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

Thibauld Feneuil[1,2] and Matthieu Rivain[1]

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

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.

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.

Construction	**Sharing Scheme**	**Priv.**	**Rob.**	**Soundness**	**Restriction**
[IKOS07, Sec. 3]	Additive	2	0	$1-\frac{1}{\binom{N}{2}}$	-
[IKOS07, Sec. 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, Sec. 4	LSSS	ℓ	0	$\frac{1}{\binom{N}{\ell}}+p\frac{\ell(N-\ell)}{\ell+1}$	Broadcast Linear operations
Our work, Sec. 5.2	LSSS with threshold gap $\Delta+1$	ℓ	0	$\frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}}+p\cdot\frac{\ell}{\ell+\Delta+1}\cdot\binom{N-\ell}{\Delta+1}$	Broadcast Linear operations

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.

Paper organization. The paper is organized as follows: In Section 2, we introduce the necessary background on zero-knowledge proofs and secure multi-party computation. We present in Section 3 our general MPC model and recall the MPC-in-the-Head paradigm in this context. In Section 4, we show how to apply threshold LSSS to this framework, and analyze the obtained soundness and the performances. Section 5 describes the generalizations and improvements of our approach. Finally, in Section 6, we present the application of our techniques to the considered use cases.

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.

A function $\mu : \mathbb{N} \to \mathbb{R}$ is said *negligible* if, for every positive polynomial $p(\cdot)$, there exists an integer $N_p > 0$ such that for every $\lambda > N_p$, we have $|\mu(\lambda)| < 1/p(\lambda)$. When not made explicit, a negligible function in λ is denoted $\mathsf{negl}(\lambda)$ while a polynomial function in λ is denoted $\mathsf{poly}(\lambda)$. We further use the notation $\mathsf{poly}(\lambda_1, \lambda_2, ...)$ for a polynomial function in several variables.

Two distributions $\{D_\lambda\}_\lambda$ and $\{E_\lambda\}_\lambda$ indexed by a security parameter λ are (t, ε)-*indistinguishable* (where t and ε are $\mathbb{N} \to \mathbb{R}$ functions) if, for any algorithm $\mathcal{A}$ running in time at most $t(\lambda)$ we have

$$\left| \Pr[\mathcal{A}^{D_\lambda}() = 1] - \Pr[\mathcal{A}^{E_\lambda}() = 1] \right| \leq \varepsilon(\lambda) ,$$

with $\mathcal{A}^{Dist}$ meaning that $\mathcal{A}$ has access to a sampling oracle of distribution $Dist$. The two distributions are said

- *computationally indistinguishable* if $\varepsilon \in \mathsf{negl}(\lambda)$ for every $t \in \mathsf{poly}(\lambda)$;
- *statistically indistinguishable* if $\varepsilon \in \mathsf{negl}(\lambda)$ for every (unbounded) t;
- *perfectly indistinguishable* if $\varepsilon = 0$ for every (unbounded) t.

2.1 Standard Cryptographic Primitives

Definition 1 (Pseudorandom Generator (PRG)). *Let $G : \{0,1\}^* \to \{0,1\}^*$ and let $\ell(\cdot)$ be a polynomial such that for any input $s \in \{0,1\}^\lambda$ we have $G(s) \in \{0,1\}^{\ell(\lambda)}$. Then, G is a (t, ϵ)-secure pseudorandom generator if the following two conditions hold:*

- *Expansion: $\ell(\lambda) > \lambda$;*
- *Pseudorandomness: the distributions*

$$\{G(s) \mid s \leftarrow \{0,1\}^\lambda\} \quad and \quad \{r \mid r \leftarrow \{0,1\}^{\ell(\lambda)}\}$$

are (t, ε)-indistinguishable.

In this paper we shall make use of a *tree PRG* which is a pseudorandom generator that expands a root seed mseed into N subseeds in a structured way. The principle is to label the root of a binary tree of depth $\lceil \log_2 N \rceil$ with mseed. Then, one inductively labels the children of each node with the output of a standard PRG applied to the node's label. The subseeds $(\mathsf{seed}_i)_{i \in [N]}$ are defined as the labels of the N leaves of the tree. A tree PRG makes it possible to reveal all the subseeds but a small subset $E \subset [N]$ by only revealing $|E| \cdot \log(N/|E|)$ labels of the tree (which is presumable much smaller than $N - |E|$). The principle is to reveal the labels on the siblings of the paths from the root of the tree to leaves $i \notin E$ (excluding the labels of those paths themselves). Those labels allow the verifier to reconstruct $(\mathsf{seed}_i)_{i \in E}$ while still hiding $(\mathsf{seed}_i)_{i \notin E}$.

Definition 2 (Collision-Resistant Hash Functions). *A family of functions $\{\mathrm{Hash}_k : \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$; $k \in \{0,1\}^{\kappa(\lambda)}\}_\lambda$ indexed by a security parameter λ is collision-resistant if there exists a negligible function ν such that, for any PPT algorithm $\mathcal{A}$, we have*

$$\Pr\left[\begin{array}{c} x \neq x' \\ \cap \ \mathrm{Hash}_k(x) = \mathrm{Hash}_k(x') \end{array} \middle| \begin{array}{l} k \leftarrow \{0,1\}^{\kappa(\lambda)}; \\ (x, x') \leftarrow \mathcal{A}(k) \end{array} \right] \leq \nu(\lambda) .$$

A collision resistant hash function can be used to build a *Merkle tree* (a.k.a. *hash tree*). This is a binary tree in which every leaf node is labelled with the cryptographic hash of a data block v_i, and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. Given a collision-resistant hash function $\mathrm{Hash}(\cdot)$, the Merkle hash root for $N = 2^n$ input data blocks $v_1, \ldots, v_N$, denoted $\mathrm{MerkleTree}(v_1, \ldots, v_N)$, is hence defined as

$$\mathrm{MerkleTree}(v_1, \ldots, v_N) = \begin{cases} \mathrm{Hash}\left(\mathrm{MerkleTree}(v_1, \ldots, v_{N/2}) \parallel \mathrm{MerkleTree}(v_{N/2+1}, \ldots, v_N) \right) & \text{if } N > 1 \\ \mathrm{Hash}(v_1) & \text{if } N = 1 \end{cases}$$

A Merkle tree makes it possible to show the consistence of a small subset $E \subset [N]$ of revealed inputs $(v_i)_{i \in E}$ with the hash root $h = \text{MerkleTree}(v_1, \ldots, v_N)$ without having to communicate all the other inputs $(v_i)_{i \notin E}$ (or their corresponding hash). The principle is to reveal the sibling paths of $(v_i)_{i \in E}$ in the Merkle tree, that we shall denote $\mathsf{auth}((v_1, \ldots, v_N), E)$, and which contains at most $|E| \cdot \log(N/|E|)$ hash values.

We now formally introduce the notion of commitment scheme which is instrumental in many zero-knowledge protocols.

Definition 3 (Commitment Scheme). *A commitment scheme is a triplet of algorithms* $(\mathsf{KeyGen}, \text{Com}, \mathsf{Verif})$ *such that*

- KeyGen *is a PPT algorithm that, on input* 1^λ*, outputs some public parameters* $\text{PP} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ *containing a definition of the message space, the randomness space and the commitment space.*
- Com *is a deterministic polynomial-time algorithm that, on input the public parameters* PP*, a message* x *and the randomness* ρ*, outputs a commitment* c*.*
- Verif *is a deterministic polynomial-time algorithm that, on input the public parameters* PP*, a message* x*, a commitment* c *and the randomness* ρ*, outputs a bit* $b \in \{0,1\}$*.*

In this article, the public parameter input PP will be made implicit in the calls to Com and Verif.

Definition 4 (Correctness Property). *A commitment scheme achieves correctness, if for any message* x *and any randomness* ρ*:*

$$\Pr[\mathsf{Verif}(x, c, \rho) = 1 \mid c \leftarrow \text{Com}(x; \rho)] = 1 \ .$$

Definition 5 (Hiding Property). *A commitment scheme is said computationally (resp. statistically, resp. perfectly) hiding if, for any two messages* x_0 *and* x_1*, the following distributions*

$$\{c \mid c \leftarrow \text{Com}(x_0; \rho), \rho \leftarrow \$\} \text{ and } \{c \mid c \leftarrow \text{Com}(x_1; \rho), \rho \leftarrow \$\}$$

are computationally (resp. statistically, resp. perfectly) indistinguishable.

Definition 6 (Binding Property). *A commitment scheme is binding if there exists a negligible function* ν *such that, for every (PPT) algorithm* $\mathcal{A}$*, we have*

$$\Pr\left[\begin{array}{c} x \neq x' \\ \cap \mathsf{Verif}(\text{PP}, x, c, \rho) = 1 \\ \cap \mathsf{Verif}(\text{PP}, x', c, \rho') = 1 \end{array} \middle| \begin{array}{l} \text{PP} \leftarrow \mathsf{KeyGen}(); \\ (x, x', \rho, \rho', c) \leftarrow \mathcal{A}(\text{PP}) \end{array}\right] \leq \nu(\lambda) \ ,$$

where the probability is taken over the randomness of $\mathcal{A}$ *and* KeyGen*. If we restrict* $\mathcal{A}$ *to being PPT, then the scheme is computationally binding. If the computation time of* $\mathcal{A}$ *is unbounded, then the scheme is statistically binding.*

2.2 Interactive Protocols

A two-party protocol is a triplet $\Pi = (\text{Init}, \mathcal{A}, \mathcal{B})$ where Init is an initialization algorithm that, on input 1^λ, produces a pair $(in_\mathcal{A}, in_\mathcal{B})$, and where $\mathcal{A}$ and $\mathcal{B}$ are two stateful algorithms, called the *parties*. The parties originally receive their inputs $in_\mathcal{A}$ and $in_\mathcal{B}$ then interacts by exchanging messages, and finally one of the parties, say $\mathcal{B}$, produces the output of the protocol. More formally, an execution of the protocol consists in a sequence:

$$\begin{array}{l} \mathsf{state}_\mathcal{A} \leftarrow \mathcal{A}(in_\mathcal{A}) \\ \mathsf{state}_\mathcal{B} \leftarrow \mathcal{B}(in_\mathcal{B}) \\ (\text{MSG}_\mathcal{A}[0], \mathsf{state}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{state}_\mathcal{A}) \\ \quad \vdots \\ (\text{MSG}_\mathcal{B}[i], \mathsf{state}_\mathcal{B}) \leftarrow \mathcal{B}(\mathsf{state}_\mathcal{B}, \text{MSG}_\mathcal{A}[i-1]) \end{array}$$

$(\textsc{Msg}_{\mathcal{A}}[i], \mathsf{state}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{state}_{\mathcal{A}}, \textsc{Msg}_{\mathcal{B}}[i])$

$\vdots$

$out \leftarrow \mathcal{B}(\mathsf{state}_{\mathcal{B}}, \textsc{Msg}_{\mathcal{A}}[n])$

The sequence of exchanged messages is called the *transcript* of the execution, which is denoted

$$\mathrm{View}(\langle \mathcal{A}(in_{\mathcal{A}}), \mathcal{B}(in_{\mathcal{B}})\rangle) := (\textsc{Msg}_{\mathcal{A}}[0], \textsc{Msg}_{\mathcal{B}}[1], \ldots, \textsc{Msg}_{\mathcal{A}}[n]) \ .$$

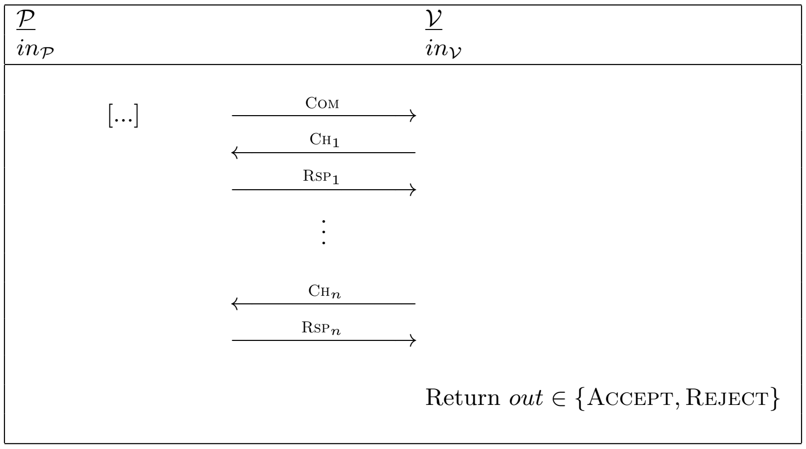
An execution producing an output *out* is further denoted

$$\langle \mathcal{A}(in_{\mathcal{A}}), \mathcal{B}(in_{\mathcal{B}})\rangle \to out \ .$$

In our exposition, the state of the parties shall be made implicit. We shall then say that an algorithm has *rewindable black-box access* to a party $\mathcal{A}$ if this algorithm can copy the state of $\mathcal{A}$ at any moment, relaunch $\mathcal{A}$ from a previously copied state, and query $\mathcal{A}$ (with its current state) on input messages. A variable x is said to be *extractable* from $\mathcal{A}$ if there exists a PPT algorithm $\mathcal{E}$ which, given a rewindable black-box access to $\mathcal{A}$, returns x after a polynomial number of queries to $\mathcal{A}$.

2.3 Zero-Knowledge Proofs of Knowledge

We will focus on a special kind of two-party protocol called an *interactive proof* which involves a *prover* $\mathcal{P}$ and a *verifier* $\mathcal{V}$. In such a protocol, $\mathcal{P}$ tries to prove a statement to $\mathcal{V}$. The first message sent by $\mathcal{P}$ is called a *commitment*, denoted $\textsc{Com}$. From this commitment $\mathcal{V}$ produces a first *challenge* $\textsc{Ch}_1$ to which $\mathcal{P}$ answers with a response $\textsc{Rsp}_1$, followed by a next challenge $\textsc{Ch}_2$ from $\mathcal{V}$, and so on. After receiving the last response $\textsc{Rsp}_n$, $\mathcal{V}$ produces a binary output: either $\textsc{Accept}$, meaning that she was convinced by $\mathcal{P}$, or $\textsc{Reject}$ otherwise. Such an m-round interactive proof with $m = 2n+1$ (1 commitment + n challenge-response pairs) is illustrated on Protocol 1.



Protocol 1: Structure of a m-round interactive proof with $m = 2n+1$.

The sequence of exchanged messages is called the *transcript* of the execution, which is denoted

$$\mathrm{View}(\langle \mathcal{P}(in_{\mathcal{P}}), \mathcal{V}(in_{\mathcal{V}})\rangle) := (\textsc{Com}, \textsc{Ch}_1, \textsc{Rsp}_1, \ldots, \textsc{Ch}_n, \textsc{Rsp}_n)$$

where $in_{\mathcal{P}}$ and $in_{\mathcal{V}}$ respectively denote the prover and verifier inputs. An execution producing an output $out \in \{\textsc{Accept}, \textsc{Reject}\}$ is further denoted

$$\langle \mathcal{P}(in_{\mathcal{P}}), \mathcal{V}(in_{\mathcal{V}})\rangle \to out \ .$$

Definition 7 (Proof of Knowledge). *Let x be a statement of language L in NP, and $W(x)$ the set of witnesses for x such that the following relation holds:*

$$\mathcal{R} = \{(x, w) : x \in L, w \in W(x)\} \ .$$

A proof of knowledge for relation $\mathcal{R}$ with soundness error ϵ is a two-party protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with the following two properties:

– *Perfect completeness: If $(x, w) \in \mathcal{R}$, then a prover $\mathcal{P}$ who knows a witness w for x succeeds in convincing the verifier $\mathcal{V}$ of his knowledge. More formally:*

$$\Pr[\langle \mathcal{P}(x, w), \mathcal{V}(x)\rangle \rightarrow \textsc{Accept}] = 1,$$

i.e. given the interaction between the prover $\mathcal{P}$ and the verifier $\mathcal{V}$, the probability that the verifier is convinced is 1*.*

– *Soundness: If there exists a PPT prover $\tilde{\mathcal{P}}$ such that*

$$\tilde{\epsilon} := \Pr[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x)\rangle \rightarrow \textsc{Accept}] > \epsilon,$$

then there exists an algorithm $\mathcal{E}$ (called an extractor) which, given rewindable black-box access to $\tilde{\mathcal{P}}$, outputs a witness w' for x in time $\mathsf{poly}(\lambda, (\tilde{\epsilon} - \epsilon)^{-1})$ with probability at least $1/2$.

Informally, a proof of knowledge has soundness error ϵ if a prover $\tilde{\mathcal{P}}$ without knowledge of the witness cannot convince the verifier with probability greater than ϵ assuming that the underlying problem (recovering a witness for the input statement) is hard. Indeed, if a prover $\tilde{\mathcal{P}}$ can succeed with a probability greater than ϵ, then the existence of the extractor (algorithm $\mathcal{E}$) implies that $\tilde{\mathcal{P}}$ can be used to compute a witness $w' \in W(x)$.

We now recall the notion of honest-verifier zero-knowledge proof:

Definition 8 (Honest-Verifier Zero-Knowledge Proof). *A proof of knowledge is {computationally, statistically, perfectly} honest-verifier zero-knowledge (HVZK) if there exists a PPT algorithm $\mathcal{S}$ (called simulator) whose output distribution is {computationally, statistically, perfectly} indistinguishable from the distribution* $\mathrm{View}(\langle \mathcal{P}(x, w), \mathcal{V}(x)\rangle)$ *obtained with an honest $\mathcal{V}$.*

Informally, the previous definition says a genuine execution of the protocol can be simulated without any knowledge of the witness. In other words, the transcript of an execution between the prover and an honest verifier does not reveal any information about the witness.

2.4 Secret Sharing and Multi-Party Computation

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 $[\![s]\!]_J := \big([\![s]\!]_i\big)_{i \in J}$.

Definition 9 (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*** $(\boldsymbol{t-1})$***-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 10 (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* $1 \le t_1 < t_2 \le N$*. 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 9 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 11 (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 12 (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*.*

A *multiparty computation* (MPC) protocol is an interactive protocol (as formally introduced in Section 2.2) involving multiple –possibly more than two– parties $\mathcal{P}_1$, …, $\mathcal{P}_N$. Each of these parties receives as input one share of a sharing $[\![x]\!]$. All together, the parties run the MPC protocol to compute $f(x)$ for some function f. At the end of the protocol, each party $\mathcal{P}_i$ outputs its own computed value of $f(x)$, denoted $f_i(x)$. In this paper, we only consider *complete* protocols for which an execution with honest parties results in all the parties outputting the right value $f(x)$. The *view* of a party $\mathcal{P}_i$ is composed of its input share $[\![x]\!]_i$, its random tape and all its received messages from the other parties (the sent messages can further be deterministically deduced from the other elements of the view).

We simply recall hereafter the notion of t-privacy for an MPC protocol in the *semi-honest model.* The interested reader is referred to [CDN15, EKR18] for more background and formalism about multiparty computation.

Definition 13 (Privacy in the Semi-Honest Model). *Let t and N be integers such that $1 \leq t < N$. Let Π_f be an MPC protocol with N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$, computing a function f. The protocol Π_f is* t-private in the semi-honest model *if for all $I \subset [N]$ such that $|I| \leq t$, there exists a PPT algorithm $\mathcal{S}$ such that $\mathcal{S}(I, [\![x]\!]_I, f_I(x))$ is perfectly indistinguishable from the joint distributions of the views of the parties in I, where $f_I(x) := \{f_i(x) \mid i \in I\}$.*

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 (see Definition 13 for a formal definition).

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 bound by $1 - \binom{N-2}{\ell-2}/\binom{N}{\ell}$, which ensures the soundness of the proof.

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

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 $\beta \leftarrow \psi(w, \varepsilon^1, \varepsilon^2, \dots; r)$ where $\varepsilon^1, \varepsilon^2, \dots$ are the previous outputs of $\mathcal{O}_R$ and where r is a fresh random tape.

Hints enable to build more efficient MPC protocols. For example, instead of computing a product of two shared values, the parties can get this product using $\mathcal{O}_H$ and simply check that the product is correct, which is cheaper in communication [BN20].

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 $[\![\alpha]\!]_1, \dots, [\![\alpha]\!]_N$ 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.[3] 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 $\alpha^1, \dots, \alpha^t$ satisfy the relation

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

for a given function g.

Protocol 2 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 3. For the sake of simplicity, we shall consider a single iteration in our presentation (as in Protocol 2) but we stress that the considered techniques and proofs equally apply to the multi-iteration setting (*i.e.* while replacing step (c) of Protocol 2 by Protocol 3).

Output distribution. In the following, we shall denote $\vec{\varepsilon} := (\varepsilon^1, \dots, \varepsilon^t)$, $\vec{\beta} := (\beta^1, \dots, \beta^t)$, $\vec{\alpha} := (\alpha^1, \dots, \alpha^t)$ and $\vec{r} := (r^1, \dots, r^t)$. 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:

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

where Φ is the deterministic function mapping $(w, \vec{\varepsilon}, \vec{\beta})$ to $\vec{\alpha}$ (defined by the coordinate functions $\varphi^1, \dots, \varphi^t$). 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:

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

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

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 3.*
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 2: General MPC protocol.

(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 3: General MPC protocol – Iterative computing & broadcasting step for iteration j (with η_j denoting the number of inner iterations).

– 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*.

To summarize, we consider a setting in which the output of the protocol has a probability distribution of the form described in Table 2.

The general MPC model introduced above captures a wide majority of the protocols used in the MPCitH context. To the best of our knowledge, all the practical instantiations of the MPCitH paradigm comply

	Output of f	
	Accept	Reject
$w : (x, w) \in \mathcal{R}$	1	0
$w : (x, w) \notin \mathcal{R}$	$\leq p$	$\geq 1 - p$

Table 2: Probability distribution of the output of the MPC protocol.

with this model except the ZKBoo [GMO16] and Ligero [AHIV17] proof systems. In particular, our model captures:

- the KKW18 protocol [KKW18] which computes arbitrary arithmetic circuits (used in Picnic2 and Picnic3),
- the product checking protocols in [BN20] and BN++ [KZ22],
- the product checking protocols in Limbo [DOT21] and Helium [KZ22],
- the MPC protocols in BBQ [DDOS19] and Banquet [BDK$^+$21],
- the MPC protocols in LegRoast [BD20] and PorcRoast [BD20],
- the MPC protocol in SDitH [FJR22],
- the MPC protocols in [FMRV22].

Example. As an illustration, we recall BN20 protocol to show how it fits our model. This MPC protocol takes as inputs three sharings $[\![x]\!]$, $[\![y]\!]$ and $[\![z]\!]$ where $x, y, z \in \mathbb{F}$ and aims to check that $z = x \cdot y$. It proceeds as follows:

- The parties get from $\mathcal{O}_H$ three hint sharings $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ such that $a, b \leftarrow \mathbb{F}$ and $c = a \cdot b$.
- The parties get from $\mathcal{O}_R$ a common random point $\varepsilon \leftarrow \mathbb{F}$.
- The parties locally compute and broadcast

$$[\![\alpha]\!] = \varepsilon \cdot [\![x]\!] + [\![a]\!] \quad \text{and} \quad [\![\beta]\!] = [\![y]\!] + [\![b]\!] \ .$$

- The parties publicly reconstruct α and β from $[\![\alpha]\!]$ and $[\![\beta]\!]$.
- The parties locally compute and broadcast

$$[\![v]\!] = \varepsilon \cdot [\![z]\!] - [\![c]\!] + \alpha \cdot [\![b]\!] + \beta \cdot [\![a]\!] - \alpha \cdot \beta.$$

- The parties publicly reconstruct v from $[\![v]\!]$.
- The parties output Accept if $v = 0$ and Reject otherwise.

The idea of this protocol is to take (as hint) a multiplicative triple (a, b, c) satisfying $c = a \cdot b$ and to "sacrifice" it using the randomness ε to check that $z = x \cdot y$. If $z \neq x \cdot y$ (or if the hint is not well-constructed), the protocol will output Reject with probability $1 - \frac{1}{|\mathbb{F}|}$, thus its false positive rate is $p = \frac{1}{|\mathbb{F}|}$. The protocol fits our model. Indeed, the number of round is $t := 1$ and we have

- ψ^1 is a randomized function that returns a triple (a, b, c) such that (a, b) is random and $c = a \cdot b$;
- ϵ^1 is a random field element;
- φ^1 is split in two subfunctions $\varphi^{1,1}$ and $\varphi^{1,2}$, as described in Protocol 3 when $\eta_1 = 2$:

$$\varphi^{1,1}(x, y, z, a, b, c, \epsilon) := (x + \epsilon \cdot a, y + b)$$
$$\varphi^{1,2}(x, y, z, a, b, c, \epsilon, \alpha, \beta) := \epsilon \cdot z - c + \alpha \cdot b + \beta \cdot a + \alpha \cdot \beta$$

where $(\alpha, \beta) := \varphi^{1,1}(x, y, z, a, b, c, \epsilon)$.

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 4: 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.

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 $i \in \{1, \dots, N\}$, for some commitment randomness ρ_i^j;
   (c) the prover sends
   $$h_j := \begin{cases} \mathrm{Hash}(\mathsf{com}_1^1, \dots, \mathsf{com}_N^1) & \text{if } j = 1 \\ \mathrm{Hash}(\mathsf{com}_1^j, \dots, \mathsf{com}_N^j, [\![\alpha^{j-1}]\!]) & \text{if } j > 1 \end{cases}$$
   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 3 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 $([\![w]\!]_i, ([\![\beta^j]\!]_i, \rho_i^j)_{j\in[t]})_{i\neq i^*}$, $\mathsf{com}_{i^*}^1, \dots, \mathsf{com}_{i^*}^t$, $[\![\alpha^1]\!]_{i^*}, \dots, [\![\alpha^t]\!]_{i^*}$ 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 $t + 1$,
   $$h_j \overset{?}{=} \begin{cases} \mathrm{Hash}(\mathsf{com}_1^1, \dots, \mathsf{com}_N^1) & \text{if } j = 1 \\ \mathrm{Hash}(\mathsf{com}_1^j, \dots, \mathsf{com}_N^j, [\![\alpha^{j-1}]\!]) & \text{if } j > 1 \\ \mathrm{Hash}([\![\alpha^t]\!]) & \text{if } j = t+1 \end{cases}$$
   (b) the output of the MPC protocol is Accept, *i.e.*
   $$g(\alpha^1, \dots, \alpha^t) \overset{?}{=} 0.$$

Protocol 4: Zero-knowledge protocol - Application of the MPCitH principle to Protocol 2.

Protocols with preprocessing phase. An additional technique can be used in this framework: one can assume that the parties take an auxiliary input which is a sharing $[\![y]\!]$ of a value y satisfying some public equation $E(y) = 0$ (where E might depend on the public statement x). For example, the MPC protocol can take as input three sharings $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ for random a, b and c satisfying $c = a \cdot b$. In practice, this assumption is verified thanks to a cut-and-choose strategy run before the emulation of the MPC protocol. Introduced by [KKW18] using a preprocessing phase, this technique has been formalized in [Beu20] as *protocols with helper*. Although we omit such a preprocessing phase from our formalism for the sake of simplicity, we stress that the theory developed in the present article is compatible with this technique and adding a preprocessing phase to the protocol would not change our results.

Soundness. Assuming that the underlying MPC protocol follows the model of Section 3.1 with a false positive rate p, the soundness error of Protocol 4 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 2 in the more general context of any (threshold) linear secret sharing (see Theorem 2).

Performances. The communication of Protocol 4 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 β^1, $\ldots$, β^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 in com_N^1 (instead of committing $[\![\beta^{\mathrm{unif}}]\!]_N$). This way the prover can save communication by revealing seed_N instead of $[\![\beta^{\mathrm{unif}}]\!]_N$ 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 4 of

– when $i^* \neq N$,

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

– when $i^* = N$,

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

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

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 9, which shares a secret $s \in \mathbb{F}$ into N shares $[\![s]\!] \in \mathbb{F}^N$. In particular, the vector spaces of Definition 9 are simply defined as $\mathbb{V}_1 = \mathbb{V}_2 = \mathbb{F}$ hereafter (other definitions of these sets will be considered in Section 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 Appendix B). 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 $[\![s]\!]_{J_0}$.

Remark 1. In the case of the additive sharing scheme, we have $\ell + 1 = N$ so that the algorithm Expand is trivial (it simply consists of the identity function). In the case of Shamir's secret sharing scheme (see Definition 12), the algorithm Expand builds the underlying polynomial and evaluates it into each party' point.

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, $\mathsf{Reconstruct}$ 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[4], when there exists J such that $[\![v]\!] = \mathsf{Expand}_J([\![v]\!]_J)$.
- **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 2). 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 5, 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, see Definition 13). This is formalized in the following theorem (see proof in Appendix C).

Theorem 1. *Let us consider an MPC protocol Π_{add} complying with the protocol format described in Protocol 2. 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 5) 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 5) 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

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

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

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
   $$[\![\alpha^j]\!] := \varphi^j_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j}}\left([\![w]\!], ([\![\beta^i]\!])_{i\le j}\right) ,$$
   - 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 3.
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 5: General MPC protocol Π_{LSSS} with LSSS.

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 $[\![\alpha^j]\!] = ([\![\alpha^j]\!]_1, \ldots, [\![\alpha^j]\!]_N)$ 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 6.

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 3).*

   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 $t+1$,
   $$h_j \overset{?}{=} \begin{cases} \tilde{h}_j & \text{if } j = 1 \\ \mathrm{Hash}(\tilde{h}_j, [\![\alpha^{j-1}]\!]_S) & \text{if } 2 \le j \le t \\ \mathrm{Hash}([\![\alpha^{j-1}]\!]_S) & \text{if } j = t+1 \end{cases}$$
   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.*
   $$g(\alpha^1, \ldots, \alpha^t) \overset{?}{=} 0 .$$

Protocol 6: Zero-knowledge protocol: application of the MPCitH principle to Protocol 5 with an $(\ell+1, N)$-threshold linear secret sharing scheme.

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

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

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 $\alpha^1, \dots, \alpha^t$ (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 $\alpha^1, \dots, \alpha^t$ such that $g(\alpha^1, \dots, \alpha^t) = 0$, 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

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

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 5 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 6 built from Π_{LSSS} satisfies the three following properties:*

- ***Completeness.*** *A prover $\mathcal{P}$ who knows a witness w such that $(x, w) \in \mathcal{R}$ and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*
- ***Soundness.*** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input x, convinces the honest verifier $\mathcal{V}$ to accept with probability*
$$\tilde{\epsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle(x) \to 1] \;>\; \epsilon$$
where the soundness error ϵ is defined as
$$\epsilon := \frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell \cdot (N-\ell)}{\ell+1} \;.$$
Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, outputs either a witness w satisfying $(x, w) \in \mathcal{R}$, or a commitment collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by
$$\frac{4}{\tilde{\epsilon} - \epsilon} \cdot \left(1 + \tilde{\epsilon} \cdot \frac{8 \cdot (N-\ell)}{\tilde{\epsilon} - \epsilon}\right).$$
- ***Honest-Verifier Zero-Knowledge.*** *There exists an efficient simulator $\mathcal{S}$ which, given the random challenge I outputs a transcript which is indistinguishable from a real transcript of Protocol 6.*

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 Appendix D for the soundness proof.

Remark 2. 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 3. 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 3 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).

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.

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

Table 3: Number of party emulations to achieve a soundness error of $2^{-\lambda}$ (assuming a negligible false positive rate p).

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 Section 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.

Remark 4. As in the additive case, the prover can generate the input shares from seeds for ℓ parties, and those seeds can be built using a seed tree. However, this tweak will improve (significantly) the communication cost only when the underlying LSSS has a high threshold (as *e.g.* in the case of additive sharing).

Let us illustrate the overhead in communication cost when $\ell = 1$ and $t = 1$ and negligible unif (which is often small in practice). In this setting, we obtain an average overhead of $\Delta\mathsf{Cost} \approx \tau \cdot \lambda \cdot (\log_2 N - 2)$. When targeting a λ-bit security, we have $\tau \approx \frac{\lambda}{\log_2 N}$, which gives

$$\Delta\mathsf{Cost} \approx \lambda^2 \cdot \left(1 - \frac{2}{\log_2 N}\right).$$

We can observe that the communication overhead due to the use of a LSSS is fixed for a given security parameter, and roughly independent on the underlying MPC protocol. When targeting a 128-bit security, this base cost is around 2 KB (for the case $\ell = 1$ and $t = 1$).

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. Informally, a linearly homomorphic commitment scheme is such that

$$\forall \alpha, \beta, x, y, \rho_x, \rho_y,$$
$$\alpha \cdot \mathrm{Com}(x; \rho_x) + \beta \cdot \mathrm{Com}(y; \rho_y) = \mathrm{Com}(\alpha \cdot x + \beta \cdot y; \alpha \cdot \rho_x + \beta \cdot \rho_y).$$

Instead of committing all the parties' input shares, the prover can just commit the input shares for the $\ell + 1$ parties in $S := \{s_0, \ldots, s_\ell\}$:

$$\begin{aligned}
\mathsf{com}_0 &= \mathrm{Com}([\![w]\!]_{s_0}; \rho_0)\\
\mathsf{com}_1 &= \mathrm{Com}([\![w]\!]_{s_1}; \rho_1)\\
&\vdots\\
\mathsf{com}_\ell &= \mathrm{Com}([\![w]\!]_{s_\ell}; \rho_\ell).
\end{aligned}$$

By linearity of Expand_S, there exists a matrix $M \in \mathbb{F}^{N\times(\ell+1)}$ such that

$$\mathsf{Expand}_S(\llbracket w \rrbracket_S) = M \cdot \llbracket w \rrbracket_S \ .$$

To open a party i, the prover then only needs to open the commitment com'_i defined as

$$\begin{aligned} \mathsf{com}'_i := \sum_{j=0}^{\ell} M_{i,j} \cdot \mathsf{com}_j &= \mathrm{Com}\Big(\sum_{j=0}^{\ell} M_{i,j} \llbracket w \rrbracket_j \ ; \ \sum_{j=0}^{\ell} M_{i,j}\rho_i\Big) \\ &= \mathrm{Com}\Big(\llbracket w \rrbracket_i \ ; \ \sum_{j=0}^{\ell} M_{i,j}\rho_i\Big) \end{aligned}$$

Instead of committing N shares, the prover just needs to commit $\ell+1$ of them which makes the commitment phase much more efficient. Another benefit of this approach is to enforce that the committed sharing corresponds to a valid sharing, which improves the soundness error to the standard formula:

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

which is substantially better than the soundness error from Theorem 2 whenever p is not negligible.

The resulting proof size highly depends on the underlying homomorphic commitment scheme. This technique allows to produce zero-knowledge protocols with soundness as good as with algebraic techniques (when there exist). For example, to prove a statement about lattices on field $\mathbb{F}_q$ with $q \approx 2^{32}$, it would be possible to use the MPCitH paradigm with threshold LSSS to get a single-iteration protocol with soundness error of 2^{32} when taking $\ell = 1$, scaling to 2^{123} when taking $\ell = 4$.

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 6 must be chosen such that $\llbracket v \rrbracket_S$ enables to deduce all the shares $\llbracket v \rrbracket_{[N]}$. 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 $\llbracket \alpha^1 \rrbracket_{i^*}, \ldots, \llbracket \alpha^t \rrbracket_{i^*}$ for one non-opened party $i^* \in S$ might not be enough to enable the verifier to recompute $\llbracket \alpha^j \rrbracket_S$ for all j. Therefore the size of S and the number of additional shares $\llbracket \alpha^j \rrbracket_i$ 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 $\Pi_{QT\text{-}LSSS}$ complying with the protocol format described in Protocol 5, but using an $(\ell, \ell+\Delta+1, N)$-quasi-threshold LSSS in place of an $(\ell+1, N)$-threshold LSSS, and such that $\Pi_{QT\text{-}LSSS}$ is ℓ-private in the semi-honest model and of false positive rate p. Then, Protocol 6 built from $\Pi_{QT\text{-}LSSS}$ satisfies the three following properties:*

– ***Completeness.*** *A prover $\mathcal{P}$ who knows a witness w such that $(x, w) \in \mathcal{R}$ and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*
– ***Soundness.*** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input x, convinces the honest verifier $\mathcal{V}$ to accept with probability*
$$\tilde{\epsilon} := \Pr[\langle\tilde{\mathcal{P}}, \mathcal{V}\rangle(x) \to 1] > \epsilon$$
where the soundness error ϵ is equal to
$$\frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+\Delta+1} \cdot \binom{N-\ell}{\Delta+1}.$$
Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces with either a witness w satisfying $(x, w) \in \mathcal{R}$, or a commitment collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by
$$\frac{4}{\tilde{\epsilon} - \epsilon} \cdot \left(1 + \tilde{\epsilon} \cdot \frac{8 \cdot (N - \ell)}{\tilde{\epsilon} - \epsilon}\right).$$
– ***Honest-Verifier Zero-Knowledge.*** *There exists an efficient simulator $\mathcal{S}$ which, given random challenge I outputs a transcript which is indistinguishable from a real transcript of Protocol 6.*

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 Appendix E 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). Some sharing schemes on algebraic curves, which are not (tight) threshold but quasi-threshold, have been proposed in [CC06] to handle this issue.

Assuming a negligible false positive rate p, the soundness error is $\binom{\ell+\Delta}{\ell}/\binom{N}{\ell}$ for a quasi-threshold scheme instead of $1/\binom{N}{\ell}$ for a tight threshold scheme. Let us focus on the case $\ell = 1$. The soundness error for a quasi-threshold scheme then satisfies
$$\frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}} = \frac{\Delta+1}{N}\ .$$
In order to gain in soundness (and hence in performances), the above formula should be lower than $1/|\mathbb{F}|$ which is the minimal achievable soundness error for a (tight) threshold scheme (since $N \leq \mathbb{F}$).

In [CC06], the gap Δ is $2g$ where g is the genus of the underlying curve. We must then search for quasi-threshold sharing schemes such that
$$\frac{2g+1}{N} \leq \frac{1}{|\mathbb{F}|} \quad \Leftrightarrow \quad N \geq |\mathbb{F}| \cdot (2g+1)\ ,$$
while for such sharing, we have $N \leq |C(\mathbb{F})|$ where $|C(\mathbb{F})|$ is the order of the underlying curve over the field $\mathbb{F}$. However, according to the Hasse-Weil inequality [Was08], we have
$$|C(\mathbb{F})| \leq |\mathbb{F}| + 1 + 2g\sqrt{|\mathbb{F}|}$$
which shows the impossibility of finding an algebraic geometric secret sharing scheme satisfying the above constraint. We deduce that a direct application of algebraic geometric secret sharing schemes [CC06] does not achieve better soundness (and hence better performances) than standard threshold sharing schemes in our context.

We stress that the above argument focuses on the case $\ell = 1$ for simplicity (and since it is relevant to optimize the performances with our approach), but the above argument also holds for any $\ell \geq 1$.

While the above analysis discards the interest in using quasi-threshold LSSS based on algebraic geometry to improve the soundness-performances trade-off of our scheme, we let this question open for other quasi-threshold schemes. However, [CDN15, Theorem 11.121] gives that an $(\ell, \ell+\Delta+1, N)$-quasi-threshold $\mathbb{F}$-linear secret-sharing scheme satisfies

$$\Delta + 1 \geq \frac{N+2}{2|\mathbb{F}|-1}.$$

Thus, we get a lower bound on the soundness error (on the case $\ell = 1$):

$$\epsilon := \frac{\Delta+1}{N} \geq \frac{1}{N} \cdot \frac{N+2}{2|\mathbb{F}|-1} \geq \frac{1}{2|\mathbb{F}|-1},$$

implying that such sharing schemes could only have a limited interest to optimize the soundness error.

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 simultaneously. 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 2). 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

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

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 2) must be of the form

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

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 (see Figure 1 from Section 6 for illustration). 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 Figure 1 compares the three strategies for batched proofs of the SDitH scheme [FJR22].

Remark 5. In our analysis, we use the number of emulated parties as an indicator of the computational performance. As explained in Section 4, we would also need to take into account the computation cost for computing and committing the input sharings (and hints' sharings) which is not negligible, but this cost is hard to estimate without a concrete implementation. We yet remind that the latter cost only impacts the prover and not the verifier. The verification time is soundly predicted by the number of party emulations.

6 Applications

In the past few years, many proof systems relying on the MPC-in-the-Head paradigm have been published. Table 4 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 Section 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 Section 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.

In the following, we present two applications of our strategy with Shamir's secret sharing as threshold LSSS:

- we first apply our general strategy to the SDitH signature scheme [FJR22] to obtain a new variant with faster signing and verification times;
- we then apply our batching technique (see Section 5.3) to the SDitH scheme (batch proofs for syndrome decoding) and to the Limbo proof system [DOT21] (batched proofs for general arithmetic circuits).

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

Table 4: 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.

6.1 Application to the SDitH Signature Scheme

We can transform the zero-knowledge proofs of knowledge described in Section 4 into signature schemes using the Fiat-Shamir's heuristic [FS87]. We describe the signature scheme obtained when following this approach for the 5-round case (*i.e.* for $t = 1$ iteration in the MPC protocol) in Appendix F and further prove that this scheme achieves $\mathsf{EUF\text{-}CMA}$ security in the random oracle model.

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].

The security of the SDitH signature scheme relies on the hardness of solving a syndrome decoding (SD) problem. This problem is one of the oldest problems in code-based cryptography: given a matrix $H \in \mathbb{F}_{\mathrm{SD}}^{(m-k)\times m}$ and a vector $y \in \mathbb{F}_{\mathrm{SD}}^{m-k}$, it consists to find a vector $x \in \mathbb{F}_{\mathrm{SD}}^{m}$ such that $y = Hx$ and such that x has a given Hamming weight w (or a Hamming weight lower than w), *i.e.* has (at most) w non-zero coordinates. The authors of the SDitH scheme describe an MPC protocol that checks whether a vector x is a correct witness of a public SD instance (H, y), and convert it into a (non-interactive) zero-knowledge proof using the MPC-in-the-Head paradigm. Using the same notations as in [FJR22], the MPC protocol involves three fields $\mathbb{F}_{\mathrm{SD}} \subseteq \mathbb{F}_{\mathrm{poly}} \subseteq \mathbb{F}_{\mathrm{points}}$ which are extensions of each other and such that $\mathbb{F}_{\mathrm{SD}}$ is the base field of the SD instance. This MPC protocol fits the model introduced in Section 3.1 (see Protocol 2), with the number t of loop iterations equal to 1. Using the same notations as in Section 4, the proof size involves the following quantities:

- $\mathsf{inputs} = k \cdot \log_2 |\mathbb{F}_{\mathrm{SD}}| + 2w \cdot \log_2 |\mathbb{F}_{\mathrm{poly}}| + t' \cdot \log_2 |\mathbb{F}_{\mathrm{points}}|$,
- $\mathsf{unif} = 2 \cdot d \cdot t' \cdot \log_2 |\mathbb{F}_{\mathrm{points}}|$,
- $\mathsf{comm} = 2 \cdot d \cdot t' \cdot \log_2 |\mathbb{F}_{\mathrm{points}}|$,

where (d, t') are additional parameters (see [FJR22] for more details). The signature size (in bits), including a 2λ-bit salt, is then given by

$$\textsc{Size} = 6\lambda + \tau \cdot \Big(\mathsf{inputs} + \mathsf{comm} + \lambda \cdot \log_2(N) + 2\lambda\Big)$$

where N is the number of MPC parties, τ the number of executions and λ is the security level. Its security is given by the attack of [KZ20a] and is equal to

$$\mathrm{cost}_{\mathrm{forge}} := \min_{\tau_1,\tau_2:\tau_1+\tau_2=\tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i}} + N^{\tau_2} \right\}$$

where p is the false positive rate of the used MPC protocol satisfying

$$p \le \sum_{i=0}^{t'} \frac{\max_{\ell \le (m+w)/d-1} \left\{ \binom{\ell}{i} \binom{|\mathbb{F}_{\mathrm{points}}|-\ell}{t'-i} \right\}}{\binom{|\mathbb{F}_{\mathrm{points}}|}{t'}} \left(\frac{1}{|\mathbb{F}_{\mathrm{points}}|} \right)^{t'-i}.$$

The authors of the SDitH scheme propose several instantiations of their scheme: some of them with $\mathbb{F}_{\mathrm{SD}} := \mathbb{F}_2$ and the others with $\mathbb{F}_{\mathrm{SD}} := \mathbb{F}_{256}$, some of them aiming for short signatures and the others aiming for fast signature/verification time. We apply the ideas of Section 4 to this scheme using Shamir's secret sharing. Since the number N of parties is limited by the field size, $N \le |\mathbb{F}_{\mathrm{SD}}|$,[5] 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 Section 4.4, we obtain signatures of size (in bits):

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

[5] 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).

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 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 Section 4.3 and formally analyzed in our proof of soundness (see Appendix D), 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. Instead of emulating $\ell+1$ parties as described in Protocol 6, the implementation runs the MPC protocol directly on the coefficients of the degree-ℓ polynomials of the Shamir's secret sharing, thus avoiding costly polynomial evaluations and interpolations.

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

Let us remark that, as in [FJR22], we did not investigate software optimizations (like vectorization or bitslicing), so there is room for improvement in the reported performances.

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$^+$-128s	-	-	-	-	-	-	7.7 KB	239 ms	0.7 ms

Table 5: 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]).

6.2 Application of the Batching Strategy

Application to SDitH. As the first application of the batching strategy described in Section 5.3, we show how to batch proofs of knowledge for the syndrome decoding problem. Specifically, we consider a context where, from a public parity-check matrix $H \in \mathbb{F}_{\text{SD}}^{(m-k)\times m}$ and vectors $y_1, \dots, y_u \in \mathbb{F}_{\text{SD}}^{m-k}$, one wants to batch proofs of knowledge for u syndromes $x_1, \dots, x_u \in \mathbb{F}_{\text{SD}}^{m}$ such that

$$\forall i \in [u], y_i = H \cdot x_i \ .$$

For this purpose, we apply our batching strategy to the SDitH scheme [FJR22]. However, in its original version, this scheme is not compatible with the application of batched Shamir's secret sharing (on which rely our batching strategy) since it involves multiplications by witness-dependent scalars. Those appear in the final product verification based on the [BN20] protocol. To overcome this issue, we propose a tweak of the SDitH scheme to verify the final multiplication triple without relying on [BN20]. Specifically, to prove that three sharings $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ verify $c = a \cdot b$ ($a, b, c \in \mathbb{F}_{\text{points}}$), the parties proceed as follows:

- they get as hints

$$\begin{aligned} &[\![r_1]\!] \text{ where } r_1 \leftarrow \mathbb{F}_{\text{points}} \\ &[\![r_2]\!] \text{ where } r_2 \leftarrow \mathbb{F}_{\text{points}} \\ &[\![h_1]\!] \text{ where } h_1 = r_1 \cdot b + r_2 \cdot a \\ &[\![h_2]\!] \text{ where } h_2 = r_1 \cdot r_2, \end{aligned}$$

- they get a random point $\xi \leftarrow \mathbb{F}_{\text{points}} \backslash \{0\}$,
- they locally compute and broadcast

$$\begin{cases} [\![v_1]\!] = \xi \cdot [\![r_1]\!] + [\![a]\!] \\ [\![v_2]\!] = \xi \cdot [\![r_2]\!] + [\![b]\!] \\ [\![v_3]\!] = \xi^2 \cdot [\![h_2]\!] + \xi \cdot [\![h_1]\!] + [\![c]\!], \end{cases}$$

- they output Accept if $v_1 \cdot v_2 = v_3$, Reject otherwise.

If $c = a \cdot b$ and the hints are correctly computed, then the polynomial

$$P(X) := (r_1 X + a) \cdot (r_2 X + b) - (h_2 X^2 + h_1 X + c)$$

equals 0 so that $v_1 \cdot v_2 - v_3 = P(\xi) = 0$ and the parties accept. If $c \neq a \cdot b$, then $P(X)$ is different from zero. Thus according to the Schwartz-Zippel Lemma, the probability that the parties accept is at most $\frac{2}{|\mathbb{F}_{\text{points}}|-1}$ (for any adversarial choice of the hints).

Replacing BN20 with the above protocol in the SDitH scheme increases the number of rounds in the underlying zero-knowledge protocol from 5 to 7. Indeed the challenge ξ cannot be sent at the same time as the evaluation point anymore, since $[\![h_1]\!]$ and $[\![h_2]\!]$ must be committed before receiving ξ. We then restrict our variant of the SDitH protocol to a single evaluation point ($t' = 1$) to ease the analysis of the [KZ20a] attack while turning to the non-interactive setting. Using the same notations as Section 4, the new proof size involves the following quantities:

- $\mathsf{inputs} = k \cdot \log_2 |\mathbb{F}_{\text{SD}}| + 2w \cdot \log_2 |\mathbb{F}_{\text{poly}}| + \underbrace{2 \cdot \log_2 |\mathbb{F}_{\text{points}}|}_{[\![h_1]\!],[\![h_2]\!]}$,
- $\mathsf{unif} = \underbrace{2 \cdot \log_2 |\mathbb{F}_{\text{points}}|}_{[\![r_1]\!],[\![r_2]\!]}$,
- $\mathsf{comm} = \underbrace{2 \cdot \log_2 |\mathbb{F}_{\text{points}}|}_{[\![v_1]\!],[\![v_2]\!]}$,

We apply the three batching strategies described in Section 5.3 for the syndrome decoding parameters $(m, k, w) = (256, 128, 80)$ and using the field extension $\mathbb{F}_{\text{points}} := \mathbb{F}_{2^{192}}$. Figure 1 illustrates the resulting performance in terms of proof size and number of party emulations. Batching with additive sharings (using the naive strategy) with $N = 256$ is represented by the dashed line for which the achieved amortized cost per statement is around 6 KB.

Naive strategy: When using the naive strategy with low-threshold Shamir's secret sharing (for $N = 256$), the proof size is a bit larger than for the additive case, but the number of emulations is divided by more than 100.

SSS-based strategy: When the number u of batched proofs is small enough ($u \leq 80$), this strategy outperforms the additive case in terms of proof size. But it grows exponentially as u increases (slower when ℓ is larger). As explained in Section 5.3, this behavior is amplified when the underlying field is small. Here the number N of parties is limited by $N \leq |\mathbb{F}_{256}| + 1 - u = 257 - u$. For instance, when $u = 110$ and $\ell = 2$, the soundness error ϵ is approximately 0.57 with N maximal ($N = 147$), requiring a high number of executions $\tau \geq 150$.

Hybrid strategy: As expected, for such a context with limited N, the hybrid approach gives the best results. We get an amortized proof size of around 2.3 KB when $\ell = 1$ and around 0.83 KB when $\ell = 8$.

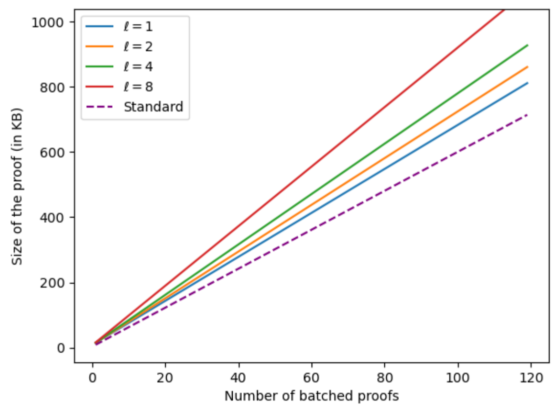
Application to Limbo. We now apply our batching strategy to Limbo [DOT21], a proof system for general arithmetic circuits. In Limbo, sharings are multiplied by scalars in three different contexts:

- After sampling a random challenge $\{r_i\}_{i=1}^m$, a set of multiplicative triples $([\![x_i]\!], [\![y_i]\!], [\![z_i]\!])_{i=1}^m$ is verified as an inner product $\langle a, b\rangle = c$ where

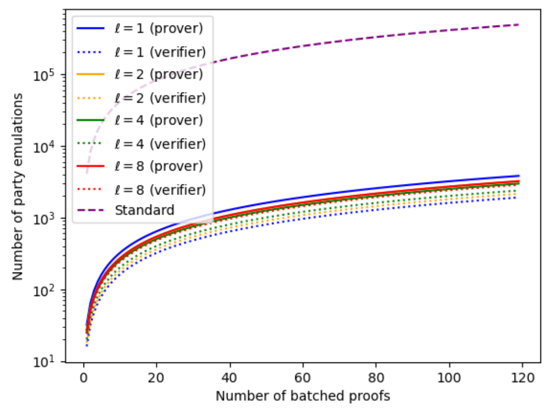
$$[\![a]\!] = \begin{pmatrix} r_1 \cdot [\![x_1]\!] \\ \vdots \\ r_m \cdot [\![x_m]\!] \end{pmatrix}, \quad [\![b]\!] = \begin{pmatrix} [\![y_1]\!] \\ \vdots \\ [\![y_m]\!] \end{pmatrix}, \quad [\![c]\!] = \sum_{i=1}^m r_i \cdot [\![z_i]\!].$$

If $\langle a, b\rangle = c$ for a random choice of $\{r_i\}_{i=1}^m$, then we can deduce that $x_i \cdot y_i = z_i$ for every $i \in [m]$ with high probability.[6]

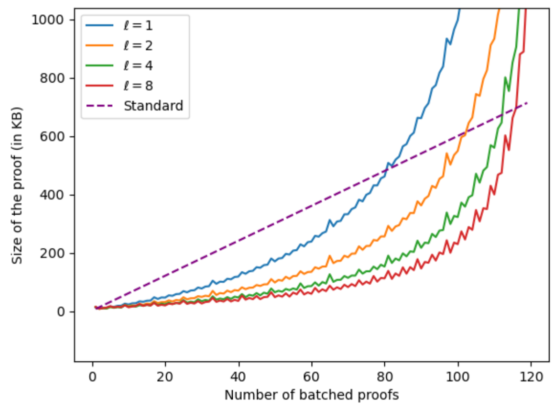
[6] In Limbo, the challenge is of the form $\{r_i\}_{i=1}^m := \{r^i\}_{i=1}^m$ for a random $r \in \mathbb{F}$.



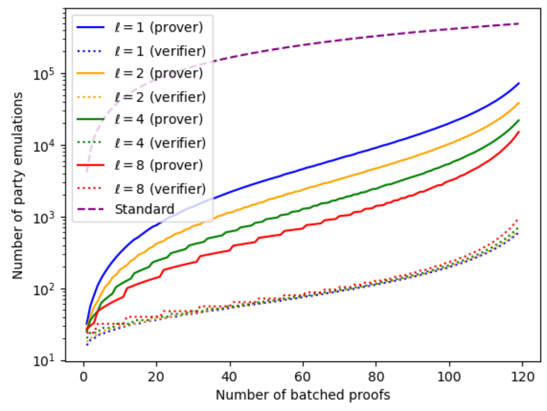
(a) Naive strategy: proof size



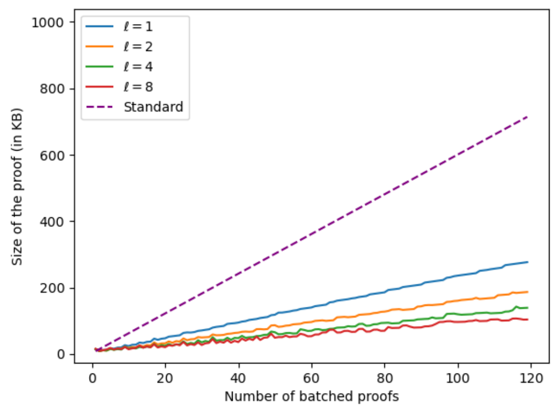
(b) Naive strategy: # party emul.



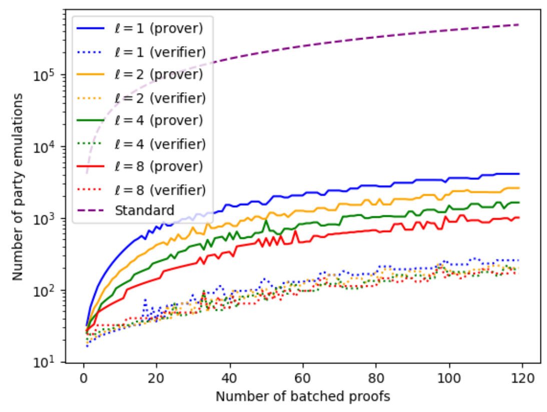
(c) SSS-based strategy: proof size



(d) SSS-based strategy: # party emul.



(e) Hybrid strategy: proof size



(f) Hybrid strategy: # party emul.

Fig. 1: Performances of the three batching strategies applied to our tweaked SDitH scheme.

- To interpolate a polynomial f of degree d such that

$$[\![f]\!](e_i) = [\![x_i]\!] \text{ for } i \in \{0, \ldots, d\}$$

where $e_0, \ldots, e_d$ are fixed distinct public points. With the Lagrange interpolation formula, we get that

$$[\![f]\!](X) = \sum_{i=0}^{d} [\![x_i]\!] \cdot \underbrace{\prod_{j=0, j\neq i}^{m} \frac{X - e_j}{e_i - e_j}}_{\ell_i(X)}$$

where $\{\ell_i(X)\}_i$ are public constant polynomials.
- To evaluate a polynomial $[\![f]\!](X) := \sum_{i=0}^{d} [\![a_i]\!] X^i$ in a challenge point s:

$$[\![f(s)]\!] = \sum_{i=0}^{d} [\![a_i]\!] \cdot s^i.$$

In all those three contexts, the scalars which multiply the shares are constant or derived from random challenges. In particular, they are witness-independent which makes our batching strategy applicable (see Section 5.3 for details).

Let us briefly explain the high-level idea of the MPC protocol underlying Limbo. This protocol aims to check a list of m multiplicative triples. To proceed, it first converts the list into an inner product of dimension m (as recalled above), and then repeats a compression step that reduces the inner product dimension until reaching a small enough instance. Given a compression parameter k, the compression step works as follows (the process is illustrated in Figure 2):

1. it splits the current inner product into k smaller inner products,
2. it compresses the k inner products into a single inner product. Let us consider the k values $v^{(1)}$, ..., $v^{(k)}$ of these inner products in the same position. To compress these k values into a single one v, it builds the polynomial P of degree $k-1$ such that

$$\forall i \in \{1, \ldots, k\},\ P(i) = v^{(i)}$$

and it computes v as $P(\xi)$ where ξ is a random verifier challenge.

Let us consider the case where Limbo will always finish on a final inner product of dimension 1, after $\rho := \lceil \log_k(m) \rceil$ compression steps. Using the same notations as in Section 4, the proof size involves the following quantities:

- $\mathsf{inputs} = \underbrace{(|w| + |C|) \cdot \log_2 \mathbb{F}}_{\text{Protocol inputs}} + \underbrace{\rho \cdot (k-1) \cdot \log_2 \mathbb{G}}_{\text{Splitting cost}} + \underbrace{(\rho \cdot (k-1) + 2) \cdot \log_2 \mathbb{G}}_{\text{Compression cost}},$
- $\mathsf{unif} = 2 \cdot \log_2 \mathbb{G}$,
- $\mathsf{comm} = 2 \cdot \log_2 \mathbb{G}$,

where $|w|$ is the size of the witness (*i.e.* the circuit input), $|C|$ is the number of multiplications in the circuit, $\mathbb{F}$ is the base field of the circuit, $\mathbb{G}$ is a field extension of $\mathbb{F}$ and k is the compression factor (see [DOT21] for more details). Limbo's proof size (in bits) is then given by:

$$\textsc{Size} = (\rho + 2) \cdot 2\lambda + \tau \cdot \Big(\mathsf{inputs} + \mathsf{comm} + \lambda \cdot \log_2(N) + 2\lambda\Big)$$

where N is the number of MPC parties, τ is the number of executions and λ is the security level. Limbo's soundness error is given by:

$$\left(\frac{1}{N} + p_k \cdot (1 - \frac{1}{N})\right)^{\tau}$$

$$\left\langle \begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_\ell^{(1)} \\ \hline \vdots \\ \hline x_1^{(k)} \\ \vdots \\ x_\ell^{(k)} \end{pmatrix}, \begin{pmatrix} y_1^{(1)} \\ \vdots \\ y_\ell^{(1)} \\ \hline \vdots \\ \hline y_1^{(k)} \\ \vdots \\ y_\ell^{(k)} \end{pmatrix} \right\rangle = z^{(1)} + \ldots + z^{(k)}$$

$$\text{Step 1} \Big\Downarrow \text{Splitting}$$

$$\left\langle \begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_\ell^{(1)} \end{pmatrix}, \begin{pmatrix} y_1^{(1)} \\ \vdots \\ y_\ell^{(1)} \end{pmatrix} \right\rangle = z^{(1)}, \ \ldots, \ \left\langle \begin{pmatrix} x_1^{(k)} \\ \vdots \\ x_\ell^{(k)} \end{pmatrix}, \begin{pmatrix} y_1^{(k)} \\ \vdots \\ y_\ell^{(k)} \end{pmatrix} \right\rangle = z^{(k)}$$

$$\text{Step 2} \Big\Downarrow \text{Compression}$$

$$\left\langle \begin{pmatrix} x_1 \\ \vdots \\ x_\ell \end{pmatrix}, \begin{pmatrix} y_1 \\ \vdots \\ y_\ell \end{pmatrix} \right\rangle = z$$

where $v = \sum_{i=1}^{k} v^{(i)} \cdot \prod_{j=1, j\neq i}^{k} \frac{\xi - j}{i - j}$
with x_u, y_u or z for v
and with ξ a verifier challenge.

Fig. 2: Limbo's compression step - from dimension $k\ell$ to dimension ℓ

where p_k is the false positive rate of the underlying MPC protocol (which depends on k) – see [DOT21, Proposition 5].

Let us assume that we want to batch u proofs for an arbitrary arithmetic circuit C defined over a base field $\mathbb{F}$. We use the Limbo proof system in interactive mode targeting a security of 40 bits, which is one of the considered use-cases of the original paper [DOT21]. If $\mathbb{F}$ is large enough, we can apply our batching strategy described in Section 5.3. Tables 6 and 7 summarize the achieved performances for the base fields $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{32}}$ and for circuit sizes ($|w|+|C|$ where $|C|$ is the number of multiplications in C) 2^8 and 2^{16}. While for $\mathbb{F}_{2^8}$ we are limited in the number of parties since $N \leq |\mathbb{F}|+1-u$, this is not an issue for $\mathbb{F}_{2^{32}}$. From these tables, we can observe that our batching strategy drastically reduces the amortized cost in terms of proof size and number of emulated parties (particularly for the verifier). Considering a batching of $u = 100$ statements, the amortized proof size is more than 10 times smaller than the standard version in all the considered settings and this ratio is closer to 1/20 for the larger circuit or field.

Acknowledgments. The authors would like to thank Antoine Joux for fruitful discussions related to this work.

u	ℓ	$\lvert w\rvert + \lvert C\rvert = 2^8$							$\lvert w\rvert + \lvert C\rvert = 2^{16}$						
		N	u/ν	τ	size	size/u	$\#_{\mathcal{P}}$	$\#_{\mathcal{V}}$	N	u/ν	τ	size	size/u	$\#_{\mathcal{P}}$	$\#_{\mathcal{V}}$
1	ADD	256	-	6	5.5	5.55	1536	1530	256	-	6	389.4	389.42	1536	1530
	1	256	1	6	9	9.22	12	6	256	1	6	398	397.59	12	6
100	1	223	34	16	61	0.61	1680	48	207	50	21	2762	27.62	2142	42
	4	223	34	4	58	0.58	456	48	223	34	4	3151	31.51	456	48
10000	1	225	32	15	5865	0.59	154935	4695	203	54	22	269290	26.93	225060	4092
	4	210	47	5	5434	0.54	54315	4260	200	57	6	277894	27.79	64416	4224

Table 6: Performances of batched Limbo proofs for arithmetic circuits on $\mathbb{F}_{2^8}$ in the interactive setting (40-bit of security). The compression factor k of Limbo is 16 and the extension field $\mathbb{G}$ is $\mathbb{F}_{2^{64}}$ when $\ell = 1$ and $\mathbb{F}_{2^{96}}$ when $\ell = 4$. All the sizes are given in kilobytes. $\#_{\mathcal{P}}$ and $\#_{\mathcal{V}}$ correspond respectively to the number of emulated parties for the prover and the verifier. The first row ($\ell =$ ADD) is the baseline Limbo scheme with additive sharing.

u	ℓ	$\lvert w\rvert + \lvert C\rvert = 2^8$							$\lvert w\rvert + \lvert C\rvert = 2^{16}$						
		N	u/ν	τ	size	size/u	$\#_{\mathcal{P}}$	$\#_{\mathcal{V}}$	N	u/ν	τ	size	size/u	$\#_{\mathcal{P}}$	$\#_{\mathcal{V}}$
1	ADD	256	-	6	10.0	10.03	1536	1530	256	-	6	1536.9	1536.92	1536	1530
	1	256	1	6	14	13.72	12	6	256	1	6	1550	1549.59	12	6
100	1	256	50	18	96	0.96	1836	36	256	100	31	8053	80.53	3131	31
	4	256	100	8	91	0.91	832	32	256	100	8	8284	82.84	832	32
10000	1	256	55	19	8170	0.82	193648	3458	256	100	31	801486	80.15	313100	3100
	4	256	74	6	7130	0.71	63648	3264	256	88	7	823455	82.35	73416	3192

Table 7: Performances of batched Limbo proofs for arithmetic circuits on $\mathbb{F}_{2^{32}}$ in the interactive setting (40-bit of security). The compression factor k of Limbo is 16 and the extension field $\mathbb{G}$ is $\mathbb{F}_{2^{64}}$ when $\ell = 1$ and $\mathbb{F}_{2^{96}}$ when $\ell = 4$. All the sizes are given in kilobytes. $\#_{\mathcal{P}}$ and $\#_{\mathcal{V}}$ correspond respectively to the number of emulated parties for the prover and the verifier. The first row ($\ell =$ ADD) is the baseline Limbo scheme with additive sharing.

References

ABB+22. Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. SPHINCS+ – Submission to the 3rd round of the NIST post-quantum project. v3.1, 2022.

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

BBHR19. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, Heidelberg, August 2019.

BCR+19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

BD20. Ward Beullens and Cyprien Delpech de Saint Guilhem. LegRoast: Efficient post-quantum signatures from the Legendre PRF. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 130–150. Springer, Heidelberg, 2020.

BDK+21. Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 266–297. Springer, Heidelberg, May 2021.

Beu20. Ward Beullens. Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 183–211. Springer, Heidelberg, May 2020.

BFH$^+$20. Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In *ACM CCS 2020*, pages 2025–2038. ACM Press, November 2020.

BHH01. Dan Boneh, Shai Halevi, and Nick Howgrave-Graham. The modular inversion hidden number problem. In *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 36–51. Springer, Heidelberg, December 2001.

BN20. Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 495–526. Springer, Heidelberg, May 2020.

Bus52. K. A. Bush. Orthogonal Arrays of Index Unity. *The Annals of Mathematical Statistics*, 23(3):426 – 434, 1952.

CC06. Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In *CRYPTO 2006*, volume 4117 of *LNCS*, pages 521–536. Springer, Heidelberg, August 2006.

CDG$^+$17. Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS 2017*, pages 1825–1842. ACM Press, October / November 2017.

CDN15. Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

DDOS19. Cyprien Delpech de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. BBQ: Using AES in picnic signatures. In *SAC 2019*, volume 11959 of *LNCS*, pages 669–692. Springer, Heidelberg, August 2019.

DKR$^+$21. Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. Shorter signatures based on tailor-made minimalist symmetric-key crypto. Cryptology ePrint Archive, Report 2021/692, 2021.

DOT21. Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge MPCitH-based arguments. In *ACM CCS 2021*, pages 3022–3036. ACM Press, November 2021.

EKR18. David Evans, Vladimir Kolesnikov, and Mike Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*. NOW Publishers, 2018.

FJR22. Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 541–572. Springer, Heidelberg, August 2022.

FMRV22. Thibauld Feneuil, Jules Maire, Matthieu Rivain, and Damien Vergnaud. Zero-knowledge protocols for the subset sum problem from MPC-in-the-head with rejection. In *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 371–402. Springer, Heidelberg, December 2022.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

GMO16. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016.

GSV21. Yaron Gvili, Sarah Scheffler, and Mayank Varia. Booligero: Improved sublinear zero knowledge proofs for boolean circuits. *FC*, pages 476 – 496, 2021.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.

KZ20a. Daniel Kales and Greg Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In *CANS 20*, volume 12579 of *LNCS*, pages 3–22. Springer, Heidelberg, December 2020.

KZ20b. Daniel Kales and Greg Zaverucha. Improving the performance of the Picnic signature scheme. *IACR TCHES*, 2020(4):154–188, 2020.

KZ21. Daniel Kales and Greg Zaverucha. Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures. Preliminary Draft, October 29, 2021, 2021.

KZ22. Daniel Kales and Greg Zaverucha. Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures. Cryptology ePrint Archive, Report 2022/588, 2022.

MS10. F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes. 9th Edition.* Discrete Mathematics and its Applications. Elsevier Science, 1978, 2010.

PS00. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

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

Was08. Lawrence C. Washington. *Elliptic Curves. Number Theory and Cryptography. 2nd Edition.* Discrete Mathematics and its Applications. Chapman & Hall/CRC Press, 2008.

– Supplementary Material –

A Technical Lemmas

In our proofs, we shall make use of the following lemmas:

Lemma 3 (Splitting Lemma [PS00]). *Let X and Y be two finite sets, and let $A \subseteq X \times Y$ such that*

$$\Pr\left[(x,y) \in A \mid (x,y) \leftarrow X \times Y\right] \geq \varepsilon \ .$$

For any $\alpha \in [0,1)$, let

$$B = \left\{(x,y) \in X \times Y \;\middle|\; \Pr\left[(x,y') \in A \mid y' \leftarrow Y\right] \geq (1-\alpha) \cdot \varepsilon\right\} \ .$$

We have:

1. $\Pr\left[(x,y) \in B \mid (x,y) \leftarrow X \times Y\right] \geq \alpha \cdot \varepsilon$
2. $\Pr\left[(x,y) \in B \mid (x,y) \leftarrow A\right] \geq \alpha$.

Proof. See [PS00] for the proof.

Lemma 4. *Let $E_1, \ldots, E_M$ be arbitrary events. For $i \in \{1, \ldots, M\}$ and a bit $b \in \{0,1\}$, let us define*

$$A_i^b := \begin{cases} E_i & \text{if } b = 1, \\ \bar{E}_i & \text{if } b = 0. \end{cases}$$

We have those following relations, where $\mathrm{wt}_H(x)$ denotes the Hamming weight of x (i.e. the number of non-zero coordinates of x):

$$\sum_{x \in \{0,1\}^M} \Pr[A_1^{x_1}, A_2^{x_2}, \ldots, A_M^{x_M}] = 1$$

$$\sum_{x \in \{0,1\}^M} \mathrm{wt}_H(x) \cdot \Pr[A_1^{x_1}, A_2^{x_2}, \ldots, A_M^{x_M}] = \Pr[E_1] + \ldots + \Pr[E_M]$$

Proof. The first equality comes from the fact the underlying events form a partition. The second equality can be proved by induction on the integer M. The case $M = 1$ is trivial:

$$\sum_{x \in \{0,1\}} \mathrm{wt}_H(x) \cdot \Pr[A_1^{x_1}] = 0 \cdot \Pr[\bar{E}_1] + 1 \cdot \Pr[E_1] = \Pr[E_1]$$

Let assume the relation for a positive integer M. By noting for some $x \in \{0,1\}^M$ $A^{\mathbf{x}} := \{A_1^{x_1}, \ldots, A_M^{x_M}\}$, we have

$$\begin{aligned}
\sum_{x \in \{0,1\}^{M+1}} & \mathrm{wt}_H(x) \cdot \Pr[A_1^{x_1}, A_2^{x_2}, \ldots, A_{M+1}^{x_{M+1}}] \\
&= \sum_{x \in \{0,1\}^M} \mathrm{wt}_H(x) \cdot \Pr[A^{\mathbf{x}}, \bar{E}_{M+1}] \\
&\qquad\qquad + \sum_{x \in \{0,1\}^M} (\mathrm{wt}_H(x) + 1) \cdot \Pr[A^{\mathbf{x}}, E_{M+1}] \\
&= \sum_{x \in \{0,1\}^M} \mathrm{wt}_H(x) \cdot (\Pr[A^{\mathbf{x}}, \bar{E}_{M+1}] + \Pr[A^{\mathbf{x}}, E_{M+1}]) \\
&\qquad\qquad + \sum_{x \in \{0,1\}^M} \Pr[A^{\mathbf{x}}, E_{M+1}] \\
&= \sum_{x \in \{0,1\}^M} \mathrm{wt}_H(x) \cdot \Pr[A^{\mathbf{x}}] + \Pr[E_{M+1}] \\
&= (\Pr[E_1] + \ldots + \Pr[E_M]) + \Pr[E_{M+1}].
\end{aligned}$$

□

Lemma 5. *Let us have $c \geq 0$, $e \geq 1$, $d \geq 0$ such that $c \cdot (e-1) \geq d$. Then, for all $a \geq 0$,*

$$\frac{c+d}{e} a \leq ca + d.$$

Proof.

$$\begin{aligned}
ca + d - \left(\frac{c+d}{e} a\right) &= a\left(c - \frac{c+d}{e}\right) + d \\
&= \underbrace{\frac{a}{e}}_{\geq 0} \underbrace{(c(e-1) - d)}_{\geq 0} \geq 0
\end{aligned}$$

B Lemmas for Threshold LSSS

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 $\mathbb{F}$ 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 $N \leq |\mathbb{F}|+1$.*

Proof. For $s \in \mathbb{F}$ and $[\![s]\!]$ an $(N, \ell+1)$-threshold sharing of s, the concatenation of s and $[\![s]\!]$ forms an $[n, k, d]$ linear code of length $n = N+1$, dimension $k = \ell+1$ and distance $d = N - \ell + 1$, which is *maximum distance separable* (a.k.a. MDS code). We consider two cases:

1. if $k \geq |\mathbb{F}|+1$ (meaning $\ell \geq |\mathbb{F}|$), then by [Bus52], we either have $n = k$ (meaning $N = \ell$ which is impossible) or $n = k+1$ (meaning $N = \ell + 1$ which is not considered in the lemma statement);
2. if $k \leq |\mathbb{F}|$ (meaning $\ell \leq |\mathbb{F}| - 1$), then the MDS conjecture [MS10] gives an upper bound to the code length, which is $n \leq |\mathbb{F}|+1$ giving $N \leq |\mathbb{F}|$, except for $|\mathbb{F}|$ a power of 2 and $k = 3$ or $k = |\mathbb{F}|-1$, in which case $n \leq |\mathbb{F}|+2$, giving $N \leq |\mathbb{F}|+1$ whenever $\ell = 2$ or $\ell = |\mathbb{F}|-2$.

□

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 $[\![s]\!]_{J_0}$.

This lemma can be obtained from the equivalence between threshold linear secret sharing schemes and interpolation codes [CDN15, Theorem 11.103]. For the sake of completeness, we propose below an alternative simple proof.

Proof. Let $s \in \mathbb{F}$ and $J_0 \subseteq [N]$ with $|J_0| = \ell+1$. Consider two sharings $[\![s_1]\!], [\![s_2]\!] \in \mathbb{F}^N$ such that

$$[\![s_1]\!]_{J_0} = [\![s_2]\!]_{J_0}$$

and such that

$$\forall J \text{ s.t. } |J| = \ell+1, \begin{cases} \mathsf{Reconstruct}_J([\![s_1]\!]_J) = s \\ \mathsf{Reconstruct}_J([\![s_2]\!]_J) = s \end{cases}$$

Let us define $\Delta = [\![s_2]\!] - [\![s_1]\!]$. We have $\Delta_{J_0} = 0$ and by linearity

$$\forall J \text{ s.t. } |J| = \ell+1, \mathsf{Reconstruct}_J(\Delta_J) = 0.$$

Let us assume that there exists i^* such that $\Delta_{i^*} \neq 0$. We will show that this leads to a contradiction.

Let us consider $I \subset J_0$ such that $|I| = \ell$. Let us further consider $v_1, v_2 \in \mathbb{F}$ such that $v_1 \neq v_2$. By the ℓ-privacy of the sharing, there exist random tapes $r_1, r_2 \in R$ such that $[\![v_1]\!] := \mathsf{Share}(v_1; r_1)$ satisfies $[\![v_1]\!]_I = [\![s_1]\!]_I$ and $[\![v_2]\!] := \mathsf{Share}(v_2; r_2)$ satisfies $[\![v_2]\!]_I = [\![s_2]\!]_I$. Denoting $\Delta' = [\![v_1]\!] - [\![v_2]\!]$, we obtain $\Delta'_I = \Delta_I = 0$. Then, by linearity of the sharing, we have

$$\begin{aligned} v_1 - v_2 &= \mathsf{Reconstruct}_{I\cup\{i^*\}}\left(\Delta'_{I\cup\{i^*\}}\right) \\ &= \mathsf{Reconstruct}_{I\cup\{i^*\}}\left((\Delta'_{i^*} \cdot \Delta_{i^*}^{-1}) \cdot \Delta_{I\cup\{i^*\}}\right) \\ &= (\Delta'_{i^*} \cdot \Delta_{i^*}^{-1}) \cdot \mathsf{Reconstruct}\left(\Delta_{I\cup\{i^*\}}\right) \\ &= 0 \end{aligned}$$

Since, by definition $v_1 - v_2 \neq 0$, we get a contradiction. We deduce that $\Delta = 0$, which means that $[\![s_1]\!] = [\![s_2]\!]$.

Now that we have shown the unicity, let us (constructively) show the existence of the algorithm Expand_{J_0}. The function $\mathsf{Reconstruct}_J$ is a linear application from $\mathbb{F}^{|J|}$ to $\mathbb{F}$, thus there exists a vector $v_J \in (\mathbb{F}^*)^{|J|}$ such that

$$\mathsf{Reconstruct}_J(\cdot) = \langle v_J, \cdot \rangle.$$

(All the coefficients of v_J are non-zero since otherwise the privacy property would be broken.) On input $[\![s]\!]_{J_0}$, the algorithm Expand_{J_0} then builds each share $[\![s]\!]_{i^*}$ as follows:

- it chooses a $J \subset J_0 \cup \{i^*\}$ such that $|J| = \ell+1$;
- it solves the linear equation $\langle v_J, [\![s]\!]_{J_0\cup\{i^*\}} \rangle = s$ where $[\![s]\!]_{i^*}$ is the only unknown.

□

C Proof of Privacy

Proof. In the MPC protocol Π_{LSSS} using an $(\ell+1, N)$-threshold LSSS, all the values computed by the parties are shares from the underlying LSSS. The parties take such sharings as input and the latters are stable by the operations performed in Π_{LSSS} (since the computation over the shares is linear). Therefore, the MPC protocol Π_{LSSS} is well-defined.

By assumption, we have that the MPC protocol Π_{add} is $(N-1)$-private when using an additive sharing. This implies that there exists a simulator $\mathsf{Sim}_{\text{add}}$ which takes as inputs a set $I' \subset [N]$ of size $N-1$, $[\![w']\!]_{I'}$, $[\![\vec{\beta}']\!]_{I'}$ and the outcome (Accept or Reject) of the real-world execution and which outputs views for parties in I' whose joint distribution is perfectly indistinguishable from the joint views of the same parties in a real-world execution of the MPC protocol (with the same outcome).

We first describe the simulator $\mathsf{Sim}_{\text{LSSS}}(I, [\![w]\!]_I, [\![\vec{\beta}]\!]_I, \vec{\varepsilon}, y)$:

1. Sample w' and $\vec{\beta}'$ randomly.
2. Compute $[\![w']\!] \leftarrow \mathsf{Share}_{\text{add}}(w'; r_{w'})$ for some fresh randomness $r_{w'}$.
3. Compute $[\![\vec{\beta}']\!] \leftarrow \mathsf{Share}_{\text{add}}(\vec{\beta}'; r_{\vec{\beta}'})$ for some fresh randomness $r_{\vec{\beta}'}$.
4. Choose a set $I' \subset [N]$ such that $|I'| = N-1$.
5. Call the simulator $\mathsf{view}_{\text{add}} \leftarrow \mathsf{Sim}_{\text{add}}(I', [\![w']\!]_{I'}, [\![\vec{\beta}']\!]_{I'}, \vec{\varepsilon}, y)$.
6. Extract $\vec{\alpha}'$ from $\mathsf{view}_{\text{add}}$ and let $\vec{\alpha} := \vec{\alpha}'$.
7. For $j = 1$ to t:
   - Compute $[\![\alpha^j]\!]_I = \varphi^j_{(\varepsilon^i)_{i\le j}, (\alpha^i)_{i<j}}\big([\![w]\!]_I, ([\![\beta^i]\!]_I)_{i\le j}\big)$,
   - Deduce $[\![\alpha^j]\!]$ from $[\![\alpha^j]\!]_I$ and α^j.
8. Output $[\![\vec{\alpha}]\!]$, $[\![w]\!]_I$, $[\![\vec{\beta}]\!]_I$.

In what follows, we show that the above simulator outputs a distribution which is perfectly indistinguishable from the views of the same parties in a real execution setting, which proves the ℓ-privacy of the protocol Π_{LSSS}.

Let us fix a set $I \subset [N]$ such that $|I| = \ell$. Let us also take w and $\vec{\beta}$ and share them as

$$[\![w]\!] = \mathsf{Share}_{\text{LSSS}}(w; r_1)$$
$$[\![\vec{\beta}]\!] = \mathsf{Share}_{\text{LSSS}}(\vec{\beta}; r_2)$$

using some random tapes r_1, r_2. Finally, let us sample a random $\vec{\varepsilon}$.

We propose below three experiments. We denote Exp_1, Exp_2 and Exp_3 the distribution of their respective outputs. The first experiment corresponds to the simulation of the joint view of the parties in I.

Experiment 1. We compute the output $y \in \{\text{Accept}, \text{Reject}\}$ of the protocol using $[\![w]\!]$, $[\![\vec{\beta}]\!]$ and $\vec{\varepsilon}$:

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

Then, we output the views returned by $\mathsf{Sim}_{\text{LSSS}}(I, [\![w]\!]_I, [\![\vec{\beta}]\!]_I, \vec{\varepsilon}, y)$ together with y.

Experiment 2. The difference with the previous experiment is that w' and $\vec{\beta}'$ are respectively defined as w and $\vec{\beta}$ (instead of being random). Thanks to the privacy of the additive sharing we get that $[\![w']\!]_I$ and $[\![\vec{\beta}']\!]_I$ are perfectly indistinguishable from $[\![w]\!]_I$ and $[\![\vec{\beta}]\!]_I$ which implies

$$\text{Exp}_1 \equiv \text{Exp}_2 \ .$$

(Here $\text{Exp}_1 \equiv \text{Exp}_2$ means that the two distributions are perfectly indistinguishable).

Experiment 3. The difference with the previous experiment is that $\vec{\alpha}$ is defined as $\vec{\alpha} = \Phi(w, \vec{\varepsilon}, \vec{\beta})$ (where Φ is defined as in (1)) instead of $\vec{\alpha} = \vec{\alpha}'$. Thanks to the $(N-1)$-privacy of the protocol Π_{add}, we get that the joint views $\mathsf{view}_{\mathsf{add}}$ returned by the simulator are perfectly indistinguishable from the same views in real-world protocol execution. In particular, $[\![\vec{\alpha}']\!]$ is identically distributed in the two experiments, which implies

$$\mathrm{Exp}_2 \equiv \mathrm{Exp}_3 \ .$$

Finally, it is not hard to check that the output of Experiment 3 corresponds to the real-world joint views of the parties in I, which concludes the proof.

□

D Proof of Soundness

This appendix provides a proof for the soundness property in the following theorem:

Theorem 2. *Let us consider an MPC protocol Π_{LSSS} complying with the protocol format described in Protocol 5 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 6 built from Π_{LSSS} satisfies the three following properties:*

- ***Completeness.*** *A prover $\mathcal{P}$ who knows a witness w such that $(x, w) \in \mathcal{R}$ and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*
- ***Soundness.*** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input x, convinces the honest verifier $\mathcal{V}$ to accept with probability*
$$\tilde{\epsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle(x) \to 1] \ > \ \epsilon$$
where the soundness error ϵ is defined as
$$\epsilon := \frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell \cdot (N-\ell)}{\ell+1} \ .$$
Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, outputs either a witness w satisfying $(x, w) \in \mathcal{R}$, or a commitment collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by
$$\frac{4}{\tilde{\epsilon} - \epsilon} \cdot \left(1 + \tilde{\epsilon} \cdot \frac{8 \cdot (N-\ell)}{\tilde{\epsilon} - \epsilon} \right) .$$
- ***Honest-Verifier Zero-Knowledge.*** *There exists an efficient simulator $\mathcal{S}$ which, given the random challenge I outputs a transcript which is indistinguishable from a real transcript of Protocol 6.*

For any set $\mathcal{T}$ of successful transcripts corresponding to the same commitment,

- either the revealed shares of $[\![w]\!]$ are not unique, and then we find a commitment collision or a hash collision,
- or the openings are unique.

Along this proof, we consider that the extractor only gets transcripts with unique revealed shares since otherwise the extractor would find a commitment collision or a hash collision.

We shall denote by R_h the randomness of $\tilde{\mathcal{P}}$ which is used to generate the initial commitment h_1 (which determines the witness sharing $[\![w]\!]$), and we denote r_h a possible realization of R_h. Throughout the proof, we denote $\mathsf{succ}_{\tilde{\mathcal{P}}}$ the event that $\tilde{\mathcal{P}}$ succeeds in convincing $\mathcal{V}$. By hypothesis, we have $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}] = \tilde{\epsilon}$.

D.1 When restraining to only bad witnesses

Let r_h be a possible realization of R_h. Given $R_h = r_h$, we have a unique hash commitment h_1 in the transcript. This hash commitment uniquely defines the shares of the witness $[\![w]\!]_1, \ldots, [\![w]\!]_N$ (by assumption on the absence of hash/commitment collisions). In the following, we shall denote $w^{(J)}$ the witness corresponding to the shares $\{[\![w]\!]_i\}_{i\in J}$, for $|J| = \ell + 1$. We have a total of $\binom{N}{\ell+1}$ possibly-distinct witnesses $w^{(J)}$. We shall say that $w^{(J)}$ is a *good witness* whenever $(x, w^{(J)}) \in \mathcal{R}$, otherwise we call $w^{(J)}$ a *bad witness*.

For any transcript produced by $\tilde{\mathcal{P}}$ (with h_1 as first hash commitment), the hash commitments $h_2, \ldots, h_t$ uniquely define the shares $\{[\![\alpha^j]\!]_i\}_{i\in S}$, for $j \in [t]$. In the following, we shall denote by $[\![\alpha^j]\!] = ([\![\alpha^j]\!]_1, \ldots, [\![\alpha^j]\!]_N)$ the full $(\ell + 1, N)$-sharing consistent with the shares $\{[\![\alpha^j]\!]_i\}_{i\in S}$. The hash commitments $h_2, \ldots, h_t$ also uniquely define the shares $\{[\![\beta^j]\!]_i\}_{i\in[N]}$, for $j \in [t]$. We shall denote $\mathcal{H}$ the set of *honest parties*, *i.e.* the set of the parties for which the committed shares $[\![\alpha^1]\!]_i, \ldots, [\![\alpha^t]\!]_i$, are consistent with the committed input shares $[\![w]\!]_i$ and $[\![\beta^1]\!]_i, \ldots, [\![\beta^t]\!]_i$. More formally,

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

We further denote Y the random variable which corresponds to the number of honest parties, *i.e.* $Y = |\mathcal{H}|$. We stress that $[\![\alpha^1]\!], \ldots, [\![\alpha^t]\!]$, $\mathcal{H}$ and Y depend on the randomness of the (malicious) prover and the randomness of the verifier used before Step 3 of Protocol 6.

For every $i \in [N]$ and $j \in [t]$, we shall further denote $[\![\bar{\alpha}^j]\!]_i$ the share obtained through an honest computation from the committed input shares, that is:

$$[\![\bar{\alpha}^j]\!]_i = \varphi^j_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j}}([\![w]\!]_i, ([\![\beta^k]\!]_i)_{k\le j}) .$$

We stress that $[\![\bar{\alpha}^j]\!]_i$ might not be equal to $[\![\alpha^j]\!]_i$. We actually have $i \in \mathcal{H}$ if and only if $[\![\bar{\alpha}^j]\!]_i = [\![\alpha^j]\!]_i$ for every $j \in [t]$. In the following, we shall say that witness $w^{(J)}$ gives rise to a false positive in the MPC protocol Π, and denote this probability event E_J, whenever

$$g(\bar{\alpha}^1_J, \ldots, \bar{\alpha}^t_J) = 0$$

where $\bar{\alpha}^1_J, \ldots, \bar{\alpha}^t_J$ are the plain values corresponding to the sharings $\{[\![\bar{\alpha}^1]\!]_i\}_{i\in J}, \ldots, \{[\![\bar{\alpha}^t]\!]_i\}_{i\in J}$. By definition of the MPC protocol Π, we have:

$$\forall J \text{ s.t. } |J| = \ell + 1, \ \Pr'[E_J] \le p.$$

For the first step of the proof, we shall consider a subset D of parties, *i.e.* $D \subset \{1, \ldots, N\}$, and we denote

$$N' := |D| \qquad \text{and} \qquad \mathcal{J} = \{J \subset D : |J| = \ell + 1\}.$$

We shall further denote

$$\Pr'[\,\cdot\,] := \Pr[\ \cdot \mid R_h = r_h, I \subset D].$$

We will show that, if $\{w^{(J)}\}_{J\in\mathcal{J}}$ are all *bad* witnesses, then the probability $\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}]$ is upper bounded by

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}] \le \frac{\binom{N}{\ell}}{\binom{N'}{\ell}} \cdot \epsilon \tag{2}$$

where ϵ is the soundness error defined in the theorem statement, which is

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

The rest of this subsection is devoted to the demonstration of (2).

For $J \in \mathcal{J}$ and $b \in \{0,1\}$, let us introduce the notation

$$A_J^b = \begin{cases} E_J & \text{if } b = 1, \\ \bar{E}_J & \text{if } b = 0. \end{cases}$$

Let $x = (x_J)_{J \in \mathcal{J}} \in \{0,1\}^{|\mathcal{J}|}$ and let $y \in \{0, \ldots, N\}$. Let us assume that $\mathsf{succ}_{\tilde{\mathcal{P}}}$, $Y = y$ and $\{A_J^{x_J}\}_{J \in \mathcal{J}}$ jointly occur. Because $\mathsf{succ}_{\tilde{\mathcal{P}}}$ occurs, we have that

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

where $\alpha^1, \ldots, \alpha^t$ are the values corresponding to the sharings $[\![\alpha^1]\!], \ldots, [\![\alpha^t]\!]$. Then for each set $J \in \mathcal{J}$ ($w^{(J)}$ is a bad witness) such that $J \subset \mathcal{H}$ (the parties in J are honest), we have that $[\![\bar{\alpha}^j]\!]_i = [\![\alpha^j]\!]_i$ for every $i \in J$ and every $j \in [t]$, which implies

$$g(\bar{\alpha}^1_J, \ldots, \bar{\alpha}^t_J) = g(\alpha^1, \ldots, \alpha^t) = 0 \ .$$

Namely, a false positive necessarily occurs for $w^{(J)}$, *i.e.* $x_J = 1$, whenever $J \in \mathcal{J}$ with $J \subset \mathcal{H}$. Thus

$$\mathrm{wt}_H(x) \geq \sum_{J \in \mathcal{J} : J \subset \mathcal{H}} x_J = \binom{y}{\ell + 1} \ .$$

By defining

$$y_{\max} := \max\{y : \mathrm{wt}_H(x) \geq \binom{y}{\ell+1}\} \ ,$$

we get that

$$\Pr{}'[\mathsf{succ}_{\tilde{\mathcal{P}}}, Y = y \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] = 0 \qquad \text{if } y > y_{\max}$$

and so

$$\Pr{}'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] = \sum_{y=0}^{y_{\max}} \Pr{}'[\mathsf{succ}_{\tilde{\mathcal{P}}}, Y = y \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \ .$$

The only way for the transcript to be successful is that the set I of challenged opened parties only contains honest parties, *i.e.* $I \subset \mathcal{H}$. Thus,

$$\Pr{}'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}, Y = y] \leq \Pr[I \subset \mathcal{H} \mid I \subset D, Y = y] = \frac{\binom{y}{\ell}}{\binom{N'}{\ell}} \ .$$

We deduce

$$\Pr{}'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \ \leq \ \sum_{y=0}^{y_{\max}} \frac{\binom{y}{\ell}}{\binom{N'}{\ell}} \cdot \Pr{}'[Y = y \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \ \leq \ \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} \ .$$

Since $\mathrm{wt}(x)$ is a non-negative integer, we have $y_{\max} \geq \ell$. Let us consider three cases:

Case 1: $y_{\max} = \ell$, it means that $\mathrm{wt}(x) = 0$, then

$$\Pr{}'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \leq \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} = \frac{1}{\binom{N'}{\ell}} = \frac{\mathrm{wt}_H(x) \cdot \ell + 1}{\binom{N'}{\ell}}$$

Case 2: $y_{\max} = \ell + 1$, it means that $\mathrm{wt}(x) \geq 1$, then

$$\Pr{}'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \leq \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} = \frac{\ell + 1}{\binom{N'}{\ell}} \leq \frac{\mathrm{wt}_H(x) \cdot \ell + 1}{\binom{N'}{\ell}}$$

Case 3: $y_{\max} \geq \ell + 2$, then

$$\begin{aligned}\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] &\leq \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} = \frac{\frac{\ell+1}{y_{\max}-\ell}\binom{y_{\max}}{\ell+1}}{\binom{N'}{\ell}} \\ &\leq \frac{\ell+1}{2}\cdot\frac{\mathrm{wt}_H(x)}{\binom{N'}{\ell}} \leq \frac{\mathrm{wt}_H(x)\cdot\ell+1}{\binom{N'}{\ell}}\end{aligned}$$

In any case, we have the relation

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] \leq \frac{\mathrm{wt}_H(x)\cdot\ell+1}{\binom{N'}{\ell}}. \tag{3}$$

for any $x \in \{0,1\}^{|\mathcal{J}|}$. And so, we have

$$\begin{aligned}\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}] &= \sum_{x\in\{0,1\}^{\mathcal{J}}} \Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}]\cdot\Pr'[\{A_J^{x_J}\}_{J\in\mathcal{J}}] \\ &\leq \sum_{x\in\{0,1\}^{\mathcal{J}}} \frac{\mathrm{wt}_H(x)\cdot\ell+1}{\binom{N'}{\ell}}\cdot\Pr'[\{A_J^{x_J}\}_{J\in\mathcal{J}}] && \text{using (3)} \\ &= \frac{1}{\binom{N'}{\ell}} + \frac{\ell}{\binom{N'}{\ell}}\cdot\sum_{x\in\{0,1\}^{\mathcal{J}}}\mathrm{wt}_H(x)\cdot\Pr'[\{A_J^{x_J}\}_{J\in\mathcal{J}}] && \text{using Lemma 4} \\ &\leq \frac{1}{\binom{N'}{\ell}} + \frac{\ell}{\binom{N'}{\ell}}\cdot\sum_{J\in J}\Pr'[E_J] && \text{using Lemma 4} \\ &\leq \frac{1}{\binom{N'}{\ell}} + \frac{\ell}{\binom{N'}{\ell}}\cdot|\mathcal{J}|\cdot p \\ &= \frac{\binom{N}{\ell}}{\binom{N'}{\ell}}\cdot\left(\frac{1}{\binom{N}{\ell}} + \frac{\ell}{\binom{N}{\ell}}\cdot\underbrace{\binom{N'}{\ell+1}}_{\leq\binom{N}{\ell+1}}p\right) \leq \frac{\binom{N}{\ell}}{\binom{N'}{\ell}}\cdot\epsilon\ .\end{aligned}$$

Thus we obtain the desired inequality (2).

D.2 Building of the extractor

In the previous subsection, we proved that the probability that a malicious prover $\tilde{\mathcal{P}}$ produces a valid transcript when the opened parties are restricted to a set of N' parties for which the shares only encode bad witnesses is upper-bound by

$$\frac{\binom{N}{\ell}}{\binom{N'}{\ell}}\cdot\epsilon\ .$$

We now show how to build an extractor which outputs a witness w satisfying $(x, w) \in \mathcal{R}$ (if not a hash or commitment collision) when giving rewindable black-box access to a malicious prover $\tilde{\mathcal{P}}$ which produces successful transcripts with a probability $\tilde{\epsilon} > \epsilon$.

Let us fix an arbitrary value $\delta \in (0, 1)$ such that $(1-\delta)\tilde{\epsilon} > \epsilon$ (such δ exists since $\tilde{\epsilon} > \epsilon$). Let r_h be a possible realization of R_h. We will say that r_h is *good* if it is such that

$$\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \geq (1-\delta)\cdot\tilde{\epsilon}\ . \tag{4}$$

By the Splitting Lemma 3 (see Appendix A) we have

$$\Pr[R_h \text{ good} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}] \geq \delta\ . \tag{5}$$

Our extractor first runs the $\tilde{\mathcal{P}}$ with honest verifier requests until obtaining a successful transcript T_0 by running. If this T_0 corresponds to a good r_h, then we can obtain further successful transcripts with "high" probability (*i.e.* probability greater than $(1-\delta)\cdot\tilde{\epsilon}$) by rewinding the protocol just after the initial commitment h_1. Based on the assumption that r_h is good, a sub-extractor $\mathcal{E}_0$ will build a list of *successful* transcripts $\mathcal{T}$, all with same initial commitment. We denote $P(\mathcal{T})$ the set of the parties which have been open in at least one transcript of $\mathcal{T}$, *i.e.* $P(\mathcal{T}) := \bigcup_{T\in\mathcal{T}} I_T$ where I_T is the set of opened parties of the transcript T.

For a certain number N_1 of iterations, the sub-extractor $\mathcal{E}_0$ tries to feed the list $\mathcal{T}$ until there exist a good witness among the open input shares. We formally describe the sub-extractor routine in the following pseudocode:

```
Sub-extractor $\mathcal{E}_0$ (on input a successful transcript $T_0$):

1. $\mathcal{T} = \{T_0\}$
2. Do $N_1$ times:
3.   Run $\tilde{\mathcal{P}}$ with honest $\mathcal{V}$ and same $r_h$ as $T_0$ to get transcript $T$
4.   If $T$ is a successful transcript,
5.       $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$.
6.       If $\mathcal{T}$ contains a good witness $w$, return $w$.
7. Return $\emptyset$.
```

Let us evaluate the probability that the stop condition is reached in a given number of iteration N_1. Consider a loop iteration in $\mathcal{E}_0$ at the beginning of which we have a list $\mathcal{T}$ of successful transcripts (which does not contain a good witness since the stop condition has not been reached) and a transcript T sampled at Step 3. We denote Z the event that a new party is open (a party which is not in $P(\mathcal{T})$) in the transcript T. This event is defined with respect to the randomness of the verifier challenges in T.

Let us lower bound the probability to have a successful transcript T and the event Z occurring in the presence of a good R_h:

$$p_g := \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \cap Z \mid R_h \text{ good}] \ .$$

We have:

$$\begin{aligned} p_g &= \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}] - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \cap \bar{Z} \mid R_h \text{ good}] \\ &= \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}] - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}, \bar{Z}] \cdot \Pr[\bar{Z} \mid R_h \text{ good}] \\ &\geq (1-\delta)\cdot\tilde{\epsilon} - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}, \bar{Z}] \cdot \Pr[\bar{Z} \mid R_h \text{ good}] \end{aligned}$$

where the last inequality holds by (4).

The probability that a new party is not opened corresponds to the probability that the set I of opened parties is a subset of $P(\mathcal{T})$, *i.e.*

$$\Pr[\bar{Z} \mid R_h \text{ good}] = \Pr[\bar{Z}] = \frac{\binom{|P(\mathcal{T})|}{\ell}}{\binom{N}{\ell}} \ .$$

The success probability knowing that that no new party is open corresponds to the success probability when restricting to the $|P(\mathcal{T})|$ parties which have been already open. By assumption ($\mathcal{T}$ does not contain a good witness), the shares of those parties only correspond to bad witnesses. Thus, this probability can be upper bounded using the inequality (2) of Section D.1 with $N' = |P(\mathcal{T})|$:

$$\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}, \bar{Z}] \leq \frac{\binom{N}{\ell}}{\binom{|P(\mathcal{T})|}{\ell}} \cdot \epsilon \ .$$

Thus, we get

$$p_g \geq (1-\delta)\cdot\tilde{\epsilon} - \epsilon \ .$$

To summarize, in the presence of a good R_h, the probability of the event $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap Z$ (*i.e.* getting a successful transcript T which opens a new party) is lower bounded by $(1-\delta)\cdot\tilde{\epsilon}-\epsilon > 0$. Moreover, the event $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap Z$ can occur at most $N-\ell$ times, because T_0 already opens ℓ parties and there are N parties in total. We deduce that after $N-\ell$ occurrences of $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap Z$, the list $\mathcal{T}$ contains a good witness.

Let us now define

$$N_1 = \frac{4(N-\ell)}{p_0} \quad \text{with} \quad p_0 := (1-\delta)\cdot\tilde{\epsilon}-\epsilon\ . \tag{6}$$

And let $X \sim \mathcal{B}(N_1, p_0)$ a binomial distributed random variable with parameters (N_1, p_0). The probability that $\mathcal{E}_0$ reaches the stop condition and returns a (good) witness for a successful transcript T_0 with good R_h satisfies:

$$\begin{aligned}
\Pr[\mathcal{E}_0(T_0) \neq \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0} \cap R_h \text{ good}] &\geq \Pr[X > N-\ell] \\
&= \Pr\left[\frac{X}{N_1} - p_0 > \frac{N-\ell}{N_1} - p_0\right] \\
&= 1 - \Pr\left[\frac{X}{N_1} - p_0 \leq \frac{N-\ell}{N_1} - p_0\right] \\
&= 1 - \Pr\left[\frac{X}{N_1} - p_0 \leq -\frac{3}{4}p_0\right] \\
&\geq 1 - \Pr\left[|\frac{X}{N_1} - p_0| \geq \frac{3}{4}p_0\right] \\
&\geq 1 - \frac{p_0\cdot(1-p_0)}{N_1\cdot p_0^2\cdot\left(\frac{3}{4}\right)^2} \qquad (7) \\
&= 1 - \frac{16}{9}\cdot\frac{1-p_0}{4\cdot(N-\ell)} = 1 - \frac{4}{9}\cdot\frac{1-p_0}{N-\ell} \\
&\geq 1 - \frac{4}{9} \geq \frac{1}{2}
\end{aligned}$$

The inequality (7) holds from the Bienaymé-Techbychev inequality. Thus, using $N_1 = \frac{4(N-\ell)}{p_0}$, the probability to reach the stop condition assuming a good R_h is at least 1/2. Without assumption on R_h, the probability to reach the stop condition satisfies:

$$\Pr[\mathcal{E}_0(T_0) \neq \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \geq \Pr[R_h \text{ good} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \cdot \Pr[\mathcal{E}_0(T_0) \neq \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0} \cap R_h \text{ good}] \geq \frac{\delta}{2}\ .$$

Let us now describe the complete extractor procedure:

```
Extractor $\mathcal{E}$:

1. Repeat $+\infty$ times:
2.     Run $\tilde{\mathcal{P}}$ with honest $\mathcal{V}$ to get transcript $T_0$
3.     If $T_0$ is not a successful transcript, go to the next iteration
4.     Call $\mathcal{E}_0$ on $T_0$ to get list of transcripts $\mathcal{T}$
5.     If $\mathcal{T} \neq \emptyset$, return $\mathcal{T}$
```

Let C denotes the number of calls to $\tilde{\mathcal{P}}$ made by the extractor before ending. While entering a new iteration:

– the extractor makes one call to $\tilde{\mathcal{P}}$ to obtain T_0,

– if T_0 is not successful, which occurs with probability $(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}])$,

  - the extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$,
- if T_0 is successful, which occurs with probability $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}]$,
  - the extractor makes at most N_1 calls to $\tilde{\mathcal{P}}$ in the loop of $\mathcal{E}_0$,
  - then $\mathcal{E}_0$ returns an empty list (the stop condition is not reached), which occurs with probability $\Pr[\mathcal{E}_0(T_0) = \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}]$, the extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$,
  - otherwise, if $\mathcal{E}_0(T_0)$ returns a non-empty list, the extractor stops and no more calls to $\tilde{\mathcal{P}}$ are necessary.

The mean number of calls to $\tilde{\mathcal{P}}$ hence satisfies the following equality:

$$\mathbb{E}[C] = 1 + \underbrace{(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}]) \cdot \mathbb{E}[C]}_{T_0 \text{ unsuccessful}} + \underbrace{\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \cdot \big(N_1 + \Pr[\mathcal{E}_0(T_0) = \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \cdot \mathbb{E}[C]}_{T_0 \text{ successful}}\big)$$

which gives

$$\begin{aligned}
\mathbb{E}[C] &\leq 1 + (1-\tilde{\epsilon}) \cdot \mathbb{E}[C] + \tilde{\epsilon} \cdot \Big(N_1 + \big(1 - \frac{\delta}{2}\big) \cdot \mathbb{E}[C]\Big) \\
&\leq 1 + \tilde{\epsilon} \cdot N_1 + \mathbb{E}[C]\Big(1 - \frac{\tilde{\epsilon} \cdot \delta}{2}\Big) \\
&\leq \frac{2}{\delta \cdot \tilde{\epsilon}} \cdot (1 + \tilde{\epsilon} \cdot N_1) \\
&= \frac{2}{\delta \cdot \tilde{\epsilon}} \cdot \left(1 + \tilde{\epsilon} \cdot \frac{4 \cdot (N - \ell)}{(1-\delta) \cdot \tilde{\epsilon} - \epsilon}\right)
\end{aligned}$$

To obtain an δ-free formula, let us take δ such that $(1-\delta)\cdot\tilde{\epsilon} = \frac{1}{2}(\tilde{\epsilon} + \epsilon)$. We have $\delta = \frac{1}{2}\left(1 - \frac{\epsilon}{\tilde{\epsilon}}\right)$ and the average number of calls to $\tilde{\mathcal{P}}$ is upper bounded as

$$\mathbb{E}[C] \leq \frac{4}{\tilde{\epsilon} - \epsilon} \cdot \left(1 + \tilde{\epsilon} \cdot \frac{8 \cdot (N - \ell)}{\tilde{\epsilon} - \epsilon}\right)$$

which concludes the proof.

E Proof of Soundness for Quasi-Threshold LSSS

This appendix provides a proof for the soundness property in the following theorem:

Theorem 3. *Let us consider an MPC protocol $\Pi_{QT\text{-}LSSS}$ complying with the protocol format described in Protocol 5, but using an $(\ell, \ell+\Delta+1, N)$-quasi-threshold LSSS in place of an $(\ell+1, N)$-threshold LSSS, and such that $\Pi_{QT\text{-}LSSS}$ is ℓ-private in the semi-honest model and of false positive rate p. Then, Protocol 6 built from $\Pi_{QT\text{-}LSSS}$ satisfies the three following properties:*

- ***Completeness.*** *A prover $\mathcal{P}$ who knows a witness w such that $(x, w) \in \mathcal{R}$ and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*
- ***Soundness.*** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input x, convinces the honest verifier $\mathcal{V}$ to accept with probability*
$$\tilde{\epsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle(x) \to 1] > \epsilon$$
where the soundness error ϵ is equal to
$$\frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+\Delta+1} \cdot \binom{N-\ell}{\Delta+1}.$$

Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces with either a witness w satisfying $(x, w) \in \mathcal{R}$, or a commitment collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by

$$\frac{4}{\tilde{\epsilon} - \epsilon} \cdot \left(1 + \tilde{\epsilon} \cdot \frac{8 \cdot (N - \ell)}{\tilde{\epsilon} - \epsilon}\right).$$

– ***Honest-Verifier Zero-Knowledge.*** *There exists an efficient simulator $\mathcal{S}$ which, given random challenge I outputs a transcript which is indistinguishable from a real transcript of Protocol 6.*

Remark 6. The proof for soundness with a quasi-threshold linear secret sharing scheme is very close to the proof in the threshold case. In fact, we can use exactly the same extractor (the extractor defined in Appendix D.2). We just need to prove Inequality (2) in the quasi-threshold case. In what follows, we use the same notations as in Appendix D.

For any set $\mathcal{T}$ of successful transcripts corresponding to the same commitment,

– either the revealed shares of $[\![w]\!]$ are not unique, and then we find a commitment collision or a hash collision,
– or the openings are unique.

Along this proof, we consider that the extractor only gets transcripts with unique revealed shares since otherwise the extractor would find a commitment collision or a hash collision.

We shall denote by R_h the randomness of $\tilde{\mathcal{P}}$ which is used to generate the initial commitment h_1 (which determines the witness sharing $[\![w]\!]$), and we denote r_h a possible realization of R_h. Throughout the proof, we denote $\mathsf{succ}_{\tilde{\mathcal{P}}}$ the event that $\tilde{\mathcal{P}}$ succeeds in convincing $\mathcal{V}$. By hypothesis, we have $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}] = \tilde{\epsilon}$.

E.1 When restraining to only bad witnesses

Let r_h be a possible realization of R_h. Given $R_h = r_h$, we have a unique hash commitment h_1 in the transcript. This hash commitment uniquely defines the shares of the witness $[\![w]\!]_1, \ldots, [\![w]\!]_N$ (by assumption on the absence of hash/commitment collisions). In the following, we shall denote $w^{(J)}$ the witness corresponding to the shares $\{[\![w]\!]_i\}_{i \in J}$, for $|J| = \ell + \Delta + 1$. We have a total of $\binom{N}{\ell+\Delta+1}$ possibly-distinct witnesses $w^{(J)}$. We shall say that $w^{(J)}$ is a *good witness* whenever $(x, w^{(J)}) \in \mathcal{R}$, otherwise we call $w^{(J)}$ a *bad witness.*

For any transcript produced by $\tilde{\mathcal{P}}$ (with h_1 as first hash commitment), the hash commitments h_2, $\ldots$, h_t uniquely define the shares $\{[\![\alpha^j]\!]_i\}_{i \in S}$, for $j \in [t]$. In the following, we shall denote by $[\![\alpha^j]\!] = ([\![\alpha^j]\!]_1, \ldots, [\![\alpha^j]\!]_N)$ the full $(\ell, \ell+\Delta+1, N)$-sharing consistent with the shares $\{[\![\alpha^j]\!]_i\}_{i \in S}$. The hash commitments h_2, $\ldots$, h_t also uniquely define the shares $\{[\![\beta^j]\!]_i\}_{i \in [N]}$, for $j \in [t]$. We shall denote $\mathcal{H}$ the set of *honest parties*, *i.e.* the set of the parties for which the committed shares $[\![\alpha^1]\!]_i$, $\ldots$, $[\![\alpha^t]\!]_i$, are consistent with the committed input shares $[\![w]\!]_i$ and $[\![\beta^1]\!]_i$, $\ldots$, $[\![\beta^t]\!]_i$. More formally,

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

We further denote Y the random variable which corresponds to the number of honest parties, *i.e.* $Y = |\mathcal{H}|$. We stress that $[\![\alpha^1]\!]$, $\ldots$, $[\![\alpha^t]\!]$, $\mathcal{H}$ and Y depend on the randomness of the (malicious) prover and the randomness of the verifier used before Step 3 of Protocol 6.

For every $i \in [N]$ and $j \in [t]$, we shall further denote $[\![\bar{\alpha}^j]\!]_i$ the share obtained through an honest computation from the committed input shares, that is:

$$[\![\bar{\alpha}^j]\!]_i = \varphi^j_{(\varepsilon^i)_{i \le j}, (\alpha^i)_{i<j}}([\![w]\!]_i, ([\![\beta^k]\!]_i)_{k \le j}) .$$

We stress that $[\![\bar{\alpha}^j]\!]_i$ might not be equal to $[\![\alpha^j]\!]_i$. We actually have $i \in \mathcal{H}$ if and only if $[\![\bar{\alpha}^j]\!]_i = [\![\alpha^j]\!]_i$ for every $j \in [t]$. In the following, we shall say that witness $w^{(J)}$ gives rise to a false positive in the MPC protocol Π, and denote this probability event E_J, whenever

$$g(\bar{\alpha}^1_J, \ldots, \bar{\alpha}^t_J) = 0$$

where $\bar{\alpha}^1_J, \ldots, \bar{\alpha}^t_J$ are the plain values corresponding to the sharings $\{[\![\bar{\alpha}^1]\!]_i\}_{i \in J}, \ldots, \{[\![\bar{\alpha}^t]\!]_i\}_{i \in J}$. By definition of the MPC protocol Π, we have:

$$\forall J \text{ s.t. } |J| = \ell + \Delta + 1,\ \ \Pr'[E_J] \leq p.$$

For the first step of the proof, we shall consider a subset D of parties, *i.e.* $D \subset \{1, \ldots, N\}$, and we denote

$$N' := |D| \qquad \text{and} \qquad \mathcal{J} = \{J \subset D : |J| = \ell + \Delta + 1\}.$$

We shall further denote

$$\Pr'[\,\cdot\,] := \Pr[\ \cdot\ \mid R_h = r_h, I \subset D].$$

We will show that, if $\{w^{(J)}\}_{J \in \mathcal{J}}$ are all *bad* witnesses, then the probability $\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}]$ is upper bounded by

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}] \leq \frac{\binom{N}{\ell}}{\binom{N'}{\ell}} \cdot \epsilon \tag{8}$$

where ϵ is the soundness error defined in the theorem statement, which is

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

The rest of this subsection is devoted to the demonstration of (8).

For $J \in \mathcal{J}$ and $b \in \{0, 1\}$, let us introduce the notation

$$A^b_J = \begin{cases} E_J & \text{if } b = 1, \\ \bar{E}_J & \text{if } b = 0. \end{cases}$$

Let $x = (x_J)_{J \in \mathcal{J}} \in \{0,1\}^{|\mathcal{J}|}$ and let $y \in \{0, \ldots, N\}$. Let us assume that $\mathsf{succ}_{\tilde{\mathcal{P}}}$, $Y = y$ and $\{A^{x_J}_J\}_{J \in \mathcal{J}}$ jointly occur. Because $\mathsf{succ}_{\tilde{\mathcal{P}}}$ occurs, we have that

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

where $\alpha^1, \ldots, \alpha^t$ are the values corresponding to the sharings $[\![\alpha^1]\!], \ldots, [\![\alpha^t]\!]$. Then for each set $J \in \mathcal{J}$ ($w^{(J)}$ is a bad witness) such that $J \subset \mathcal{H}$ (the parties in J are honest), we have that $[\![\bar{\alpha}^j]\!]_i = [\![\alpha^j]\!]_i$ for every $i \in J$ and every $j \in [t]$, which implies

$$g(\bar{\alpha}^1_J, \ldots, \bar{\alpha}^t_J) = g(\alpha^1, \ldots, \alpha^t) = 0\ .$$

Namely, a false positive necessarily occurs for $w^{(J)}$, *i.e.* $x_J = 1$, whenever $J \in \mathcal{J}$ with $J \subset \mathcal{H}$.

Thus

$$\mathrm{wt}_H(x) \geq \sum_{J \in \mathcal{J} : J \subset \mathcal{H}} x_J = \binom{y}{\ell + \Delta + 1}\ .$$

By defining

$$y_{\max} := \max\{y : \mathrm{wt}_H(x) \geq \binom{y}{\ell + \Delta + 1}\}\ ,$$

we get that

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}, Y = y \mid \{A^{x_J}_J\}_{J \in \mathcal{J}}] = 0 \qquad \text{if } y > y_{\max}$$

and so

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] = \sum_{y=0}^{y_{\max}} \Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}, Y = y \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] \ .$$

The only way for the transcript to be successful is that the set I of challenged opened parties only contains honest parties, *i.e.* $I \subset \mathcal{H}$. Thus,

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}, Y = y] \le \Pr[I \subset \mathcal{H} \mid I \subset D, Y = y] = \frac{\binom{y}{\ell}}{\binom{N'}{\ell}} \ .$$

We deduce

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] \ \le \ \sum_{y=0}^{y_{\max}} \frac{\binom{y}{\ell}}{\binom{N'}{\ell}} \cdot \Pr'[Y = y \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] \ \le \ \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} \ .$$

Since $\mathrm{wt}(x)$ is a non-negative integer, we have $y_{\max} \ge \ell$. Let us consider three cases:

Case 1: $y_{\max} = \ell + \Delta$, it means that $\mathrm{wt}(x) = 0$, then

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] \le \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} = \frac{\binom{\ell+\Delta}{\ell}}{\binom{N'}{\ell}} = \frac{\mathrm{wt}_H(x)\cdot\binom{\ell+\Delta}{\ell-1} + \binom{\ell+\Delta}{\ell}}{\binom{N'}{\ell}}$$

Case 2: $y_{\max} = \ell + \Delta + 1$, it means that $\mathrm{wt}(x) \ge 1$, then

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] \le \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} = \frac{\binom{\ell+\Delta+1}{\ell}}{\binom{N'}{\ell}} \le \frac{\mathrm{wt}_H(x)\cdot\binom{\ell+\Delta}{\ell-1} + \binom{\ell+\Delta}{\ell}}{\binom{N'}{\ell}}$$

since $\binom{\ell+\Delta+1}{\ell} = \binom{\ell+\Delta}{\ell-1} + \binom{\ell+\Delta}{\ell}$.

Case 3: $y_{\max} \ge \ell + \Delta + 2$, then

$$\begin{aligned}\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] &\le \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} = \frac{\binom{\ell+\Delta+1}{\ell}}{\binom{y_{\max}-\ell}{\Delta+1}} \cdot \frac{\binom{y_{\max}}{\ell+\Delta+1}}{\binom{N'}{\ell}} \\ &\le \frac{\binom{\ell+\Delta+1}{\ell}}{\binom{y_{\max}-\ell}{\Delta+1}} \cdot \frac{\mathrm{wt}_H(x)}{\binom{N'}{\ell}} \le \frac{\binom{\ell+\Delta+1}{\ell}}{\Delta+2} \cdot \frac{\mathrm{wt}_H(x)}{\binom{N'}{\ell}} \\ &\le \frac{\mathrm{wt}_H(x)\cdot\binom{\ell+\Delta}{\ell-1} + \binom{\ell+\Delta}{\ell}}{\binom{N'}{\ell}}.\end{aligned}$$

The last inequality can be prove using Lemma 5 since

$$\binom{\ell+\Delta+1}{\ell} = \binom{\ell+\Delta}{\ell-1} + \binom{\ell+\Delta}{\ell}.$$

In any case, we have the relation

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J\in\mathcal{J}}] \le \frac{\mathrm{wt}_H(x)\cdot\binom{\ell+\Delta}{\ell-1} + \binom{\ell+\Delta}{\ell}}{\binom{N'}{\ell}}. \tag{9}$$

for any $x \in \{0,1\}^{|\mathcal{J}|}$. And so, we have

$$\begin{aligned}
\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}] &= \sum_{x \in \{0,1\}^{\mathcal{J}}} \Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \cdot \Pr'[\{A_J^{x_J}\}_{J \in \mathcal{J}}] \\
&\leq \sum_{x \in \{0,1\}^{\mathcal{J}}} \frac{\mathrm{wt}_H(x) \cdot \binom{\ell+\Delta}{\ell-1} + \binom{\ell+\Delta}{\ell}}{\binom{N'}{\ell}} \cdot \Pr'[\{A_J^{x_J}\}_{J \in \mathcal{J}}] && \text{using 9} \\
&= \frac{\binom{\ell+\Delta}{\ell}}{\binom{N'}{\ell}} + \frac{\binom{\ell+\Delta}{\ell-1}}{\binom{N'}{\ell}} \cdot \sum_{x \in \{0,1\}^{\mathcal{J}}} \mathrm{wt}_H(x) \cdot \Pr'[\{A_J^{x_J}\}_{J \in \mathcal{J}}] && \text{using Lemma 4} \\
&\leq \frac{\binom{\ell+\Delta}{\ell}}{\binom{N'}{\ell}} + \frac{\binom{\ell+\Delta}{\ell-1}}{\binom{N'}{\ell}} \cdot \sum_{J \in J} \Pr'[E_J] && \text{using Lemma 4} \\
&\leq \frac{\binom{\ell+\Delta}{\ell}}{\binom{N'}{\ell}} + \frac{\binom{\ell+\Delta}{\ell-1}}{\binom{N'}{\ell}} \cdot |\mathcal{J}| \cdot p \\
&= \frac{\binom{N}{\ell}}{\binom{N'}{\ell}} \cdot \left(\frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}} + \frac{\binom{\ell+\Delta}{\ell-1}}{\binom{N}{\ell}} \cdot \underbrace{\binom{N'}{\ell+\Delta+1}}_{\leq \binom{N}{\ell+\Delta+1}} p \right) \leq \frac{\binom{N}{\ell}}{\binom{N'}{\ell}} \cdot \epsilon
\end{aligned}$$

since

$$\begin{aligned}
\frac{\binom{\ell+\Delta}{\ell-1}\binom{N}{\ell+\Delta+1}}{\binom{N}{\ell}} &= \frac{N! \cdot (\ell+\Delta)! \cdot \ell! \cdot (N-\ell)!}{N! \cdot (\ell+\Delta+1)! \cdot (\ell-1)!(N-\ell-\Delta-1)!(\Delta+1)!} \\
&= \frac{\ell}{\ell+\Delta+1}\binom{N-\ell}{\Delta+1}.
\end{aligned}$$

Thus we obtain the desired inequality (2).

E.2 Building of the extractor

Since Equation 8 proved in the previous subsection and Equation 2 proved in the case of threshold scheme (Section D.1) are the same, we can use the extractor $\mathcal{E}$ described in Appendix D.2.

F Signature Scheme and Proof of Unforgeability

We can transform the zero-knowledge proofs of knowledge described in Section 4 into signature schemes using the Fiat-Shamir's heuristic [FS87]. Protocols 7 and 8 describe the signing and verification algorithms obtained when following this approach for the 5-round case (*i.e.* for $t = 1$ iteration in the MPC protocol).

When applying the Fiat-Shamir transform, we compute the verifier challenges $(\varepsilon^{[e]})_{e \in [\tau]}$ and $(I^{[e]})_{e \in [\tau]}$ as:

$$\begin{aligned}
&h_1 = \mathrm{Hash}_1(m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[\tau]}) \\
&(\varepsilon^{[e]})_{e \in [\tau]} \leftarrow \mathrm{Expand}(h_1)
\end{aligned}$$

and

$$\begin{aligned}
&h_2 = \mathrm{Hash}_2(m, \mathsf{salt}, h_1, [\![\alpha^{[1]}]\!]_S, \ldots, [\![\alpha^{[N]}]\!]_S) \\
&(I^{[e]})_{e \in [\tau]} \leftarrow \mathrm{Expand}(h_2)
\end{aligned}$$

where m is the input message, Hash_1 and Hash_2 are cryptographic hash functions, Expand is an extendable output hash function, and $(\tilde{h}^{[e]}, [\![\alpha^{[e]}]\!]_S)_{e \in [\tau]}$ are the commitments and the broadcast shares merged for the

τ repetitions. We introduce a value salt called *salt* which is sampled from $\{0,1\}^{2\lambda}$ at the beginning of the signing process. This value is then used for each commitment to the parties' states. Since the signature security relies on the random oracle model, we can safely replace the commitment scheme Com of Protocol 6 by a single hash function Hash_0. Moreover, we derive all the randomness used in the scheme from a root seed for performance reason and to make the scheme easily turnable into a deterministic signature scheme. In particular, the randomness used for the sharings is derived from this root seed using a pseudo-random generator PRG which is made explicit in the description (while it was implicit in the description of the zero knowledge protocol). Finally, we denote Hash_m the hash function involved for the Merkle trees.

In this signature scheme, a secret key is a witness w and a public key is a statement x, with $(x, w) \in \mathcal{R}$ for the considered relation $\mathcal{R}$. We assume the existence of a function F which maps every witness to the corresponding statement:

$$F : w \mapsto x \quad \text{ s.t. } (x, w) \in \mathcal{R}.$$

We further assume that F is an $(t_{\mathrm{OWF}}, \epsilon_{\mathrm{OWF}})$*-hard one-way function*, namely an adversary $\mathcal{A}$ receiving a random statement x has a probability at most ϵ_{OWF} to output the corresponding witness w in time at most t_{OWF}.

The zero-knowledge protocol for relation $\mathcal{R}$ which we transform into the present signature scheme depends on two functions: ψ which computes the hints and φ which corresponds to the party computation. In practice, those two functions depend on the statement x (*i.e.*, on the public key in the case of a signature scheme), so for the sake of completeness, we will index them with x in this section: ψ_x and φ_x.

We first consider the notion of *unforgeability against key-only attacks* (EUF-KO). In this setting, the adversary is only given a public key x and she attempts to generate a pair (m, σ) such that σ is a valid signature of m with respect to x. The following lemma shows the EUF-KO of the signature scheme in the random oracle model and with respect to the OWF security of the function F.

Lemma 6. *Let* Hash_0, Hash_1, Hash_2, Hash_m *and* Expand *be modeled as random oracles, and let* (N, τ, λ, p) *be parameters of the signature scheme. Let* $\mathcal{A}$ *be an adversary against the EUF-KO security of the scheme running in time* $t_{\mathcal{A}}$ *and making a total of* Q *random oracle queries. Assuming that* F *is an* $(t_{\mathcal{A}}, \epsilon_{\mathrm{OWF}})$*-hard one-way function, then* $\mathcal{A}$*'s advantage in the EUF-KO game is*

$$\epsilon_{\text{EUF-KO}} \leq \epsilon_{\mathrm{OWF}} + \frac{(\tau N + 1)Q^2}{2^{2\lambda}} + \Pr[X + Y = \tau],$$

with

- $X = \max_{q_1 \in \mathcal{Q}_1}\{X_{q_1}\}$ *with* $X_{q_1} \sim \mathfrak{B}\left(\tau, \binom{N}{\ell+1} p\right)$ *and*
- $Y = \max_{q_2 \in \mathcal{Q}_2}\{Y_{q_2}\}$ *with* $Y_{q_2} \sim \mathfrak{B}\left(\tau - X, 1/\binom{N}{\ell}\right)$,

where $\mathfrak{B}(n_0, p_0)$ *denotes the binomial distribution with* n_0 *the number of trials and* p_0 *the success probability of each trial.*

The following proof is highly inspired (and carbon copied where relevant) from the proofs of [BDK+21, Lemma 2] and [KZ22, Lemma 5].

Proof. We give an algorithm $\mathcal{B}$ (the reduction) which uses the EUF-KO adversary $\mathcal{A}$ to compute a pre-image for the key generation function F.

Algorithm $\mathcal{B}$ simulates the EUF-KO game using the random oracles Hash_0, Hash_1, Hash_2 and Hash_m and query lists Q_c, Q_1, Q_2 and Q_m. In addition, $\mathcal{B}$ maintains two tables $\mathcal{T}_{\mathsf{sh}}$ and $\mathcal{T}_{\mathsf{wit}}$ which respectively store the shares of the parties and the corresponding witnesses (for a given set of $\ell + 1$ shares J) that $\mathcal{B}$ recovers from $\mathcal{A}$'s RO queries. $\mathcal{B}$ also maintains a set Bad to keep track of the outputs of all four random oracles. We also ignore calls to Expand in our analysis, since they are used to expand outputs from Hash_1 and Hash_2: when Expand is a random function this is equivalent to increasing the output lengths of Hash_1 and Hash_2.

Inputs: A secret key w, a public key $x := F(w)$ and a message $m \in \{0,1\}^*$.

Phase 0: Initialization.

1. Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.
2. Sample a root seed $\rho \leftarrow \{0,1\}^{2\lambda}$.

Phase 1: Preparation of the MPC-in-the-Head inputs. For each iteration $e \in [\tau]$,

1. Derive randomness $r_w^{[e]}$, $r_\beta^{[e]}$ and $r_\psi^{[e]}$ from the root seed ρ:
$$r_w^{[e]}, r_\beta^{[e]}, r_\psi^{[e]} \leftarrow \mathrm{PRG}(\mathsf{salt}, e, \rho).$$
2. Share the witness w into an $(\ell+1, N)$-threshold linear secret sharing $[\![w^{[e]}]\!]$:
$$[\![w^{[e]}]\!] \leftarrow \mathsf{Share}(w; r_w^{[e]}).$$
3. Compute
$$\beta^{[e]} \leftarrow \psi_x(w; r_\psi^{[e]})$$
and share it:
$$[\![\beta^{[e]}]\!] \leftarrow \mathsf{Share}(\beta^{[e]}; r_\beta^{[e]}).$$
4. Compute the commitments
$$\mathsf{com}_i^{[e]} := \mathrm{Hash}_0(\mathsf{salt}, e, i, [\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$$
for all $i \in [N]$, and compute the Merkle root
$$\tilde{h}^{[e]} := \mathrm{MerkleTree}(\mathsf{com}_1^{[e]}, \ldots, \mathsf{com}_N^{[e]}).$$

Phase 2: First challenge (randomness for the MPC protocol).

1. Compute $h_1 = \mathrm{Hash}_1(m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[\tau]})$.
2. Expand h_1 as $(\varepsilon^{[e]})_{e \in [\tau]} \leftarrow \mathrm{Expand}(h_1)$.

Phase 3: Simulation of the MPC protocol. For each iteration $e \in [\tau]$,

1. Computes, for $i \in S$,
$$[\![\alpha^{[e]}]\!]_i := \varphi_{x,\varepsilon^{[e]}}\left([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i\right)$$
and recomposes $\alpha^{[e]}$.

This step is repeated as many times as in the MPC protocol (cf Protocol 3).

Phase 4: Second challenge (parties to be opened).

1. Compute $h_2 = \mathrm{Hash}_2(\mathsf{salt}, h_1, [\![\alpha^{[1]}]\!]_S, \ldots, [\![\alpha^{[N]}]\!]_S)$.
2. Expand h_2 as $(I^{[e]})_{e \in [\tau]} \leftarrow \mathrm{Expand}(h_2)$ where, for every e, $I^{[e]} \subset [N]$ is a subset of ℓ parties (*i.e.* $|I^{[e]}| = \ell$).

Phase 5: Building of the signature. Output the signature σ built as
$$\mathsf{salt} \mid h_1 \mid h_2 \mid \left(([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)_{i \in I}, \mathsf{auth}^{[e]}, [\![\alpha^{[e]}]\!]_{i^{*[e]}} \right)_{e \in [\tau]}$$
where $\mathsf{auth}^{[e]}$ is the authentication path for $\{\mathsf{com}_i^{[e]}\}_{i \in I}$ w.r.t. Merkle root $\tilde{h}^{[e]}$ and $i^{*[e]} \in S \setminus I^{[e]}$.

Protocol 7: Signature Scheme – Signing algorithm

Inputs: A public key x, a signature σ and a message $m \in \{0,1\}^*$.

1. Parse the signature σ as
$$\mathsf{salt} \mid h_1 \mid h_2 \mid \Big((\llbracket w^{[e]} \rrbracket_i, \llbracket \beta^{[e]} \rrbracket_i)_{i \in I}, \mathsf{auth}^{[e]}, \llbracket \alpha^{[e]} \rrbracket_{i^{*[e]}} \Big)_{e \in [\tau]} \leftarrow \sigma$$
2. Expand h_1 as $(\varepsilon^{[e]})_{e\in[\tau]} \leftarrow \mathrm{Expand}(h_1)$.
3. Expand h_2 as $(I^{[e]})_{e\in[\tau]} \leftarrow \mathrm{Expand}(h_2)$ where, for every e, $I^{[e]} \subset [N]$ is a subset of ℓ parties (*i.e.* $|I^{[e]}| = \ell$).
4. For each iteration $e \in [\tau]$,
   - Computes the commitments $\mathsf{com}_i^{[e]}$ and the broadcast values $\llbracket \alpha^{[e]} \rrbracket_i$ for $i \in I^{[e]}$ from $(\llbracket w^{[e]} \rrbracket_i, \llbracket \beta^{[e]} \rrbracket_i)_{i\in I}$: for all $i \in I^{[e]}$,
   $$\mathsf{com}_i^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, i, \llbracket w^{[e]} \rrbracket_i, \llbracket \beta^{[e]} \rrbracket_i)$$
   $$\llbracket \alpha^{[e]} \rrbracket_i = \varphi_{x,\varepsilon^{[e]}}\left(\llbracket w^{[e]} \rrbracket_i, \llbracket \beta^{[e]} \rrbracket_i\right)$$
   - Recover $\alpha^{[e]}$, by
   $$\alpha^{[e]} = \mathsf{Reconstruct}_{I^{[e]} \cup \{i^{*[e]}\}}(\llbracket \alpha^{[e]} \rrbracket_{I \cup \{i^{*[e]}\}}).$$
   - Compute the Merkle root $\tilde{h}^{[e]}$ using $\left(\{\mathsf{com}_i^{[e]}\}_{i\in I}, \mathsf{auth}^{[e]}\right)$.
   - Compute the shares $\llbracket \alpha^{[e]} \rrbracket_S$ using $\llbracket \alpha^{[e]} \rrbracket = \mathsf{Expand}_{I\cup\{i^*\}}\left(\llbracket \alpha^{[e]} \rrbracket_{I\cup\{i^*\}}\right)$;
5. Compute $h_1' = \mathrm{Hash}_1(m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[\tau]})$.
6. Compute $h_2' = \mathrm{Hash}_2(\mathsf{salt}, h_1, \llbracket \alpha^{[1]} \rrbracket_S, \ldots, \llbracket \alpha^{[N]} \rrbracket_S)$.
7. Output ACCEPT iff $h_1' \overset{?}{=} h_1$, $h_2' \overset{?}{=} h_2$ and $\forall e \in [\tau], g(\alpha^{[e]}) \overset{?}{=} 0$.

Protocol 8: Signature Scheme – Verification algorithm

Behavior of $\mathcal{B}$. On input x, a OWF challenge, algorithm $\mathcal{B}$ forwards it to $\mathcal{A}$ as a signature public key for the EUF-KO game. It lets $\mathcal{A}$ run and answer its random oracle queries in the following way. We assume (wlog.) that Algorithm 1, Algorithm 2, Algorithm 3 and Algorithm 4 only consider queries that are correctly formed, and ignore duplicate queries. In these algorithms, the notation $v \to \mathcal{Q}$ is used to mean $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{v\}$.

– Hash_0: When $\mathcal{A}$ queries the commitment random oracle, $\mathcal{B}$ records the query to learn which commitment corresponds to which input share. See Algorithm 1.
– Hash_m: As Hash_0, it records the query to link the commitment with the Merkle roots. See Algorithm 2.
– Hash_1: When $\mathcal{A}$ sends the Merkle roots for the share commitments, $\mathcal{B}$ checks whether these roots were output by a right use of a Merkle tree simulated by Hash_m and for which the leaves were output by its simulation of Hash_0. If any were for some e and i, then $\mathcal{B}$ is able to reconstruct the shares for party i in repetition e. If $\mathcal{B}$ is able to reconstruct the shares for a subset J of $\ell + 1$ parties for an execution e, then it can extract the corresponding witness value $w^{[e](J)}$ used by $\mathcal{A}$ (for this execution e and this subset of parties J). See Algorithm 3. *Note: The algorithm description also include the computation of further values that are part of the protocol which are useless to Algorithm 3 and only included for notation purpose in order to make the analysis of the case* $\Pr[\mathcal{A}$ *wins* $\mid \mathcal{B}$ *outputs* $\perp]$ *easier to follow.*
– Hash_2: No extraction takes place during this random oracle simulation. See Algorithm 4.

```
Hash_0($q_0 = (\mathsf{salt}, e, i, \llbracket w \rrbracket_i, \llbracket \beta \rrbracket_i)$):
    $\mathsf{com} \overset{\$}{\leftarrow} \{0,1\}^{2\lambda}$.
    If $\mathsf{com} \in \mathsf{Bad}$, then abort.
    $\mathsf{com} \to \mathsf{Bad}$.
    $(q_0, \mathsf{com}) \to \mathcal{Q}_0$.
    Return $\mathsf{com}$.
```

Algorithm 1: Hash function Hash_0

```
Hash$_m$($q_m = (h_1, h_2)$):
    $h_1 \to \mathsf{Bad}$
    $h_2 \to \mathsf{Bad}$
    $h_m \xleftarrow{\$} \{0,1\}^{2\lambda}$.
    If $h_m \in \mathsf{Bad}$, then abort.
    $h_m \to \mathsf{Bad}$.
    $(q_m, h_m) \to \mathcal{Q}_m$.
    Return $h_m$.
```

Algorithm 2: Hash function Hash$_m$

In the rest of the proof, we assume that $\mathcal{A}$ returns a pair (m, σ) if and only if it is valid (with probability 1). This is wlog. since $\mathcal{A}$ can check that (m, σ) passes the verification before returning it without any degradation of her success probability. As a consequence, the hash h_1 in the returned (valid) signature has necessarily been obtained through a query q_1 to Hash$_1$ of the form $q_1 = (m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[\tau]})$. Moreover, all the hash computations from the commitments $\mathsf{com}_i^{[e]}$, with $i \in I^{[e]}$, to the Merkle root $\tilde{h}^{[e]}$ must have been obtained through valid requests to Hash$_m$ (otherwise the verification of $\mathsf{auth}^{[e]}$ would fail with overwhelming probability). Similarly, all the commitments $\mathsf{com}_i^{[e]}$, with $i \in I^{[e]}$, must have been obtained through valid calls to Hash$_0$. This notably implies that the table $\mathcal{T}_{\mathsf{sh}}$ filled by Algorithm 3 satisfies $\mathcal{T}_{\mathsf{sh}}[q_1, e, i] \neq \emptyset$ for every (e, i) such that $i \in I^{[e]}$.

When $\mathcal{A}$ terminates, $\mathcal{B}$ checks the $\mathcal{T}_{\mathsf{wit}}$ table for any entry where the extracted $w^{[e](J)}$ is consistent with x. If a match is found, $\mathcal{B}$ outputs $w^{[e](J)}$ as a pre-image for the OWF, otherwise $\mathcal{B}$ outputs $\perp$.

Advantage of the reduction. Given the behavior presented above, we have the following by the law of total probability:

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ ouputs } \perp] \\ &\quad + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ ouputs } w] \\ &\leq \Pr[\mathcal{B} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \mid \mathcal{B} \text{ ouputs } \perp] \\ &\quad + \Pr[\mathcal{B} \text{ ouputs } w]. \end{aligned} \tag{10}$$

Let Q_{com}, Q_m, Q_1 and Q_2 denote the number of queries made by $\mathcal{A}$ to each respective random oracle. Given the way in which values are added to Bad, we have:

$$\begin{aligned}\Pr[\mathcal{B} \text{ aborts}] &= (\#\text{times a digest is sampled}) \cdot \Pr[\mathcal{B} \text{ aborts at that digest}] \\ &\leq (Q_{\mathsf{com}} + Q_m + Q_1 + Q_2) \cdot \frac{\max|\mathsf{Bad}|}{2^{2\lambda}} \\ &= (Q_{\mathsf{com}} + Q_m + Q_1 + Q_2) \cdot \frac{Q_{\mathsf{com}} + 3Q_m + (\tau N + 1)Q_1 + 2Q_2}{2^{2\lambda}} \\ &\leq \frac{(\tau N + 1)(Q_{\mathsf{com}} + Q_m + Q_1 + Q_2)^2}{2^{2\lambda}} \end{aligned} \tag{11}$$

By definition, we also have

$$\Pr[\mathcal{B} \text{ ouputs } w] \leq \epsilon_{\mathrm{OWF}} \ .$$

It remains to deal with the term $\Pr[\mathcal{A} \text{ wins} \mid \mathcal{B} \text{ ouputs } \perp]$. Namely, we now analyze the probability of $\mathcal{A}$ winning the EUF-KO experiment conditioned on the event that $\mathcal{B}$ outputs $\perp$, i.e., no pre-image to x was found on the query lists. For the rest of the proof, we assume that $\mathcal{B}$ outputs $\perp$.

```
$\mathrm{Hash}_1(q_1)$:
    Parse $q_1$ as $(m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[\tau]})$
    For $e \in [\tau], i \in [N]$, do $\tilde{h}^{[e]} \to \mathsf{Bad}$.

    For $(e, i) \in [\tau] \times [N]$ such that $\exists \mathsf{com}_i^{[e]}$: $\mathsf{com}_i^{[e]}$ is the $i^{\text{th}}$ leaf
              of the Merkle tree with root $\tilde{h}^{[e]}$ where nodes are in $\mathcal{Q}_m$, do
        If $\exists (\llbracket w \rrbracket_i, \llbracket \beta \rrbracket_i)$ s.t. $((\mathsf{salt}, e, i, \llbracket w \rrbracket_i, \llbracket \beta \rrbracket_i), \mathsf{com}_i^{[e]}) \in \mathcal{Q}_0$, then
            $(\llbracket w \rrbracket_i, \llbracket \beta \rrbracket_i) \to \mathcal{T}_{\mathsf{sh}}[q_1, e, i]$.

    For each $e \in [\tau]$ and $J \subset [N]$, do
        If $\mathcal{T}_{\mathsf{sh}}[q_1, e, i] \neq \emptyset, \forall i \in J$, then
            $w^{[e](J)} \leftarrow \mathsf{Reconstruct}_J(\llbracket w \rrbracket_J)$.
            $\beta^{[e](J)} \leftarrow \mathsf{Reconstruct}_J(\llbracket \beta \rrbracket_J)$.
            $w^{[e](J)} \to \mathcal{T}_{\mathsf{wit}}[q_1, e, J]$.

    $h_1 \xleftarrow{\$} \{0,1\}^{2\lambda}$.
    If $h_1 \in \mathsf{Bad}$, then abort.
    $h_1 \to \mathsf{Bad}$.
    $(q_1, h_1) \to \mathcal{Q}_1$.

    $\triangleright$ This gray block is for notation purpose only
    $(\varepsilon^{[e]})_{e \in [\tau]} \leftarrow \mathrm{Expand}(h_1)$
    For each $e \in [\tau]$ and $J \subset [N]$:$\mathcal{T}_{\mathsf{wit}}[q_1, e, J] \neq \emptyset$, do
        $\alpha^{[e](J)} = \varphi_{\varepsilon^{[e]}}\left(w^{[e](J)}, \beta^{[e](J)}\right)$
    For each $(e, i) \in [\tau] \times [N]$:$\mathcal{T}_{\mathsf{sh}}[q_1, e, i] \neq \emptyset$, do
        $\llbracket \bar{\alpha}^{[e]} \rrbracket_i = \varphi_{\varepsilon^{[e]}}\left(\llbracket w^{[e]} \rrbracket_i, \llbracket \beta^{[e]} \rrbracket\right)$

    Return $h_1$.
```

Algorithm 3: Hash function Hash_1

```
$\mathrm{Hash}_2(q_2)$:
    Parse $q_2$ as $(\mathsf{salt}, h_1, (\llbracket \alpha^{[e]} \rrbracket_S)_{e \in [\tau]})$.
    $h_1 \to \mathsf{Bad}$.
    $h_2 \xleftarrow{\$} \{0,1\}^{2\lambda}$.
    If $h_2 \in \mathsf{Bad}$, then abort.
    $h_2 \to \mathsf{Bad}$.
    $(q_2, h_2) \to \mathcal{Q}_2$.
    Return $h_2$.
```

Algorithm 4: Hash function Hash_2

Cheating in the first round. For any query $(q_1, h_1) \in \mathcal{Q}_1$, and its corresponding expanded answer $(\varepsilon^{[e]})_{e\in[\tau]}$, let $G_1(q_1, h_1)$ be the set of indices $e \in [\tau]$ of "good executions" where there exists J such that both $\mathcal{T}_{\mathsf{wit}}[q_1, e, J]$ is non-empty and $g(\alpha^{[e](J)}) = 0$, namely a false positive occurs for at least one set J for execution e (since $w^{[e](J)}$ cannot satisfy $(x, w^{[e](J)}) \in \mathcal{R}$ since $\mathcal{B}$ outputs $\perp$). We then have, for every $e \in [\tau]$,

$$\Pr[e \in G_1(q_1, h_1) \mid \mathcal{B} \text{ outputs } \perp] \leq \binom{N}{\ell+1} p$$

where p is the false-positive rate of the underlying MPC protocol, given that h_1 is distributed uniformly at random (which holds since Hash_1 and Expand are random functions).

As the response h_1 is uniform, each $e \in [\tau]$ has the same independent probability of being in $G_1(q_1, h_1)$. We therefore have that $\#G_1(q_1, h_1) \sim X_{q_1}$ where $X_{q_1} = \mathfrak{B}\left(\tau, \binom{N}{\ell+1}p\right)$, the binomial distribution with τ trials, each with success probability $\binom{N}{\ell+1}p$. Letting $(q_{1\mathsf{best}}, h_{1\mathsf{best}})$ denote the query-response pair which maximizes $\#G_1(q_1, h_1)$, we then have that

$$\#G_1(q_{1\mathsf{best}}, h_{1\mathsf{best}}) \sim X = \max_{q_1 \in \mathcal{Q}_1} \{X_{q_1}\}.$$

Cheating in the second round. Let $(q_2, h_2) \in \mathcal{Q}_2$ be the query such that h_2 is used in the valid signature returned by the adversary. Since the returned signature is valid (with probability 1), there must also exist $(q_1, h_1) \in \mathcal{Q}_1$ such that h_1 is used in the signature and q_2 is of the form $q_2 = (h_1, \ldots)$. Then for each "bad" first-round execution $e \in [\tau]\backslash G_1(q_1, h_1)$, either the verification failed, in which case $\mathcal{A}$ couldn't have won, or the verification passed, despite $\forall J, g(\alpha^{[e](J)}) \neq 0$. We shall denote $\mathcal{H}^{[e]}(q_2)$ the set of *honest parties*, *i.e.* the set of the parties for which the committed shares $[\![\alpha^{[e]}]\!]_i$, are consistent with the committed input shares $[\![w^{[e]}]\!]_i$ and $[\![\beta^{[e]}]\!]_i$. More formally,

$$\mathcal{H}^{[e]}(q_2) = \left\{ i : [\![\alpha^{[e]}]\!]_i = \varphi_{\varepsilon^{[e]}}([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i) \right\} \ .$$

Let us consider three cases:

– there is strictly less than ℓ honest parties, *i.e.* $|\mathcal{H}^{[e]}(q_2)| < \ell$, but as ℓ parties are opened, verification would fail;
– there is strictly more than ℓ honest parties, *i.e.* $|\mathcal{H}^{[e]}(q_2)| > \ell$, but it would imply that there exists a set J of $\ell + 1$ honest parties ($\mathcal{J} \subset \mathcal{H}^{[e]}(q_2), |J| = \ell + 1$). In that case, we get that $\alpha^{[e](J)} = \alpha^{[e]}$ where $\alpha^{[e]}$ is the value encoded by $[\![\alpha^{[e]}]\!]_S$ in q_2. However, since $e \in [\tau]\backslash G_1(q_1, h_1)$, $g(\alpha^{[e](J)}) \neq 0$ implies that $g(\alpha^{[e]}) \neq 0$, and so verification would fail;
– there is exactly ℓ honest parties, *i.e.* $|\mathcal{H}^{[e]}(q_2)| = \ell$, which is the only possible case given that $\mathcal{A}$ wins with q_2.

Since the expanded $h_2 = (I^{[e]})_{e\in[\tau]} \in \{I \subset [N] : |I| = \ell\}^\tau$ is distributed uniformly at random, the probability that the verification passes while cheating for all such "bad" first-round executions e is

$$\left(\frac{1}{\binom{N}{\ell}}\right)^{\tau - \#G_1(q_1, h_1)} \leq \left(\frac{1}{\binom{N}{\ell}}\right)^{\tau - \#G_1(q_{1\mathsf{best}}, h_{1\mathsf{best}})} .$$

The probability that this happens for at least one of the Q_2 queries made to Hash_2 is

$$\Pr\left[\mathcal{A} \text{ wins} \mid \begin{array}{c} \mathcal{B} \text{ outputs } \perp \\ \#G_1(q_{1\mathsf{best}}, h_{1\mathsf{best}}) = \tau_1 \end{array}\right] \leq 1 - \left(1 - \left(\frac{1}{\binom{N}{\ell}}\right)^{\tau - \tau_1}\right)^{Q_2} .$$

Finally conditioning on $\mathcal{B}$ outputting $\perp$ and summing over all values of τ_1, we have that

$$\Pr[A \text{ wins} \mid \mathcal{B} \text{ outputs } \perp] \leq \Pr[X + Y = \tau] \qquad (12)$$

where X is as before, and $Y = \max_{q_2 \in \mathcal{Q}_2}\{Y_{q_2}\}$ where the Y_{q_2} variables are independently and identically distributed as $\mathcal{B}(\tau - X, 1/\binom{N}{\ell})$.

Conclusion. Bringing Equation (10), Equation (11) and Equation (12) together, we obtain the following:

$$\Pr[\mathcal{A} \text{ wins}] \leq \frac{(\tau N+1)(Q_{\mathsf{com}}+Q_m+Q_1+Q_2)^2}{2^{2\lambda}} + \Pr[X+Y=\tau] + \Pr[\mathcal{B} \text{ outputs } w].$$

Assuming KeyGen is an ϵ_{OWF}-secure OWF and setting $Q = Q_{\mathsf{com}} + Q_m + Q_1 + Q_2$ gives the required bound and concludes the proof.

□

We now consider the notion of *unforgeability against chosen message attacks* (EUF-CMA). In this setting, the adversary is given a public key x and she can ask an oracle (called the *signature oracle*) to sign messages $(m_1, \ldots, m_r)$ that she can select at will. The goal of the adversary is to generate a pair (m, σ) such that m is not one of requests to the signature oracle and such that σ is a valid signature of m with respect to x. The following theorem shows the EUF-CMA of the signature scheme in the random oracle model.

Theorem 4. *Let* Hash_0, Hash_1, Hash_2, Hash_m *and* Expand *be modeled as random oracles, and let* (N, τ, λ, p) *be parameters of the signature scheme. Let* $\mathcal{A}$ *be an adversary against the EUF-CMA security of the scheme running in time* $t_{\mathcal{A}}$ *and making a total of* Q_{RO} *random oracle queries and* Q_{sign} *signing queries. Assuming that* F *is an* $(t_{\mathcal{A}}, \epsilon_{\text{OWF}})$*-hard one-way function and that* PRG *is a* $(t_{\mathcal{A}}, \epsilon_{\text{PRG}})$*-secure pseudorandom generator then* $\mathcal{A}$*'s advantage in the EUF-CMA game is*

$$\epsilon_{\text{EUF-CMA}} \leq \epsilon_{\text{OWF}} + \epsilon_{\text{PRG}} + \frac{(\tau N+2)Q^2}{2^{2\lambda}} + \Pr[X+Y=\tau],$$

where $Q = Q_{RO} + N_{Hash} \cdot Q_{Sig}$ *with* $N_{Hash} = 2 + \tau(2N-1)$ *the number of hash computations in a signature generation, and where* X, Y *are defined as in Lemma 6.*

Proof. We consider the reduction algorithm $\mathcal{B}$ described in the proof of Lemma 6 which we extend to answer to the signing queries of the adversary $\mathcal{A}$. We consider three different games:

- Game 0: $\mathcal{B}$ uses a signature oracle $\mathcal{O}_{\text{Sig}}(w, x, \cdot)$ which perfectly answers signing queries from $\mathcal{A}$;
- Game 1: the signature oracle is replaced by $\mathcal{O}'_{\text{Sig}}(w, x, \cdot)$ which perfectly answers signing queries from $\mathcal{A}$ except that the calls to the PRG are replaced with true randomness;
- Game 2: the signature oracle is replaced by a simulator $\mathcal{S}_{\text{Sig}}(x, \cdot)$ answering signing queries from $\mathcal{A}$ without being given the secret witness as input.

In Game 0, $\mathcal{B}$ behaves exactly as in the proof of Lemma 6 while additionally answering the signing queries from $\mathcal{A}$ using $\mathcal{O}_{\text{Sig}}(w, x, \cdot)$. The signing oracle $\mathcal{O}_{\text{Sig}}(w, x, \cdot)$ makes queries to the random oracles Hash_0, Hash_1, Hash_2 and Hash_m which are answered by $\mathcal{B}$ as in the proof of Lemma 6, and it computes the signature from the input message and the key pair (w, x) as described in Protocol 7. The total number of random oracle queries is hence of $Q = Q_{\text{RO}} + N_{\text{Hash}} \cdot Q_{\text{Sig}}$. The signing queries being perfectly answered, $\mathcal{A}$ produces a valid signature with probability $\epsilon_{\text{EUF-CMA}}$. Then, by the proof of Lemma 6, $\mathcal{B}$ interacting with $\mathcal{A}$ and $\mathcal{O}_{\text{Sig}}(w, x, \cdot)$ recovers w with probability ϵ_{Game0} such that

$$\epsilon_{\text{EUF-CMA}} \leq \epsilon_{\text{Game0}} + \frac{(\tau N+1)Q^2}{2^{2\lambda}} + \Pr[X+Y=\tau] \ .$$

We note that this is insufficient to prove our security statement since the reduction $\mathcal{B}$ should not have access to the oracle $\mathcal{O}_{\text{Sig}}(w, x, \cdot)$, which is why we need to transit to Game 2.

Game 1 is similar to Game 0 but the signing oracle $\mathcal{O}_{\text{Sig}}(w, x, \cdot)$ is replaced by an oracle $\mathcal{O}'_{\text{Sig}}(w, x, \cdot)$. The latter works in the exact same way as the original signing oracle except that the the pseudo-randomness $(r_w^{[e]}, r_\beta^{[e]}, r_\psi^{[e]})$ of Phase 1, *i.e.* the outputs of $\text{PRG}(\mathsf{salt}, e, \rho)$, is replaced by true randomness (independent of the root seed ρ). This is indistinguishable from the previous reduction given that PRG is a secure pseudo-random generator. We deduce that the success probability ϵ_{Game1} of $\mathcal{B}$ to recover w while interacting with $\mathcal{A}$ and $\mathcal{O}'_{\text{Sig}}(w, x, \cdot)$ satisfies

$$|\epsilon_{\text{Game0}} - \epsilon_{\text{Game1}}| \leq \epsilon_{\text{PRG}} \ .$$

Finally, Game 2 is similar to the previous games but the signing oracle is replaced by a simulator $\mathcal{S}_{\mathrm{Sig}}(x, \cdot)$ which does not take the secret key w as input. Additionally, $\mathcal{B}$ keeps a list $\mathcal{S}l$ of all the salts appearing in random oracle queries. Namely, Algorithm 1 (Hash_0), Algorithm 3 (Hash_1) and Algorithm 4 (Hash_2) further perform $\mathsf{salt} \to \mathcal{S}l$ on any query with salt as salt. This simulator is depicted in Algorithm 5. Whenever $\mathcal{S}_{\mathrm{Sig}}$ aborts, $\mathcal{B}$ also aborts.

```
$\underline{\mathcal{S}_{\mathrm{Sig}}(x, m)}$:

    Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$
    If $\mathsf{salt} \in \mathcal{S}l$, then abort
    Sample random hashes $h_1 \leftarrow \{0,1\}^{2\lambda}$ and $h_2 \leftarrow \{0,1\}^{2\lambda}$
    If $h_1 \in \mathsf{Bad}$ or $h_2 \in \mathsf{Bad}$ then abort
    Expand $h_1$ as $(\varepsilon^{[e]})_{e\in[\tau]} \leftarrow \mathrm{Expand}(h_1)$
    Expand $h_2$ as $(I^{[e]})_{e\in[\tau]} \leftarrow \mathrm{Expand}(h_2)$
    Randomly generate $\alpha^{[e]}$ for every $e \in [\tau]$ s.t. $g(\alpha^{[e]}) = 0$
    Randomly generate the shares $[\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i$ for every $e \in [\tau]$ and $i \in I^{[e]}$
    Compute $[\![\alpha^{[e]}]\!]_i := \varphi_{x,\varepsilon^{[e]}}\left([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i\right)$ for every $e \in [\tau]$ and $i \in I^{[e]}$
    From $[\![\alpha^{[e]}]\!]_{I^{[e]}}$ and $\alpha^{[e]}$, reconstruct the shares $[\![\alpha^{[e]}]\!]_S$ for every $e \in [\tau]$
    Compute $\mathsf{com}_i^{[e]} := \mathrm{Hash}_0(\mathsf{salt}, e, i, [\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$ for every $e \in [\tau]$ and $i \in I^{[e]}$
    Sample random commitments $\mathsf{com}_i^{[e]} \leftarrow \{0,1\}^{2\lambda}$ for every $e \in [\tau]$ and $i \notin I^{[e]}$
    If $\mathsf{com}_i^{[e]} \in \mathsf{Bad}$ for some $e \in [\tau]$ and $i \notin I^{[e]}$, then abort
    Compute $\tilde{h}^{[e]} := \mathrm{MerkleTree}(\mathsf{com}_1^{[e]}, \ldots, \mathsf{com}_N^{[e]})$ for every $e \in [\tau]$
    For $q_1 = (m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[\tau]})$, let $(q_1, h_1) \to Q_1$
    For $q_2 = (\mathsf{salt}, h_1, ([\![\alpha^{[e]}]\!]_S)_{e\in[\tau]})$, let $(q_2, h_2) \to Q_2$
    Let $\mathsf{auth}^{[e]}$ the authentication path for $\{\mathsf{com}_i^{[e]}\}_{i\in I}$ w.r.t. Merkle root $\tilde{h}^{[e]}$
    Let $i^{*[e]} \in S \setminus I^{[e]}$
    Return $\sigma := \left[\mathsf{salt} \mid h_1 \mid h_2 \mid \left(([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)_{i\in I}, \mathsf{auth}^{[e]}, [\![\alpha^{[e]}]\!]_{i^{*[e]}}\right)_{e\in[\tau]}\right]$
```

Algorithm 5: Signing simulator.

Let us stress that the random generation of $\alpha^{[e]}$ such that $g(\alpha^{[e]}) = 0$ is done according to the real distribution of $\alpha^{[e]}$ in a valid signature. This distribution depends on the MPC protocol and is independent of w (by the zero-knowledge property).

The signing simulator follows the same principle as the zero-knowledge simulator of the proof-of-knowledge protocol. By knowing the challenges beforehand, it can generate a signature with perfect distribution without knowing the secret witness. In the random oracle model, this simply means randomly generating the answers h_1 and h_2 of the oracles Hash_1 and Hash_2 before they are actually queried. In the absence of abortion, we thus get that an answer of $\mathcal{S}_{\mathrm{Sig}}$ on input (x, m) is identically distributed to an answer of $\mathcal{O}'_{\mathrm{Sig}}$ on input $(w, x, \cdot)$.

We have to deal with an additional subtlety. By randomly generating the commitments $\mathsf{com}_i^{[e]}$ for every $e \in [\tau]$ and $i \notin I^{[e]}$, the simulator implicitly define some outputs of the Hash_0 oracle for which she does not know the input. Since the sharing $[\![w^{[e]}]\!]$ is fully defined by the shares $[\![w^{[e]}]\!]_{I^{[e]}}$ and the witness w, and the sharing $[\![\beta^{[e]}]\!]$ is fully defined by the shares $[\![\beta^{[e]}]\!]_{I^{[e]}}$ and the plain value $\beta^{[e]} = \psi_x(w; r_\psi^{[e]})$, Hash_0 should be constrained to answer a future request $q_0 = (\mathsf{salt}, e, i, [\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$ by the $\mathsf{com}_i^{[e]}$ randomly sampled by the simulator whenever a match occurs, *i.e.*, whenever

1. (salt, e, i) are such that salt corresponds to a previous signing request for which $i \neq I^{[e]}$,
2. the shares $[\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i$ in the request q_0 are consistent with the full sharings $[\![w^{[e]}]\!]$ and $[\![\beta^{[e]}]\!]$ defined by the shares $[\![w^{[e]}]\!]_{I^{[e]}}$ and $[\![\beta^{[e]}]\!]_{I^{[e]}}$ of this previous signing request together with the witness w.

To deal with this, we simply modify Algorithm 1 (Hash_0) in the following way. If a new request $q_0 = (\mathsf{salt}, e, i, [\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$ is made for which condition 1 above is satisfied, the algorithm reconstructs a candidate witness w^* from the shares $[\![w^{[e]}]\!]_{I^{[e]}}$ from the previous signing query and the share $[\![w^{[e]}]\!]_i$ from the current q_0 query. In case w^* is a valid witness, *i.e.* $F(w^*) = x$, $\mathcal{B}$ returns w^*. We thus have that such event can only occur if $\mathcal{B}$ outputs w and does not affect the event ($\mathcal{A}$ wins | $\mathcal{B}$ ouputs $\perp$) that we are considering here.

The probability of abortion due to collisions in Bad is the same for $\mathcal{O}'_{\mathrm{Sig}}$ and $\mathcal{S}_{\mathrm{Sig}}$. Indeed, they both do the same amounts of queries to the random oracles, the simulator just do them in a different order and handle the queries for h_1 and h_2 directly. $\mathcal{S}_{\mathrm{Sig}}$ may further aborts in case of salt collision, which happens with probability at most $(Q_{\mathrm{RO}} + Q_{\mathrm{Sig}})/2^{2\lambda}$. We deduce that the success probability $\epsilon_{\mathrm{Game2}} = \epsilon_{\text{EUF-CMA}}$ of $\mathcal{B}$ to recover w while interacting with $\mathcal{A}$ and simulating signing queries with $\mathcal{S}_{\mathrm{Sig}}(x, \cdot)$ satisfies

$$|\epsilon_{\mathrm{Game1}} - \epsilon_{\mathrm{Game2}}| \leq \frac{Q_{\mathrm{Sig}}(Q_{\mathrm{RO}} + Q_{\mathrm{Sig}})}{2^{2\lambda}} \leq \frac{Q^2}{2^{2\lambda}}$$

which concludes the proof.