Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head

Thibauld Feneuil[1,2(✉)] and Matthieu Rivain[1]

[1] CryptoExperts, Paris, France
thibauld.feneuil@cryptoexperts.com
[2] Sorbonne Université, CNRS, INRIA, Institut de Mathématiques de Jussieu-Paris Rive Gauche, Ouragan, Paris, France

Abstract. The MPC-in-the-Head paradigm is a popular framework to build zero-knowledge proof systems using techniques from secure multi-party computation (MPC). While this paradigm is not restricted to a particular secret sharing scheme, all the efficient instantiations for small circuits proposed so far rely on additive secret sharing.

In this work, we show how applying a threshold linear secret sharing scheme (threshold LSSS) can be beneficial to the MPC-in-the-Head paradigm. For a general *passively-secure* MPC protocol model capturing most of the existing MPCitH schemes, we show that our approach improves the soundness of the underlying proof system from $1/N$ down to $1/\binom{N}{\ell}$, where N is the number of parties and ℓ is the privacy threshold of the sharing scheme. While very general, our technique is limited to a number of parties $N \leq |\mathbb{F}|$, where $\mathbb{F}$ is the field underlying the statement, because of the MDS conjecture.

Applying our approach with a low-threshold LSSS also boosts the performance of the proof system by making the MPC emulation cost independent of N for both the prover and the verifier. The gain is particularly significant for the verification time which becomes logarithmic in N (while the prover still has to generate and commit the N input shares). We further generalize and improve our framework: we show how linearly-homomorphic commitments can get rid of the linear complexity of the prover, we generalize our result to any quasi-threshold LSSS, and we describe an efficient batching technique relying on Shamir's secret sharing.

We finally apply our techniques to specific use-cases. We first propose a variant of the recent SDitH signature scheme achieving new interesting trade-offs. In particular, for a signature size of 10 KB, we obtain a verification time lower than 0.5 ms, which is competitive with SPHINCS$^+$, while achieving much faster signing. We further apply our batching technique to two different contexts: batched SDitH proofs and batched proofs for general arithmetic circuits based on the Limbo proof system. In both cases, we obtain an amortized proof size lower than 1/10 of the baseline scheme when batching a few dozen statements, while the amortized performances are also significantly improved.

J. Guo and R. Steinfeld (Eds.): ASIACRYPT 2023, LNCS 14438, pp. 441–473, 2023.
https://doi.org/10.1007/978-981-99-8721-4_14

1 Introduction

Zero-knowledge proofs are an important tool for many cryptographic protocols and applications. Such proofs enable a *prover* to prove a statement by interacting with a *verifier* without revealing anything more than the statement itself. Zero-knowledge proofs find applications in many contexts: secure identification and signature, (anonymous) credentials, electronic voting, blockchain protocols, and more generally, privacy-preserving cryptography.

Among all the possible techniques to build zero-knowledge proofs, the MPC-in-the-Head framework introduced by Ishai, Kushilevitz, Ostrovsky and Sahai in [IKOS07] has recently gained popularity. This framework relies on secure multi-party computation (MPC) techniques: the prover emulates "in her head" an ℓ-private MPC protocol with N parties and commits each party's view independently. The verifier then challenges the prover to reveal the views of a random subset of ℓ parties. By the privacy of the MPC protocol, nothing is revealed about the plain input, which implies the zero-knowledge property. On the other hand, a malicious prover needs to cheat for at least one party, which shall be discovered by the verifier with high probability, hence ensuring the soundness property.

The MPC-in-the-Head (MPCitH) paradigm provides a versatile way to build (candidate) quantum-resilient proof systems and signature schemes. This approach has the advantage to rely on security assumptions that are believed to be robust in the quantum setting, namely the security of commitment schemes and/or hash functions. Many recent works have proposed new MPCitH techniques which can be applied to general circuits and/or specific problems, some of them leading to efficient candidate post-quantum signature schemes, see for instance [GMO16, CDG+17, AHIV17, KKW18, DDOS19, KZ20b, BFH+20, BN20, BD20, BDK+21, DOT21, DKR+21, KZ22, FJR22, FMRV22]. Proof systems built from the MPCitH paradigm can be divided in two categories:

- Schemes targeting small circuits (*e.g.* to construct efficient signature schemes), such as [KKW18, BN20, KZ22]. In these schemes, the considered MPC protocol only needs to be secure in the *semi-honest model*, enabling efficient constructions, but the resulting proof is *linear* in the circuit size. Previous schemes in this category are all based on additive secret sharing.
- Schemes such as [AHIV17, GSV21] in which the considered MPC protocol is secure in the *malicious model* and the proof is *sublinear* in the circuit size (in $O(\sqrt{|C|})$ with $|C|$ being the circuit size). Due to their sublinearity, these schemes are more efficient for *middle-size circuits* (while the former remain more efficient for smaller circuits arising *e.g.* in signature schemes).

We note that other quantum-resilient proof systems exist (a.k.a. SNARK, STARK) which do not rely on the MPCitH paradigm and which achieve polylogarithmic proof size (w.r.t. the circuit size), see *e.g.* [BCR+19, BBHR19]. These schemes are hence better suited for *large circuits*.

Our work belongs to the first category of MPCitH-based schemes (*i.e.* targeting small circuits). Currently, the best MPCitH-based schemes in this

scope rely on $(N-1)$-private passively-secure MPC protocols with N parties [KKW18,BN20,DOT21,KZ22], where the parameter N provides different trade-offs between communication (or signature size) and execution time. In these schemes, the proof is composed of elements of size solely depending on the target security level λ (the "incompressible" part) and other elements of size $\mathcal{O}(\lambda^2/\log N)$ bits (the "variable" part). To obtain short proofs or signatures, one shall hence take a large number of parties N. On the other hand, the prover and verifier running times scale linearly with N (because of the MPC emulation) and hence quickly explode while trying to minimize the proof size.

In this paper, we improve this state of affairs. While previous efficient instantiations of the MPCitH paradigm for small circuits all rely on additive secret sharing, we show how to take advantage of using threshold linear secret sharing. Using our approach, we can decrease the soundness error from $1/N$ to $1/\binom{N}{\ell}$ (still using passively-secure protocols), for a small constant ℓ, while making the cost of the MPC emulation independent of N, for both the prover and the verifier. The prover running time remains globally linear in N (because of the initial sharing and commitment phase) but is still significantly improved in practice. On the other hand, the verification time becomes logarithmic in N and is hence drastically reduced (both asymptotically and in practice).

Our Contribution. We first describe a general model of multiparty computation protocol (with additive secret sharing) which captures a wide majority of the protocols used in the MPCitH context. (To the best of our knowledge, our model applies to all the MPCitH schemes except those derived from ZKBoo or Ligero.) Given a statement x and a relation $\mathcal{R}$, these MPC protocols aim to evaluate a randomized function f on a secret witness w such that f outputs Accept when $(x,w) \in \mathcal{R}$ and Reject with high probability otherwise. The *false-positive rate* of the MPC protocol corresponds to the probability that f outputs Accept even if $(x,w) \notin \mathcal{R}$. We further recall the general transformation of such a protocol into a zero-knowledge proof which achieves a soundness error of

$$\frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right)$$

where N is the number of parties and p is the false-positive rate of the MPC protocol. We then show how to apply an arbitrary threshold linear secret sharing scheme (LSSS) to our general MPC model and how to transform the obtained MPC protocol into a zero-knowledge proof achieving the following soundness error:

$$\frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell \cdot (N-\ell)}{\ell+1},$$

where ℓ is the threshold of the LSSS (any ℓ shares leak no information while the secret can be reconstructed from any $\ell+1$ shares). Our theorems cover all the MPC protocols complying with our general model, and for any threshold LSSS (covering additive sharing as a particular case).

Besides improving soundness, using an LSSS with a small threshold implies significant gains in terms of timings. Indeed, the prover and the verifier do not need to emulate all the N parties anymore, but only a small number of them ($\ell+1$ for the prover and ℓ for the verifier). For instance, when working with Shamir's secret sharing [Sha79] with polynomials of degree $\ell = 1$, the prover only needs to emulate 2 parties (instead of N) and the verifier only needs to emulate 1 party (instead of $N-1$) while keeping a soundness error about $\frac{1}{N}$ (assuming a small false positive rate p). On the other hand, the proof size is slightly larger than in the standard case (with additive sharing) since one needs to use a Merkle tree for the commitments (and include authentication paths for the opened commitments in the proof). Overall, our approach provides better trade-offs between proof size and performances for MPCitH schemes while drastically reducing the verification time in particular.

We further improve and generalize our approach in different ways. We first show how using linearly-homomorphic commitments can make both the prover and verifier times independent of N (which opens the doors to efficient schemes with large N). The main issue with this approach given the context of application of MPCitH is the current absence of post-quantum candidates for homomorphic commitment schemes. We also generalize our approach to quasi-threshold LSSS, for which a gap Δ exists between the number of parties ℓ which leak no information and the number of parties $\ell + 1 + \Delta$ necessary to reconstruct the secret. We particularly analyze algebraic geometric quasi-threshold schemes [CC06] but our result is mostly negative: we show that using such schemes does not bring a direct advantage to our framework. We then show that our result on quasi-threshold schemes is still useful in the context of batched proofs (*i.e.* proving simultaneously several statements with a single verification process). We propose a batching technique based on Shamir's secret sharing which enables to efficiently batch proofs in our framework (for a subset of the existing MPCitH schemes).

Finally, we describe some applications of our techniques. We first adapt the SDitH signature scheme [FJR22] to our framework with Shamir's secret sharing. We obtain a variant of this scheme that achieves new interesting size-performance trade-offs. For instance, for a signature size of 10 KB, we obtain a signing time of around 3 ms and a verification time lower than 0.5 ms, which is competitive with SPHINCS$^+$ [ABB+22] in terms of size and verification time while achieving much faster signing. We further apply our batching technique to two different contexts: batched proofs for the SDitH scheme and batched proofs for general arithmetic circuits based on the Limbo proof system [DOT21]. In both cases and for the considered parameters, we obtain an amortized proof size lower than 1/10 of the baseline scheme when batching a few dozen statements, while the amortized performances are also significantly improved (in particular for the verifier).

Related Works. The MPC-in-the-Head paradigm was introduced in the seminal work [IKOS07]. The authors propose general MPCitH constructions relying on MPC protocols in the *semi-honest model* and in the *malicious model*. In the

former case (semi-honest model), they only consider 2-private MPC protocols using an additive sharing as input (they also propose an alternative construction with 1-private protocols). In the latter case (malicious model), they are not restricted to any type of sharing. The exact security of [IKOS07] is analyzed in [GMO16]. As other previous works about the MPCitH paradigm, our work can be seen as a specialization of the IKOS framework. In particular, we restrict the considered MPC model, optimize the communication in this model and provide a refined analysis for the soundness (in the exact security setting) to achieve good practical performances.

To the best of our knowledge, besides [IKOS07], the only previous work which considers MPCitH without relying on an additive secret sharing scheme is Ligero [AHIV17]. Ligero is a practical MPCitH-based zero-knowledge proof system for generic circuits which uses Shamir's secret sharing (or Reed-Solomon codes). The authors consider a particular type of MPC protocol in the *malicious model* and analyze the soundness of the resulting proof system. Ligero achieves sublinear communication cost by packing several witness coordinates in one sharing which is made possible by the use of Shamir's secret sharing.

In comparison, our work formalizes the MPC model on which many recent MPCitH-based schemes (with additive sharing) rely and shows how using LSSS in this model can be beneficial. We consider a slightly more restricted MPC model than the one of Ligero: we impose that the parties only perform linear operations on the sharings. On the other hand, we only need the MPC protocol to be secure in the *semi-honest model* and not in the malicious model as Ligero. In fact, this difference of settings (semi-honest versus malicious) makes our techniques and Ligero's different in nature. While Ligero makes use of proximity tests to get a robust MPC protocol, we can use lighter protocols in our case (since we do not need robustness). Moreover, for a given number of parties and a given privacy threshold, the soundness error of our work is smaller than the one of [AHIV17]. On the other hand, we consider MPC protocols which only performs linear operations on shares which, in the current state of the art, cannot achieve sublinearity. For this reason, our work targets proofs of knowledge for small circuits (for example, to build efficient post-quantum signature schemes) while Ligero remains better for middle-size circuits (thanks to the sublinearity).

Finally, let us cite [DOT21] which is another article providing a refined analysis for the transformation of a general MPC model. The scope of the transformation differs from ours, since it covers $(N-1)$-private MPC protocols using broadcast.

In Table 1, we sum up all the MPC models considered in the state of the art of the MPC-in-the-Head paradigm with the soundness errors and limitations of the general schemes.

Concurrent Work. A concurrent and independent work [AGH+23] proposes an optimization of the MPCitH-based schemes based on additive secret sharing. The authors propose a "hypercube" technique which enables the prover to emulate the entire MPC protocol by performing the computation of only $\log_2 N + 1$ parties instead of N, while keeping the same communication cost. While both

Table 1. Existing general transformations of an MPC protocol into a zero-knowledge proof, with associated MPC model and resulting soundness error. The column "Priv." indicates the privacy threshold of the MPC protocol, while the column "Rob." indicates its robustness threshold. N denotes the number of parties in the MPC protocol, δ denotes the robustness error, and p denotes the false positive rate as defined in this work.

Construction	SharingScheme	Priv.	Rob.	Soundness	Restriction
[IKOS07, Sect. 3]	Additive	2	0	$1 - \frac{1}{\binom{N}{2}}$	–
[IKOS07, Sect. 4]	Any	t	t	When $N = \Omega(t), 2^{-\Omega(t)}$	–
[GMO16]	Any	t	r	$\max\left\{\frac{\binom{r}{t}}{\binom{N}{t}}, \sum_{j=0}^{k} 2^j \frac{\binom{k}{j}\binom{N-2k}{t-j}}{\binom{N}{t}}\right\}$ with $k = \lfloor r/2 \rfloor + 1$	–
[AHIV17]	Any	t	r	$\left(1 - \frac{r}{N}\right)^t + \delta$	Broadcast
[DOT21]	Additive	$N-1$	0	$\frac{1}{N} + p\left(1 - \frac{1}{N}\right)$	Broadcast
Our work, Sect. 4	LSSS	ℓ	0	$\frac{1}{\binom{N}{\ell}} + p\frac{\ell(N-\ell)}{\ell+1}$	BroadcastLinear operations
Our work, Sect. 5.2	LSSS withthresholdgap $\Delta+1$	ℓ	0	$\frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+\Delta+1} \cdot \binom{N-\ell}{\Delta+1}$	BroadcastLinear operations

the hypercube approach and our approach enables to significantly speed up MPCitH-based schemes, they provide different interesting trade-offs and their relative performances shall depend on the context.

2 Preliminaries

Throughout the paper, $\mathbb{F}$ shall denote a finite field. For any $m \in \mathbb{N}^*$, the integer set $\{1, \ldots, m\}$ is denoted $[m]$. For a probability distribution D, the notation $s \leftarrow D$ means that s is sampled from D. For a finite set S, the notation $s \leftarrow S$ means that s is uniformly sampled at random from S. For an algorithm $\mathcal{A}$, $out \leftarrow \mathcal{A}(in)$ further means that out is obtained by a call to $\mathcal{A}$ on input in (using uniform random coins whenever $\mathcal{A}$ is probabilistic). Along the paper, probabilistic polynomial time is abbreviated PPT.

In this paper, we shall use the standard cryptographic notions of indistinguishability, secure pseudo-random generator (PRG), tree PRG, collision-resistant hash function, (hiding and binding) commitment scheme, (honest verifier) zero-knowledge proof of knowledge and secure multiparty computation protocols (in the semi-honest model). Those notions are formally recalled in the full version [FR22]. We recall hereafter the definition of (quasi-)threshold linear secret sharing.

Along the paper, the sharing of a value s is denoted $[\![s]\!] := ([\![s]\!]_1, \ldots, [\![s]\!]_N)$ with $[\![s]\!]_i$ denoting the share of index i for every $i \in [N]$. For any subset of indices $J \subseteq [N]$, we shall further denote $[\![s]\!]_J := \left([\![s]\!]_i\right)_{i \in J}$.

Definition 1 (Threshold LSSS). *Let $\mathbb{F}$ be a finite field and let $\mathbb{V}_1$ and $\mathbb{V}_2$ be two vector spaces over $\mathbb{F}$. Let t and N be integers such that $1 < t \leq N$. A (t, N)-*threshold linear secret sharing scheme *is a method to share a secret $s \in \mathbb{V}_1$ into N shares $[\![s]\!] := ([\![s]\!]_1, \ldots, [\![s]\!]_N) \in \mathbb{V}_2^N$ such that the secret can be reconstructed from any t shares while no information is revealed on the secret from the knowledge of $t-1$ shares.*

Formally, an (t, N)-threshold LSSS consists of a pair of algorithms:

$$\begin{cases} \mathsf{Share} : \mathbb{V}_1 \times R \mapsto \mathbb{V}_2^N \\ \mathsf{Reconstruct}_J : \mathbb{V}_2^t \mapsto \mathbb{V}_1 \end{cases}$$

where $R \subseteq \{0,1\}^*$ denotes some randomness space and where $\mathsf{Reconstruct}_J$ is indexed by a set (and defined for every) $J \subset [N]$ such that $|J| = t$. This pair of algorithms satisfies the three following properties:

1. ***Correctness:*** *for every $s \in \mathbb{V}_1$, $r \in R$, and $J \subset [N]$ s.t. $|J| = t$, and for $[\![s]\!] \leftarrow \mathsf{Share}(s; r)$, we have:*

$$\mathsf{Reconstruct}_J([\![s]\!]_J) = s.$$

2. ***Perfect $(t-1)$-privacy:*** *for every $s_0, s_1 \in \mathbb{V}_1$ and $I \subset [N]$ s.t. $|I| = t-1$, the two distributions*

$$\left\{ [\![s_0]\!]_I \mid \begin{array}{c} r \leftarrow R \\ [\![s_0]\!] \leftarrow \mathsf{Share}(s_0; r) \end{array} \right\} \quad \text{and} \quad \left\{ [\![s_1]\!]_I \mid \begin{array}{c} r \leftarrow R \\ [\![s_1]\!] \leftarrow \mathsf{Share}(s_1; r) \end{array} \right\}$$

   are perfectly indistinguishable.

3. ***Linearity:*** *for every $v_0, v_1 \in \mathbb{V}_2^t$, $\alpha \in \mathbb{F}$, and $J \subset [N]$ s.t. $|J| = t$,*

$$\mathsf{Reconstruct}_J(\alpha \cdot v_0 + v_1) = \alpha \cdot \mathsf{Reconstruct}_J(v_0) + \mathsf{Reconstruct}_J(v_1).$$

Definition 2 (Quasi-Threshold LSSS). *Let $\mathbb{F}$ be a finite field and let $\mathbb{V}_1$ and $\mathbb{V}_2$ be two vector spaces over $\mathbb{F}$. Let t_1, t_2 and N be integers such that $1 \leq t_1 < t_2 \leq N$. A (t_1, t_2, N)-*quasi-threshold linear secret sharing scheme *is a method to share a secret $s \in \mathbb{V}_1$ into N shares $[\![s]\!] := ([\![s]\!]_1, \ldots, [\![s]\!]_N) \in \mathbb{V}_2^N$ such that the secret can be reconstructed from any t_2 shares while no information is revealed on the secret from the knowledge of t_1 shares.*

The formal definition of (t_1, t_2, N)-quasi-threshold LSSS is similar to Definition 1 with the $\mathsf{Reconstruct}_J$ function defined over $\mathbb{V}_2^{t_2}$ (instead of $\mathbb{V}_2^t$) with cardinalities $|I| = t_1$ and $|J| = t_2$ (instead of $|I| = t-1$ and $|J| = t$). In particular an $(t-1, t, N)$-quasi-threshold LSSS is an (t, N)-threshold LSSS.

Definition 3 (Additive Secret Sharing). *An additive secret sharing scheme over $\mathbb{F}$ is an (N, N)-threshold LSSS for which the Share algorithm is defined as*

$$\mathsf{Share} : \big(s\,;\, (r_1, \ldots, r_{N-1})\big) \mapsto [\![s]\!] := \Big(r_1, \ldots r_{N-1},\, s - \sum_{i=1}^{N-1} r_i\Big),$$

with randomness space $R = \mathbb{F}^{N-1}$, and the $\mathsf{Reconstruct}_{[N]}$ algorithm simply outputs the sum of all the input shares.

Definition 4 (Shamir's Secret Sharing). *The Shamir's Secret Sharing over $\mathbb{F}$ is an $(\ell+1, N)$-threshold LSSS for which the Share algorithm builds a sharing $[\![s]\!]$ of $s \in \mathbb{F}$ as follows:*

- *sample* $r_1, \ldots, r_\ell$ *uniformly in* $\mathbb{F}$,
- *build the polynomial* P *as* $P(X) := s + \sum_{i=1}^{\ell} r_i X^i$,
- *build the shares* $[\![s]\!]_i$ *as evaluations* $P(e_i)$ *of* P *for each* $i \in \{1, \ldots, N\}$, *where* $e_1, \ldots, e_N$ *are public non-zero distinct points of* $\mathbb{F}$.

For any subset $J \subseteq [N]$, *s.t.* $|J| = \ell + 1$, *the* $\mathsf{Reconstruct}_J$ *algorithm interpolates the polynomial* P *from the input* $\ell + 1$ *evaluation points* $[\![s]\!]_J = (P(e_i))_{i \in J}$ *and outputs the constant term* s.

3 The MPC-in-the-Head Paradigm

The MPC-in-the-Head (MPCitH) paradigm is a framework introduced by Ishai, Kushilevitz, Ostrovsky and Sahai in [IKOS07] to build zero-knowledge proofs using techniques from secure multi-party computation (MPC). We first recall the general principle of this paradigm before introducing a formal model for the underlying MPC protocols and their transformation into zero-knowledge proofs.

Assume we want to build a zero-knowledge proof of knowledge of a witness w for a statement x such that $(x, w) \in \mathcal{R}$ for some relation $\mathcal{R}$. To proceed, we shall use an MPC protocol in which N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ securely and correctly evaluate a function f on a secret witness w with the following properties:

- each party $\mathcal{P}_i$ takes a share $[\![w]\!]_i$ as input, where $[\![w]\!]$ is a sharing of w;
- the function f outputs Accept when $(x, w) \in \mathcal{R}$ and Reject otherwise;
- the protocol is ℓ-private in the semi-honest model, meaning that the views of any ℓ parties leak no information about the secret witness.

We can use this MPC protocol to build a zero-knowledge proof of knowledge of a witness w satisfying $(x, w) \in \mathcal{R}$. The prover proceeds as follows:

- she builds a random sharing $[\![w]\!]$ of w;
- she simulates locally ("in her head") all the parties of the MPC protocol;
- she sends a commitment of each party's view to the verifier, where such a view includes the party's input share, its random tape, and its received messages (the sent messages can further be deterministically derived from those elements);
- she sends the output shares $[\![f(w)]\!]$ of the parties, which should correspond to a sharing of Accept.

Then the verifier randomly chooses ℓ parties and asks the prover to reveal their views. After receiving them, the verifier checks that they are consistent with an honest execution of the MPC protocol and with the commitments. Since only ℓ parties are opened, the revealed views leak no information about the secret witness w, which ensures the zero-knowledge property. On the other hand, the random choice of the opened parties makes the cheating probability upper bounded[1] by $1 - \binom{N-2}{\ell-2} / \binom{N}{\ell}$, which ensures the soundness of the proof.

[1] The optimal strategy for a malicious prover is to have an inconsistency only between two parties. The soundness error is thus the probability that these two parties are not simultaneously in the set of the ℓ opened views.

The MPCitH paradigm simply requires the underlying MPC protocol to be secure in the semi-honest model (and not in the malicious model), meaning that the parties are assumed to be honest but curious: they follow honestly the MPC protocol while trying to learn secret information from the received messages.

Several simple MPC protocols have been proposed that yield fairly efficient zero-knowledge proofs and signature schemes in the MPCitH paradigm, see for instance [KZ20b,BD20,BDK+21,FJR22]. These protocols lie in a specific subclass of MPC protocols in the semi-honest model which we formalize hereafter.

3.1 General Model of MPC Protocol with Additive Sharing

We consider a passively-secure MPC protocol that performs its computation on a base finite field $\mathbb{F}$ so that all the manipulated variables (including the witness w) are tuples of elements from $\mathbb{F}$. In what follows, the sizes of the different tuples involved in the protocol are kept implicit for the sake of simplicity. The parties take as input an additive sharing $[\![w]\!]$ of the witness w (one share per party). Then the parties compute one or several rounds in which they perform three types of actions:

Receiving randomness: the parties receive a random value (or random tuple) ε from a randomness oracle $\mathcal{O}_R$. When calling this oracle, all the parties get the same random value ε. This might not be convenient in a standard multi-party computation setting (since such an oracle would require a trusted third party or a possibly complex coin-tossing protocol), but in the MPCitH context, these random values are provided by the verifier as challenges.

Receiving hint: the parties can receive a sharing $[\![\beta]\!]$ (one share per party) from a hint oracle $\mathcal{O}_H$. The hint β can depend on the witness w and the previous random values sampled from $\mathcal{O}_R$. Formally, for some function ψ, the hint is sampled as $\beta \leftarrow \psi(w, \varepsilon^1, \varepsilon^2, \ldots; r)$ where $\varepsilon^1, \varepsilon^2, \ldots$ are the previous outputs of $\mathcal{O}_R$ and where r is a fresh random tape.

Computing & broadcasting: the parties can locally compute $[\![\alpha]\!] := [\![\varphi(v)]\!]$ from a sharing $[\![v]\!]$ where φ is an $\mathbb{F}$-linear function, then broadcast all the shares $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ to publicly reconstruct $\alpha := \varphi(v)$. If φ is in the form $v \mapsto Av + b$, then the parties can compute $[\![\varphi(v)]\!]$ from $[\![v]\!]$ by letting

$$[\![\varphi(v)]\!]_i := A[\![v]\!]_i + [\![b]\!]_i \quad \text{for each party } i$$

where $[\![b]\!]$ is a publicly-known sharing of b.[2] This process is usually denoted $[\![\varphi(v)]\!] = \varphi([\![v]\!])$. The function φ can depend on the previous random values $\{\varepsilon^i\}_i$ from $\mathcal{O}_R$ and on the previous broadcasted values.

After t rounds of the above actions, the parties finally output Accept if and only if the publicly reconstructed values $\alpha^1, \ldots, \alpha^t$ satisfy the relation

$$g(\alpha^1, \ldots, \alpha^t) = 0$$

[2] Usually, $[\![b]\!]$ is chosen as $(b, 0, \ldots, 0)$ in the case of the additive sharing.

for a given function g.

Protocol 1 gives a general description of an MPC protocol in this paradigm, which we shall use as a model in the rest of the paper. In general, the computing & broadcasting step can be composed of several iterations, which is further depicted in Protocol 2. For the sake of simplicity, we shall consider a single iteration in our presentation (as in Protocol 1) but we stress that the considered techniques and proofs equally apply to the multi-iteration setting (*i.e.* while replacing step (c) of Protocol 1 by Protocol 2).

1. The parties take as input a sharing $[\![w]\!]$.
2. For $j = 1$ to t, the parties:
   (a) get a sharing $[\![\beta^j]\!]$ from the hint oracle $\mathcal{O}_H$, such that
   $$\beta^j \leftarrow \psi^j(w, \varepsilon^1, \ldots, \varepsilon^{j-1}; r^j)$$
   for a uniform random tape r^j;
   (b) get a common random ε^j from the oracle $\mathcal{O}_R$;
   (c) for some $\mathbb{F}$-linear function $\varphi^j_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j}}$, compute
   $$[\![\alpha^j]\!] := \varphi^j_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j}}\left([\![w]\!], ([\![\beta^i]\!])_{i\le j}\right) ,$$
   broadcast $[\![\alpha^j]\!]$, and then publicly reconstruct α^j. *Note: This step can be composed of several iterations as described in Protocol 2.*
3. The parties finally accept if $g(\alpha^1, \ldots, \alpha^t) = 0$ and reject otherwise.

Note: In the above description w, β^j, ε^j, α^j are elements from the field $\mathbb{F}$ or tuples with coordinates in $\mathbb{F}$ (whose size is not made explicit to keep the presentation simple).

Protocol 1: General MPC protocol Π_{add}.

Output Distribution. In the following, we shall denote $\vec{\varepsilon} := (\varepsilon^1, \ldots, \varepsilon^t)$, $\vec{\beta} := (\beta^1, \ldots, \beta^t)$, $\vec{\alpha} := (\alpha^1, \ldots, \alpha^t)$ and $\vec{r} := (r^1, \ldots, r^t)$. From the above description, we have that the output of the protocol deterministically depends on the broadcasted values $\vec{\alpha}$ (through the function g), which in turn deterministically depend on the input witness w, the sampled random values $\vec{\varepsilon}$, and the hints $\vec{\beta}$ (through the functions φ's). It results that the functionality computed by the protocol can be expressed as:

$$f(w, \vec{\varepsilon}, \vec{\beta}) = \begin{cases} \textsc{Accept} & \text{if } g(\vec{\alpha}) = 0, \\ \textsc{Reject} & \text{otherwise,} \end{cases} \qquad \text{with } \vec{\alpha} = \Phi(w, \vec{\varepsilon}, \vec{\beta}), \tag{1}$$

(c) for $k = 1$ to η_j:
- compute a sharing

$$[\![\alpha^{j,k}]\!] := \varphi^{j,k}_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j},(\alpha^{j,i})_{i<k}}\left([\![w]\!],([\![\beta^i]\!])_{i\le j}\right),$$

  for some $\mathbb{F}$-linear function $\varphi^{j,k}_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j},(\alpha^{j,i})_{i<k}}$;
- broadcast their shares $[\![\alpha^{j,k}]\!]$;
- publicly reconstruct $\alpha^{j,k}$;

We denote $\alpha^j := (\alpha^{j,1}, \ldots, \alpha^{j,\eta_j})$.

Protocol 2: General MPC protocol Π_{add} – Iterative computing & broadcasting step for iteration j (with η_j denoting the number of inner iterations).

where Φ is the deterministic function mapping $(w, \vec{\varepsilon}, \vec{\beta})$ to $\vec{\alpha}$ (defined by the coordinate functions $\varphi^1, \ldots, \varphi^t$). We shall restrict our model to MPC protocols for which the function f satisfies the following properties:

- If w is a *good witness*, namely w is such that $(x, w) \in \mathcal{R}$, and if the hints $\vec{\beta}$ are genuinely sampled as $\beta^j \leftarrow \psi^j(w, (\varepsilon^i)_{i<j}; r^j)$ for every j, then the protocol always accepts. More formally:

$$\Pr_{\vec{\varepsilon},\vec{r}}\left[f(w,\vec{\varepsilon},\vec{\beta}) = \textsc{Accept} \;\middle|\; \begin{array}{c}(x,w)\in\mathcal{R}\\ \forall j, \beta^j \leftarrow \psi^j(w,(\varepsilon^i)_{i<j};r^j)\end{array}\right] = 1.$$

- If w is a *bad witness*, namely w is such that $(x, w) \notin \mathcal{R}$, then the protocol rejects with probability at least $1 - p$, for some constant probability p. The latter holds even if the hints $\vec{\beta}$ are not genuinely computed. More formally, for any (adversarially chosen) deterministic functions $\chi^1, \ldots, \chi^t$, we have:

$$\Pr_{\vec{\varepsilon},\vec{r}}\left[f(w,\vec{\varepsilon},\vec{\beta}) = \textsc{Accept} \;\middle|\; \begin{array}{c}(x,w)\notin\mathcal{R}\\ \forall j, \beta^j \leftarrow \chi^j(w,(\varepsilon^i)_{i<j};r^j)\end{array}\right] \le p.$$

We say that a *false positive* occurs whenever the MPC protocol outputs Accept on input a bad witness w, and we call p the *false positive rate*.

The general MPC model introduced above captures a wide majority of the protocols used in the MPCitH context, such as [KKW18, DDOS19, KZ20b, BFH+20, BN20, BD20, DOT21, BDK+21, DKR+21, KZ22, FJR22, FMRV22]. To the best of our knowledge, our model applies to all the MPCitH schemes in the literature except those derived from the ZKBoo [GMO16] and Ligero [AHIV17] proof systems. An example of protocol fitting this model is the [BN20] protocol where the random multiplication (Beaver) triples to sacrifice are given by the hint oracle. Another example is Limbo [DOT21] for which the hint oracle $\mathcal{O}_H$ corresponds to the untrusted subroutines $\Pi_{\mathsf{InnerProd}}$ and Π_{Rand} while the randomness oracle $\mathcal{O}_R$ is RandomCoin.

3.2 Application of the MPCitH Principle

Any MPC protocol complying with the above description gives rise to a practical short-communication zero-knowledge protocol in the MPCitH paradigm. The resulting zero-knowledge protocol is described in Protocol 3: after sharing the witness w, the prover emulates the MPC protocol "in her head", commits the parties' inputs, and sends a hash digest of the broadcast communications; finally, the prover reveals the requested parties' inputs as well as the broadcast messages of the unopened party, thus enabling the verifier to emulate the computation of the opened parties and to check the overall consistency.

Soundness. Assuming that the underlying MPC protocol follows the model of Sect. 3.1 with a false positive rate p, the soundness error of Protocol 3 is

$$\frac{1}{N} + \left(1 - \frac{1}{N}\right) \cdot p.$$

The above formula results from the fact that a malicious prover might successfully cheat with probability $1/N$ by corrupting the computation of one party or with probability p by making the MPC protocol produce a false positive. This soundness has been formally proven in some previous works, see *e.g.* [DOT21, BN20, FJR22]. In the present article, we provide a general proof for any protocol complying with the format of Protocol 1 in the more general context of any (threshold) linear secret sharing (see Theorem 2).

Performances. The communication of Protocol 3 includes:

- the input shares $([\![w]\!]_i, [\![\beta^1]\!]_i, \ldots, [\![\beta^t]\!]_i)$ of the opened parties. In practice, a seed $\mathsf{seed}_i \in \{0,1\}^\lambda$ is associated to each party so that for each committed variable v (among the witness w and the hints $\beta^1, \ldots, \beta^t$) the additive sharing $[\![v]\!]$ is built as
$$\begin{cases} [\![v]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i) \text{ for } i \neq N \\ [\![v]\!]_N = v - \sum_{i=1}^{N-1} [\![v]\!]_i. \end{cases}$$
Thus, instead of committing $([\![w]\!]_i, [\![\beta^1]\!]_i)$, the initial commitments simply include the seeds for $i \neq N$, and com_i^j becomes useless for $j \geq 2$ and $i \neq N$. Formally, we have:
$$\mathsf{com}_i^j = \begin{cases} \mathrm{Com}(\mathsf{seed}_i; \rho_i^1) & \text{for } j = 1 \text{ and } i \neq N \\ \mathrm{Com}([\![w]\!]_N, [\![\beta^1]\!]_N; \rho_N^1) & \text{for } j = 1 \text{ and } i = N \\ \emptyset & \text{for } j > 1 \text{ and } i \neq N \\ \mathrm{Com}([\![\beta^j]\!]_N; \rho_N^j) & \text{for } j > 1 \text{ and } i = N \end{cases}$$
Some coordinates of the β^j might be uniformly distributed over $\mathbb{F}$ (remember that the β^j are tuples of $\mathbb{F}$ elements). We denote β^{unif} the sub-tuple composed of those uniform coordinates. In this context, the last share $[\![\beta^{\mathrm{unif}}]\!]_N$ can be built as $[\![\beta^{\mathrm{unif}}]\!]_N \leftarrow \mathrm{PRG}(\mathsf{seed}_N)$ so that a seed seed_N can be committed

1. The prover shares the witness w into a sharing $[\![w]\!]$.
2. The prover emulates "in her head" the N parties of the MPC protocol.
   For $j = 1$ to t:
   (a) the prover computes
   $$\beta^j = \psi^j(w, (\varepsilon^i)_{i<j}),$$
   shares it into a sharing $[\![\beta^j]\!]$;
   (b) the prover computes the commitments
   $$\mathsf{com}_i^j := \begin{cases} \mathrm{Com}([\![w]\!]_i, [\![\beta^1]\!]_i; \rho_i^1) & \text{if } j = 1 \\ \mathrm{Com}([\![\beta^j]\!]_i; \rho_i^j) & \text{if } j > 1 \end{cases}$$
   for all $i \in \{1, \ldots, N\}$, for some commitment randomness ρ_i^j;
   (c) the prover sends
   $$h_j := \begin{cases} \mathrm{Hash}(\mathsf{com}_1^1, \ldots, \mathsf{com}_N^1) & \text{if } j = 1 \\ \mathrm{Hash}(\mathsf{com}_1^j, \ldots, \mathsf{com}_N^j, [\![\alpha^{j-1}]\!]) & \text{if } j > 1 \end{cases}$$
   to the verifier;
   (d) the verifier picks at random a challenge ε^j and sends it to the prover;
   (e) the prover computes
   $$[\![\alpha^j]\!] := \varphi^j_{(\varepsilon^i)_{i\le j}, (\alpha^i)_{i<j}}\big([\![w]\!], ([\![\beta^i]\!])_{i\le j}\big)$$
   and recomposes α^j.
   Note: This step is computed according to Protocol 2 in case of an iterative computing & broadcasting step.
   The prover further computes $h_{t+1} := \mathrm{Hash}([\![\alpha^t]\!])$ and sends it to the verifier.
3. The verifier picks at random a party index $i^* \in [N]$ and sends it to the prover.
4. The prover opens the commitments of all the parties except party i^* and further reveals the commitments and broadcast messages of the unopened party i^*. Namely, the prover sends $([\![w]\!]_i, ([\![\beta^j]\!]_i, \rho_i^j)_{j\in[t]})_{i\neq i^*}$, $\mathsf{com}_{i^*}^1, \ldots, \mathsf{com}_{i^*}^t$, $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ to the verifier.
5. The verifier recomputes the commitments com_i^j and the broadcast values $[\![\alpha^j]\!]_i$ for $i \in [N] \setminus \{i^*\}$ and $j \in [t]$ from $([\![w]\!]_i, ([\![\beta^j]\!]_i, \rho_i^j)_{j\in[t]})_{i\neq i^*}$ in the same way as the prover.
6. The verifier accepts if and only if:
   (a) the views of the opened parties are consistent with each other, with the committed input shares and with the hash digest of the broadcast messages, *i.e.* for $j = 1$ to $t+1$,
   $$h_j \stackrel{?}{=} \begin{cases} \mathrm{Hash}(\mathsf{com}_1^1, \ldots, \mathsf{com}_N^1) & \text{if } j = 1 \\ \mathrm{Hash}(\mathsf{com}_1^j, \ldots, \mathsf{com}_N^j, [\![\alpha^{j-1}]\!]) & \text{if } j > 1 \\ \mathrm{Hash}([\![\alpha^t]\!]) & \text{if } j = t+1 \end{cases}$$
   (b) the output of the MPC protocol is Accept, *i.e.*
   $$g(\alpha^1, \ldots, \alpha^t) \stackrel{?}{=} 0.$$

Protocol 3: Zero-knowledge protocol - Application of the MPCitH principle to Protocol 1.

in com_N^1 (instead of committing $[\![\beta^{\text{unif}}]\!]_N$). This way the prover can save communication by revealing seed_N instead of $[\![\beta^{\text{unif}}]\!]_N$ whenever the latter is larger;
- the messages $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ broadcasted by the unopened party. Let us stress that one can sometimes save communication by sending only some elements of $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ and use the relation $g(\alpha^1, \ldots, \alpha^t) = 0$ to recover the missing ones;
- the hash digests $h_1, \ldots, h_{t+1}$ and the unopened commitments $\mathsf{com}_{i^*}^1, \ldots, \mathsf{com}_{i^*}^t$ (as explained above, we have $\mathsf{com}_{i^*}^j = \emptyset$ for $j > 1$ if $i^* \neq N$).

Moreover, instead of revealing the $(N-1)$ seeds of the opened parties, one can generate them from a generation tree as suggested in [KKW18]. One then only needs to reveal $\log_2 N$ λ-bit seeds. We finally obtain a total communication cost for Protocol 3 of

- when $i^* \neq N$,

$$\mathsf{Cost} = \underbrace{(t+1)\cdot 2\lambda}_{h_1,h_2,\ldots,h_{t+1}} + \Big(\underbrace{\mathsf{inputs}}_{[\![w]\!]_N, [\![\beta^1]\!]_N, \ldots,} + \underbrace{\mathsf{comm}}_{[\![\alpha^1]\!]_{i^*},\ldots,[\![\alpha^t]\!]_{i^*}} + \underbrace{\lambda \cdot \log_2 N}_{\mathsf{seed}_i \text{ for } i \neq i^*} + \underbrace{2\lambda}_{\mathsf{com}_{i^*}^1} \Big).$$

- when $i^* = N$,

$$\mathsf{Cost} = \underbrace{(t+1)\cdot 2\lambda}_{h_1,h_2,\ldots,h_{t+1}} + \Big(\underbrace{\mathsf{comm}}_{[\![\alpha^1]\!]_{i^*},\ldots,[\![\alpha^t]\!]_{i^*}} + \underbrace{\lambda \cdot \log_2 N}_{\mathsf{seed}_i \text{ for } i \neq i^*} + \underbrace{t \cdot 2\lambda}_{\mathsf{com}_{i^*}^1,\ldots,\mathsf{com}_{i^*}^t} \Big).$$

where inputs denote the bitsize of $(w, \beta^1, \ldots, \beta^t)$ excluding the uniformly distributed elements β^{unif}, and where comm denotes the bitsize of $(\alpha^1, \ldots, \alpha^t)$ excluding the elements which can be recovered from $g(\alpha^1, \ldots, \alpha^t) = 0$.

To achieve a soundness error of $2^{-\lambda}$, one must repeat the protocol $\tau = \frac{\lambda}{\log_2 N}$ times. The resulting averaged cost is the following:

$$\mathsf{Cost} = (t+1)\cdot 2\lambda + \tau \cdot \left(\frac{N-1}{N} \cdot \mathsf{inputs} + \mathsf{comm} + \lambda \cdot \log_2 N + \frac{N-1+t}{N} \cdot 2\lambda \right).$$

Several recent works based on the MPCitH paradigm [BD20,KZ21,FJR22] provides zero-knowledge identification protocols with communication cost below 10 KB for a 128-bit security level. Unfortunately, to obtain a small communication cost, one must take a large number of parties N, which induces an important computational overhead compared to other approaches to build zero-knowledge proofs. Indeed, the prover must emulate N parties in her head for each of the τ repetitions of the protocol, which makes a total of $\frac{\lambda N}{\log_2 N}$ party emulations to achieve a soundness error of $2^{-\lambda}$. Thus, increasing N has a direct impact on the performances. For instance, scaling from $N = 16$ to $N = 256$ roughly halves the communication but increases the computation by a factor of eight. Given this state of affairs, a natural question is the following:

Can we build zero-knowledge proofs in the MPC-in-the-head paradigm while avoiding this computational overhead?

In what follows, we show how applying (low-threshold) linear secret sharing to the MPCitH paradigm provides a positive answer to this question.

4 MPC-in-the-Head with Threshold LSS

4.1 General Principle

Let ℓ and N be integers such that $1 \leq \ell < N$. We consider an $(\ell+1, N)$-threshold linear secret sharing scheme (LSSS), as formally introduced in Definition 1, which shares a secret $s \in \mathbb{F}$ into N shares $[\![s]\!] \in \mathbb{F}^N$. In particular, the vector spaces of Definition 1 are simply defined as $\mathbb{V}_1 = \mathbb{V}_2 = \mathbb{F}$ hereafter (other definitions of these sets will be considered in Sect. 5). We recall that such a scheme implies that the secret can be reconstructed from any $\ell + 1$ shares while no information is revealed on the secret from the knowledge of ℓ shares. The following lemmas shall be useful to our purpose (see proofs in the full version [FR22]). The first lemma holds assuming the MDS conjecture [MS10] while the second one comes from the equivalence between threshold LSSS and interpolation codes [CDN15, Theorem 11.103].

Lemma 1. *Let $\mathbb{F}$ be a finite field and let ℓ, N be integers such that $1 \leq \ell < N-1$. If an $(\ell+1, N)$-threshold LSSS exists for $\mathbb{F}$, and assuming the MDS conjecture, then $N \leq |\mathbb{F}|$ with the following exception: if $|\mathbb{F}|$ is a power of 2 and $\ell \in \{2, |\mathbb{F}|-2\}$ then $N \leq |\mathbb{F}| + 1$.*

Lemma 2. *Let (Share, Reconstruct) be an $(\ell+1, N)$-threshold LSSS. For every tuple $v_0 \in \mathbb{V}_2^{\ell+1}$ and every subset $J_0 \subseteq [N]$ with $|J_0| = \ell+1$, there exists a unique sharing $[\![s]\!] \in \mathbb{V}_2^N$ such that $[\![s]\!]_{J_0} = v_0$ and such that*

$$\forall J \text{ s.t. } |J| = \ell + 1, \mathsf{Reconstruct}_J([\![s]\!]_J) = s,$$

where $s := \mathsf{Reconstruct}_{J_0}(v_0)$. Moreover, there exists an efficient algorithm Expand_{J_0} which returns this unique sharing from $[\![s]\!]_{J_0}$.

In the rest of the paper we shall frequently use the following notions:

- **Sharing of a tuple.** If v is a tuple, a secret sharing $[\![v]\!]$ is defined coordinate-wise. The algorithms Share, Reconstruct and Expand (from Lemma 2) further apply coordinate-wise.
- **Valid sharing.** We say that a sharing $[\![v]\!]$ is *valid* when there exists v such that

$$\forall J \text{ s.t. } |J| = \ell + 1, \mathsf{Reconstruct}_J([\![v]\!]_J) = v,$$

or equivalently[3], when there exists J such that $[\![v]\!] = \mathsf{Expand}_J([\![v]\!]_J)$.

[3] This second formulation is true only for threshold schemes (and not for quasi-threshold schemes that we will introduce latter).

– **Consistent shares.** We say that shares $[\![v]\!]_{i_1}, \ldots, [\![v]\!]_{i_z}$ are *consistent* when there exist other shares $[\![v]\!]_{[N]\setminus\{i_1,\ldots,i_z\}}$ such that $[\![v]\!]$ is a valid sharing.

Application to the MPCitH Paradigm. We suggest applying a threshold LSSS to the MPCitH paradigm instead of a simple additive sharing scheme. Let us consider a protocol Π_{add} complying with the MPC model introduced in the previous section (Protocol 1). We can define a protocol Π_{LSSS} similar to Π_{add} with the following differences:

– the parties initially receive an $(\ell+1, N)$-threshold linear secret sharing of the witness w,
– when invoked for a hint β_j, the oracle $\mathcal{O}_H$ returns an $(\ell+1, N)$-threshold linear secret sharing of β^j,
– when the shares of α^j are broadcasted, the value α^j is reconstructed using the algorithm $\mathsf{Reconstruct}$. Namely, the parties arbitrarily choose $\ell+1$ shares $([\![\alpha^j]\!]_i)_{i\in J_0}$, run the algorithm $\mathsf{Reconstruct}_{J_0}$ to get α^j, and check that all the broadcast shares are consistent with the output of Expand_{J_0}. If the check fails, the protocol returns Reject.

The resulting MPC protocol, formally described in Protocol 4, is well-defined and ℓ-private in the semi-honest model (meaning that the views of any ℓ parties leak no information about the secret). This is formalized in the following theorem (see proof in the full version [FR22]).

Theorem 1. *Let us consider an MPC protocol Π_{add} complying with the protocol format described in Protocol 1. If Π_{add} is well-defined and $(N-1)$-private, then the protocol Π_{LSSS} corresponding to Π_{add} with an $(\ell+1, N)$-threshold linear secret sharing scheme (see Protocol 4) is well-defined and ℓ-private.*

4.2 Conversion to Zero-Knowledge Proofs

We can convert the MPC protocol using threshold linear secret sharings into a zero-knowledge protocol using the MPC-in-the-Head paradigm. Instead of requesting the views of $N-1$ parties, the verifier only asks for the views of ℓ parties. Since the MPC protocol is ℓ-private, we directly get the zero-knowledge property. One key advantage of using a threshold LSSS is that only $\ell+1$ parties out of N need to be computed by the prover, which we explain further hereafter.

Besides the commitments on the input sharing $[\![w]\!]$, and the hints' sharings $[\![\beta^1]\!], \ldots, [\![\beta^t]\!]$, the prover must send to the verifier the communication between the parties, which for the considered MPC model (see Protocol 4) consists of the broadcast sharings $[\![\alpha^1]\!], \ldots, [\![\alpha^t]\!]$. Observe that such a sharing $[\![\alpha^j]\!]$ is also an LSSS sharing of the underlying value α^j since it is computed as

$$[\![\alpha^j]\!] := \varphi^j_{(\varepsilon^i,\alpha^i)_{i\le j}}\left([\![w]\!], ([\![\beta^i]\!])_{i\le j}\right)$$

where $[\![w]\!], [\![\beta^1]\!], \ldots, [\![\beta^t]\!]$ are LSSS sharings and φ^j is an affine function. This notably implies that, for all i, the broadcast sharing $[\![\alpha^j]\!] = ([\![\alpha^j]\!]_1, \ldots, [\![\alpha^j]\!]_N)$

1. The parties take as input an $(\ell+1, N)$-threshold linear sharing $[\![w]\!]$.
2. For $j = 1$ to t, the parties:
   (a) get an $(\ell+1, N)$-threshold linear sharing $[\![\beta^j]\!]$ from the hint oracle $\mathcal{O}_H$, such that
   $$\beta^j \leftarrow \psi^j(w, \varepsilon^1, \ldots, \varepsilon^{j-1}; r^j)$$
   for a uniform random tape r^j;
   (b) get a common random ε^j from the oracle $\mathcal{O}_R$;
   (c) for some $\mathbb{F}$-linear function $\varphi^j_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j}}$,
   – compute
   $$[\![\alpha^j]\!] := \varphi^j_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j}}\big([\![w]\!], ([\![\beta^i]\!])_{i\le j}\big) \ ,$$
   – broadcast $[\![\alpha^j]\!]$,
   – compute
   $$\alpha^j := \mathsf{Reconstruct}_{J_0}([\![\alpha^j]\!]_{J_0})$$
   for some J_0 of size $\ell + 1$,
   – verify that $\mathsf{Expand}_{J_0}([\![\alpha]\!]_{J_0})$ is consistent with $[\![\alpha^j]\!]$ (*i.e.* that $[\![\alpha^j]\!]$ forms a valid sharing) and reject otherwise.

   Note: This step can be composed of several iterations as described in Protocol 2.
3. The parties finally accept if $g(\alpha^1, \ldots, \alpha^t) = 0$ and reject otherwise.

Note: In the above description w, β^j, ε^j, α^j are elements from the field $\mathbb{F}$ or tuples with coordinates in $\mathbb{F}$ (whose size is not made explicit to keep the presentation simple).

Protocol 4: General MPC protocol Π_{LSSS} with LSSS.

contains redundancy. According to Lemma 2, in order to uniquely define such a sharing, one only needs to commit $\ell + 1$ shares of $[\![\alpha^j]\!]$. In other words, we can choose a fixed subset S of $\ell + 1$ parties and only commit the broadcast shares from these parties, which then acts as a commitment of the full sharing $[\![\alpha^j]\!]$. For all $j \in [t]$, the prover needs to send the broadcast share $[\![\alpha^j]\!]_{i^*}$ of an arbitrary unopened party i^*. To verify the computation of the ℓ opened parties $I = \{i_1, \ldots, i_\ell\} \subseteq [N]$, the verifier can recompute the shares $[\![\alpha^j]\!]_{i_1}, \ldots, [\![\alpha^j]\!]_{i_\ell}$. Then, from these ℓ shares together with $[\![\alpha^j]\!]_{i^*}$, the verifier can reconstruct the shares $[\![\alpha^j]\!]_S$ using $\mathsf{Expand}_{\{i^*,i_1,\ldots,i_\ell\}}$ and check their commitments.

By committing the broadcast messages of only a subset S of parties, the proof becomes independent of the computation of the other parties. It means

that the prover must commit the input shares of all the parties but only need to emulate $\ell+1$ parties to commit their broadcast shares. When ℓ is small with respect to N, this has a great impact on the computational performance of the prover. The resulting zero-knowledge protocol is described in Protocol 5.

4.3 Soundness

Consider a malicious prover $\tilde{\mathcal{P}}$ who does not know a correct witness w for the statement x but still tries to convince the verifier that she does. We shall say that such a malicious prover cheats for some party $i \in [N]$ if the broadcast shares $[\![\alpha^1]\!]_i, \ldots, [\![\alpha^t]\!]_i$ recomputed from the committed input/hint shares $[\![w]\!]_i$, $[\![\beta^1]\!]_i, \ldots, [\![\beta^t]\!]_i$ are not consistent with the committed broadcast shares $([\![\alpha^1]\!]_S, \ldots, [\![\alpha^t]\!]_S)$.

Let us first consider the simple case of false positive rate $p = 0$. If a malicious prover cheats on less than $N-\ell$ parties, then at least $\ell+1$ parties have broadcast shares which are consistent with $([\![\alpha^1]\!]_i, \ldots, [\![\alpha^t]\!]_i)_{i\in S}$ and give rise to broadcast values $\alpha^1, \ldots, \alpha^t$ for which the protocol accepts, *i.e.* $g(\alpha^1, \ldots, \alpha^t) = 0$. Since $p = 0$, the input shares of those $\ell+1$ parties necessarily define a good witness w (*i.e.* satisfying $(x, w) \in \mathcal{R}$), which is in contradiction with the definition of a malicious prover. We deduce that in such a zero-false-positive scenario, a malicious prover (who does not know a good witness) has to cheat for at least $N - \ell$ parties. Then, if the malicious prover cheats on more than $N - \ell$ parties, the verifier shall always discover the cheat since she shall necessarily ask for the opening of a cheating party. We deduce that a malicious prover must necessarily cheat on exactly $N - \ell$ parties, and the only way for the verifier to be convinced is to ask for the opening of the exact ℓ parties which have been honestly emulated. The probability of this event to happen is

$$\frac{1}{\binom{N}{N-\ell}} = \frac{1}{\binom{N}{\ell}},$$

which corresponds to the soundness error of the protocol, assuming $p = 0$.

Let us now consider a false positive rate p which is not zero. A malicious prover can then rely on a false positive to get a higher probability to convince the verifier. In case the committed input shares $[\![w]\!]_1, \ldots, [\![w]\!]_N$ were consistent (*i.e.* they formed a valid secret sharing), the soundness error would be

$$\frac{1}{\binom{N}{\ell}} + \left(1 - \frac{1}{\binom{N}{\ell}}\right) \cdot p.$$

However, we cannot enforce a malicious prover to commit a valid secret sharing $[\![w]\!]$ since the verifier never sees more than the shares of ℓ parties. More precisely, let us denote

$$\mathcal{J} := \{J \subset [N] : |J| = \ell + 1\}$$

and let $w^{(J)}$ be the witness corresponding to the shares $[\![w]\!]_J$ for some subset $J \in \mathcal{J}$, formally $w^{(J)} := \mathsf{Reconstruct}_J([\![w]\!]_J)$. Then we could have

$$w^{(J_1)} \neq w^{(J_2)}$$

1. The prover shares the witness w into an $(\ell+1, N)$-threshold linear secret sharing $[\![w]\!]$.
2. The prover emulates "in her head" a (public) subset S of $\ell+1$ parties of the MPC protocol. For $j = 1$ to t:
   (a) the prover computes
   $$\beta^j = \psi^j(w, (\varepsilon^i)_{i<j}),$$
   shares it into an $(\ell+1, N)$-threshold linear secret sharing $[\![\beta^j]\!]$;
   (b) the prover computes the commitments
   $$\mathsf{com}_i^j := \begin{cases} \mathrm{Com}([\![w]\!]_i, [\![\beta^j]\!]_i; \rho_i^j) & \text{if } j = 1 \\ \mathrm{Com}([\![\beta^j]\!]_i; \rho_i^j) & \text{if } j > 1 \end{cases}$$
   for all $i \in [N]$, for some commitment randomness ρ_i^j, and computes the Merkle root
   $$\tilde{h}_j := \mathrm{MerkleTree}(\mathsf{com}_1^j, \ldots, \mathsf{com}_N^j).$$
   (c) the prover sends
   $$h_j := \begin{cases} \tilde{h}_j & \text{if } j = 1 \\ \mathrm{Hash}(\tilde{h}_j, [\![\alpha^{j-1}]\!]_S) & \text{if } j > 1 \end{cases}$$
   to the verifier;
   (d) the verifier picks at random a challenge ε^j and sends it to the prover;
   (e) the prover computes, for $i \in S$,
   $$[\![\alpha^j]\!]_i := \varphi^j_{(\varepsilon^k)_{k\le j}, (\alpha^k)_{k<j}}\big([\![w]\!]_i, ([\![\beta^k]\!]_i)_{k\le j}\big)$$
   and recomposes α^j. *This step is repeated as many times as in the MPC protocol (cf Protocol 2).*

   The prover further computes $h_{t+1} := \mathrm{Hash}([\![\alpha^t]\!]_S)$ and sends it to the verifier.
3. The verifier picks at random a subset $I \subset [N]$ of ℓ parties (*i.e.* $|I| = \ell$) and sends it to the prover.
4. The prover opens the commitments of all the parties in I, namely she sends $([\![w]\!]_i, ([\![\beta^j]\!]_i, \rho_i^j)_{j\in[t]})_{i\in I}$ to the verifier. The prover further sends the authentication paths $\mathsf{auth}_1, \ldots, \mathsf{auth}_t$ to these commitments, *i.e.* auth_j is the authentication path for $\{\mathsf{com}_i^j\}_{i\in I}$ w.r.t. Merkle root $\tilde{h}_j$ for every $j \in [t]$. Additionally, the prover sends broadcast shares $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ of an unopened party $i^* \in S \setminus I$.
5. The verifier recomputes the commitments com_i^j and the broadcast values $[\![\alpha^j]\!]_i$ for $i \in I$ and $j \in [t]$ from $([\![w]\!]_i, ([\![\beta^j]\!]_i, \rho_i^j)_{j\in[t]})_{i\in I}$. Then she recovers $\alpha^1, \ldots, \alpha^t$, by
   $$\alpha^j = \mathsf{Reconstruct}_{I\cup\{i^*\}}([\![\alpha^j]\!]_{I\cup\{i^*\}})$$
   for every $j \in [t]$.
6. The verifier accepts if and only if:
   (a) the views of the opened parties are consistent with each other, with the committed input shares and with the hash digest of the broadcast messages, *i.e.* for $j = 1$ to $t+1$,
   $$h_j \stackrel{?}{=} \begin{cases} \tilde{h}_j & \text{if } j = 1 \\ \mathrm{Hash}(\tilde{h}_j, [\![\alpha^{j-1}]\!]_S) & \text{if } 2 \le j \le t \\ \mathrm{Hash}([\![\alpha^{j-1}]\!]_S) & \text{if } j = t+1 \end{cases}$$
   where $\tilde{h}_j$ is the Merkle root deduced from $\big(\{\mathsf{com}_i^j\}_{i\in I}, \mathsf{auth}_j\big)$ and $[\![\alpha^{j-1}]\!]_S$ are the shares in subset S deduced from $[\![\alpha^{j-1}]\!] = \mathsf{Expand}_{I\cup\{i^*\}}\big([\![\alpha^{j-1}]\!]_{I\cup\{i^*\}}\big)$;
   (b) the output of the opened parties are Accept, *i.e.*
   $$g(\alpha^1, \ldots, \alpha^t) \stackrel{?}{=} 0\ .$$

Protocol 5: Zero-knowledge protocol: application of the MPCitH principle to Protocol 4 with an $(\ell+1, N)$-threshold linear secret sharing scheme.

for distinct subsets $J_1, J_2 \in \mathcal{J}$. A malicious prover can exploit this degree of freedom to increase the soundness error.

Soundness Attack. Let us take the example of the [BN20] protocol on a field $\mathbb{F}$. In this protocol, the MPC functionality f outputs Accept for a bad witness w (*i.e.* such that $(x, w) \notin \mathcal{R}$) with probability $p = \frac{1}{|\mathbb{F}|}$, *i.e.* if and only if the oracle $\mathcal{O}_R$ samples a specific element ε_w of $\mathbb{F}$. In this context, a possible strategy for the malicious prover is the following:

1. Build the shares $[\![w]\!]_1, \ldots, [\![w]\!]_N$ such that
$$\forall J_1, J_2 \in \mathcal{J},\ \varepsilon_{w^{(J_1)}} \neq \varepsilon_{w^{(J_2)}}.$$
We implicitly assume here that $\binom{N}{\ell+1} \leq |\mathbb{F}|$ and that constructing such collision-free input sharing is possible. We assume that $(x, w^{(J)}) \notin \mathcal{R}$ for every J (otherwise the malicious prover can recover a good witness by enumerating the $w^{(J)}$'s).
2. After receiving the initial commitments, the verifier sends the challenge ε.
3. If there exists $J_0 \in \mathcal{J}$ such that $\varepsilon = w^{(J_0)}$, which occurs with probability $\binom{N}{\ell+1} \cdot p$ since all the $\varepsilon^{(J)}$ are distinct, then the malicious prover defines the broadcast values $\alpha^1, \ldots, \alpha^t$ (and the broadcast shares in the set S) according to the broadcast shares of the parties in J_0. It results that the computation of the parties in J_0 is correct and the prover will be able to convince the verifier if the set I of opened parties is a subset of J_0 ($I \subset J_0$).
4. Otherwise, if no subset $J_0 \in \mathcal{J}$ is such that $\varepsilon = w^{(J_0)}$, the malicious prover is left with the option of guessing the set I. Namely, she (randomly) chooses a set I_0 of ℓ parties as well as broadcast values $\alpha^1, \ldots, \alpha^t$ such that $g(\alpha^1, \ldots, \alpha^t) = 0$, and then she deduces and commits the broadcast shares $[\![\alpha^j]\!]_S$ from the $[\![\alpha^j]\!]_{I_0}$ (computed from the committed input shares) and the chosen α^j's. The malicious prover will be able to convince the verifier if and only if the challenge set I matches the guess I_0.

The probability p_{attack} that the malicious prover convinces the verifier using the above strategy satisfies

$$\begin{aligned} p_{\text{attack}} &:= \overbrace{\binom{N}{\ell+1}p}^{\Pr[\exists J_0 : \varepsilon = w^{(J_0)}]} \cdot \overbrace{\frac{\binom{\ell+1}{\ell}}{\binom{N}{\ell}}}^{\Pr[I \subset J_0]} + \overbrace{\left(1 - \binom{N}{\ell+1}p\right)}^{\Pr[\forall J, \varepsilon \neq w^{(J)}]} \cdot \overbrace{\frac{1}{\binom{N}{\ell}}}^{\Pr[I = I_0]} \\ &= \frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell \cdot (N - \ell)}{\ell + 1} \geq \underbrace{\frac{1}{\binom{N}{\ell}} + \left(1 - \frac{1}{\binom{N}{\ell}}\right) \cdot p}_{\text{Soundness error if the committed sharing is well-formed.}}. \end{aligned}$$

Soundness Proof. We can prove that the above strategy to forge successful transcripts for the [BN20] protocol is actually optimal and that it further applies to other protocols complying with our model. This is formalized in the following theorem (together with the completeness and HVZK property of the protocol).

Theorem 2. *Let us consider an MPC protocol Π_{LSSS} complying with the protocol format described in Protocol 4 using an $(\ell+1, N)$-threshold LSSS, such that Π_{LSSS} is ℓ-private in the semi-honest model and of false positive rate p. Then, Protocol 5 built from Π_{LSSS} is* complete*,* sound *and* honest-verifier zero-knowledge*, with a soundness error ϵ defined as*

$$\epsilon := \frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell \cdot (N-\ell)}{\ell+1}.$$

Proof. The completeness holds from the completeness property of the underlying MPC protocol. The zero-knowledge property directly comes from the ℓ-privacy property of the MPC protocol with an $(\ell+1, N)$-threshold linear secret sharing scheme. See the full version [FR22] for the soundness proof.

Remark 1. Let us remark that the above theorem includes the MPCitH setting with additive sharing as a particular case. Indeed, when $\ell = N-1$, we obtain the usual formula for the soundness error, that is:

$$\ell = N-1 \quad \Longrightarrow \quad \epsilon = \frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right).$$

Remark 2. When $\ell = 1$, we have $\epsilon \approx \frac{1}{N}$ (assuming p is small). It can look as surprising that we can have such soundness error by revealing a single party's view. Since the communication is only broadcast, a verifier does not need to check for inconsistency between several parties, she just needs to check that the revealed views are consistent with the committed broadcast messages. Moreover, the verifier has the guarantee that the shares broadcast by all the parties form *a valid sharing of the open value*. It means that even if the prover reveals only one party's view, the latter can be inconsistent with the committed broadcast. Assuming we use Shamir's secret sharing, committing to a valid broadcast sharing consists in committing a degree-ℓ polynomial such that evaluations are the broadcast shares. By interpolating the broadcast shares of ℓ honest parties (and given the plain value of the broadcast message), one shall entirely fix the corresponding Shamir's polynomial, and the other parties can not be consistent with this polynomial without being consistent with the honest parties (and the latter can only occur if there is a false positive).

4.4 Performances

The advantage of using a threshold LSSS over a standard additive sharing mainly resides in a much faster computation time, for both the prover and the verifier. Indeed, according to the above description, the prover only emulates $\ell+1$ parties

while the verifier only emulates ℓ parties, which is particularly efficient for a small ℓ. For example, assuming that p is negligible and taking $\ell = 1$, the soundness error is $1/N$ (which is similar to standard MPCitH with additive sharing) and the prover only needs to emulate $\ell + 1 = 2$ parties (instead of N) while the verifier only needs to emulate $\ell = 1$ party (instead of $N - 1$).

When targeting a soundness error of λ bits, one needs to repeat the protocol $\tau := \frac{-\lambda}{\log_2 \epsilon}$ times and thus the number of times that a prover emulates a party is multiplied by τ. Table 2 summarizes the number of party emulations for the prover and the verifier for the standard case (additive sharing) and for the case of an $(\ell + 1, N)$-threshold LSSS. Interestingly, we observe that the emulation phase is more expensive when increasing N for the additive sharing case while it becomes cheaper for the threshold LSSS case (with some constant ℓ). For the sake of comparison, we also give in Table 2 the numbers corresponding to the hypercube optimization from the concurrent work [AGH+23].

The computational bottleneck for the prover when using an LSSS with low threshold ℓ and possibly high N becomes the generation and commitment of all the parties' input shares, which is still linear in N. Moreover the sharing generation for a threshold LSSS might be more expensive than for a simple additive sharing. On the other hand, the verifier does not suffer from this bottleneck since she only has to verify ℓ opened commitments (per repetition). One trade-off to reduce the prover commitment bottleneck is to increase ℓ, which implies a smaller τ (for the same N) and hence decreases the number of commitments.

Table 2. Number of party emulations to achieve a soundness error of $2^{-\lambda}$ (assuming a negligible false positive rate p).

	With additive sharing		With threshold LSSS	
	Traditional	Hypercube	$\ell = 1$	Any ℓ
Prover	$\approx \lambda \frac{N}{\log_2 N}$	$\approx \lambda \frac{\log_2 N + 1}{\log_2 N}$	$\approx \lambda \frac{2}{\log_2 N}$	$\approx \lambda \frac{\ell+1}{\log_2 \binom{N}{\ell}}$
Verifier	$\approx \lambda \frac{N-1}{\log_2 N}$	$\approx \lambda \frac{\log_2 N}{\log_2 N}$	$\approx \lambda \frac{1}{\log_2 N}$	$\approx \lambda \frac{\ell}{\log_2 \binom{N}{\ell}}$

In terms of communication, using a threshold LSSS implies a slight overhead. In particular, since only ℓ parties out of N are opened, we use Merkle tree for the commitments and include the authentication paths in the communication.

Let us recall the notations defined in Sect. 3.2:

- inputs: the bitsize of $(w, \beta^1, \ldots, \beta^t)$ excluding the uniformly-distributed elements β^{unif}, and
- comm: the bitsize of $([\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*})$ excluding the elements which can be recovered from $g(\alpha^1, \ldots, \alpha^t) = 0$.

We denote unif the bitsize of the uniformly-distributed elements β^{unif}. Then, the proof size (in bits) when repeating the protocol τ times is

$$\mathsf{Cost} = \underbrace{(t+1)\cdot 2\lambda}_{h_1,h_2,\ldots,h_{t+1}} + \tau \cdot \Big(\underbrace{\ell \cdot (\mathsf{inputs} + \mathsf{unif})}_{\{[\![w]\!]_i,[\![\beta^1]\!]_i,\ldots,[\![\beta^t]\!]_i\}_{i\in I}} + \underbrace{\mathsf{comm}}_{[\![\alpha^1]\!]_{i^*},\ldots,[\![\alpha^t]\!]_{i^*}} + \underbrace{2\lambda \cdot t \cdot \ell \cdot \log_2 \frac{N}{\ell}}_{\mathsf{auth}_1,\ldots,\mathsf{auth}_t} \Big).$$

Let us remark that the bitsize unif appears here while it was not the case for additive sharings. This comes from the fact that, even if β^{unif} is uniformly sampled, $[\![\beta^{\text{unif}}]\!]$ has some structure (*i.e.* some redundancy) when using an arbitrary linear secret sharing scheme.

5 Further Improvements

In this section, we suggest potential ways to further improve and generalize our approach.

5.1 Using Linearly Homomorphic Commitments

As explained previously, one of the bottlenecks of this construction is that the prover must realize N commitments. Although we decrease the cost of emulating the MPC protocol (from N parties to a constant number), we still need to commit the inputs of all the parties which is still linear in N. For this reason, we cannot arbitrarily increase the number of parties N even while working on large fields (*e.g.* $\mathbb{F}_{2^{32}}$ or larger). One natural strategy to improve this state of affairs and get rid of those N commitments is to use a linearly homomorphic commitment scheme. When relying on such a scheme, the prover can just commit the input shares for the $\ell+1$ parties in S, instead of committing all the parties' input shares. Then the commitment of any party can be expressed as a linear combination of these commitments. For applications to the post-quantum setting (which is a context of choice for MPCitH schemes), one could rely on lattice-based homomorphic commitment schemes. To the best of our knowledge, most of these schemes are only additively homomorphic (not linearly) and they support a bounded number of additions which makes their application to our context not straightforward. This is yet an interesting question for future research.

5.2 Using Quasi-Threshold Linear Secret Sharing

Theorem 2 only considers linear secret sharing schemes, but we can generalize the result to any quasi-threshold linear secret sharing scheme. In such schemes, ℓ shares leak no information about the secret and $\ell+1+\Delta$ shares are necessary to reconstruct the secret, with $\Delta > 0$, namely we have a gap between the two thresholds. In our context, this gap shall impact the soundness of the protocol. Indeed, the prover just needs to cheat for $N-\ell-\Delta$ parties (such that there is less than $\ell+\Delta$ honest parties), but the verifier asks to open only ℓ parties.

Considering quasi-threshold schemes bring more versatility to our approach and opens the door to techniques that are not possible with tight threshold schemes (*e.g.* batching such as proposed below).

Let us remark that the set S of emulated parties in Protocol 5 must be chosen such that $[\![v]\!]_S$ enables to deduce all the shares $[\![v]\!]_{[N]}$. In the tight threshold case, such a set S is always of size $\ell+1$ (see Lemma 2), but in the case of quasi-threshold LSSS, this set S might be larger than $\ell+\Delta+1$. Moreover, sending shares $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ for one non-opened party $i^* \in S$ might not be enough to enable the verifier to recompute $[\![\alpha^j]\!]_S$ for all j. Therefore the size of S and the number of additional shares $[\![\alpha^j]\!]_i$ to be revealed depend on the underlying quasi-threshold linear secret sharing, which impacts the communication cost. On the other hand, the soundness error of the obtained proof of knowledge is not impacted.

Theorem 3. *Let us consider an MPC protocol $\Pi_{\text{QT-LSSS}}$ complying with the protocol format described in Protocol 4, but using an $(\ell, \ell+\Delta+1, N)$-quasi-threshold LSSS in place of an $(\ell+1, N)$-threshold LSSS, and such that $\Pi_{\text{QT-LSSS}}$ is ℓ-private in the semi-honest model and of false positive rate p. Then, Protocol 5 built from $\Pi_{\text{QT-LSSS}}$ is* complete, sound *and* honest-verifier zero-knowledge, *with a soundness error ϵ defined as*

$$\epsilon := \frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+\Delta+1} \cdot \binom{N-\ell}{\Delta+1}.$$

Proof. The completeness holds from the completeness property of the underlying MPC protocol. The zero-knowledge property directly comes from the ℓ-privacy property of the MPC protocol with an $(\ell, \ell+\Delta+1, N)$-threshold linear secret sharing scheme. See the full version [FR22] for the proof of the soundness.

Using Algebraic Geometric Secret Sharing? One drawback while using a tight threshold LSSS is that the number N of parties is limited by the size of the underlying field $\mathbb{F}$, specifically we have $N \leq |\mathbb{F}|$ (see Lemma 1). In the full version [FR22], we investigate whether quasi-threshold LSSS based on algebraic geometry [CC06] can improve this state of affairs. Our result is negative: we show that we cannot tackle this issue by using such schemes. We also show that the soundness error (with $\ell = 1$) is at least $1/(2|\mathbb{F}| - 1)$ for any quasi-threshold LSSS. This implies that such sharing schemes could only have a limited interest to achieve smaller sizes, since it could decrease the soundness error by a factor at most *two* compared to the case with the Shamir's secret sharing scheme.

We show hereafter that the above generalization to quasi-threshold LSSS is useful for another purpose, namely an efficient batching technique in our framework.

5.3 Batching Proofs with Shamir's Secret Sharing

Principle. Shamir's secret sharing is traditionally used to share a single element of the underlying field, but it can be extended to share several elements simulta-

neously. To share $v_1, v_2, \ldots, v_u \in \mathbb{F}$, we can sample ℓ random elements $r_1, \ldots, r_\ell$ of $\mathbb{F}$ and build the polynomial P of degree $\ell + u - 1$ such that, given distinct fixed field elements $e_1, \ldots, e_{u+\ell}$,

$$\begin{cases} P(e_1) = v_1 \\ P(e_2) = v_2 \\ \quad\vdots \\ P(e_u) = v_u \end{cases} \quad \text{and} \quad \begin{cases} P(e_{u+1}) = r_1 \\ \quad\vdots \\ P(e_{u+\ell}) = r_\ell \end{cases}$$

The shares are then defined as evaluations of P on fixed points of $\mathbb{F}\backslash\{e_1, \ldots, e_u\}$. Revealing at most ℓ shares does not leak any information about the shared values $v_1, \ldots, v_u$, while one needs at least $\ell + u$ shares to reconstruct all of them. In other words, this is an $(\ell, \ell + u, N)$-quasi-threshold linear secret sharing scheme for the tuple $(v_1, \ldots, v_u)$. Thus, while applying such a sharing to our context, the soundness error is given by (see Theorem 3)

$$\frac{\binom{\ell+u-1}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+u} \cdot \binom{N-\ell}{u}.$$

When running an MPC protocol on such batch sharing, the operations are simultaneously performed on all the shared secrets $v_1, \ldots, v_u$. It means that we can batch the proof of knowledge of several witnesses which have the same verification circuit (*i.e.* the same functions φ^j in our MPC model – see Protocol 1). Using this strategy, the soundness error is slightly larger, but we can save a lot of communication by using the same sharing for several witnesses.

Specifically, the proof size while batching u witnesses is impacted as follows. The parties' input shares are not more expensive, but to open the communication, the prover now needs to send u field elements by broadcasting (instead of a single one). Thus the communication cost for τ executions is given by

$$\mathsf{Cost} = \underbrace{(t+1)\cdot 2\lambda}_{h_1,h_2,\ldots,h_{t+1}} + \tau \cdot \Big(\underbrace{\ell \cdot (\mathsf{inputs} + \mathsf{rtapes})}_{\{[\![w]\!]_i, [\![\beta^1]\!]_i, \ldots, [\![\beta^t]\!]_i\}_{i\in I}} + \underbrace{u \cdot \mathsf{comm}}_{\alpha^1,\ldots,\alpha^t} + \underbrace{2\lambda \cdot t \cdot \ell \cdot \log_2 \frac{N}{\ell}}_{\mathsf{auth}_1,\ldots,\mathsf{auth}_t} \Big).$$

Unfortunately, the scope of application of this batching technique is limited. In particular, while we can multiply the batched shared secrets by the same scalar, with

$$[\![\begin{pmatrix} \gamma \cdot v_1 \\ \vdots \\ \gamma \cdot v_u \end{pmatrix}]\!] := \gamma \cdot [\![\begin{pmatrix} v_1 \\ \vdots \\ v_u \end{pmatrix}]\!]$$

for some $\gamma \in \mathbb{F}$, we cannot compute

$$[\![\begin{pmatrix} \gamma_1 \cdot v_1 \\ \vdots \\ \gamma_u \cdot v_u \end{pmatrix}]\!] \quad \text{from} \quad [\![\begin{pmatrix} v_1 \\ \vdots \\ v_u \end{pmatrix}]\!]$$

for distinct scalars $\gamma_1, \ldots, \gamma_u$ (whenever at least two scalars are distinct). This restriction implies that the scalar factors used in the verification circuit must be independent of the different witnesses which are batched together. More precisely, it implies that the functions φ^j in our MPC model (see Protocol 1) must be of the form

$$\varphi^j_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j}}(\cdot) = \underbrace{\bar{\varphi}^j_{(\varepsilon^i)_{i\leq j}}(\cdot)}_{\substack{\text{Linear function with} \\ \varepsilon^i\text{-dependent coefficients}}} + \underbrace{b^j_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j}}}_{\substack{\text{Constant offset which} \\ \text{depends on the } \varepsilon^i \text{ 's and } \alpha^i\text{'s}}}$$

This restriction prevents the use of this batching strategy for several MPCitH protocols. For example, all the protocols using the multiplication checking protocol from [BN20] as a subroutine cannot use this batching strategy. To the best of our knowledge, the only protocols in the current state of the art which support this batching strategy are Banquet [BDK+21] and Limbo [DOT21].

Batching Strategies. In what follows, we propose three strategies to batch MPCitH proofs relying on the same verification circuit:

Naive strategy: The naive way to batch u MPCitH proofs is to emulate u independent instances of MPC protocol, one for each input witness. Compared to sending u independent proofs, one can save communication by using the same seed trees and the same commitments for the u instances. This strategy can be applied for standard MPCitH schemes based on additive sharing as well as for our framework of threshold LSSS-based MPCitH. When using additive sharings, the main drawback of this strategy is that the prover and the verifier need to emulate the party computation a large number of times, *i.e.* N times (or $N-1$ times for the verifier) per iteration and per statement. When batching $u \geq 25$ statements with $N = 256$, the prover and the verifier must emulate more than 100 000 parties to achieve a security of 128 bits. When using a low-threshold LSSS, the emulation cost is much cheaper, but the proof transcript is larger. While batching u statements, the emulation cost and the soundness error are given by the following table:

	# Emulations	Soundness Error
Prover	$\tau \cdot (\ell+1) \cdot u$	$\frac{1}{\binom{N}{\ell}} + p \cdot \frac{(N-\ell)\cdot\ell}{\ell+1}$
Verifier	$\tau \cdot \ell \cdot u$	

SSS-based strategy: We can use the batching strategy based on Shamir's secret sharing (SSS) described above. Instead of having u independent input sharings (one per witness), we have a single input sharing batching the u witnesses. The number of MPC emulations is lower than for the naive strategy. The proof size is also smaller and (mostly) below that of the standard setting

for small u, but it grows exponentially when considering a small field $\mathbb{F}$. Each batched statement consumes one evaluation point (in $\mathbb{F}$), the number N of parties is hence limited by $N \leq |\mathbb{F}|+1-u$. Because of this limitation together with the security loss due to the use of a quasi-threshold sharing scheme, the soundness error of this batched protocol degrades rapidly as u grows. While batching u statements using Shamir's secret sharings, the emulation cost and the soundness error are given by the following table:

	# Emulations	Soundness Error
Prover	$\tau \cdot (\ell + u)$	$\frac{\binom{\ell+u-1}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+u} \cdot \binom{N-\ell}{u}$
Verifier	$\tau \cdot \ell$	

Hybrid strategy: In the previous strategy, the proof size is convex w.r.t. the number u of batched proofs and, for small some u, the curve slope is flatter than the slope in the additive case. It means that using a hybrid approach can achieve smaller proof sizes (as well as better performances) than with the two above strategies. Specifically, instead of having one input sharing encoding the u witnesses (one per batched statement) and a single emulation of the MPC protocol, we can use ν input sharings each of them encoding $\frac{u}{\nu}$ witnesses and have then ν emulations of the MPC protocol. Using this hybrid strategy, the emulation cost and the soundness error are given by the following table:

	# Emulations	Soundness Error
Prover	$\tau \cdot (\ell + \frac{u}{\nu}) \cdot \nu$	$\frac{\binom{\ell+u/\nu-1}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+u/\nu} \cdot \binom{N-\ell}{u/\nu}$
Verifier	$\tau \cdot \ell \cdot \nu$	

Section 6.2 presents some application results for these batching strategies. In particular the full version [FR22] compares the three strategies for batched proofs of the SDitH scheme [FJR22].

6 Applications

In the past few years, many proof systems relying on the MPC-in-the-Head paradigm have been published. Table 3 provides a tentatively exhaustive list of these schemes while indicating for each scheme:

- the base field (or ring) of the function computed by the underlying MPC protocol,
- whether the underlying MPC protocol fits our general model (see Sect. 3.1),

– the hard problem (or one-way function) for which the witness knowledge is proved.

In column *Base Ring*, the notation "$\mathbb{F}$ ($\mathbb{K}$)" means that the function computed by the underlying MPC protocol is composed of $\mathbb{F}$-linear functions and multiplications over $\mathbb{K}$. For example, the schemes for AES use $\mathbb{F}_2$-linear functions and $\mathbb{F}_{256}$-multiplications.

Applying our framework with an arbitrary (low-)threshold linear secret sharing scheme instead of an additive sharing scheme is possible whenever

– the underlying MPC protocol fits the model introduced in Sect. 3.1,
– the underlying MPC protocol is defined over a field (and not only a ring),
– this base field is large enough (since the number of parties N is limited by the size of the field).

Because of this last condition, all the proof systems for Boolean circuits and/or one-way functions with $\mathbb{F}_2$ operations (*e.g.* AES, Rain, SDitH over $\mathbb{F}_2$) do not support our framework of MPCitH based on (low-)threshold LSSS. Same for the scheme recently proposed in [FMRV22] and which achieves short communication using secret sharing over the integers: this idea is not compatible with our approach.

6.1 Application to the SDitH Signature Scheme

We can transform the zero-knowledge proofs of knowledge described in Sect. 4 into signature schemes using the Fiat-Shamir's heuristic [FS87].

In the following, we focus on the signature scheme obtained when applying this approach to the SDitH protocol (SDitH for "Syndrome Decoding in the Head") [FJR22]on the base field $\mathbb{F}_{\mathrm{SD}}$. We apply the ideas of Sect. 4 to this scheme using Shamir's secret sharing. Since the number N of parties is limited by the field size, $N \leq |\mathbb{F}_{\mathrm{SD}}|$,[4] we consider the instance with $\mathbb{F}_{\mathrm{SD}} := \mathbb{F}_{256}$ as base field. As explained previously, our MPCitH strategy with $(\ell+1, N)$-threshold LSSS does not make the signature smaller but substantially improves the signing and verification times. According to Sect. 4.4, we obtain signatures of size (in bits):

$$\textsc{Size} = 6\lambda + \tau \cdot \left(\ell \cdot (\mathsf{inputs} + \mathsf{unif}) + \mathsf{comm} + 2\lambda \cdot \ell \cdot \log_2 \frac{N}{\ell}\right)$$

where inputs, unif, and comm are such as defined in Sect. 3 (see the full version [FR22] for explicit values for the SDitH scheme).

In [FJR22], the authors choose p a bit lower than 2^{-64} which implies that the number of executions τ just needs to be increased by one while turning to the

[4] The Shamir's secret sharing over a field $\mathbb{F}$ can have at most $|\mathbb{F}|-1$ shares (one share by non-zero evaluation point), but we can have an additional share by defining it as the leading coefficient of the underlying polynomial (*i.e.* using the point at infinity as evaluation point).

Table 3. Generic MPC-in-the-Head Techniques and Signature Schemes from MPC-in-the-Head Techniques. All the signature sizes are in kilobytes and target a security of 128 bits. The *original* signature sizes correspond to values given by the underlying articles. The *normalized* signature sizes are given for a range of $8 - 32$ parties (in the underlying MPC protocol) when there is a preprocessing phase and for a range of $32 - 256$ parties otherwise. The column "Model" indicates whether the underlying MPC protocol fits our general model.

Scheme Name	Year	Base Ring	Model	#Rounds	Helper	Hard Problem	Signature Size	
							Original	Normalized
ZKBoo [GMO16]	2016	Any ring	✗	3	✗	Any $(2,3)$-decomposition circuit	–	–
ZKB++ [CDG+17]	2017	Any ring	✗	3	✗		–	–
Ligero [AHIV17]	2017	Any field	✗	5	✗	Any arithmetic circuit C (additions and multiplications)	–	–
Ligero++ [BFH+20]	2020	Any field	✗	5	✗		–	–
KKW [KKW18]	2018	Any ring	✓	3 or 5	✓		–	–
BN [BN20]	2020	Any field	✓	5	✗		–	–
Limbo [DOT21]	2021	Any field	✓	$\log \lvert C \rvert$	✗		–	–
BN++ [KZ22]	2021	Any field	✓	5	✗		–	–
Helium [KZ22]	2021	Any field	✓	7	✗		–	–
Picnic1 [CDG+17]	2016	$\mathbb{F}_2$	✗	3	✗	LowMC (partial)	32.1	–
Picnic2 [KKW18]	2018	$\mathbb{F}_2$	✓	3	✓		12.1	12.1 – 15.4
Picnic3 [KZ20b]	2019	$\mathbb{F}_2$	✓	3	✓	LowMC (full)	12.3	11.1 – 13.7
Helium+LowMC [KZ22]	2022	$\mathbb{F}_2$ ($\mathbb{F}_8$)	✓	7	✗		5.4 – 12.1	6.4 – 9.2
BBQ [DDOS19]	2020	$\mathbb{F}_2$ ($\mathbb{F}_{256}$)	✓	3	✓	AES	30.9	31.8 – 48.6
Banquet [BDK+21]	2021	$\mathbb{F}_2$ ($\mathbb{F}_{256}$)	✓	7	✗		13.0 – 19.3	13.0 – 17.1
Limbo-Sign [DOT21]	2021	$\mathbb{F}_2$ ($\mathbb{F}_{256}$)	✓	13	✗		14.2 – 17.9	14.2 – 17.9
Helium+AES [KZ22]	2022	$\mathbb{F}_2$ ($\mathbb{F}_{256}$)	✓	7	✗		9.7 – 17.2	9.7 – 14.4
LegRoast [BD20]	2020	$\mathbb{F}_{2^{127}-1}$	✓	7	✗	Legendre PRF	12.2 – 16.0	12.2 – 14.8
PorcRoast [BD20]	2020	$\mathbb{F}_{2^{127}-1}$	✓	7	✗	Higher-Power Residue Characters	6.3 – 8.6	6.3 – 7.8
Rainier-128 [DKR+21]	2021	$\mathbb{F}_2$ ($\mathbb{F}_{128}$)	✓	5	✗	Rain [DKR+21]	5.1 – 9.4	5.9 – 8.1
BN++Rain [KZ22]	2022	$\mathbb{F}_2$ ($\mathbb{F}_{128}$)	✓	5	✗		4.4 – 5.8	4.9 – 6.4
SDitH [FJR22]	2022	$\mathbb{F}_2$	✓	5	✗	Syndrome Decoding over $\mathbb{F}_2$	11.8 – 17.0	10.9 – 15.6
	2022	$\mathbb{F}_{256}$	✓	5	✗	Syndrome Decoding over $\mathbb{F}_{256}$	8.3 – 11.5	8.3 – 11.5
[FMRV22]	2022	$\mathbb{Z}$	✓	5	✓/✗	Subset-Sum Problem	21.1 – 33.2	24.3 – 34.8
	2022	$\mathbb{Z}$	✓	5	✗	BHH PRF [BHH01]	4.8	4.8 – 6.5

non-interactive case. Here, by taking $\ell > 1$, we decrease τ and each execution has more impact on the communication cost. Therefore we take p negligible in order to avoid to increase τ while turning to the non-interactive setting. At the same time, it means that we can apply an idea from Limbo [DOT21] which consists in using the same first challenge for all parallel executions of the underlying MPC protocol.

As explained in Sect. 4.3, in case of a non-negligible false positive rate, an adversary can try to forge a proof of knowledge by committing an invalid sharing of the witness (which is not possible in the case of additive sharing). This ability is also exploitable in the non-interactive setting while considering the attack of [KZ20a]. In order to thwart this type of attack on our variant of the SDitH scheme, we make the conservative choice of taking a false positive rate p satisfying

$$\tau \cdot \binom{N}{\ell + 1} \cdot p \leq 2^{-128}.$$

This way, the probability that a single witness encoded by a subset of $\ell + 1$ shares among N leads to a false positive (in at least one of the τ iterations) is upper bounded by 2^{-128} so that any attack strategy which consists to guess (even partially) the first challenge shall cost at least 2^{128} operations. Then, we simply need to take τ such that $\binom{N}{\ell}^{\tau} \geq 2^{128}$ in order to achieve a 128-bit security in the non-interactive setting. We propose four possible instances of our scheme for $\ell \in \{1, 3, 7, 12\}$ and $N = 256$ (the maximal number of parties).

We have implemented our variant of the SDitH signature scheme in C. In our implementation, the pseudo-randomness is generated using AES in counter mode and the hash function is instantiated with SHAKE. We have benchmarked our implementation on a 3.8 GHz Intel Core i7 CPU with support of AVX2 and AES instructions. All the reported timings were measured on this CPU while disabling Intel Turbo Boost.

Table 4 summarizes the obtained performances for the different sets of parameters. We observe that the verification time is significantly smaller –between one and two orders of magnitude– than for the original scheme. This was expected since the verifier only emulates the views of ℓ parties instead of $N - 1$. The gain in signing time is more mitigated: even if the signer emulates only few parties, she must still commit the input shares of N parties. Nevertheless, the number of executions τ decreases while increasing the threshold ℓ, which further improves the signing time. The resulting signatures are slightly larger than for the original scheme with the same number of parties (the short version), but our scheme gains a factor 10 in signing and verification time. Compared to the fast version of the original signature scheme (which uses a lower number of parties $N = 32$) and for similar signature size, our scheme gains a factor 3 in signing time and a factor 10 in verification time.

Table 4 further compares our scheme with recent MPCitH schemes based on AES (both AES and SD for random linear codes being deemed as a conservative assumption) as well as with SPHINCS$^+$ [ABB+22] as a baseline conservative scheme. We can observe that our scheme outperforms AES-based candidates for comparable signature sizes (around 10 KB). In particular, compared to Helium+AES [KZ22], signing is 5 times faster with our scheme while verification is 40 times faster. Fast versions of those schemes have signatures about twice larger, while being still slower than ours in signing and verification. Compared to SPHINCS$^+$, our scheme achieves slightly better verification time and much better trade-offs for signature size vs. signing time. Some other MPCitH signature schemes reported in Table 3 achieve smaller signature sizes (down to 5KB) but they are based on less conservative assumptions (LowMC, Rain, BHH PRF). Yet none of these schemes achieve fast verification as SPHINCS$^+$ or our scheme.

6.2 Application of the Batching Strategy

We apply in the full version [FR22] our batching technique to two different contexts:

– We batch non-interactive proofs of knowledge for the syndrome decoding problem using the SDitH scheme [FJR22]. Since SDitH is not compatible

Table 4. Parameters, performances and comparison. The parameters for [FJR22] and our scheme are $(m, k, w) = (256, 128, 80)$ and $\mathbb{F}_{\text{SD}} = \mathbb{F}_{\text{poly}} = \mathbb{F}_{256}$. Timings for [FJR22] and our scheme have been benchmarked on a 3.8 Ghz Intel Core i7. Timings for Banquet, Helium and SPHINCS$^+$ have been benchmarked on a 3.6 GHz Intel Xeon W-2133 CPU [BDK+21, KZ22]. Timings for Limbo have been benchmarked on a 3.1 GHz Intel i9-9900 CPU [DOT21].

Scheme	N	τ	ℓ	t'	$\lvert\mathbb{F}_{\text{points}}\rvert$	$\log_2 p$	$\lvert\mathsf{sgn}\rvert$	t_{sgn}	t_{verif}
Our scheme	256	16	1	3	2^{64}	-167	10.47 KB	7.1 ms	0.46 ms
	256	6	3	3	2^{64}	-167	**9.97 KB**	3.2 ms	**0.38 ms**
	256	3	7	4	2^{64}	-222	11.10 KB	2.5 ms	0.47 ms
	256	2	12	4	2^{64}	-222	11.99 KB	**2.2 ms**	0.51 ms
[FJR22] - Var3f	32	27	–	5	2^{24}	-78	11.5 KB	6.4 ms	5.9 ms
[FJR22] - Var3s	256	17	–	5	2^{24}	-78	8.26 KB	30 ms	27 ms
Banquet (AES)	16	41	–	1	2^{32}	$(-32, -27)$	19.3 KB	6.4 ms	4.9 ms
	255	21	–	1	2^{48}	$(-48, -43)$	13.0 KB	44 ms	40 ms
Limbo-Sign (AES)	16	40	–	–	2^{48}	-40	21.0 KB	2.7 ms	2.0 ms
	255	24	–	–	2^{48}	-40	14.2 KB	29 ms	27 ms
Helium+AES	17	31	–	1	2^{144}	$(-136, -144)$	17.2 KB	6.4 ms	5.8 ms
	256	16	–	1	2^{144}	$(-136, -144)$	9.7 KB	16 ms	16 ms
SPHINCS$^+$-128f	–	–	–	–	–	–	16.7 KB	14 ms	1.7 ms
SPHINCS$^+$-128 s	–	–	–	–	–	–	7.7 KB	239 ms	0.7 ms

with our batching strategy, we propose a tweak of it. We achieve an amortized proof size around 2.3 KB using $\ell = 1$ and around 0.83 KB using $\ell = 8$, instead of around 8 KB (proof size when non batched).

– We batch proofs for general arithmetic circuits using the Limbo proof system [DOT21]. We obtain an amortized proof size lower than 1/10 of the baseline scheme when batching, while the amortized performances are also significantly improved (in particular for the verifier).

References

[ABB+22] Aumasson, J.-P., et al.: SPHINCS+ - Submission to the 3rd round of the NIST post-quantum project. v3.1 (2022)

[AGH+23] Aguilar-Melchor, C., Gama, N., Howe, J., Hülsing, A., Joseph, D., Yue, D.: The return of the SDitH. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023. LNCS, vol. 14008, pp. 564–596. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30589-4_20

[AHIV17] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: lightweight sublinear arguments without a trusted setup. In: ACM CCS 2017, pp. 2087–2104. ACM Press (2017)

[BBHR19] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 701–732. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_23

[BCR+19] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4

[BD20] Beullens, W., Delpech de Saint Guilhem, C.: LegRoast: efficient post-quantum signatures from the Legendre PRF. In: Ding, J., Tillich, J.-P. (eds.) PQCrypto 2020. LNCS, vol. 12100, pp. 130–150. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_8

[BDK+21] Baum, C., de Saint Guilhem, C.D., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: short and fast signatures from AES. In: Garay, J.A. (ed.) PKC 2021. LNCS, vol. 12710, pp. 266–297. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75245-3_11

[BFH+20] Bhadauria, R., Fang, Z., Hazay, C., Venkitasubramaniam, M., Xie, T., Zhang, Y.: Ligero++: a new optimized sublinear IOP. In: ACM CCS 2020, pp. 2025–2038. ACM Press (2020)

[BHH01] Boneh, D., Halevi, S., Howgrave-Graham, N.: The modular inversion hidden number problem. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 36–51. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_3

[BN20] Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020. LNCS, vol. 12110, pp. 495–526. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45374-9_17

[CC06] Chen, H., Cramer, R.: Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 521–536. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_31

[CDG+17] Chase, M., et al.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: ACM CCS 2017, pp. 1825–1842. ACM Press (2017)

[CDN15] Cramer, R., Damgård, I.B., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press, Cambridge (2015)

[DDOS19] de Saint Guilhem, C.D., De Meyer, L., Orsini, E., Smart, N.P.: BBQ: using AES in picnic signatures. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 669–692. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38471-5_27

[DKR+21] Dobraunig, C., Kales, D., Rechberger, C., Schofnegger, M., Zaverucha, G.: Shorter signatures based on tailor-made minimalist symmetric-key crypto. Cryptology ePrint Archive, Report 2021/692 (2021)

[DOT21] de Saint Guilhem, C.D., Orsini, E., Tanguy, T.: Limbo: efficient zero-knowledge MPCitH-based arguments. In: ACM CCS 2021, pp. 3022–3036. ACM Press (2021)

[FJR22] Feneuil, T., Joux, A., Rivain, M.: Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022. LNCS, vol. 13508, pp. 541–572. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-15979-4_19

[FMRV22] Feneuil, T., Maire, J., Rivain, M., Vergnaud, D.: Zero-knowledge protocols for the subset sum problem from MPC-in-the-head with rejection. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022. LNCS, vol. 13792, pp. 371–402. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22966-4_13

[FR22] Feneuil, T., Rivain, M.: Threshold linear secret sharing to the rescue of MPC-in-the-head. Cryptology ePrint Archive, Report 2022/1407 (2022)

[FS87] Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12

[GMO16] Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: faster zero-knowledge for Boolean circuits. In: USENIX Security 2016, pp. 1069–1083. USENIX Association (2016)

[GSV21] Gvili, Y., Scheffler, S., Varia, M.: BooLigero: improved sublinear zero knowledge proofs for Boolean circuits. In: Borisov, N., Diaz, C. (eds.) FC 2021. LNCS, vol. 12674, pp. 476–496. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64322-8_23

[IKOS07] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: 39th ACM STOC, pp. 21–30. ACM Press (2007)

[KKW18] Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: ACM CCS 2018, pp. 525–537. ACM Press (2018)

[KZ20a] Kales, D., Zaverucha, G.: An attack on some signature schemes constructed from five-pass identification schemes. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 2020. LNCS, vol. 12579, pp. 3–22. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65411-5_1

[KZ20b] Kales, D., Zaverucha, G.: Improving the performance of the Picnic signature scheme. IACR TCHES **2020**(4), 154–188 (2020)

[KZ21] Kales, D., Zaverucha, G.: Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures. Preliminary Draft, 29 October 2021

[KZ22] Kales, D., Zaverucha, G.: Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures. Cryptology ePrint Archive, Report 2022/588 (2022)

[MS10] MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes, 9th edn. Discrete Mathematics and its Applications. Elsevier Science (1978/2010)

[Sha79] Shamir, A.: How to share a secret. Commun. Assoc. Compu. Mach. **22**(11), 612–613 (1979)