 3, the honest parties are simulated without knowledge of the secret witness w. Hence, the adversary's advantage ε_3 in Game 3 is upper bounded by the advantage of the adversary against the TS-EUF-KO security, which conclude the proof.

Remark 3. A concrete instantiation of the arithmetic black box might open the door to active attacks from the corrupted parties to induce errors in the revealed values $\bar{\boldsymbol{Q}}$ and $\{v_e^{(\mathsf{wit})}\}_{e\in E}$. For instance, using SPDZ-like authenticated sharings, corrupted parties might manage to fool the MAC verification with some small probability. Denoting ε_{ABB} such fooling probability, we would have $|\varepsilon_2 - \varepsilon_1| \leq 2 \cdot q_{\mathsf{sign}} \cdot \varepsilon_{\text{ABB}}$, instead of $|\varepsilon_2 - \varepsilon_1| = 0$ in the second transition. This is because this transition implicitly relies on the fact that the revealed values $\bar{\boldsymbol{Q}}$ and $\{v_e^{(\mathsf{wit})}\}_{e\in E}$ are consistent with a genuine computation from (w, r). But in case the adversary would succeed in inducing an error in these values without provoking an abort of the ABB protocol, the simulated shares $\{[\![v'_e]\!]_i\}_{e\in E, i\in\mathcal{H}}$ in Game 2 would not be consistent with the distribution in Game 1, hence the additional term $2 \cdot q_{\mathsf{sign}} \cdot \varepsilon_{\text{ABB}}$ in the advantage loss.

6 Threshold PCS from Merkle Trees

6.1 Thresholdizing PCS based on Merkle Trees

To commit to a polynomial, a widely used approach in the post-quantum setting is to use a Merkle tree-based commitment scheme. In such a scheme, one computes the evaluations of the input polynomial $\boldsymbol{P}$ over an evaluation domain $\mathbb{E}$, and commits to these evaluations through a Merkle tree. Namely, one proceeds as follows:

- Compute $\mathsf{com}_e := \mathsf{Hash}(\boldsymbol{P}(e), \rho_e)$ for all $e \in \mathbb{E}$, where ρ_e is a random nonce used to ensure the hiding property.[4]
- Compute $h_{\mathsf{MT}} := \mathsf{MerkleRoot}(\{\mathsf{com}_e\}_{e\in\mathbb{E}})$, and output h_{MT} as the commitment digest.

To open an evaluation of the committed polynomial, one then simply reveals its authentication path in the Merkle tree. Let us note that such a commitment only allows the opening of evaluations over a restricted domain $\mathbb{E}$ and not the full field. Moreover, it must be completed by a way to ensure that the committed polynomial is of the right degree (as one could commit to arbitrary evaluations). A widely used approach consists in performing a low-degree test (or proximity test), which provides a relaxed guarantee about the degree of the committed polynomial. Another related approach, recently introduced in [FR25b, FR25a] and known as *degree-enforcing commitment scheme* (DECS) provides a strong guarantee on the degree of the committed polynomial (see Section 6.2 hereafter for more details).

We present a general technique for efficiently thresholdizing Merkle tree-based commitment schemes that follow the above format, independently of the specific degree test or degree-enforcement mechanism used. In such schemes, the digests com_e leak no sensitive information about $\boldsymbol{P}$; they can therefore be made public, allowing all parties to locally compute the Merkle tree. Moreover, deriving an evaluation sharing $[\![\boldsymbol{P}(e)]\!]$ from a polynomial sharing $[\![\boldsymbol{P}]\!]$ at a point $e \in \mathbb{E}$ is straightforward and efficient, since polynomial evaluation is a linear operation in the polynomial's coefficients. The main challenge lies in computing com_e from $[\![\boldsymbol{P}(e)]\!]$: a naive solution would involve a multiparty computation of the hash function, which is prohibitively expensive, especially given the large number of evaluations that must be committed to.

[4] In practice, if the evaluations $\{\boldsymbol{P}(e)\}$ contain sufficient entropy, the nonce ρ_e can be omitted.

A way to avoid these costly MPC hash evaluations is to commit to an evaluation $\boldsymbol{P}(e)$ by committing to its sharing $[\![\boldsymbol{P}(e)]\!]$. Concretely, each party locally commits to its own share by computing and broadcasting $\mathsf{Hash}([\![\boldsymbol{P}(e)]\!]_i)$. The commitment digest com_e is then defined as the concatenation of the digests of all shares:

$$\mathsf{com}_e := \mathsf{Hash}([\![\boldsymbol{P}(e)]\!]_1) \mid \cdots \mid \mathsf{Hash}([\![\boldsymbol{P}(e)]\!]_T)\,.$$

While computationally efficient, this approach has a major drawback. Opening the commitment com_e requires revealing all shares $[\![\boldsymbol{P}(e)]\!]_1, \ldots, [\![\boldsymbol{P}(e)]\!]_T$, which must all be included in the opening proof to enable verification. We introduce a simple tweak to mitigate this overhead.

Instead of directly committing to the shares of $[\![\boldsymbol{P}(e)]\!]$, we commit to a masked evaluation $\boldsymbol{P}(e) + \mathsf{mask}$ together with a commitment to the associated mask. Specifically, each party samples a random mask share mask_i and broadcasts the value $[\![\boldsymbol{P}(e)]\!]_i + \mathsf{mask}_i$, along with a commitment $\mathsf{com}_e^{(i)}$ to mask_i. From the broadcast values, the parties can reconstruct the masked evaluation $\boldsymbol{P}(e) + \mathsf{mask}$. During the opening phase, the parties reveal their mask shares mask_i, which allows the verifier to unmask the evaluation and check consistency with the commitment. The key observation is that the mask shares mask_i can be generated using a pseudorandom generator from short seeds ρ_i. As a result, during the opening it suffices to reveal the seeds ρ_i instead of the full mask shares mask_i. This optimization makes the opening communication cost independent of the size of the committed elements, allowing the cost to be amortized across multiple openings.

Concretely, the commitment digest to an evaluation $\boldsymbol{P}(e)$ is defined as

$$\mathsf{com}_e := \mathsf{Hash}\big(\boldsymbol{P}(e) + \mathsf{mask} \mid \mathsf{Hash}(\rho_1) \mid \cdots \mid \mathsf{Hash}(\rho_T)\big)\,,$$

where

$$\boldsymbol{P}(e) + \mathsf{mask} = \sum_{i=1}^{T} \big([\![\boldsymbol{P}(e)]\!]_i + \mathsf{mask}_i\big) \quad \text{with} \quad \mathsf{mask}_i \leftarrow \mathsf{XOF}(\rho_i) \text{ for all } i \in [T]\,,$$

for an extendable output function $\mathsf{XOF}(\cdot)$. To open the commitment to $\boldsymbol{P}(e)$, the parties simply reveal the seeds $\rho_1, \ldots, \rho_T$, which together with the value $\boldsymbol{P}(e)$ allow the verifier to recompute the commitment digest com_e. The communication cost of opening a single evaluation is thus reduced to T λ-bit seeds (in addition to the evaluation itself). Importantly, the size of the opening proof no longer depends on the size of the committed elements and remains small for practical values of T.

In the next section, we describe a concrete instantiation of a threshold PCS based on this approach.

6.2 Threshold Degree-Enforcing Commitment Scheme

The degree-enforcing commitment scheme (DECS) [FR25b, FR25a] is a zero-knowledge small-domain PCS based on Merkle trees allowing to commit to a degree-d vector polynomial $\boldsymbol{P} \in (\mathbb{F}[X])^n$ defined over a finite field. The DECS commitment algorithm first samples a random degree-d vector polynomial $\boldsymbol{M} \in \mathbb{F}^\eta$, called *masking polynomial*, which is needed for the zero-knowledge property. It then computes the Merkle leaves $\{\mathsf{com}_e = \mathsf{Hash}(\boldsymbol{P}(e), \boldsymbol{M}(e))\}_{e \in \mathbb{E}}$ and hashes them into a Merkle tree as $h_{\mathrm{MT}} = \mathsf{MerkleRoot}(\{\mathsf{com}_e\}_{e \in \mathbb{E}})$, following the general blueprint described in the previous section. It derives a challenge (through Fiat-Shamir) as $\Gamma \leftarrow \mathsf{XOF}(h_{\mathrm{MT}})$, where Γ is a matrix from $\mathbb{F}^{\eta \times n}$. It finally computes the *degree-enforcing* polynomial $\boldsymbol{R}$ as

$$\boldsymbol{R}(X) = \Gamma \cdot \boldsymbol{P}(X) + \boldsymbol{M}(X)$$

which is a vector polynomial with η components of degree $\leq d$ assuming that $\boldsymbol{P}$ is of degree $\leq d$. The commitment is defined as $\mathsf{com} = (h_{\mathrm{MT}}, \boldsymbol{R})$ – where the format of the commitment constrains $\boldsymbol{R}$ to be of degree $\leq d$ – and the opening key is $\mathsf{key} = (\boldsymbol{P}, \boldsymbol{M})$. The commitment opening step returns some evaluations $\{\boldsymbol{P}(e)\}_{e \in E}$ as well as the opening proof $\pi = (\{\boldsymbol{M}(e)\}_{e \in E}, \pi_{\mathrm{MT}})$, with π_{MT} being the authentication path with respect to the leaves $\{\mathsf{com}_e\}_{e \in E}$ and Merkle root h_{MT}.

Using the opened evaluations $\{\boldsymbol{P}(e)\}_e$, the verification algorithm recomputes the evaluations $\{\boldsymbol{R}(e)\}_e$ and checks that they are consistent with the committed degree-enforcing polynomial $\boldsymbol{R}$. Given an invalid

witness (with degree $> d$), the probability that a single linear combination has degree $\leq d$ is $\binom{|\mathbb{E}|}{d+2}/|\mathbb{F}|$. Because $\boldsymbol{R}$ is made of η coordinates (with independent randomness), we obtain a soundness error of:

$$\varepsilon = \frac{\binom{|\mathbb{E}|}{d+2}}{|\mathbb{F}|^\eta} \ .$$

Let us now describe how to thresholdize this PCS using the general transformation from Section 6.1. We denote $[\![\boldsymbol{P}']\!] = [\![\boldsymbol{P}, \boldsymbol{M}]\!]$ where $\boldsymbol{P}$ is the polynomial to commit and $\boldsymbol{M}$ the masking polynomial. The parties first locally compute all the evaluations $\boldsymbol{P}'(e)$ for $e \in \mathbb{E}$ from $[\![\boldsymbol{P}']\!]$ and derive the commitment com_e to $\boldsymbol{P}(e)$ as explained in the previous section. Each party can then locally compute the Merkle tree. The next step consists in computing the degree-enforcing polynomial $\boldsymbol{R}$ in a threshold manner. Since $\boldsymbol{R}$ is linear in $\boldsymbol{P}'$, this can simply be done by having the parties locally compute the sharing $[\![\boldsymbol{R}]\!]$ from $[\![\boldsymbol{P}']\!]$. The obtained shares are broadcast, allowing the parties to reconstruct $\boldsymbol{R}$.

Remark 4. In practice, the polynomial $\boldsymbol{R}$ is not fully included in the commitment. To reduce the signature size, one replaces it by a hash digest $h_{\mathsf{R}} = \mathsf{Hash}(\mathsf{sid}, \boldsymbol{R})$. For the verifier to check the hash, one includes some evaluations of $\boldsymbol{R}$ from a set E^* disjoint from the opening set E. This way, the verifier can recompute the evaluations of $\boldsymbol{R}$ over E using the openings of $\boldsymbol{P}, \boldsymbol{M}$, and use the remaining evaluations of $\boldsymbol{R}$ over E^* to reconstruct $\boldsymbol{R}$ and check the hash digest's consistency.

The concrete commitment and opening protocols and verification algorithm of the resulting threshold PCS, which we call *threshold DECS* (TDECS) are depicted Figure 6, Figure 7 and Figure 8 respectively. In this description, we use a hash function Hash and an extended output function XOF that are both modeled as a random oracles in our security analysis. We further denote $h_{\mathsf{MT}} := \mathsf{MerkleRoot}(\{\mathsf{com}_e\}_{e\in\mathbb{E}})$ the process of computing a Merkle tree from the leaves $\{\mathsf{com}_e\}_{e\in\mathbb{E}}$, with hash function Hash, and outputting the root h_{MT}.

In Appendix B, we further describe how to extend this scheme to the case where the committed polynomial $\boldsymbol{P}$ is defined over a base field $\mathbb{F}' \subset \mathbb{F}$ while the commitment scheme is defined over $\mathbb{F}$. The principle is to add a *field-enforcing polynomial* to the commitment in order to prove (with overwhelming probability) that the committed polynomial's evaluations are indeed in $\mathbb{F}'$.

6.3 Security of the Threshold DECS

The following theorems state the extractable-binding and zero-knowledge security of the threshold DECS scheme.

Theorem 3. *The threshold PCS described in Figure 6, Figure 7 and Figure 8 is (t, ε)-extractable binding in the ROM (where Hash and XOF are modeled as random oracles) with:*

$$\varepsilon \leq \frac{Q^2}{2^{2\lambda}} + \frac{\binom{N}{d+2}}{|\mathbb{F}|^\eta}$$

where Q is the number of random oracle queries made by the adversary ($Q \leq t$).

Proof. Let $\mathsf{com} = (h_{\mathsf{MT}}, h_{\mathsf{R}})$, $\pi = (\{\boldsymbol{M}(e), \{\rho_e^{(i)}\}_{i\in[T]}\}_{e\in E}, \{\boldsymbol{R}(e)\}_{e\in E^*}, \pi_{\mathsf{MT}})$, the set of evaluations $\{\boldsymbol{v}_e\}_{e\in E}$. The extractor first calls the verification on the inputs. If the output is Reject, the extractor fails.

We denote $\mathcal{Q}_{\mathsf{H}} = \{(q_i, r_i)\}_i$ the set of query-response pairs made by $\mathcal{A}$ to $\mathsf{H}(\cdot)$. The extractor Ext behaves as follows. Given the Merkle root h_{MT}, the extractor reverts the Merkle tree using the random oracle queries. It first looks for a query of the form $(h_0 \mid h_1, h_{\mathsf{MT}})$, where $|h_0| = |h_1| = 2\lambda$. It then proceeds recursively and tries to find queries of the form $(h_{00} \mid h_{01}, h_0)$, $(h_{10} \mid h_{11}, h_1)$, and follows the same algorithm recursively a total of $h = \lceil \log_2 |\mathbb{E}| \rceil$ times. The extractor should find at most $|\mathbb{E}|/2$ pairs of the form $\mathsf{com}_{2i} \mid \mathsf{com}_{2i+1}$ for $i \in [0 : |\mathbb{E}|/2]$.

$\mathsf{TDECS.Commit}(\mathsf{sid}, [\![\boldsymbol{P}]\!]) \to \mathsf{com}, \{\mathsf{key}_i\}_{i\in[T]}$

1. Each party locally samples random seeds $\rho_e^{(i)} \leftarrow \{0,1\}^\lambda$ for all $e \in \mathbb{E}$ and derives the associated masks:
$$\mathsf{mask\_p}_e^{(i)}, \mathsf{mask\_m}_e^{(i)} = \mathsf{XOF}(\mathsf{sid}, i, e, \rho_e^{(i)}) \ .$$
2. Each party locally samples a random degree-d vector polynomial $[\![\boldsymbol{M}]\!]_i \in (\mathbb{F}[X])^\eta$. We denote $[\![\boldsymbol{M}]\!] := ([\![\boldsymbol{M}]\!]_1, \ldots, [\![\boldsymbol{M}]\!]_T)$ the associated sharing.
3. For all $e \in \mathbb{E}$, the parties locally compute and broadcast:
$$[\![\hat{\boldsymbol{P}}]\!]_i(e) = [\![\boldsymbol{P}]\!]_i(e) + \mathsf{mask\_p}_e^{(i)} \ , \quad [\![\hat{\boldsymbol{M}}]\!]_i(e) = [\![\boldsymbol{M}]\!]_i(e) + \mathsf{mask\_m}_e^{(i)} \ ,$$
$$\mathsf{com}_e^{(i)} = \mathsf{Hash}(\mathsf{sid}, i, e, \rho_e^{(i)}) \ .$$
4. For all $e \in \mathbb{E}$, the parties reconstruct $\hat{\boldsymbol{P}}(e)$ and $\hat{\boldsymbol{M}}(e)$, and locally compute:
$$\mathsf{leaf}_e = (\hat{\boldsymbol{P}}(e), \hat{\boldsymbol{M}}(e), \{\mathsf{com}_e^{(i)}\}_{i\in[T]}) \quad \text{and} \quad \mathsf{com}_e = \mathsf{Hash}(\mathsf{sid}, \mathsf{leaf}_e) \ .$$
5. The parties locally compute
$$h_{\mathsf{MT}} := \mathsf{MerkleRoot}(\{\mathsf{com}_e\}_{e\in\mathbb{E}}) \ , \quad \Gamma = \mathsf{XOF}(h_{\mathsf{MT}}) \in \mathcal{D}(\mathbb{F}^{\eta\times n}) \ .$$
6. The parties locally compute and broadcast the sharing $[\![\boldsymbol{R}]\!] = \Gamma \cdot [\![\boldsymbol{P}]\!] + [\![\boldsymbol{M}]\!]$ and reconstruct $\boldsymbol{R}$. They compute a hash commitment $h_{\mathsf{R}} := \mathsf{Hash}(\mathsf{sid}, \mathsf{R})$.
7. The parties locally compute and output:
$$\mathsf{com} := (h_{\mathsf{MT}}, h_{\mathsf{R}}) \ , \quad \mathsf{key}_i := \left(\{\rho_e^{(i)}\}_{e\in\mathbb{E}}, \{\mathsf{com}_e\}_{e\in\mathbb{E}}, [\![\boldsymbol{P}]\!]_i, [\![\boldsymbol{M}]\!]_i\right) \ .$$

Fig. 6. Threshold DECS – Commitment protocol.

For all $e \in [0, |\mathbb{E}|-1]$, the extractor now searches for queries of the form $(\hat{\boldsymbol{P}}(e) \mid \hat{\boldsymbol{M}}(e) \mid \{\mathsf{com}_e^{(i)}\}_{i\in[T]}, \mathsf{com}_i)$, where the input has size $(|\boldsymbol{P}| + |\boldsymbol{M}|) \cdot \log_2 |\mathbb{K}| + T \cdot 2\lambda$. For each query found, Ext now search for queries of the form $(\mathsf{sid} \mid i \mid e \mid \rho_e^{(i)}, \mathsf{com}_e^{(i)})$. If it finds T such queries, it can extract the seeds $\{\rho_e^{(i)}\}_{i\in[T]}$ and use them to unmask $\hat{\boldsymbol{P}}(e)$. We define $E_{\mathsf{Val}} \subseteq \mathbb{E}$ the subset for which Ext succeeds in extracting the evaluations of the polynomial, and $E_{\mathsf{Inv}} = \mathbb{E} \setminus E_{\mathsf{Val}}$ the set for which Ext fails to extract the corresponding evaluations. If $E \not\subseteq E_{\mathsf{val}}$: the set of opened leaves do not correspond to a valid Merkle tree, so the algorithm aborts. If $|E_{\mathsf{val}}| \geq d+1$, Ext can retrieve $\boldsymbol{P}$ by interpolation. If the subset has size $\leq d$, Ext fails. The extractor also fails if the polynomial has degree $\geq d$.

Let $\boldsymbol{P}$ be the polynomial returned by the extractor. The adversary wins the game if:

$$\exists e \in E : \boldsymbol{P}(e) \neq \boldsymbol{v}_e \ ,$$

while the commitment/opening it generates passes the verification algorithm.

In the following, we bound the probability that the adversary succeeds in performing such an attack.

Game0. This is the game defined by the Definition 2 property.

Game1. This game is the same as Game0 but aborts if the adversary succeeds in finding a hash collision of the form $(*, \mathsf{com}_e)$ with $e \in E_{\mathsf{Inv}}$. This happens with probability:

$$\varepsilon_{\mathsf{col}} = \frac{Q^2}{2^{2\lambda}}$$

where Q is the number of random oracle queries.

$\mathsf{TDECS.Open}(\mathsf{sid}, \mathsf{com}, E, \{\mathsf{key}_i\}_{i\in[T]}) \to \{v_e\}_{e\in E}, \pi$:

1. From $\mathsf{key}_i := \big(\{\rho_e^{(i)}\}_{e\in\mathbb{E}}, \{\mathsf{com}_e\}_{e\in\mathbb{E}}, [\![\boldsymbol{P}]\!]_i, [\![\boldsymbol{M}]\!]_i\big)$ each party locally computes the shares $[\![\boldsymbol{P}]\!]_i(e)$ and $[\![\boldsymbol{M}]\!]_i(e)$ for all $e \in E$. The parties broadcast $\{[\![\boldsymbol{P}]\!]_i(e), [\![\boldsymbol{M}]\!]_i(e), \rho_e^{(i)}\}_{e\in E}$ and reconstruct $\hat{\boldsymbol{P}}(e)$ and $\hat{\boldsymbol{M}}(e)$ for all $e \in E$.
2. The parties locally compute the Merkle proof π_{MT} for the open leaves $\{\mathsf{com}_e\}_{e\in E}$ from the full set of leaves $\{\mathsf{com}_e\}_{e\in\mathbb{E}}$.
3. The parties output:

$$\{v_e\}_{e\in E} := \{\boldsymbol{P}(e)\}_{e\in E}\ ,\ \pi := \Big(\big\{\boldsymbol{M}(e), \{\rho_e^{(i)}\}_{i\in[T]}\big\}_{e\in E}, \{\boldsymbol{R}(e)\}_{e\in E^*}, \pi_{\mathsf{MT}}\Big)$$

where E is the polynomial's evaluation set and $E^* \subset \mathbb{E} \setminus E$ of size s contains the necessary evaluations required to check h_{R}'s consistency.

Fig. 7. Threshold DECS – Opening algorithm.

$\mathsf{TDECS.Verif}(\mathsf{sid}, \mathsf{com}, E, \{v_e\}_{e\in E}, \pi) \to$ Accept / Reject:

1. Parse π as $\Big(\big\{\boldsymbol{M}(e), \{\rho_e^{(i)}\}_{i\in[T]}\big\}_{e\in E}, \{\boldsymbol{R}(e)\}_{e\in E^*}, \pi_{\mathsf{MT}}\Big)$ and com as $(h_{\mathsf{MT}}, h_{\mathsf{R}})$
2. Compute $\Gamma = \mathsf{XOF}(h_{\mathsf{MT}}) \in \mathcal{D}(\mathbb{F}^{\eta\times n})$.
3. For all $e \in E$, compute:

$$\mathsf{com}_e^{(i)} = \mathsf{Hash}(\mathsf{sid}, i, e, \rho_e^{(i)}) \ \text{ and } \ \mathsf{com}_e = \mathsf{Hash}(\mathsf{leaf}_e)$$

where $\mathsf{leaf}_e := (\hat{\boldsymbol{P}}(e), \hat{\boldsymbol{M}}(e), \{\mathsf{com}_e^{(i)}\}_{i\in[N]})$.
4. From $\{\mathsf{com}_e\}_{e\in E}$ and π_{MT}, recompute the Merkle root h'_{MT}. Check that $h_{\mathsf{MT}} = h'_{\mathsf{MT}}$. If the check fails return Reject.
5. For all $e \in E$, verify the equality $\boldsymbol{R}(e) = \Gamma \cdot \boldsymbol{P}(e) + \boldsymbol{M}(e)$. If one check fails return Reject.
6. If all the checks pass, output Accept, output Reject otherwise.

Fig. 8. Threshold DECS – Verification algorithm.

Game2. This game is the same as Game1 but also stops if the adversary breaks the degree-enforcing commitment scheme. Namely, given a challenge matrix Γ, $\mathcal{A}$ needs to find a set E for which it stands that:

$$\boldsymbol{R}(e) = \Gamma \cdot \boldsymbol{P}(e) \text{ for all } e \in E$$
$$\boldsymbol{P}^{(E)} = \mathsf{Interpol}(E, \{\boldsymbol{P}(e)\}_{e\in E}) \text{ is of degree} > d$$

From [FR25b], the probability that such a set exists is upper bounded by:

$$\varepsilon_{\mathsf{def}} = \frac{\binom{N}{d+2}}{|\mathbb{F}|^{\eta}}\ .$$

The game aborts if such a set exists, while if it does not, the adversary cannot win the game. This concludes the proof. □

Theorem 4. *The threshold PCS described in Figure 6, Figure 7 and Figure 8 is (t, ε)-zero knowledge in the ROM (where* Hash *and* XOF *are modeled as random oracles) with $\varepsilon \le Q/2^{\lambda}$, where Q is the number of random oracle queries made by the adversary ($Q \le t$).*

Proof. Let $\mathcal{C} \subset [T]$ the corrupted parties, and $\mathcal{H} = [T] \setminus \mathcal{C}$ the subset of honest parties. The stateful simulator Sim is defined in the following.

Commit Phase: For a session identifier sid, the simulator Sim runs as follows.

1. Sample random matrix $\Gamma \in \mathbb{F}^{\eta \times n}$.
2. For all $i \in \mathcal{H}$ and for all $e \in \mathbb{E}$, pick at random $\mathsf{com}_e^{(i)} \xleftarrow{\$} \{0,1\}^{2\lambda}$.
3. For all $e \in \mathbb{E}$, and for all $i \in \mathcal{H}$, sample random $[\![\hat{\boldsymbol{P}}]\!]_i(e), [\![\hat{\boldsymbol{M}}]\!]_i(e) \xleftarrow{\$} \mathbb{K}^n$, and broadcast:
$$\left\{\{[\![\hat{\boldsymbol{P}}]\!]_i(e), [\![\hat{\boldsymbol{M}}]\!]_i(e), \mathsf{com}_e^{(i)}\}_{e\in\mathbb{E}}\right\}_{i\in\mathcal{H}} .$$
4. Receive the first round of communication from the corrupted parties:
$$\left\{\{[\![\hat{\boldsymbol{P}}]\!]_i(e), [\![\hat{\boldsymbol{M}}]\!]_i(e), \mathsf{com}_e^{(i)}\}_{e\in\mathbb{E}}\right\}_{i\in\mathcal{C}} .$$
5. For all $e \in \mathbb{E}$, compute $\mathsf{com}_e = \mathsf{Hash}(\mathsf{sid}, \hat{\boldsymbol{P}}(e), \hat{\boldsymbol{M}}(e), \{\mathsf{com}_e^{(i)}\}_{i\in[T]})$ where
$$\hat{\boldsymbol{P}}(e), \hat{\boldsymbol{M}}(e) \leftarrow \sum_{i\in[T]} [\![\hat{\boldsymbol{P}}]\!]_i(e), \sum_{i\in[T]} [\![\hat{\boldsymbol{M}}]\!]_i(e) .$$
6. Compute $h_{\mathsf{MT}} = \mathsf{MerkleRoot}(\{\mathsf{com}_e\}_{e\in\mathbb{E}})$.
7. Compute $\Gamma = \mathsf{XOF}(h_{\mathsf{MT}})$.
8. For all $i \in \mathcal{H}$, pick a random $[\![\boldsymbol{R}]\!]_i \xleftarrow{\$} \mathbb{K}[X]^\eta$ of degree $\leq d$ and broadcast it.
9. Receive the second round of communication from the corrupted parties:
$$\{[\![\boldsymbol{R}]\!]_i\}_{i\in\mathcal{C}} ,$$
and computes
$$\boldsymbol{R} := \mathsf{Rec}_{[T]}(\{[\![\boldsymbol{R}]\!]_i\}_{i\in[T]}) , \quad h_{\mathsf{R}} = \mathsf{Hash}(\mathsf{sid}, \boldsymbol{R}) .$$
10. For all $i \in \mathcal{H}$ and for all $e \in \mathbb{E}$, pick a random λ-bits seed $\rho_e^{(i)}$.
11. Output the commitment:
$$\mathsf{com}^{\mathsf{Sim}} := (h_{\mathsf{MT}}, h_{\mathsf{R}}) .$$

Open Phase: For a session identifier sid, the simulator Sim runs as follows. It takes as input the evaluation set E as well as the honest parties' evaluations shares, $\{[\![\boldsymbol{v}_e]\!]_i\}_{e\in E}$. Let $[\![\boldsymbol{P}]\!]_i(e) = [\![\boldsymbol{v}_e]\!]_i$ for all $e \in E$, $i \in \mathcal{H}$.

1. For all $e \in E$, $i \in \mathcal{H}$, compute:
$$[\![\boldsymbol{M}]\!]_i(e) := [\![\boldsymbol{R}]\!]_i(e) - \Gamma \cdot [\![\boldsymbol{P}]\!]_i(e) .$$
2. For all $e \in E$, $i \in \mathcal{H}$, compute:
$$\begin{aligned} \mathsf{mask\_p}_e^{(i)} &:= [\![\hat{\boldsymbol{P}}]\!](e) - [\![\boldsymbol{P}]\!](e) \\ \mathsf{mask\_m}_e^{(i)} &:= [\![\hat{\boldsymbol{M}}]\!](e) - [\![\boldsymbol{M}]\!](e) . \end{aligned}$$
3. Program the random oracle as for all $e \in E$, $i \in \mathcal{H}$:
$$\mathsf{mask\_p}_e^{(i)}, \mathsf{mask\_m}_e^{(i)} = \mathsf{XOF}(\mathsf{sid}, i, e, \rho_e^{(i)}) .$$

4. For all $e \in E$, and for all $i \in \mathcal{H}$, sample a random seed

$$\rho_e^{(i)} \xleftarrow{\$} \{0,1\}^\lambda$$

and program the random oracle such that:

$$\mathsf{com}_e^{(i)} = \mathsf{Hash}(\mathsf{sid}, i, e, \rho_e^{(i)}) \ .$$

5. For all $i \in \mathcal{H}$ and for all $e \in E$, broadcast $\rho_e^{(i)}$.
6. Receive $\rho_e^{(i)}$, $e \in E$, for all $i \in \mathcal{C}$ from the corrupted parties.
7. Retrieve $\hat{\boldsymbol{P}}(e), \hat{\boldsymbol{M}}(e)$ for all $e \in E$. Using the seeds $\rho_e^{(i)}$, $e \in E$, for all $i \in \mathcal{C}$, derive masks and use them to fully recover $\boldsymbol{P}(e), \boldsymbol{M}(e)$. Retrieve authentication path π_{MT} of the subset E w.r.t. Merkle root h_{MT}.

Sim outputs:

$$\{v_e\}_{e\in E} := \{\boldsymbol{P}(e)\}_{e\in E} \ , \quad \pi := \left(\left\{\boldsymbol{M}(e), \{\rho_e^{(i)}\}_{i\in[T]}\right\}_{e\in E}, \{\boldsymbol{R}(e)\}_{e\in E^*}, \pi_{\mathsf{MT}}\right) \ .$$

We now bound the advantage of the adversary succeeding in distinguishing the simulation from a genuine execution of the protocol using the following game transitions.

$\mathsf{Game0}$. This is the standard game from Definition 3.

$\mathsf{Game1}$. This game is the same as $\mathsf{Game0}$ but it aborts whenever, upon an adversary's request to the random oracle, the random response equals a commitment broadcasted by the simulator in the commit phase, namely one ot the $\mathsf{com}_e^{(i)}$ for $e \in \mathbb{E}$, $i \in \mathcal{H}$. Such an abort happens with probability:

$$\varepsilon_{\mathsf{pre}} = \frac{Q_{\mathsf{pre}}}{2^\lambda}$$

where Q_{pre} is the number of RO queries.

$\mathsf{Game2}$. This game is the same as $\mathsf{Game1}$ but it aborts whenever in the open phase the simulator picks a random seed $\rho_e^{(i)}$ for some $e \in E$, $i \in \mathcal{H}$ for which there already exists a random oracle query of the form $\mathsf{Hash}(\mathsf{sid}, i, e, \rho_e^{(i)})$ or $\mathsf{XOF}(\mathsf{sid}, i, e, \rho_e^{(i)})$. Such an abort happens with probability:

$$\varepsilon_{\mathsf{prog}} = \frac{Q_{\mathsf{prog}}}{2^\lambda}$$

where Q_{prog} is the number of RO queries.

If $\mathsf{Game2}$ does not abort, the simulated commitment and opening are identically distributed as a genuine execution of the protocol. Specifically, we have:

- Γ is uniformly random from $\mathbb{F}^{\eta\times n}$,
- $\boldsymbol{R}$ is uniformly random due to the randomness of $[\![\boldsymbol{R}]\!]_i$, $i \in \mathcal{H}$,
- For all $e \in \mathbb{E}$, $i \in \mathcal{H}$, $\{[\![\hat{\boldsymbol{P}}]\!]_i(e), [\![\hat{\boldsymbol{M}}]\!]_i(e), \mathsf{com}_e^{(i)}\}_{e\in\mathbb{E}}$ are picked at random and have the same distribution as in a real execution (due to the random masks and properties of the random oracle output),
- For all $e \in E$, $i \in \mathcal{H}$, $[\![\boldsymbol{P}]\!]_i(e), [\![\boldsymbol{M}]\!]_i(e)$ are consistent with the degree enforcing test and the masked evaluations shares from the first broadcast, and the seeds $\rho_e^{(i)}$ are consistent with the hash commitments,
- For all $e \in E$, $\boldsymbol{P}(e)$ (resp. $\boldsymbol{M}(e)$) is uniformly random due to the randomness of $\hat{\boldsymbol{P}}(e)$ (resp. $\hat{\boldsymbol{M}}(e)$). This follows the same distribution as in the real world when considering a set E of size $\leq \ell$,
- The seeds $\{\rho_e^{(i)}\}_{i\in\mathcal{H}}$ are uniformly random,
- Since the simulator computes the Merkle tree as in an honest execution of the protocol and the leaves of the tree are public, π_{MT} is consistent with the opened leaves and the Merkle tree h_{MT}.

□

7 Concrete Threshold Signatures from MQ and AES

In this section, we provide concrete instantiations of our framework to build threshold MPC-in-the-Head signatures from MQ and AES. We first introduce the PIOP that will be used for building those signatures.

7.1 The LPPC PIOP

The LPPC PIOP (where LPPC stands for *linear and parallel polynomial constraints*), presented in the Threshold-Computation-in-the-Head framework [FR25b] is an abstraction of Ligero's protocol to check arithmetic circuits [AHIV17].[5] It aims to prove the knowledge of a witness $w \in \mathbb{F}^{n \cdot s}$ arranged in a $n \times s$ matrix:

$$\boldsymbol{w} := \begin{bmatrix} w_{1,1} & \dots & w_{1,s} \\ \vdots & \ddots & \vdots \\ w_{n,1} & \dots & w_{n,s} \end{bmatrix} .$$

Let $\boldsymbol{P}^{(\mathsf{wit})} := (P_1, \dots, P_n)$ the vector polynomial encoding the witness such that P_k encodes the k-th row of $\boldsymbol{w}$ by interpolation over fixed evaluation points $\omega_1, \dots, \omega_s$. Namely, for every row index $k \in [n]$, we have:

$$(P_k(\omega_1), \dots, P_k(\omega_s)) = (w_{k,1}, \dots, w_{k,s}) . \tag{2}$$

We call s the *packing parameter*. In addition, to ensure the zero-knowledge property of the PIOP, these polynomials are randomized by interpolation from ℓ extra random coefficients per polynomial, with ℓ the number of oracle queries in the PIOP. In other words, $\boldsymbol{P}^{(\mathsf{wit})}$ is a random vector polynomial of degree $\leq s + \ell - 1$, satisfying (2).

The LPPC PIOP checks two types of constraints on the witness $\boldsymbol{w}$:

1. Parallel polynomial constraints:

   $$\forall j \in [m_1], \forall k \in [s] \ : \ f_j(\boldsymbol{P}(\omega_k)) = 0 ,$$

   for given polynomial functions $f_1, \dots, f_{m_1} \in \mathbb{F}[X_1, \dots, X_n]$ of total degree $\leq d_f$. Namely, the PIOP checks that each f_j vanishes when evaluated on each column of the witness matrix $\boldsymbol{w}$.

2. Global linear constraints:

   $$\forall j \in [m_2] \ : \ \sum_{k=1}^{s} \langle \boldsymbol{A}_j(\omega_k), \boldsymbol{P}(\omega_k) \rangle = t_j ,$$

   for given polynomials $\boldsymbol{A}_1(X), \dots, \boldsymbol{A}_{m_2}(X) \in \mathbb{F}[X]^n$ of degree $\leq s - 1$ and $t_j \in \mathbb{F}$. In other words, for every j, the PIOP checks an inner product relation between the global witness w (i.e. the vector obtained by flattening the matrix $\boldsymbol{w}$) and constant coefficients encoded in the $\boldsymbol{A}_j$ polynomials.

To check these constraints within a PIOP fitting the model of Section 4.1, the prover could simply define $Q_j(X) = f_j(\boldsymbol{P}(X))$ for $j \in [m_1]$ and $Q'_j(X) = \langle \boldsymbol{A}_j(X), \boldsymbol{P}(X) \rangle$ for $j \in [m_2]$ and sends those polynomials to the verifier. The latter could then check that $Q_j(\omega_k) = 0$ for all $j \in [m_1]$ and $k \in [s]$, and that $\sum_{k=1}^{s} Q'_j(\omega_k) = t_j$ for all $j \in [m_2]$.

In order to reduce the communication overhead, the above polynomials are batched using verifier randomness Γ_1, Γ_2, and the batched result is masked using random polynomials $\boldsymbol{M}_1, \boldsymbol{M}_2$ to ensure the zero-knowledge property of the PIOP. This batching reduces the number of coordinates from $m_1 \cdot s + m_2$ to 2ρ, where ρ is a security parameter (see Lemma 1). The computation results in two vector polynomials $\boldsymbol{Q}_1, \boldsymbol{Q}_2$ that are sent to the verifier. The verifier can then check that $\boldsymbol{Q}_1(\omega_1), \dots, \boldsymbol{Q}_1(\omega_s) = \boldsymbol{0}$ and $\sum_{k=1}^{s} \boldsymbol{Q}_2(\omega_k) = \Gamma_2 \cdot (t_1, \dots, t_{m_2})$. The full protocol is described in Figure 9.

[5] In the formalism of [FR25b], the LPPC protocol is presented as an MPC protocol Π_{LPPC} manipulating Shamir's secret sharings and which is then compiled into a zero-knowledge interactive proof by the TCitH framework. We note that there is a direct correspondence between the MPC protocol Π_{LPPC} and the PIOP described here, with the polynomials in the PIOP corresponding to Shamir's secret sharings in the TCitH framework.

Inputs: A secret witness $w \in \mathbb{F}^{n \cdot s}$, arranged in a $n \times s$ matrix $\boldsymbol{w}$, is input to $\mathcal{P}$. A public statement $x = (f_1, \ldots, f_{m_1}, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_{m_2}, t_1, \ldots, t_{m_2})$ defining an LPPC instance is input to both $\mathcal{P}$ and $\mathcal{V}$.

Protocol:

1. $\mathcal{P}$ generates a random vector polynomial $\boldsymbol{P}^{(\mathsf{wit})} = (P_1, \ldots, P_n) \in \mathbb{F}[X]^n$ of degree $\leq s + \ell - 1$ satisfying Equation 2.
2. $\mathcal{P}$ generates a random vector polynomial $\boldsymbol{M}_1 \in \mathbb{F}[X]^\rho$ of degree $\leq d_f \cdot (\ell + s - 1) - s$.
3. $\mathcal{P}$ generates a random vector polynomial $\boldsymbol{M}_2 \in \mathbb{F}[X]^\rho$ of degree $\leq \ell + 2s - 2$ such that $\sum_{k=1}^{s} \boldsymbol{M}_2(\omega_k) = \boldsymbol{0}$.
4. $\mathcal{P}$ sends the polynomial oracle $[\boldsymbol{P}]$ to $\mathcal{V}$, where $\boldsymbol{P} := (\boldsymbol{P}^{(\mathsf{wit})}, \boldsymbol{M}_1, \boldsymbol{M}_2)$.
5. $\mathcal{V}$ samples random matrices $\varGamma_1 \leftarrow \mathbb{F}^{m_1 \times \rho}$ and $\varGamma_2 \leftarrow \mathbb{F}^{m_2 \times \rho}$, and sends them to $\mathcal{P}$.
6. $\mathcal{P}$ computes:

$$\boldsymbol{Q}_1(X) := \boldsymbol{M}_1(X) \cdot V_\varOmega(X) + \varGamma_1 \cdot \left(f_1(\boldsymbol{P}^{(\mathsf{wit})}(X)), \ldots, f_{m_1}(\boldsymbol{P}^{(\mathsf{wit})}(X))\right) , \tag{3}$$

   where $V_\varOmega(X) := \prod_{k=1}^{s}(X - \omega_k)$ is the vanishing polynomial of $\varOmega = \{\omega_1, \ldots, \omega_s\}$, and

$$\boldsymbol{Q}_2(X) := \boldsymbol{M}_2(X) + \varGamma_2 \cdot \left(\langle \boldsymbol{A}_1(X), \boldsymbol{P}^{(\mathsf{wit})}(X)\rangle, \ldots, \langle \boldsymbol{A}_{m_2}(X), \boldsymbol{P}^{(\mathsf{wit})}(X)\rangle\right) \tag{4}$$

7. $\mathcal{P}$ sends $\boldsymbol{Q} := (\boldsymbol{Q}_1, \boldsymbol{Q}_2)$ to $\mathcal{V}$.
8. $\mathcal{V}$ samples a random subset $E \subset \mathbb{E}$ and queries the oracle $[\boldsymbol{P}]$ for the evaluation $v_e = \boldsymbol{P}(e)$ for every $e \in E$.
9. $\mathcal{V}$ checks the correctness of Equation 3 and Equation 4 for every evaluation point $e \in E$ by evaluating $\boldsymbol{Q}(e)$ from the received polynomial $\boldsymbol{Q}$ on one hand and evaluating the two equations on $v_e = \boldsymbol{P}(e)$ on the other hand. $\mathcal{V}$ checks that $\boldsymbol{Q}_1(\omega_1), \ldots, \boldsymbol{Q}_1(\omega_s) = \boldsymbol{0}$ and $\sum_{k=1}^{s} \boldsymbol{Q}_2(\omega_k) = \varGamma_2 \cdot (t_1, \ldots, t_{m_2})$. If any check fails, $\mathcal{V}$ rejects. Otherwise, it accepts.

Fig. 9. LPPC PIOP.

Remark 5. The LPPC protocol can be turned into a one-round variant by avoiding the batching step. The prover directly sends the polynomials obtained by applying the constraints to the witness polynomial without batching. In this variant, the polynomials $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ have $m_1 + m_2$ coordinates instead of 2ρ. This might be of interest for statements with a small number of constraints, such as, e.g., the $\mathcal{MQ}$ problem.

The PC PIOP. A useful special case of the LPPC PIOP is obtained by disabling packing, i.e., by setting $s = 1$. Then the witness matrix has a single column and the *parallel* polynomial constraints are applied to the whole witness vector (rather than to s columns in parallel). In that case, such a constraint further encompass global linear constraints as well, since the latter can be expressed as polynomial constraints of degree 1. This yields the simpler PC PIOP (for *polynomial constraints*), introduced as the MPC protocol $\varPi_{\mathsf{PC}}$ in [FR25b] and also equivalent to the QuickSilver in the VOLE-based zero-knowledge paradigm [YSWW21, BBD+23b].

This variant is often more convenient for arithmetizations where constraints naturally target the full witness (as in many MPC-in-the-Head signatures), and it is particularly well-suited to $\mathcal{MQ}$-type statements. Importantly, our analysis and parameterization for LPPC directly encompass this case as the specialization $s = 1$. In that case, $m_2 = 0$ and the polynomial $\boldsymbol{Q}_2$ is simply discarded from the protocol.

Security of the LPPC PIOP. The following lemmas state the soundness and zero-knowledge properties of the LPPC PIOP. The soundness holds by randomness of the batching challenges $\varGamma_1, \varGamma_2$ and by the Schwartz-Zippel lemma, while the zero-knowledge property is obtained by the randomness in the polynomial $\boldsymbol{P}^{(\mathsf{wit})}$

(masking the open evaluations) and the masking polynomials $\boldsymbol{M}_1, \boldsymbol{M}_2$ (masking the polynomials $\boldsymbol{Q}_1, \boldsymbol{Q}_2$). These arguments are standard for this type of protocols, and are detailed in [FR25b]. The proofs are given in appendix for completeness.

Lemma 1. *The LPPC PIOP is $(\varepsilon_1, \varepsilon_2)$-round-by-round sound with:*

$$\varepsilon_1 = \frac{1}{|\mathbb{F}|^\rho} \quad \textit{and} \quad \varepsilon_2 = \frac{\binom{d_{\mathsf{REL}}}{\ell}}{\binom{|\mathbb{E}|}{\ell}}$$

where $d_{\mathsf{REL}} = \max(d_{\mathsf{PP}}, d_{\mathsf{GL}})$, *for* $\begin{cases} d_{\mathsf{PP}} = d_f \cdot (\ell + s - 1) \\ d_{\mathsf{GL}} = \ell + 2s - 2 \end{cases}$.

The following lemma states the honest-verifier zero-knowledge under Definition 6. This definition relies on the functions Υ and Ψ from the HS-PIOP model. We provide the concrete definition of these functions for the LPPC PIOP in the next paragraph.

Lemma 2. *The LPPC PIOP is honest-verifier zero-knowledge under Definition 6. Namely, any accepting transcript $(c, \boldsymbol{Q}, E, \{v_e\}_{e \in E})$ is uniformly distributed over its definition domain, conditioned to the event that $\Upsilon(\boldsymbol{Q}) = \boldsymbol{0}$ and for all $e \in E$, $\boldsymbol{Q}(e) = \Psi(v_e, x, c)$.*

Consistency with the HS-PIOP Model. We show hereafter that the LPPC PIOP fits into the HS-PIOP model defined in Section 4.1. For this purpose, we give the concrete definition of the functions Φ, Ψ, Υ.

- The function $\Phi : (w, r) \mapsto \boldsymbol{P} := (\boldsymbol{P}^{(\mathsf{wit})}, \boldsymbol{M}_1, \boldsymbol{M}_2)$ maps the witness w and random string r of field elements into the polynomials $\boldsymbol{P}^{(\mathsf{wit})}, \boldsymbol{M}_1, \boldsymbol{M}_2$ by interpolation. Here r is a (field element) random tape sufficiently long to provide the randomness of these polynomials. The function Φ is linear as building polynomials by interpolation from fixed points is a linear operation.
- The function $\Psi : (\boldsymbol{P}, x, c) \mapsto \boldsymbol{Q} := (\boldsymbol{Q}_1, \boldsymbol{Q}_2)$ maps the polynomials $\boldsymbol{P} = (\boldsymbol{P}^{(\mathsf{wit})}, \boldsymbol{M}_1, \boldsymbol{M}_2)$, the public statement x (which defines the LPPC constraints) and the verifier challenge $c = (\Gamma_1, \Gamma_2)$ into the polynomials $\boldsymbol{Q}_1, \boldsymbol{Q}_2$ defined by Equation 3, Equation 4. This function is algebraic of degree $\deg(f)$.
- The function Υ gathers the linear constraints on $\boldsymbol{Q}$ that are supposed to be equal to $\boldsymbol{0}$. In the LPPC PIOP, this function is defined as:

$$\Upsilon : \boldsymbol{Q} \mapsto \left(\boldsymbol{Q}_1(\omega_1), \ldots, \boldsymbol{Q}_1(\omega_s), \sum_{k=1}^{s} \boldsymbol{Q}_2(\omega_k) - \Gamma_2 \cdot \boldsymbol{t} \right) \in \mathbb{F}^{(s+1)\cdot\rho}$$

  where $\boldsymbol{t} := (t_1, \ldots, t_{m_2})$. The function Υ is linear as it is defined as a linear mapping applied to evaluations of the input polynomial $\boldsymbol{Q}$ into fixed points.

It is also clear that the LPPC PIOP is a masking HS-PIOP since $\boldsymbol{Q}$ can be written as:

$$\boldsymbol{Q} = \Psi'(\boldsymbol{P}^{(\mathsf{wit})}, x, c) + \boldsymbol{P}^{(\mathsf{mask})} \ ,$$

where $\boldsymbol{P}^{(\mathsf{mask})} := (\boldsymbol{M}_1 \cdot V_\Omega, \boldsymbol{M}_2)$ and $\Upsilon(\Psi'(\boldsymbol{P}^{(\mathsf{wit})}, x, c)) = \Upsilon(\boldsymbol{P}^{(\mathsf{mask})}) = \boldsymbol{0}$.

7.2 Threshold Signatures from the LPPC PIOP + TDECS

We just saw that the LPPC PIOP is a HS-PIOP and the definitions of the functions Φ, Ψ, Υ. By compiling the protocol from *Figure* 4 with the TDECS from Section 6.2 and the three LPPC functions, we obtain threshold signatures for any hard problem (or one-way function) that can be expressed as an LPPC instance (or PC instance as a special case). Before analyzing the obtained signature size and communication cost, we first discuss possible tweaks leading to interesting trade-offs in some contexts.

Parallel repetitions. A very common principle in MPC-in-the-Head proof systems is to use parallel repetition to amplify the soundness. This is not strictly required in our context as one can already choose N (the size of the Merkle tree) and ℓ (the number of PIOP oracle queries) such that the soundness error $\varepsilon_2 = \binom{d_{\mathsf{REL}}}{\ell} / \binom{|\mathbb{E}|}{\ell}$ is negligible. Repeating the proof τ times in parallel, however, allows to reduce the soundness error to ε_2^τ, which can be useful to decrease the parameters N and/or ℓ in each repetition. Extending our threshold signature protocol and its security to deal with parallel repetition is straightforward: it simply consists in running the same protocol τ times in parallel, while merging the Fiat-Shamir hashes over the τ repetitions. The parameters τ offers a trade-off between the signature size and the communication. Indeed, increasing τ will increase the communication between the parties and allow for smaller signatures, while decreasing τ reduces the communication, at the cost of larger signatures.

Field enforcing. Let us denote by $\mathbb{F}$ the field over which our statement is defined and by $\mathbb{K}$ the field over which the TDECS and LPPC PIOP are defined. A variant of the TDECS given in Appendix B enables to commit to polynomials over a subfield $\mathbb{F}$ of $\mathbb{K}$ and to prove that the committed polynomials are indeed defined over $\mathbb{F}$. This can be useful to arithmetize some statements over small fields, while by definition we must have $|\mathbb{K}| \geq N$ where N is the size of the Merkle tree in the TDECS. When activated, the field enforcing option give rise to extra communication and security parameter μ (see Appendix B for details).

Masking polynomials. When composing the LPPC PIOP with the TDECS, one subtle issue arises: the masking polynomials $\boldsymbol{M}_1, \boldsymbol{M}_2$ are of degree greater than $\boldsymbol{P}$ while the TDECS requires all committed polynomials to have the same degree. To circumvent this issue, we can build $\boldsymbol{M}_1, \boldsymbol{M}_2$ from extra random polynomials of degree $\leq s + \ell - 1$ (the same degree as $\boldsymbol{P}$). We denote by $n^{(\mathsf{mask})}$ the number of such polynomials required to build $\boldsymbol{M}_1, \boldsymbol{M}_2$ (see details hereafter).

Signature Size. We give hereafter the signature size for the threshold signature scheme obtained when instantiating our framework with the LPPC PIOP and the TDECS threshold PCS:

$$
\begin{aligned}
|\sigma| = \; & 6\lambda && (\mathsf{sid} \textit{ and Fiat-Shamir hashes}) \\
& + \tau \cdot \ell \cdot (n + n^{(\mathsf{mask})} \cdot \rho) \cdot \log_2 |\mathbb{K}| && (\textit{the openings } \{\boldsymbol{P}^{[k]}(e)\}_{e,k}) \\
& + \tau \cdot (d_f - 1) \cdot (s + \ell - 1) \cdot \rho \cdot \log_2 |\mathbb{K}| && (\textit{the PIOP polynomial } \{\boldsymbol{Q}_1^{[k]}\}_k) \\
& + \tau \cdot (2s - 2) \cdot \rho \cdot \log_2 |\mathbb{K}| && (\textit{the PIOP polynomial } \{\boldsymbol{Q}_2^{[k]}\}_k) \\
& + \tau \cdot 2\lambda \cdot \mathsf{auth}(\ell, |\mathbb{E}|) && (\textit{the authentication paths}) \\
& + \tau \cdot (s + \ell) \cdot \eta \cdot \log_2 |\mathbb{K}| && (\textit{the degree-enforcing polynomial}) \\
& + \tau \cdot (s + \ell) \cdot \mu \cdot \log_2 |\mathbb{K}| && (\textit{the field-enforcing polynomial}) \\
& + \tau \cdot T \cdot \ell \cdot \lambda && (\textit{the seeds for the TPCS})
\end{aligned}
$$

where:

- $\mathbb{F}$ is the field over which the LPPC instance is defined,
- $\mathbb{K}$ is the extension field over which the TDECS and LPPC PIOP are defined,
- n is the number of rows of the LPPC witness matrix,
- s is the number of columns in the LPPC witness matrix, or the packing parameter,
- ℓ is the number of oracle queries in the LPPC PIOP,
- ρ is the batching parameter in the LPPC PIOP,
- $\deg(f)$ is the maximum degree of the polynomial constraints in the LPPC instance,
- η and μ are respectively the degree-enforcing and field-enforcing parameters,
- τ is the number of parallel repetitions,

– T is the number of parties in the MPC execution of the PIOP,

– $\mathsf{auth}(\ell, |\mathbb{E}|)$ is the average size of ℓ authentication paths in a Merkle tree of size $|\mathbb{E}|$, which is upper-bounded by:

$$\mathsf{auth}(\ell, |\mathbb{E}|) \leq \ell \cdot \log_2 \left(|\mathbb{E}|/\ell\right) \ ,$$

– $n^{(\mathsf{mask})}$ denotes the number of extra committed polynomials of degree $\leq s + \ell - 1$ required to build $\boldsymbol{P}^{(\mathsf{mask})} = (\boldsymbol{M}_1, \boldsymbol{M}_2)$. It is given by:

$$n^{(\mathsf{mask})} = \underbrace{\left\lceil \frac{(\deg(f) - 1) \cdot (s + \ell - 1)}{s} \right\rceil}_{\text{for } \boldsymbol{M}_1} + \underbrace{\left\lceil \frac{2s - 1}{s} \right\rceil}_{\text{for } \boldsymbol{M}_2} \ .$$

Communication Cost. We analyze the communication cost considering two components: the "fixed cost" which is the communication of the threshold signature protocol excluding the ABB communication (which depends on the specific ABB instantiation). This fixed cost includes the communication of the TDECS and the broadcast messages in the signature protocol outside the ABB calls. For the second component (ABB communication), we give the count of secure multiplications required in the ABB calls. These multiplications are only required for the computation of $\boldsymbol{Q}$ (namely for the distributed evaluation of the algebraic Ψ).

Fixed communication cost. The fixed communication cost defined above is given by the following formula:

$$\begin{aligned}
&\tau \cdot |\mathbb{E}| \cdot \left[(n + n^{(\mathsf{mask})} \cdot \rho + \eta + \mu) \cdot \log_2 |\mathbb{K}| + 2\lambda\right] && \textit{(1st TDECS broadcast)} \\
&+ \tau \cdot (s + \ell - 1) \cdot (\eta + \mu) \cdot \log_2 |\mathbb{K}| && \textit{(2nd TDECS broadcast)} \\
&+ \tau \cdot \ell \cdot \left[(n + n^{(\mathsf{mask})} \cdot \rho + \eta + \mu) \cdot \log_2 |\mathbb{K}| + \lambda\right] && \textit{(3rd TDECS broadcast)} \\
&+ \tau \cdot (\deg(f) - 1) \cdot (s + \ell - 1) \cdot \rho \cdot \log_2 |\mathbb{K}| && \textit{(LPPC polynomials broadcast)} \\
&+ \tau \cdot (2s - 2) \cdot \rho \cdot \log_2 |\mathbb{K}| && \textit{(LPPC polynomials broadcast)}
\end{aligned}$$

Secure Multiplication Count. Let us analyze the concrete number of secure multiplications, namely multiplications involved in the evaluation of the PIOP polynomial occurring in the ABB calls. We assume in the following that the polynomial constraints in the LPPC instance have degree $\deg(f) = 2$, which is the most common case in practice.[6] We explain how to compute $\boldsymbol{Q}_1$, irrelevant of the specific LPPC instance, with a low number of multiplications. The computation of $\boldsymbol{Q}_2$ only consists of linear operations, so it is considered free. Since the $\boldsymbol{Q}_1$ has degree 2, it can be written as a quadratic form: $x^\top Ax + b^\top x + c$.

For an LPPC instance of m quadratic equations a witness of length n, checking a solution using Figure 9 means evaluating the following expression for each $k \in [\rho]$:

$$\boldsymbol{P}^{(\mathsf{wit})}(X)^\top \cdot \left(\sum_{j=1}^{m_1} \gamma_j \cdot A_j\right) \cdot \boldsymbol{P}^{(\mathsf{wit})}(X) + \left(\sum_{j=1}^{m_1} \gamma_j \cdot b_j\right)^\top \cdot \boldsymbol{P}^{(\mathsf{wit})}(X) - \sum_{j=1}^{m_1} (\gamma_j \cdot c_j) \ ,$$

where $\{A_j\}_j$ are the matrices containing the degree-2 monomials coefficients, $\{b_j\}_j$ contains the coefficients of the degree-1 terms, and $\{c_j\}_j$ the constant terms. The non-linear part of the above expression can be computed by a single dot-product of the form $\langle \boldsymbol{S}(X), \boldsymbol{P}^{(\mathsf{wit})}(X) \rangle$ where $\boldsymbol{S}(X) := \sum_{j=1}^{m_1} \gamma_j \cdot (A_j \cdot \boldsymbol{P}^{(\mathsf{wit})}(X) + b_j)$ is a vector of polynomials of degree at most $s+\ell-1$. This requires $N^{\mathsf{LPPC}}_{\mathsf{polmult}} = \rho \cdot n$ multiplications of polynomials

[6] In the case of constraints of higher degree, we can break the monomials of degree > 2 into product of smaller ones s.t. we only have to prove quadratic constraints.

of degree at most $s + \ell - 1$. We deduce that the total number of ABB multiplications over the τ repetitions of the LPPC PIOP is given by:

$$N_{\mathsf{mult}}^{\mathsf{ABB}} = \tau \cdot (2 \cdot (s + \ell) - 1) \cdot \underbrace{N_{\mathsf{polmult}}^{\mathsf{LPPC}}}_{\rho \cdot n} \tag{5}$$

(where polynomial multiplications are performed through evaluation and interpolation).

7.3 Arithmetization of MQ and AES with (LP)PC

In the following, we show how to use the (LP)PC PIOP to prove the knowledge of a witness for two different one-way functions: a multivariate quadratic ($\mathcal{MQ}$) system and an AES instance. The arithmetizations define the witness structure and the PIOP constraints: they can eventually be used to build threshold signatures using the method depicted in the beginning of Section 7.2.

Arithmetization for $\mathcal{MQ}$. We can use the LPPC PIOP to check the solution of an instance of the $\mathcal{MQ}$ problem, which is defined as $((A_j)_{j\in[m]}, (b_j)_{j\in[m]}, x, y)$ such that $\forall j \in [m]$, $A_j \in \mathbb{F}_q^{n\times n}$, $b_j \in \mathbb{F}_q^n$ and $x \in \mathbb{F}_q^n$ are uniformly sampled, and for all $j \in [m]$, the following equality holds:

$$y_j = x^T A_j x + b_j^T x \ .$$

We define the verification key as $\mathsf{vk} := (\boldsymbol{A}, \boldsymbol{b}, y)$, and the secret key is x. The LPPC polynomial constraints are directly derived from the $\mathcal{MQ}$ equations. This arithmetization does not use packing (we have $s = 1$) and does not contain any linear constraint.

Arithmetization for AES. We now explain how to build a signature based on an AES instance using the LPPC PIOP. Given an AES instance $y = \mathsf{AES}_k(x)$ or the Even-Mansour (EM) variant $y = k \oplus \mathsf{AES}_x(k)$, we let $\mathsf{vk} = (x, y)$ and $\mathsf{sk} = k$.

We recall that AES's operations typically operate of bytes so the values in lowercase are assumed to be bytes. Uppercase letters denote AES's 4×4 byte array, while R is used to denote the block cipher rounds. In [BBD+23a, BBB+24], the authors introduce arithmetizations of AES instances targeting VOLE-in-the-head zero-knowledge proof systems. While efficient with seed-tree based proof systems, these arithmetizations do not behave well when using Merkle tree commitments. Indeed, the witness is encoded bitwise, which is not efficient in a Merkle tree-based proof system: to commit to a polynomial encoding of a bit using a Merkle tree, we would need to lift the encoding polynomial into a degree-k field extension in order to have enough tree leaves, which would result in non-optimal communication. While the secret lies in $\mathbb{F}_2$, the opened evaluations of the witness polynomial would lie in $\mathbb{F}_{2^k}$. We now present an arithmetization of AES with the witness encoded as bytes. We follow a similar approach as [BBD+23a]. The aim is to extend the secret key into an extended witness that allows for verifying all the algorithm computations with constraints of reasonable degree. In FAEST's style arithmetization, proving the GF(256) inversion constraints with input and output bytes x and y as:

$$x \cdot y = 1$$

is the main bottleneck. The naive approach (checking the above equation) restricts x and y to be non-zero, which adds an extra overhead when choosing the AES instance. There are a few different ways to mitigate the problem (see [BBB+24, BBM+24]). For the sake of simplicity and better understanding, we use the method from [BBM+24] that requires checking the two following degree-3 constraints:

$$xy^2 = y \wedge x^2 y = x \ .$$

Therefore, for every inversion in the AES's encryption (that is, the number of calls to the S-Box's inversion step), the witness is extended with the input and output of the inversion to check these non-linear constraints. To reduce the constraints degree, we also include the product xy, yielding 2 quadratic constraints to verify.

In addition, the algorithm needs to check additional linear constraints. If we let y^{R-1} the output of some S-Box of round $R-1$ and x^R if the input of the corresponding S-Box of round R, it has to verify that $x^R = \Phi(y^{R-1})$ where Φ is $\mathbb{F}_2$-linear. Additional linear constraints from the beginning and the end of the encryption algoritm also need to be verified. While these steps of the encryption algorithm do not yield non-linear constraints in FAEST's arithmetization (because the other AES operations are $\mathbb{F}_2$-linear), some of these steps become non-linear when encoding the witness byte-per-byte.

The other $\mathbb{F}_{2^8}$-non-linear operations are the $\mathbb{F}_2$-affine S-Box operations. Another way of representing this affine step is to use the linearized polynomial $f(X)$ defined as (see [MR02]):

$$f(X) := \sum_{k=0}^{7} \lambda_k \cdot X^{2^k}, \text{ with } (\lambda_k)_{k\in[0:7]} = (\texttt{05}, \texttt{09}, \texttt{f9}, \texttt{25}, \texttt{f4}, \texttt{01}, \texttt{b5}, \texttt{8f}) \ .$$

We now have a second non-linear constraint to check, that is $z = f(y)$ with y being the output of the inversion part of the S-Box. However, if checked naively, this constraint has degree-128, which would imply excessive communication. We can reduce this constraint's degree by extending the witness with powers of y. The witness for this AES arithmetization is then:

$$\boldsymbol{w} = \left((k_i)_{i\in[\lambda/8]}, (x_i, x_i y_i, y_i, (y_i^{2^k})_{k>1}, z_i)_{i\in[N_{\text{SBox}}]}\right) \ .$$

where $N_{\text{SBox}} = S_{\text{ke}} + S_{\text{enc}}$ is the total number of S-Boxes. If using the EM variant the secret key bytes $(k_i)_{i\in[\lambda/8]}$ do not need to be included in the witness, the key k is actually included as the first state array $(x_i)_{i\in[\lambda/8]}$.

Remark 6. If using the standard AES algorithm, the S-Box constraints that need to be proved are those from the KeySchedule algorithm and the encryption algorithm. In the Even-Mansour variant, one would only need to prove the S-Box constraints from the encryption algorithm.

We still require the witness to satisfy some linear constraints, namely those induced by the MixColumns and AddRoundKey steps, if not using the EM variant.

Quadratic constraints for AES-based signature. We now detail what are the equations to verify when considering an arithmetization with quadratic constraints. For each S-Box, we have a witness containing $(x, xy, y, y_2, y_4, y_8, y_{16}, y_{32}, y_{64}, z)$, on which we check the constraints:

- $xy^2 = y \wedge x^2 y = x$,
- $y^2 = y_2$,
- $y_2^2 = y_4$,
- $y_4^2 = y_8$,
- $y_8^2 = y_{16}$,
- $y_{16}^2 = y_{32}$,
- $y_{32}^2 = y_{64}$,
- $\sum_{k=0}^{6} \lambda_k y_{2^k} + \lambda_7 y_{64}^2 = z$.

We remark that among the 9 non-linear operations, we have 7 squarings. As we use additive secret sharing over a field of characteristic 2 in the signature algorithm, we observe that given a sharing $([\![w]\!]_1, \ldots, [\![w]\!]_T)$, the tuple $([\![w]\!]_1^2, \ldots, [\![w]\!]_T^2)$ is an additive sharing of w^2. As a result, all the squarings are free to evaluate, we only need to perform 2 secure multiplications to verify each S-Box.

7.4 Threshold Signature Instances and Evaluation

We present in Table 1 different signature variants based on AES and $\mathcal{MQ}$ instances. The parameters are set in order to reach $\lambda = 128$ bits of statistical security. For each AES and $\mathcal{MQ}$ instance, we consider three parameter sets for three different evaluation domain size: $|\mathbb{E}| \approx 256$, $|\mathbb{E}| \approx 2^{13}$, and $|\mathbb{E}| \approx 2^{16}$. For the first setting, we use $\mathbb{K} = \mathbb{F} = \mathsf{GF}(256)$, while for the two latter setings we use $\mathbb{K} = \mathsf{GF}(2^{16})$.[7] We assume the use

[7] We recall that the evaluation domain $\mathbb{E}$ is constrained to satisfy $|\mathbb{E}| + s \leq |\mathbb{K}|$, as the evaluation point must be outside the witness support, which is why we have $|\mathbb{E}|$ slightly lower than $|\mathbb{K}|$ in the first and last settings.

of *grinding*, a standard technique to optimize MPCitH signatures. Namely, we include a α-bit proof-of-work in the Fiat-Shamir hashes, for some parameter α, increasing the number of hash computations by an average factor of 2^α for the generation of each challenge. This allows us to reduce the soundness errors ε_1 and ε_2 by a factor $2^{-\alpha}$. In other words, the parameters can be chosen so that the original soundness errors satisfy $\varepsilon_1 \leq 2^{-\lambda+\alpha}$ and $\varepsilon_2 \leq 2^{-\lambda+\alpha}$. For our instances, we fix $\alpha = 8$ bits, which means that the soundness errors are upper-bounded by 2^{-120}.

Table 1. Signature instances for $\lambda = 128$ bits of security.

	$n \cdot s$	$\lvert\mathbb{F}\rvert$	$\lvert\mathbb{K}\rvert$	$\lvert\mathbb{E}\rvert$	ℓ	s	d_f	τ	η	μ	ρ
AES	$16 + 200 \cdot 10$	256	256	248	8	8	2	5	26	0	15
AES-EM	$16 + 160 \cdot 10$	256	256	248	8	8	2	5	26	0	15
MQ-$\mathbb{F}_{2^8}$	48	256	256	255	2	1	2	10	19	0	15
MQ-$\mathbb{F}_{2^{16}}$	34	65536	65536	255	2	1	2	10	10	0	8
AES	$16 + 200 \cdot 10$	256	65536	8192	17	16	2	1	28	15	8
AES-EM	$16 + 160 \cdot 10$	256	65536	8192	17	16	2	1	28	15	8
MQ-$\mathbb{F}_{2^8}$	48	256	65536	8192	3	1	2	4	8	15	8
MQ-$\mathbb{F}_{2^{16}}$	34	65536	65536	8192	3	1	2	4	12	0	8
AES	$16 + 200 \cdot 10$	256	65536	65520	13	16	2	1	31	15	8
AES-EM	$16 + 160 \cdot 10$	256	65536	65520	13	16	2	1	31	15	8
MQ-$\mathbb{F}_{2^8}$	48	256	65536	65535	10	1	2	1	18	15	8
MQ-$\mathbb{F}_{2^{16}}$	34	65536	65536	65535	10	1	2	1	18	0	8

Evaluation. Table 2 provides key performance figures of our threshold signature instances, namely the signature size $|\sigma|$ (with respect to the threshold T), the number of multiplications $N_{\mathsf{mult}}^{\mathsf{ABB}}$ (over $\mathbb{K}$) processed by the arithmetic black box, and the communication per party. To provide concrete numbers for the communication cost and round complexity, we need to instantiate the functionality $\mathcal{F}_{\mathsf{ABB}}$. The provided numbers are based on the standard choice of using the SPDZ online protocol, relying on pre-shared multiplication triples and information-theoretic MACs [Bea92, DPSZ12]. As commonly considered in threshold signature schemes, the communication cost is split in two: (1) the communication of the *presign phase* which includes all the communication rounds that can be performed before knowing the message to be signed and that can hence be precomputed, (2) the communication of the *sign phase* that performs the rest of the protocol (once the message is known). For both phases, the figures in brackets represent the part of the communication that is due to the TDECS, which is independent of the ABB instantiation.

On the ABB instantiation. The figures of Table 2 assume that the **Compute** routine of the ABB is instantiated with SPDZ online protocol. In this setting, a field multiplication requires $2 \cdot \log_2 |\mathbb{F}|$ bits of communication per party. In the dishonest majority setting, the parties shares are augmented with λ-bit MACs to guarantee the computation's correctness (with overwhelming probability). As a result, each party's secret key sk_i contains the witness share $[\![w]\!]_i$, as well as a MAC key share $[\![\Delta]\!]_i$ and the MAC share $[\![\mu]\!]_i$ satisfying $\mu = \Delta \cdot w$. All these shares are part of the long secret key and expected to be used in multiple signing sessions. In particular, the MAC key Δ should remain private across the signing sessions, which would not be the case with the original MAC check protocol of SPDZ, as recently shown in [KNOS25]. TO avoid this issue, we instantiate ABB's **Reveal** routine using commit and open, with a batched version of the *masked MAC check* protocol introduced in [KNOS25], thus ensuring active security across multiple signing sessions. As the commitment and batched MAC check have constant communication, the reveal phase is linear in the size of the revealed values. We stress that these common protocols require the offline generation of multiplication triples ahead of the signing session. The reported communication cost does not include this offline phase.[8]

[8] The generation of such triples is well studied topic in the MPC literature and several techniques exist providing different trade-offs. While specifying this offline phase is necessary to implement the protocol in practice, we let this choice out of scope of the present paper.

Table 2. Signature size $|\sigma|$ (in kB), number of multiplications $N_{\mathsf{mult}}^{\mathsf{ABB}}$ (over $\mathbb{K}$), presignature/signature communication per party (in kB). The number in brackets represent the TDECS communication costs. The communication costs are computed based on the SPDZ online protocol and exclude the offline phase for generating the multiplication triples.

	$\lvert\mathbb{E}\rvert$	$\lvert\sigma\rvert$	$N_{\mathsf{mult}}^{\mathsf{ABB}}$	Presign com.	Sign com.	Total com.
AES	248	$22.54 + 0.64 \cdot T$	7750	483.53 (465.71)	24.4 (14.32)	507.93
AES-EM	248	$20.59 + 0.64 \cdot T$	6200	418.43 (403.71)	20.4 (12.32)	438.83
MQ-$\mathbb{F}_{256}$	255	$7.09 + 0.32 \cdot T$	36000	447.55 (375.23)	3.58 (2.62)	451.13
MQ-$\mathbb{F}_{2^{16}}$	255	$7.52 + 0.32 \cdot T$	13600	483.52 (428.8)	4.4 (3.04)	487.92
AES	8192	$13.72 + 0.27 \cdot T$	1625	3565.57 (3558.08)	11.39 (7.11)	3576.96
AES-EM	8192	$12.83 + 0.27 \cdot T$	1300	3154.67 (3148.48)	9.69 (6.26)	3164.36
MQ-$\mathbb{F}_{256}$	8192	$7.06 + 0.19 \cdot T$	10752	7842.54 (7799.34)	3.82 (2.66)	7846.35
MQ-$\mathbb{F}_{2^{16}}$	8192	$6.4 + 0.19 \cdot T$	7616	6191.33 (6160.67)	2.88 (2.06)	6194.21
AES	65520	$12.59 + 0.21 \cdot T$	1425	28838.0 (28831.38)	8.79 (5.51)	28846.79
AES-EM	65520	$11.91 + 0.21 \cdot T$	1140	25560.86 (25555.38)	7.49 (4.86)	25568.35
MQ-$\mathbb{F}_{256}$	65535	$7.48 + 0.16 \cdot T$	8064	24281.03 (24248.61)	4.5 (3.54)	24285.53
MQ-$\mathbb{F}_{2^{16}}$	65535	$6.88 + 0.16 \cdot T$	5712	20470.29 (20447.28)	3.64 (2.96)	20473.93

Comparison. In Table 3, we compare different existing post-quantum threshold signature schemes. When the round complexity analysis is not explicit in the reference paper, we only provide a lower bound based on our understanding of the protocol. As in [CBNA26], we consider the interactive calls to the ABB to be single round.[9] The size overhead per party for [CC24] is given by the formula $\tau \cdot \log_2(N) \cdot \lambda + \lambda \cdot \tau$ (where N is the number of leaves in the GGM tree and τ the number of repetitions). We used the parameters given in the introduction of [CC24], yielding 2.46 kB per party.

Table 3. Comparison of post-quantum threshold (T-out-of-N) signature schemes. Number of rounds R_{pre} and R_{sig} for presignature and signature phases, constraints on (T, N), signature size $|\sigma|$ (kB), total communication per party (kB), and underlying hardness assumption.

	R_{pre}	R_{sig}	(T, N)	$\lvert\mathsf{sig}\rvert$	Com.	Assumption
Hermine [BCdP+26]	1	1	$N \leq 64$	11.3	73.2	MSIS
Th. ML-DSA [CdPE+26]	2	4	$N \leq 6$	2.4	21-1005	MLWE
Th. MAYO [CBNA26]	≥ 5	≥ 3	no limit	0.3	145	Struct. $\mathcal{MQ}$
Th. MAYO [CBNA26]	≥ 4	≥ 1	no limit	0.3	1280	Struct. $\mathcal{MQ}$
[CC24]	≥ 4	≥ 3	no limit	$2.46 \cdot T + 6$	71.2	RSD
Ours – AES	≥ 4	≥ 2	no limit	$0.27 \cdot T + 12.9$	3290	AES
Ours – MQ	≥ 4	≥ 2	no limit	$0.32 \cdot T + 7.1$	451	$\mathcal{MQ}$

Remark 7. In order to enumerate the total number of PIOP multiplications for the $\mathcal{MQ}$ signatures, we use the formula from Equation 5. While we could also use it for the AES instances, it is not the most efficient. Indeed, each AES quadratic constraint contains at most 1 secure multiplication. The constraints on the affine part of the S-Box are free to check since the parties's shares are additive and squaring is GF(2^8) linear. As a result, the parties only need to evaluate 2 secure multiplications per S-Box for the inversion step. Therefore,

[9] For both schemes ([CBNA26] and ours), the round complexity may increase by a constant factor for a concrete ABB instantiation, especially in the dishonest majority model (e.g. by using SPDZ and implementing **Reveal** using the masked MAC check from [KNOS25]).

by directly considering Equation 3, we deduce that evaluating the PIOP requires performing the following number of secure multiplications[10].

$$N_{\mathsf{mult}}^{\mathsf{ABB}} = \tau \cdot (2 \cdot (s + \ell) - 1) \cdot \left\lceil \frac{(S_{\mathrm{ke}} + S_{\mathrm{enc}}) \cdot 2}{s} \right\rceil .$$

The repetition parameter ρ is now out of the equation: this is because neither the f_i nor $P_w(X)$ depend on it, so the functions can be evaluated only once.

An advantage of our scheme and the one of Carozza and Couteau [CC24] is to be versatile as any scheme based on the MPC-in-the-head paradigm. Namely, although beding exemplified on RSD (for [CC24]) and AES/MQ (for our scheme), they can potentially be applied to any hardness assumption (up to their efficient arithmetization). Moreover, we believe our work leaves room for further optimizations and improvements, in particular in the efficient arithmetization of the underlying functionalities and the design of more efficient TDECS.

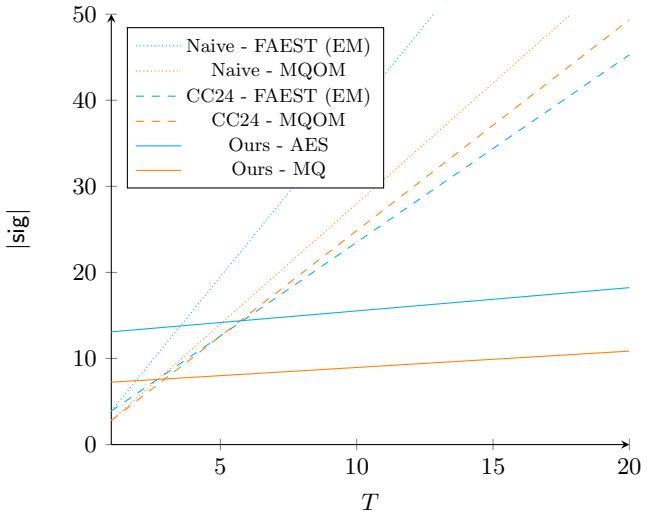

Fig. 10. Comparison between the naive approach, the technique of [CC24], and our scheme for MQ and AES-based signatures. The signature size is given in kB.

Let us further compare our scheme and the versatile alternatives, namely the naive solution of concatenating T signatures from a classic signature scheme, and the optimized solution of Carozza and Couteau [CC24]. The comparison is illustrated in Figure 10, where we use FAEST (FAEST-EM-128s) [BBB+24] and MQOM (MQOM2-L1-gf2-short-5r) [BBFR24] as the underlying signature schemes for the naive solution, and with the approach from [CC24]. In the naive solution, the signature size scales linearly with T with a constant factor equal to the size of a single signature (around 4.5 kB for FAEST and 2.8 kB for MQOM). In comparison, our scheme has a signature size that also scales linearly with T, but with a much smaller constant factor. For instance, for our AES-based signatures, the size increases by around 0.27 kB per party, while for our MQ-based signatures it increases by around 0.19 kB per party. On the other hand, the solution of Carozza and Couteau [CC24] yields a constant factor between 2 and 2.5 kB per party, which is still much larger than our scheme.

[10] Assuming an efficient use of the packing parameter s.

References

ABB+24. Carlos Aguilar Melchor, Slim Bettaieb, Loïc Bidoux, Thibauld Feneuil, Philippe Gaborit, Nicolas Gama, Shay Gueron, James Howe, Andreas Hülsing, David Joseph, Antoine Joux, Mukul Kulkarni, Edoardo Persichetti, Tovohery H. Randrianarisoa, Matthieu Rivain, and Dongze Yue. SDitH — Syndrome Decoding in the Head. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

AHIV23. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: lightweight sublinear arguments without a trusted setup. *DCC*, 91(11):3379–3424, 2023.

BBB+24. Carsten Baum, Lennart Braun, Ward Beullens, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. FAEST. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

BBD+23a. Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. FAEST. Technical report, National Institute of Standards and Technology, 2023. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`.

BBD+23b. Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 581–615. Springer, Cham, August 2023.

BBD+25. Michele Battagliola, Giacomo Borin, Giovanni Di Crescenzo, Alessio Meneghetti, and Edoardo Persichetti. Enhancing threshold group action signature schemes: Adaptive security and scalability improvements. In Ruben Niederhagen and Markku-Juhani O. Saarinen, editors, *Post-Quantum Cryptography - 16th International Workshop, PQCrypto 2025, Part I*, pages 129–161. Springer, Cham, April 2025.

BBFR24. Ryad Benadjila, Charles Bouillaguet, Thibauld Feneuil, and Matthieu Rivain. MQOM — MQ on my Mind. Technical report, National Institute of Standards and Technology, 2024. available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures`.

BBM+24. Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. One tree to rule them all: Optimizing GGM trees and OWFs for post-quantum signatures. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 463–493. Springer, Singapore, December 2024.

BBMP24. Michele Battagliola, Giacomo Borin, Alessio Meneghetti, and Edoardo Persichetti. Cutting the GRASS: Threshold GRoup action signature schemes. In Elisabeth Oswald, editor, *CT-RSA 2024*, volume 14643 of *LNCS*, pages 460–489. Springer, Cham, May 2024.

BCdP+26. Giacomo Borin, Sofía Celi, Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Guilhem Niot, Thomas Prest, and Kaoru Takemure. Hermine: An efficient lattice-based FROST-like threshold signature. Cryptology ePrint Archive, Paper 2026/419, 2026.

BCK+22. Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Cham, August 2022.

BDJ+23. Lennart Braun, Cyprien Delpech de Saint Guilhem, Robin Jadoul, Emmanuela Orsini, Nigel P. Smart, and Titouan Tanguy. ZK-for-Z2K: MPC-in-the-head zero-knowledge proofs for $\mathbb{Z}_{2^k}$. In Elizabeth A. Quaglia, editor, *19th IMA International Conference on Cryptography and Coding*, volume 14421 of *LNCS*, pages 137–157. Springer, Cham, December 2023.

Bea92. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Berlin, Heidelberg, August 1992.

BKL+25. Cecilia Boschini, Darya Kaviani, Russell W. F. Lai, Giulio Malavolta, Akira Takahashi, and Mehdi Tibouchi. Ringtail: Practical two-round threshold signatures from learning with errors. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 149–164. IEEE Computer Society Press, May 2025.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CATZ24. Rutchathon Chairattana-Apirom, Stefano Tessaro, and Chenzhi Zhu. Partially non-interactive two-round lattice-based threshold signatures. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 268–302. Springer, Singapore, December 2024.

CBNA26. Sofia Celi, Giacomo Borin, Guilhem Niot, and Diego F. Aranha. Optimizing and implementing threshold mayo. *IACR Transactions of Cryptographic Hardware and Embedded Systems*, 2026.

CC24. Eliana Carozza and Geoffroy Couteau. On threshold signatures from MPC-in-the-head. Cryptology ePrint Archive, Report 2024/1897, 2024.

CdPE+26. Sofia Celi, Rafael del Pino, Thomas Espitau, Guilhem Niot, and Thomas Prest. Efficient Threshold ML-DSA. In *USENIX 2026 - 35th USENIX Security Symposium*, Baltimore, United States, August 2026.

CEN25. Sofía Celi, Daniel Escudero, and Guilhem Niot. Share the MAYO: Thresholdizing MAYO. In Ruben Niederhagen and Markku-Juhani O. Saarinen, editors, *Post-Quantum Cryptography - 16th International Workshop, PQCrypto 2025, Part I*, pages 165–198. Springer, Cham, April 2025.

CS19. Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: Threshold post-quantum signatures. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *LNCS*, pages 128–153. Springer, Cham, December 2019.

CS20. Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In Jintai Ding and Jean-Pierre Tillich, editors, *PQCrypto 2020*, pages 169–186. Springer, Cham, 2020.

DKM+24. 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.

DKR24. 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.

DM20. Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 187–212. Springer, Cham, May 2020.

DN03. Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 247–264. Springer, Berlin, Heidelberg, August 2003.

dPKPR24. Rafaël del Pino, Shuichi Katsumata, Thomas Prest, and Mélissa Rossi. Raccoon: A masking-friendly signature proven in the probing model. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 409–444. Springer, Cham, August 2024.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Berlin, Heidelberg, August 2012.

EKT24. Thomas Espitau, Shuichi Katsumata, and Kaoru Takemure. Two-round threshold signature from algebraic one-more learning with errors. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 387–424. Springer, Cham, August 2024.

FR23. Thibauld Feneuil and Matthieu Rivain. Threshold linear secret sharing to the rescue of MPC-in-the-head. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part I*, volume 14438 of *LNCS*, pages 441–473. Springer, Singapore, December 2023.

FR25a. Thibauld Feneuil and Matthieu Rivain. SmallWood: Hash-based polynomial commitments and zero-knowledge arguments for relatively small instances. Cryptology ePrint Archive, Report 2025/1085, 2025.

FR25b. 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.

FRWJ25. Thibauld Feneuil, Matthieu Rivain, and Auguste Warmé-Janville. Masking-friendly post-quantum signatures in the threshold-computation-in-the-head framework. *IACR TCHES*, 2025(4):667–710, 2025.

GKS24. Kamil Doruk Gür, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice-based signatures from threshold homomorphic encryption. In Markku-Juhani Saarinen and Daniel Smith-Tone, editors, *PQCrypto 2024, Part II*, pages 266–300. Springer, Cham, June 2024.

KLL25. John Kelsey, Nathalie Lang, and Stefan Lucks. Turning hash-based signatures into distributed signatures and threshold signatures: Delegate your signing capability, and distribute it among trustees. *CiC*, 2(2):24, 2025.

KNOS25. Alexander Kyster, Frederik Huss Nielsen, Sabine Oechsner, and Peter Scholl. Rushing at SPDZ: On the practical security of malicious MPC implementations. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 2491–2508. IEEE Computer Society Press, May 2025.

KRT24. Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 459–491. Springer, Cham, August 2024.

MR02. Sean Murphy and Matthew J. B. Robshaw. Essential algebraic structure within the AES. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 1–16. Springer, Berlin, Heidelberg, August 2002.

PKN+25. Rafaël Del Pino, Shuichi Katsumata, Guilhem Niot, Michael Reichle, and Kaoru Takemure. Unmasking TRaccoon: A lattice-based threshold signature with an efficient identifiable abort protocol. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 423–456. Springer, Cham, August 2025.

PN25. Rafaël Del Pino and Guilhem Niot. Finally! A compact lattice-based threshold signature. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part III*, volume 15676 of *LNCS*, pages 169–199. Springer, Cham, May 2025.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

TZ23. Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 628–658. Springer, Cham, April 2023.

YSWW21. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001. ACM Press, November 2021.

ZT25. 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.

A Threshold Signatures

A.1 Formal Definitions

In this section, we provide the formal syntax and correctness and security properties of a (T, N)-threshold signature scheme [BCK+22, TZ23].

Definition 8 (Threshold Signature Scheme). *A threshold signature scheme* TSIG *is a* 4*-tuple of probabilistic polynomial-time algorithms and interactive protocols* $\mathsf{TSIG} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$*:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T, N)$*: This algorithm takes as inputs a security parameter* λ*, a number of parties* $n \geq 1$ *and a threshold* $1 \leq T \leq N$*. It outputs* public parameters pp*.*
- $(\mathsf{vk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_N)) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*: This interactive protocol (or trusted dealer algorithm) takes as inputs the* public parameter pp *and outputs a global verification key* vk *and secret key shares* sk_i *for each party* $\mathcal{U}_i \in \boldsymbol{\mathcal{U}} = \{\mathcal{U}_1, \ldots, \mathcal{U}_N\}$*.*
- $\{(\mathsf{comp}_i, \sigma_i)\}_{i \in S} \leftarrow \mathsf{Sign}\langle\{\mathcal{U}_i(\mathsf{sk}_i)\}_{i \in S}\rangle(\mathsf{msg}, \mathsf{sid})$*: This interactive protocol is executed by a subset* $S \subseteq \boldsymbol{\mathcal{U}}$ *with* $|S| \geq T$*. The common public input is a message* msg *and a unique session identifier* sid*. Each party* $\mathcal{U}_i \in S$ *uses its secret share* sk_i *as private input. Upon completion, each party* $\mathcal{U}_i$ *outputs a completion bit* $\mathsf{comp}_i \in \{0, 1\}$ *and a local output* σ_i*. The protocol is said to have* successful completion *if* $\mathsf{comp}_i = 1$ *for all* $i \in S$*. If for at least one party* $\mathcal{U}_i$*,* $\mathsf{comp}_i = 0$*, the protocol fails (e.g., due to malicious behavior or abort) and the local output* σ_i *is set to* $\perp$*.*
- $b \leftarrow \mathsf{Verif}(\mathsf{vk}, \mathsf{msg}, \sigma)$*: This (deterministic) algorithm takes as inputs a verification key* vk*, a message* msg*, and a bit string* σ*. It outputs a bit* $b \in \{0, 1\}$*. If* $b = 1$*, then* σ *is a* signature for msg under the verification key vk*.*

Correctness. For any $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, t, n)$, any $(\mathsf{vk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_N)) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, any message msg, and any subset $S \subseteq \boldsymbol{\mathcal{U}}$ with $|S| \geq T$: if all parties in S follow the Sign protocol honestly for the message msg with a unique session identifier sid, then, with probability 1, all parties $\mathcal{U}_i$ output $(\mathsf{comp}_i = 1, \sigma_i = \sigma)$ for some σ such that $\mathsf{Verif}(\mathsf{vk}, \mathsf{msg}, \sigma) = 1$. (i.e. upon successful completion, all parties output the same σ that is a signature for msg under the verification key vk).

Security Definitions. Throughout the paper, we assume a static corruption model where the adversary corrupts a subset of parties before the key generation.

Key-Only Attack. This definition captures the scenario where the adversary knows $T-1$ secret shares but cannot engage in any signing sessions with honest parties.

Definition 9 (TS-EUF-KOA). *A (T, N)-threshold signature scheme $\mathsf{TSIG} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ is (t, ε)-*existentially unforgeable under a key-only attack *if for any adversary $\mathcal{A}$ running in time t, the advantage of $\mathcal{A}$ in the experiment $\mathsf{Exp}^{\mathsf{euf\text{-}koa}}_{\mathsf{TSIG},\mathcal{A}}$ (described in Figure 11) verifies:*

$$\mathbf{Adv}^{\mathsf{euf\text{-}koa}}_{\mathsf{TSIG},\mathcal{A}} := \Pr[\mathsf{Exp}^{\mathsf{euf\text{-}koa}}_{\mathsf{TSIG},\mathcal{A}} = 1] \leq \varepsilon$$

1. **Initialization:** $\mathcal{A}$ outputs a set of corrupted parties $\mathcal{C} \subset \boldsymbol{\mathcal{U}}$ with $|\mathcal{C}| = T-1$. The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T, N)$ and $(\mathsf{vk}, \{\mathsf{sk}_i\}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. $\mathcal{A}$ is given $(\mathsf{pp}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i \in \mathcal{C}})$.
2. **Forgery:** $\mathcal{A}$ outputs a pair $(\mathsf{msg}^*, \sigma^*)$.
3. **Winning condition:** The experiment returns 1 if $\mathsf{Verif}(\mathsf{vk}, \mathsf{msg}^*, \sigma^*) = 1$.

Fig. 11. TS-EUF-KOA experiment for threshold signatures.

One-More Unforgeability. This definition captures security against an active adversary who can interleave multiple signing sessions. We model the signing protocol via an oracle $\mathcal{O}_{\mathsf{sign}}$ that maintains the state of honest parties.

Definition 10 (TS-OMUF-CMA). *A (T, N)-threshold signature scheme $\mathsf{TSIG} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ is (t, ε)-*one-more unforgeable under a chosen-message attack *if for any adversary $\mathcal{A}$ running in time t, the advantage of $\mathcal{A}$ in the experiment $\mathsf{Exp}^{\mathit{omuf\text{-}cma}}_{\mathsf{TSIG},\mathcal{A}}$ (described in Figure 12) verifies:*

$$\mathbf{Adv}^{\mathit{omuf\text{-}cma}}_{\mathsf{TSIG},\mathcal{A}} := \Pr[\mathsf{Exp}^{\mathit{omuf\text{-}cma}}_{\mathsf{TSIG},\mathcal{A}} = 1] \leq \varepsilon$$

B Field-Enforcing DECS

In this section, we introduce a tweak to the DECS that allows working in a field extension $\mathbb{K} \supseteq \mathbb{F}$ while proving that the witness is actually defined over the base field $\mathbb{F}$.

DECS in a field extension. Following the original DECS description, we have the constraint that the number of open parties per polynomial commitment is bounded by $|\mathbb{F}|$, the size of the witness field. We can overcome this issue by working in a field extension $\mathbb{K}$ of $\mathbb{F}$. In this setting, a witness $w = (w_{i,j}) \in \mathbb{F}^{n \times s}$, is encoded as a vector polynomial $\boldsymbol{P} = (P_1, \ldots, P_n) \in \mathbb{K}[X]^n$, where each polynomial P_i is a random polynomial of degree $d := s + \ell - 1$, satisfying $P_i(\omega_j) = w_{i,j}$ for all $j \in [s]$, where $\omega_1, \ldots, \omega_s$ are fixed distinct points in $\mathbb{K}$. So everything is similar to the original DECS over $\mathbb{K}$ instead of $\mathbb{F}$, except that the evaluations of $\boldsymbol{P}$ at the points ω_j are defined over $\mathbb{F}$, which should be guaranteed by the *field-enforcing* DECS.

1. **Initialization:** $\mathcal{A}$ outputs a set of corrupted indices $\bar{\mathcal{C}} \subset \boldsymbol{\mathcal{U}}$ with $|\bar{\mathcal{C}}| = T - 1$. The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T, N)$ and $(\mathsf{vk}, \{\mathsf{sk}_i\}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. $\mathcal{A}$ is given $(\mathsf{pp}, \mathsf{vk}, \{\mathsf{sk}_i\}_{i \in \bar{\mathcal{C}}})$. A counter $q_{\text{sig}} \leftarrow 0$ is initialized.
2. **Signing queries:** $\mathcal{A}$ has access to a signing oracle $\mathcal{O}_{\mathsf{sign}}$ defined as follows:
   - **Session Init:** $\mathcal{A}$ can start a new session sid for a message msg with a chosen specific set of honest participants $\mathcal{H} \subseteq \boldsymbol{\mathcal{U}} \setminus \bar{\mathcal{C}}$ and of corrupted parties $\mathcal{C} \subseteq \bar{\mathcal{C}}$ such that $\mathcal{H} \cup \mathcal{C}$ is the set of parties involved in the session and $|\mathcal{H} \cup \mathcal{C}| = T$. The oracle initializes the internal states of honest parties for $(\mathsf{sid}, \mathsf{msg})$.
   - **Interaction:** $\mathcal{A}$ sends messages of the form $(\mathsf{sid}, \mathcal{U}_j, \mathsf{msg})$ (for $j \in \mathcal{H}$) to the oracle. The oracle executes the next step of the protocol Sign on behalf of honest party $\mathcal{U}_j$ using its current state for session sid. The oracle returns $\mathcal{U}_j$'s response (if any) to $\mathcal{A}$.
   - **Completion:** If $\mathsf{comp}_j = 1$ for some honest parties $\mathcal{U}_j$ (with $j \in \mathcal{H}$) involved in a session, then the session is marked as *successfully completed*, and q_{sig} is incremented $q_{\text{sig}} \leftarrow q_{\text{sig}} + 1$.

   The adversary $\mathcal{A}$ may invoke multiple sessions concurrently with different sid values. The oracle maintains separate state $\mathsf{state}[\mathsf{sid}]$ for each session and follows the prescribed Sign protocol exactly for all honest parties. The adversary controls the message flow and can choose which honest parties to interact with in each session.
3. **Forgery:** $\mathcal{A}$ outputs a list of L pairs $\{(\mathsf{msg}_k, \sigma_k)\}_{k=1}^{L}$.
4. **Winning condition:** The experiment returns 1 if:
   - All signatures are valid: $\forall k, \mathsf{Verif}(\mathsf{vk}, \mathsf{msg}_k, \sigma_k) = 1$.
   - All pairs $(\mathsf{msg}_k, \sigma_k)$ are distinct.
   - The adversary produced more forgeries than the number of completed signing sessions: $L > q_{\text{sig}}$.

Fig. 12. TS-OMUF-CMA experiment for threshold signatures.

Enforcing the base field. When working in a field-extension as described in the previous section, we still want to ensure that the opened polynomial evaluations correspond to a valid polynomial encoding of $w \in \mathbb{F}^{n \times s}$. To do so, we extend the notion of *degree-enforcing commitment* presented in [FR25b], with a base field check inspired by the ring check protocol from [BDJ+23]. We define the notion of field-and-degree-enforcing commitment scheme that merges the two protocols that share many common points.

Definition 11 (Field-Enforcing Degree-Enforcing Commitment Scheme). *Let $\mathbb{F}$ a finite field and $\mathbb{K}$ an extension of $\mathbb{F}$. Consider the DECS protocol defined over $\mathbb{K}$. Let $\Gamma \in \mathbb{K}^{\eta \times n}$ the DECS challenge matrix. Let $\boldsymbol{P} \in \mathbb{K}[X]^n$ the input DECS polynomial, and $\boldsymbol{M} \in \mathbb{K}[X]^\eta$ the corresponding masking polynomial. The* field-enforcing DECS *(FEDECS) protocol is similar to the DECS protocol (on based field $\mathbb{K}$), where Γ and $\boldsymbol{M}$ are extended with a random matrix $\Gamma' \in \mathbb{F}^{\mu \times n}$ and a masking polynomial $\boldsymbol{M'} \in \mathbb{F}[X]^\mu$, such that the resulting "field-enforcing degree-enforcing" polynomial $\boldsymbol{R}$ is defined as:*

$$\begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{R'} \end{pmatrix} := \begin{pmatrix} \Gamma \\ \Gamma' \end{pmatrix} \cdot \boldsymbol{P} + \begin{pmatrix} \boldsymbol{M} \\ \boldsymbol{M'} \end{pmatrix} .$$

The verifier checks that $\deg \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{R'} \end{pmatrix} \leq d$ *and* $\boldsymbol{R'}(\omega_j) \in \mathbb{F}^\mu$*, for all* $j \in [s]$.

Theorem 5. *If $\exists i \in [n], j \in [s]$ s.t. $P_i(\omega_j) \in \mathbb{K} \setminus \mathbb{F}$, we have:*

$$\Pr\left[\boldsymbol{R'}(\omega_j) \in \mathbb{F}^\mu \ \forall j \in [s]\right] \leq \frac{1}{|\mathbb{F}|^\mu} .$$

Proof. Without loss of generality, we consider $\boldsymbol{\delta} \in \mathbb{F}^n$ the first row of the matrix Γ'. Let $\boldsymbol{P}$ be invalid and i, j s.t. $P_i(\omega_j) \in \mathbb{K} \setminus \mathbb{F}$. For any invertible $\delta_i \in \mathbb{F}^\times$, we have $P_i(\omega_j) \in \mathbb{F}$ if and only if $\delta_i P_i(\omega_j) \in \mathbb{F}$ since

$P_i(\omega_j) = \delta_i^{-1}\delta_i P_i(\omega_j)$, *i.e.* the test always fails on an invalid input. We then have:

$$\Pr\left[\delta_i \cdot P_i(\omega_j) \in \mathbb{F} \mid P_i(\omega_j) \in \mathbb{K} \setminus \mathbb{F}, \delta_i \xleftarrow{\$} \mathbb{F}\right] = \Pr\left[\delta_i = 0 \mid \delta_i \xleftarrow{\$} \mathbb{F}\right] \leq \frac{1}{|\mathbb{F}|} .$$

By repeating the test μ times, i.e. for each coordinate of $\boldsymbol{R}'$ with fresh randomness (*i.e.* when considering every row of the matrix Γ'), we get that the final probability that an invalid output passes the test is $1/|\mathbb{F}|^{\mu}$. □

The field-enforcing check (repeated μ times) can be seen as performing a degree-enforcing check (repeated μ times) with randomness picked if $\mathbb{F}$ instead of $\mathbb{K}$. Therefore, the field-enforcing check also performs degree-enforcing check with soundness error:

$$\varepsilon = \frac{\binom{N}{\ell+2}}{|\mathbb{F}|^{\mu}} .$$

As a consequence, we can define the extra degree-enforcing parameter η so that the total soundness satisfies:

$$\frac{\binom{N}{\ell+2}^2}{|\mathbb{F}|^{\mu} \cdot |\mathbb{K}|^{\eta}} \leq 2^{-\lambda} ,$$

which may result in a smaller η than we would get by performing the two checks independently.[11]

C Soundness of the LPPC PIOP

Proof. We first argue the value of the soundness error over the verifier challenge $\boldsymbol{\gamma}'$. Let γ^* any value in Γ_1, Γ_2. Since the challenge γ^* is chosen uniformly in $\mathbb{F}$, we have the bound

$$\Pr\left[\gamma^* \cdot x = 0 \mid \gamma^* \xleftarrow{\$} \mathbb{F}, x \in \mathbb{F}[X]\right] \leq \frac{1}{|\mathbb{F}|} .$$

By applying the bound to the batching equations Equation 3, Equation 4, we have the soundness error, for any $k \in [\rho]$:

$$\Pr\left[\Upsilon(\boldsymbol{Q}_k) = \boldsymbol{0}\right] \leq \Pr\left[\gamma^* \cdot \nu(X) = 0 \mid \gamma^* \xleftarrow{\$} \mathbb{F}\right] \leq \frac{1}{|\mathbb{F}|} ,$$

where $\nu(X)$ is either $f_j(\boldsymbol{P}^{(\mathsf{wit})}(X))$ or $\langle \boldsymbol{A}_j(X), \boldsymbol{P}^{(\mathsf{wit})}(X)\rangle$ for some j. By repeating the batching challenge ρ times with fresh randomness, we get that the final soundness error is:

$$\varepsilon_1 = \frac{1}{|\mathbb{F}|^{\rho}} .$$

Now regarding the random opening challenge (*i.e.* the polynomial oracle queries), the prover opens ℓ evaluations points from $E \subset \mathbb{E}$. Given an invalid witness, the probability that $\boldsymbol{Q}(e) = \Psi(\boldsymbol{P}(e), x, c)$ is a direct application of the Schwartz-Zippel lemma. Given an invalid witness the test accepts with probability

$$\varepsilon_2 := \frac{\binom{d_{\mathsf{REL}}}{\ell}}{\binom{|\mathbb{E}|}{\ell}} ,$$

where d_{REL} is the maximum between the degrees of $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$.

[11] This depends on the parameter set. In cases where $\binom{N}{\ell+2} > |\mathbb{F}|^{\mu}$, the most efficient method is to consider both soundness parameters η, μ independently.

D Zero-Knowledge of the LPPC PIOP

Proof. The simulator runs as follows.

1. Sample a random subset $E \subset \mathbb{E}$ of size ℓ,
2. Pick a random challenge matrices $\Gamma_1, \Gamma_2 \in (\mathbb{F}^{m_1 \times \rho} \times \mathbb{F}^{m_2 \times \rho})$,
3. Sample a random vector polynomial $\boldsymbol{P}^{(\mathsf{wit})} \in \mathbb{F}[X]^n$ of degree $\leq d$,
4. Sample random vector polynomials $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ of size ρ of respective degrees d_{PP} and d_{GL} conditionned to $\Upsilon(\boldsymbol{Q}) = \boldsymbol{0}$.
5. For all $e \in E$ and for all $k \in [\rho]$, compute:

$$\boldsymbol{M}_1(e) := \frac{\boldsymbol{Q}_1(e) - \Gamma_1 \cdot \left(f_1(\boldsymbol{P}^{(\mathsf{wit})}(e)), \ldots, f_{m_1}(\boldsymbol{P}^{(\mathsf{wit})}(e))\right)}{V_\Omega(X)}$$

and

$$\boldsymbol{M}_2(e) := \boldsymbol{Q}_2(e) - \Gamma_2 \cdot \left(\langle \boldsymbol{A}_1(e), \boldsymbol{P}^{(\mathsf{wit})}(e)\rangle, \ldots, \langle \boldsymbol{A}_{m_2}(e), \boldsymbol{P}^{(\mathsf{wit})}(e)\rangle\right) .$$

The protocol transcript is defined as:

$$(c, \boldsymbol{Q}, E, \{\boldsymbol{P}(e)\}_{e \in E})$$

where $c = (\Gamma_1, \Gamma_2)$, $\boldsymbol{Q} = (\boldsymbol{Q}_1, \boldsymbol{Q}_2)$ and $\boldsymbol{P} := (\boldsymbol{P}^{(\mathsf{wit})}, \boldsymbol{M}_1, \boldsymbol{M}_2)$.

The transcript produced by the simulator is valid, since $\boldsymbol{Q}_1, \boldsymbol{Q}_2, \{\boldsymbol{P}(e)\}_{e \in E}$ are defined according to *Equation* 3, *Equation* 4. Therefore we have $\boldsymbol{Q}(e) = \Psi(\boldsymbol{P}(e), x, c)$. Also, $\boldsymbol{Q}$ is chosen such that $\Upsilon(\boldsymbol{Q}) = \boldsymbol{0}$. It also has the same distribution as a *real* transcript of the execution of the protocol:

- In a real transcript, $\{\boldsymbol{P}^{(\mathsf{wit})}(e)\}_{e \in E}$ are uniformly random since they are evaluations of a polynomial defined using ℓ random coefficients. In the simulation, the values are random since the polynomial is picked at random.
- In a real execution $\boldsymbol{M}_1(X), \boldsymbol{M}_2(X)$ are uniformly random polynomials hence their evaluations over E are random. In the simulation, they are also random evaluations due to the randomness of $\boldsymbol{Q}_1, \boldsymbol{Q}_2$'s evaluations over E (since $\boldsymbol{Q}_1, \boldsymbol{Q}_2$ are random polynomials).
- $\boldsymbol{Q}_1(X), \boldsymbol{Q}_2(X)$ have the same distributions in real exectution and simulation. They are picked uniformly at random in the simulation, and they have the same distribution in a real execution due to the randomness of $\boldsymbol{M}_1(X), \boldsymbol{M}_2(X)$.