# Share&Shrink: (In-)Feasibility of MPC from one Broadcast-then-Asynchrony, and Delegated Computation

**Abstract.** We consider protocols for secure multi-party computation (MPC) under honest majority, i.e., for $n=2t+1$ players of which $t$ are corrupt, that achieve *guaranteed output delivery* (GOD), and operate in a single initial round of broadcast ($\mathsf{BC}$), followed by steps of asynchronous peer-to-peer (P2P) messages. The power of closely related "hybrid networks" was studied in [Fitzi-Nielsen, Disc'09], [BHN, Podc'10] and [Patra-Ravi, IEEE Tr. Inf. Theory'18]. The interest of such protocols is that they go at the actual speed of the network, and security is preserved under arbitrary network conditions (past the initial $\mathsf{BC}$).
We first complete the picture of this model with an impossibility result showing that some setup is required to achieve honest majority MPC with GOD. We then consider a bare bulletin-board PKI setup, and leverage recent advances in multi-key homomorphic encryption [BJMS, Asiacrypt'20], to state the feasibility of MPC in a tight 1-$\mathsf{BC}$-then-1 single step of asynchronous P2P messages.
We then consider efficiency. The only protocols adaptable to such a network model and setup are [BJMS, Asiacrypt'20], which does not scale well for many players, and [GLS, Crypto'15], which does not support input delegation from external resource-constrained owners (such as IoT devices or smartphones), limiting its practical use. Our main contribution is a generic design that enables MPC in 1$\mathsf{BC}$-then-asynchronous P2P. It operates over ciphertexts encrypted under a (threshold) single-key encryption scheme, resulting in the smallest sizes expectable and efficient evaluation. It can be implemented from any homomorphic encryption scheme built from linear maps (e.g., $\mathsf{GSW}$, $\mathsf{CL}$, ...). Our main building block is the squishing of the verifiable input sharing ("Share"), in parallel with the distributed key generation ($\mathsf{DKG}$) in the single $\mathsf{BC}$, followed by threshold encryption ("Shrink") in one asynchronous step. Interestingly, it can be compiled as the first constant-round YOSO protocol in 1 $\mathsf{BC}$.

**Keywords:** MPC,Verifiable Computation,Homomorphic Encryption,Yoso

## 1 Introduction

Byzantine broadcast ($\mathsf{BC}$) and round-by-round synchronous communication are handy abstractions for designing simple multi-party computation protocols. However, in practice implementing a broadcast channel requires either multiple rounds of peer-to-peer communication [60,85,112] or special channels (such as blockchains), and is therefore costly and difficult in itself. This difficulty was pointed out in the comments of NIST's recent call for standardization of multiparty threshold-cryptography[2]. Moreover, implementing BC fails beyond $t < n/3$ corruptions if the network cannot deliver a number of consecutive synchronous rounds. An example is the protocol of [109] for the closely related primitive of consensus, without trusted distributed key generation (trusted DKG) setup, i.e., without distribution of correlated secrets to players by a trusted authority, which requires an expected 36 rounds. Thus significant effort is currently being put into minimizing the use of broadcast rounds in MPC [66,21,68,104,75,56,57]. We push further this line of work by asking for only one initial call to BC, i.e. a *broadcast-optimal* protocol as studied in [49,56,75,18], without any trusted DKG setup, followed by a fully asynchronous protocol. Beyond their security under network slowdowns, a further benefit of asynchronous protocols is that they go at the actual speed of the network, i.e., are *responsive* [50,104].

Since we aim at the (arguably gold standard) of guaranteed output delivery (GOD) under honest majority, this initial call to BC is necessary, as shown by an elementary split-brain attack [21]. Overall, two main approaches exist to design constant-round MPC protocols based on fully homomorphic encryption (FHE):

**Threshold FHE-based:** A mainstream tool for round-efficient MPC is based on $(n,t)-$Threshold Fully Homomorphic Encryption schemes. The following generic approach or close variations[25], that may be called "DKG-then-Input-distribution" in two BC, consists of: $\textcircled{1}$ the distributed generation (DKG) of a common *threshold encryption key* ek, with private assignment of a *decryption key share* to each player, such that the scheme provides IND-CPA security against adversaries controlling up to $t$ key shares. It is followed $\textcircled{2}$ by the broadcast of encrypted inputs under the common threshold encryption key. Then, players locally evaluate a circuit through an algorithm Eval on the encrypted inputs, that outputs a ciphertext. Finally, players $\textcircled{3}$ perform an asynchronous threshold decryption protocol using their key shares, so that any set of $t+1$ of them can recover the output.

**Multikey-FHE-based:** Another popular tool to further reduce the number of rounds is to use a (n,n)-threshold Multikey-FHE scheme [40,89,90], for which no DKG is needed before the input distribution, effectively reducing the number of broadcast to just one: it starts $\textcircled{1}$ with the local generation of keys, directly followed by the broadcast of encrypted inputs under the different keys. Then, players proceed with the multikey evaluation of a circuit through an expensive algorithm MultikeyEval, before $\textcircled{2}$ performing a $(n,n)$-threshold decryption.

These two methods highlight a tradeoff: one allows efficient evaluation at the cost of one extra broadcast, while the other allows obtaining an MPC protocol in an optimal number of broadcasts with an expensive evaluation. In this work, our goal is to propose a new method that offers the best of both worlds:

*A protocol with an efficient evaluation and only one initial broadcast.*

Overall, our aim is to develop a new approach that eliminates the need for a DKG before the input distribution. To achieve this, we face two key challenges:

- ***Challenge 1: Delegation.*** We observe from the DKG-then-Input-distribution approach that calls to BC are used to provide players with *a common view on threshold-encrypted inputs*. In a breakthrough approach, Gordon et al. [61] proposed the first $(n, t)$-threshold multikey-FHE scheme enabling the construction of an MPC protocol with GOD requiring one single BC[1]. However, it suffers from a practical issue. Indeed, as pointed out by [19], the most important property for deploying MPC in practice is the ability to *delegate* the costly computation to an untrusted cloud, in order to handle inputs coming from external resource-constrained input-owners (e.g. Iot devices, smartphones, ...) that, after the initial distribution of their inputs, do not take further part in the computation. Numerous techniques [15,103,107,41,64,113,100] have been proposed to achieve a secure and verifiable delegation of computation, scaling [24,22,106,9] to thousands or even millions of private inputs[2]. Unfortunately, the construction of [61] cannot enable delegation. Moreover, as it is based on GSW [74], sizes of ciphertexts are $d\times$ larger (where $d$ is the lattice dimension, see Table 2) than those of the latest FHE schemes based on the ring-learning-with-errors (RLWE) assumption [96]. Unfortunately, the approach of [61] is not generic, and in particular, not portable to RLWE (see Appendix A.1).
- ***Challenge 2: Efficient Evaluation.*** To remedy the latter issue, Badrinarayanan et al.[17] built an MPC protocol based on multikey-FHE using a more generic construction. However, this approach suffers from a major drawback: the sizes of their (multikey) ciphertexts are quadratic in the number $|\mathcal{Q}|$ of input-owners, and the same quadratic dependency applies to the multikey evaluation, making it inefficient. Part of this inefficiency is due to the use of the GSW-based multikey scheme from [101]. However, even using recent works [89,90] that reduced the overhead in homomorphic evaluation complexity to linear in $|\mathcal{Q}|$, it would still be $|\mathcal{Q}|\times$ less efficient than their single-key counterparts.

In this work, we then ask the following question:

*Is there a generic method providing a common view on ciphertexts of inputs under a (threshold) single-key encryption (so of sizes independent of $|\mathcal{Q}|$), using no more than one broadcast, without a trusted* DKG *setup?*

## 1.1 Results

Before we move to our main contribution, we first complete the theoretical picture of honest majority MPC with GOD from one initial round of broadcast. We enrich it with two new (in)feasibility results, as illustrated in Table 1.

***A) Contribution 1: Feasibility of 1 BC+asynch P2P MPC with GOD under honest majority.*** We assume one initial access to a broadcast functionality BC, which guarantees output whatever the (non)behavior of the sender.

[1]plus preliminary publication of input-independent material on a bulletin board PKI

[2]see the recent real-world applications, e.g. for health statistics with the COVID-19 Exposure Notification system developed jointly by Apple and Google [1], or for secure advertising by Meta [3,100], that is used for queries with up to 1 billion records.

**Theorem 1.** *There exists an MPC protocol with GOD under honest majority, under the sole setup of a bulletin board PKI; which furthermore enjoys (i) termination in* 1BC*-then-*1 *step of asynchronous P2P messages; and (ii) allows inputs from external owners, i.e., which do not take part in the computation;*

The baseline is the protocol of [17], that, as stated, lacks properties (i) and (ii). In Section 6, we show how their three-round MPC protocol in the plain model can be modified to work in a single-BC model and support external input-owners. Our proof differs from the one of Goel et al. [75], which considers a weaker setting as explained in Appendix C.

| Setup / Comm. | Trusted DKG | bPKI + URS | bPKI | No setup |
|---|---|---|---|---|
| 1 BC + 1 Sync P2P | ← | [61]+[56] ✓ | ✓ [17] + **Thm. 1** | [71] ✗ |
| 1 BC + $\infty$ Asynch P2P | [21] ✓ | ← | ✓ | **Thm. 5** ✗ |

Table 1: Feasibility and impossibility of MPC with GOD under honest majority with different setups and communication patterns. URS stands for *uniform random string*, and bPKI for *bulletin-board PKI*.

***B) Impossibility of* 1*-Broadcast-then-Asynchronous MPC without setup.*** In Theorem 5, we show that for $t \geq 3$ and $n \leq 3t - 4$, then some functionalities are not securely implementable, without setup, in one broadcast followed by an arbitrary number of pairwise asynchronous communications. It thus parallels [104], which dealt with perfect security. The strategy adapts [71, §4.1].

***Related Works.*** A long line of recent works [57,56,75] have undertaken to fully characterize what MPC protocols with one initial broadcast round allow achieving across various setup settings (such as with or without PKI, URS, ...). For instance, [57] studied the feasibility and impossibility of two-round MPC with different guarantees and broadcast patterns considering a model in which only a URS is available, but no PKI nor correlated randomness. It has been extended in [56] when a PKI is also available. In Table 1, we complete the picture by studying the case where a bulletin board PKI is available but no URS.

***C) Main Result: Share&Shrink.*** We answer positively the main question by proposing a new generic protocol in the bulletin-board PKI model, called Share&Shrink. Considering a set $\mathcal{Q}$ of input-owners wishing to delegate a computation on their inputs to a set of $n = 2t + 1$ players, our new protocol *simultaneously* addresses the main problems raised in Section 1: a) MPC with GOD in one Broadcast, b) Delegability, and c) Efficient Evaluation.

Overall, Share&Shrink performs *in parallel*: a DKG, and the distribution of inputs encrypted under a single threshold encryption key (thus of sizes independent of $|\mathcal{Q}|$). The broadcast BC is used **only once** in parallel by both players and input-owners. The second step is performed over asynchronous point-to-point channels. It leverages two main ingredients:

– First, any linear homomorphic encryption ($\ell$-HE) scheme as defined in Definition 3. This includes close variations of some well-known schemes such as GSW [101] or BFV [62]. This linearity property allows us to express these schemes as a set of linear maps. In particular, encryption consists in the evaluation of a linear map $\Lambda^{\mathsf{ek}}_{\mathsf{Enc}}$, parametrized by an encryption key $\mathsf{ek}$, over the secret input.
– Second, (any) publicly verifiable secret sharing (PVSS) scheme [73,83], formalized as a functionality $\mathcal{F}_{\mathsf{LSS}}$ in Section 3.2. Recall that a PVSS sharing algorithm, on input a secret $s$, roughly consists in generating a $(n, t)$-linear secret sharing of $s$, encrypting each $n$ shares with one of the $n$ players keys, before returning the resulting vector of $n$ ciphertexts. Interestingly, thanks to the linearity property of the sharing, one can compute linear combinations on the shared inputs.

Share & Shrink can be described as follows.

**0. Setup.** Each player non-interactively generates then publishes a public key in a bulletin board PKI, for any public-key encryption scheme (PKE). Players also retrieve a URS (see Section 2.2), as needed in most of the practical cases.

**1. Share.** Players run a DKG protocol in one broadcast round. The pattern is as in [67], possibly with the generation of a relinearization key as done in [111], if the $\ell$-HE scheme is a RLWE-based scheme with fully homomorphic capabilities.

***In parallel,*** input-owners broadcast PVSSs of their inputs and of encryption randomnesses.

**2. Shrink.** Each player obtains a common *threshold encryption key* $\mathsf{ek}$ and perform threshold encryptions of the shared inputs under $\mathsf{ek}$, thereby "shrinking" them down to the sizes of ciphertexts encrypted under the (threshold) single-key $\mathsf{ek}$. What makes threshold encryption work in *one step of peer-to-peer asynchronous messages*, is that it simply consists in the opening of a *linear map* $\Lambda^{\mathsf{ek}}_{\mathsf{Enc}}$, parametrized by $\mathsf{ek}$, evaluated over the shared inputs (and the shared randomnesses), leveraging the functionality $\mathcal{F}_{\mathsf{LSS}}$ we introduced.

At the end, players obtain a common view on inputs encrypted under a single-key $\mathsf{ek}$. They can proceed as in the DKG-then-Input-distribution method, i.e. with evaluation and later threshold decryption, which require no further broadcast.

In short, as shown in Table 2, Share&Shrink enables building a delegated MPC protocol with GOD in one broadcast with the following properties:

– ***Generic*** **(RLWE compatible)**, i.e. it is scheme-independent, and in particular can be built from RLWE-based schemes.
– ***Short ciphertexts size*** **(efficient Eval)**, i.e. the size of the ciphertexts undergoing homomorphic evaluation does not depend on the number of inputs-owners. This matters in practice as it is linked to computation complexity. Notably, the multikey ciphertexts used in previous works [17] have sizes that depend on $|\mathcal{Q}|$, leading to homomorphic operations complexity that also grows with $|\mathcal{Q}|$.

**Theorem 2 (Share&Shrink).** *For any linear homomorphic encryption scheme in the sense of Definition 3, and evaluation algorithm (or asynchronous protocol) over encrypted inputs in the sense of Section 5.1, there exists an MPC protocol under honest majority with GOD, in 1* BC *followed by asynchronous P2P* *messages. It furthermore allows inputs from* external input-owners*. It operates on ciphertexts under a (threshold) single-key encryption, and in particular, their sizes are* independent *of the number $|\mathcal{Q}|$ of input-owners, and of $n$. Moreover, it can straightforwardly be adapted to the YOSO model (see Appendix G).*

| Protocol | 1 BC + asynch P2P | GOD for $t < n/2$ | Delegability | Size of ciphertexts |
|---|---|---|---|---|
| [88][99] [102][98] | ✗ | ✗ | ✓ | $\|\mathscr{C}\|$ |
| [40] | ✓ | ✗ | ✗ | $\|\mathcal{Q}\| \cdot \|\mathscr{C}\|$ |
| [61] | ✓ | ✓ | ✗ | $\|\mathscr{C}\|$ |
| [17] + **Thm. 1** | ✓ | ✓ | ✓ | $\|\mathcal{Q}\| \cdot \|\mathscr{C}\|$ |
| **Thm. 2** | ✓ | ✓ | ✓ | $\|\mathscr{C}\|$ |

Table 2: MPC for $n = 2t + 1$ players and $|\mathcal{Q}|$ input-owners, using FHE with lattice dimension $d$ and modulus $q$, and assuming a URS and a bulletin-board PKI. The "Size" is the one of the ciphertexts which undergo homomorphic evaluation. "Delegability" means that input-owners can outsource the computation to a set of players. Protocols that must restart if a player aborts mid-process are not marked as GOD.

It has furthermore the *reusability* property (see Appendix C), i.e., messages from input-owners are independent of the circuit, allowing multiple circuits to be evaluated on a set of distributed inputs. [19] pointed out that not handling inputs from external lightweight owners, like mobile phones, is one of the main obstacles to the deployment of MPC. We thus believe that enabling delegation, and not having a complexity that grows with the number of input-owners, are significant advantages over previous works in 1BC then asynchrony [61,17].

## 1.2 More Related Works

**Approach of [61]**: Their protocol is detailed in Appendix A.1 and based on a threshold multikey-FHE scheme. Unfortunately, as previously argued, their construction is not generic, i.e. cannot be built from any RLWE-based FHE scheme, and not delegable, i.e. cannot accommodate external input-owners.
**Approach of [17]**. Their protocol is described in Section 6 and leverages a multikey-FHE scheme. As previously argued in Section 1, its main drawback is that the multikey evaluation unavoidably introduces an overhead in $|\mathcal{Q}|$, even in $|\mathcal{Q}|^2$ in their construction that uses GSW, which makes it very expensive to use.
**Approach of [11]:** Another useful (non FHE-based) tool to locally evaluate a circuit on encrypted data is garbled circuits. Following Yao's seminal work [114], they have become a core concept for 2-party secure computation, enabling one player to "encrypt" a circuit while another privately evaluates it. For more than two players, it was extended by [20], and followed by numerous constant-round protocols [94,11,93,12,13]. However, garbled-circuit-based protocols face scalability issues as the circuit size grows linearly with the number of players, making the per-player communication cost depend on the circuit size $|C|$. In [11], this cost is $O(n^\tau|C| + n^{\tau+1}d)$, where $\tau > 2$ and $d$ is the depth of $C$, increasing data processing and communication bandwidth requirements.

## 2 Model

**General notations.** We denote $x \xleftarrow{\$} \mathscr{D}$ the sampling of $x$ according to the distribution $\mathscr{D}$. Cardinality of a set $X$ is denoted as $|X|$. For a finite set $E$, we denote the uniform distribution on $E$ as $U(E)$. The set of positive integers $[1, \ldots, n]$ is denoted $[n]$. We denote vectors in bold, e.g., $\mathbf{v}$, and $\overrightarrow{\mathscr{D}}$ a vector of distributions. We denote by $\lambda$ the security parameter throughout the paper.

We consider a positive integer $d$, denoted the *lattice dimension*; a monic polynomial $f$ of degree $d$; $k < q$ positive integers denoted as plaintext and ciphertext moduli; and $R := \mathbb{Z}[X]/f(X)$. We denote $R_k = R/(k.R)$ and $R_q = R/(q.R)$ the residue rings of $R$ modulo $k$ and $q$. We denote $[.]_k$ the reduction of an integer modulo $k$ into $R_k$. When applied to polynomials or vectors, these operations are performed coefficient-wise. All linear forms are succinctly specified as *formal linear combinations*, e.g., let $(\overline{x_i})_i$ denote *labels* of some variables $(x_i)_i$, then, $\sum_i \lambda_i \overline{x_i}$ denotes $\left\{(x_i)_i \to \sum_i \lambda_i x_i\right\}$.

### 2.1 Players, Input Owners and Corruptions.

We consider $n = 2t+1$ players $\mathscr{P} = (P_i)_{i \in [n]}$, which are probabilistic polynomial-time (PPT) machines, of public identities. We also consider PPT machines called *input-owners* $\mathcal{Q}$, and an *output-learner* $\mathcal{L}$, which are logically disjunct from players. Using the Universal Composability (UC) model [32] with static corruptions (Appendix B.8), we consider a PPT machine, denoted as the Environment Env. It fully controls an entity called the "dummy adversary" $\mathcal{A}$. At the start of the execution, $\mathcal{A}$ may corrupt up to $t$ players of its choosing, along with any number of input-owners. They behave as instructed by $\mathcal{A}$. Without loss of generality, $\mathcal{A}$ corrupts exactly $t$ players, indexed by $\mathcal{I} \subset [n]$. The remaining *honest* ones are indexed by $\mathcal{H} = [n] \setminus \mathcal{I}$. $\mathcal{A}$ notifies Env of every message received by corrupt players and from (simulated) functionalities. $\mathcal{A}$ can *rush*, i.e. all messaging functionalities (BC, bPKI, $\mathcal{F}_{\mathrm{AT}}$) let $\mathcal{A}$ learn the messages sent by honest players before choosing those sent by corrupt ones.

For simplicity, we present our protocol in the *semi-malicious* corruption model of [14], widely adopted since [61]. Details are provided in Appendix B.9.

### 2.2 Ideal Functionalities

***Broadcast with eventual termination:* BC.** We formalize in Fig. 5 in Appendix B.3 the ideal functionality of broadcast. It is parametrized by a sender, and by a set of receivers. It has the following properties: (Termination) all honest receivers eventually output, and (Consistency) any two honest receivers output the same value. Finally, (Validity) if the sender is honest and inputs value $m$, all honest receivers output the same $m$.

***(Asynchronous) Authenticated Message Transmitting:* $\mathcal{F}_{AT}$.** We formalize in Fig. 6 the ideal functionality of asynchronous *public authenticated* message transmitting with eventual delivery delay. It is parametrized by a sender and a receiver, hence the terminology *authenticated*. It delivers every message sent within a finite delay $D$, although $D$ can be adaptively increased by $\mathcal{A}$. It leaks the content of every message to $\mathcal{A}$, hence the terminology *public*.

***"Bulletin Board PKI":* $\mathsf{bPKI}$.** We present in Fig. 7 the ideal functionality of a bulletin board of public keys, denoted as $\mathsf{bPKI}$. Upon receiving a key $\mathsf{pk}_i$ from any player $P_i \in \mathscr{P}$, it stores $(P_i, \mathsf{pk}_i)$ and leaks this information to the adversary $\mathcal{A}$. Then, it *waits until it receives a public key from every honest player* in $\mathscr{P}$, and sets a timeout. After it elapsed, $\mathsf{bPKI}$ sets to $\perp$ the keys of the players which did not give a key and eventually delivers $\mathsf{pk} \leftarrow (\mathsf{pk}_i)_{i\in[n]}$.

***Global Public Uniform Random String (URS):* $\overline{\mathcal{G}}_{URS}$.** It samples uniformly at random a sequence of bits of pre-defined length $\kappa$, denoted URS, then outputs it to all players. It is further formalized in Fig. 8 of Appendix B.6.

### 2.3 Ideal Functionality of MPC $\mathcal{F}_{\mathrm{C}}$.

The ideal functionality of MPC that we aim to UC implement, is formalized as $\mathcal{F}_{\mathrm{C}}$ in Fig. 4. It returns to an output-learner $\mathcal{L}$ the evaluation of an arithmetic circuit $\mathrm{C} : (\mathscr{M} \cup \{\perp\})^n \to \mathscr{M}$ over inputs in $\mathscr{M}$. For simplicity: C has $n$ input gates, one output gate, and $\mathcal{F}_{\mathrm{C}}$ expects one single input from each owner.

The functionality works as follows. Upon receiving an input from any input-owner $\mathsf{Q}_i$, it stores $(\mathsf{Q}_i, \overline{m_i})$[3] and leaks this to $\mathcal{A}$. Before delivering the output, $\mathcal{F}_{\mathrm{C}}$ needs to wait for the inputs to be submitted. However, $\mathcal{A}$ may prevent corrupt owners from sending their inputs. To handle this, the functionality *i*) waits until it receives an input from every honest owner, *ii*) sets a timeout $T_{\mathcal{A}}$, and *iii*) sets to $\perp$ the inputs of the (corrupt) owners which did not give an input after the timeout. The evaluation is delivered after a finite $\mathsf{delay}$ chosen by $\mathcal{A}$.

## 3 Cryptographic Ingredients

We now detail the main cryptographic ingredients needed for the remainder of the paper. After defining a Linear Homomorphic Encryption scheme in Section 3.1 and detailing an example, we describe in Section 3.2, an ideal functionality for linear secret sharing, denoted $\mathcal{F}_{\mathsf{LSS}}$, and discuss its implementation.

### 3.1 Toy model of linear homomorphic encryption scheme.

We observe that in many encryption schemes, key generation, encryption, and decryption are essentially *linear maps*. For simplicity's sake, we assume that all operations take place over $\mathbb{Z}/q\mathbb{Z}$-modules, where $q$ is a known modulus[4]. A linear map between two $\mathbb{Z}/q\mathbb{Z}$-modules $g : (E, +) \to (F, *)$ satisfies $g(e_1 + e_2) = g(e_1) * g(e_2)$ for $e_1, e_2 \in E$, and, $g(a \cdot e) = a \cdot g(e)$ for all $a \in \mathbb{Z}/q\mathbb{Z}$. In such schemes, key generation is a linear function in a decryption key $\mathsf{sk}$ and some randomness, encryption is linear in a plaintext and some randomnesses, while decryption is, roughly, linear in $\mathsf{sk}$. This linearity implies that there exists a public map which, given an offset (roughly: $\mathsf{sk}'$), maps encryptions under $\mathsf{sk}$ into ciphertexts which have the same distribution as fresh encryptions under $\mathsf{sk} + \mathsf{sk}'$. This linearity also supports partial homomorphic operations, such as addition and/or

[3] Where $\overline{m_i}$ denotes the label of variable $m_i$

[4] However, this formalization can be adapted to settings where $q$ is unknown, such as in the $\mathsf{CL}$ case, where secret-sharing operates over $\mathbb{Z}$, as explained in [31].

multiplication, in some form. The following Definition 3 provides a wrapper for all such schemes, formalized using generic linear maps ($\Lambda_{\mathsf{EKeyGen}}$, $\Lambda_{\mathsf{Enc}}$ and $\Lambda_{\mathsf{Dec}}$).

***Definition 3 (Linear Homomorphic Encryption ($\ell$-HE)).*** *A $\ell$-HE scheme consists of a message space $\mathscr{M}$; spaces of decryption and encryption keys $\mathscr{X}$, $\mathscr{E}k$; randomness spaces $\mathscr{B}_{Key}$ and $\overrightarrow{\mathscr{B}_{\mathsf{Enc}}}$, both (subsets of) vector spaces over $\mathbb{Z}/q\mathbb{Z}$ modules; and a ciphertext space $\mathscr{C}$; along with the following PPT algorithms:*

- $\mathsf{Setup}(1^\lambda)$: *On input the security parameter $\lambda$, the setup algorithm outputs a set of public parameters* $\mathsf{pp}$ *and a uniform random string $a$.*
- $\mathsf{KeyGen}(\mathsf{pp}, a)$: *On input some public parameters* $\mathsf{pp}$, *and a URS $a$, the key generation algorithm samples a decryption key* $\mathsf{sk} \xleftarrow{\$} \mathscr{X}$ *and a key randomness* $\rho^{key} \xleftarrow{\$} \mathscr{B}_{Key}$. *When $\mathscr{B}_{Key}$ is not closed under addition, the randomness $\rho^{key}$ is drawn small enough such that the sum of $n$ such terms remains statistically within $\mathscr{B}_{Key}$. In our multiparty setting, this ensures that a common key resulting from a sum of $n$ keys remains valid, as illustrated in Equation* (1). *The algorithm outputs an encryption key* $\mathsf{ek} \leftarrow \Lambda^{a}_{\mathsf{EKeyGen}}(\mathsf{sk}, \rho^{key}) \in \mathscr{E}k^{5}$, *where $\Lambda^{a}_{\mathsf{EKeyGen}}$ is a public* linear map *with coefficients determined by the URS $a$.*
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{ek} \in \mathscr{E}k,\ m \in \mathscr{M})$: *On input public parameters* $\mathsf{pp}$, *an encryption key* $\mathsf{ek}$ *and a plaintext $m$, the encryption algorithm samples a vector of randomnesses* $\boldsymbol{\rho}_{\mathsf{Enc}} \xleftarrow{\$} \overrightarrow{\mathscr{B}_{Enc}}$ *and outputs a ciphertext* $\mathsf{c} \leftarrow \Lambda^{\mathsf{ek}}_{\mathsf{Enc}}(m, \boldsymbol{\rho}_{\mathsf{Enc}}) \in \mathscr{C}$.
- $\mathsf{Dec}(\mathsf{sk} \in \mathscr{X}, \mathsf{c} \in \mathscr{C})$: *On input a ciphertext* $\mathsf{c}$ *and a decryption key* $\mathsf{sk}$, *the decryption algorithm computes $\mu \leftarrow \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk})$, and either outputs a plaintext $m = \Omega_{\mathsf{Dec}}(\mu)$ or $\perp$, where $\Omega_{\mathsf{Dec}}$ denotes a non-linear decoding function.*

***Examples.*** Definition 3 applies to various classical schemes such as BGN [26], exponentiated ElGamal [81,110], GSW [74] or CL [39], as discussed in Appendix D. Interestingly, this definition wraps schemes with different degrees of homomorphic capabilities. In particular, FHE schemes that satisfy Definition 3 are categorized as $\ell$-FHE, with more details provided in Section 3.1.1.

***MPC from $\ell$-HE.*** Our main goal is to build a constant-round MPC protocol in 1 BC using efficient RLWE-based public-key FHE schemes. Modern schemes like BFV[62] or CKKS[43] require the generation of an extra "relinearization key" **rlk** for homomorphic evaluation of a circuit. To be compatible with our Share&Shrink protocol in 1 BC later described in Section 4, such FHE schemes must meet the $\ell$-HE criteria outlined in Def. 3, i.e. to be built from linear maps, including for this relinearization key. Unfortunately, most RLWE-based schemes [88,102,99] do not satisfy this, as their **rlk** are quadratic in the decryption key sk. In Section 3.1.1, we consider the only known RLWE-based scheme that meets this requirement.

**3.1.1 $\ell$-BFV: A RLWE-based $\ell$-FHE scheme with Linear Relinearization Key Generation.** In [111], Urban and Rambaud introduced $\ell$-BFV, a variant of the BFV FHE scheme, with a relinearization key generation that is linear in the decryption key, unlike the original [62].

$^5$sometimes with a *relinearization key* $\mathbf{rlk} \in \mathscr{R}lk$, which must also be linear in sk

***Definition 4.*** *($\ell$-BFV [111]) $\ell$-BFV is defined as a mere $\ell$-HE scheme, augmented with an additional linear map, $\Lambda_{\mathsf{RlkGen}}$, for generating a relinearization key. $\ell$-BFV takes two URSs as inputs:* $\mathbf{a}$ *and* $\mathbf{d}_1$*. Let $\mathscr{X}_q$, $\Psi_q$ and $\mathscr{B}_{\mathsf{Enc},q}$ be distributions over $R_q$, with coefficients distributed according to a centered discrete Gaussian with standard deviation $\sigma$ (resp $\sigma_{\mathsf{Enc}}$) and truncated support over $[-B, B]$ (resp $B_{\mathsf{Enc}}$) where $\sigma$ and $B$ are cryptosystem parameters. $\ell$-BFV leverages the well-known gadget toolkit [74,40], including the following* gadget vector*:* $\mathbf{g} \in R_q^l$*, for some parameter $l \in \mathbb{N}$, and consists of the following PPT algorithms:*

- $\ell$-BFV.KeyGen()*: Parse the URSs as $(\mathbf{a} \in R_q^l, \mathbf{d}_1 \in R_q^l)$, sample $e^{(\mathsf{pk})} \xleftarrow{\$} \Psi_q$ and* $\mathsf{sk} \xleftarrow{\$} \mathscr{X}_q$*, and set $a = \mathbf{a}[0]$. Define $\Lambda^{a}_{\mathsf{EKeyGen}} : (\mathsf{sk}, e^{(\mathsf{pk})}) \to (-a \cdot \mathsf{sk} + e^{(\mathsf{pk})}, a)$.*
  *Sample $r \leftarrow \mathscr{X}_q$, $\mathbf{e}_0^{(\mathbf{rlk})}, \mathbf{e}_2^{(\mathbf{rlk})} \leftarrow \Psi_q^l$, and define $\Lambda^{\mathbf{a},\mathbf{d}_1,\mathbf{g}}_{\mathsf{RlkGen}} : (\mathsf{sk}, r, \mathbf{e}_0^{(\mathbf{rlk})}, \mathbf{e}_2^{(\mathbf{rlk})}) \to$* $(-\mathsf{sk} \cdot \mathbf{d}_1 + \mathbf{e}_0^{(\mathbf{rlk})} + r \cdot \mathbf{g}, \mathbf{d}_1, r \cdot \mathbf{a} + \mathbf{e}_2^{(\mathbf{rlk})} + \mathsf{sk} \cdot \mathbf{g})$
  *Output $\mathsf{ek} \leftarrow \Lambda^{a}_{\mathsf{EKeyGen}}(\mathsf{sk}, e^{(\mathsf{pk})}) = (-a \cdot \mathsf{sk} + e^{(\mathsf{pk})}, a) = (b, a)$, the relinearization key* $\mathbf{rlk} \leftarrow \Lambda^{\mathbf{a},\mathbf{d}_1,\mathbf{g}}_{\mathsf{RlkGen}}(\mathsf{sk}, r, \mathbf{e}_0^{(\mathbf{rlk})}, \mathbf{e}_2^{(\mathbf{rlk})})$ *and* $\mathsf{sk}$.
- $\ell$-BFV.Enc$(\mathsf{ek} = (b, a), m \in R_k)$*: Sample the encryption randomnesses $e_0^{(\mathsf{Enc})} \xleftarrow{\$} \mathscr{B}_{\mathsf{Enc},q}$, $e_1^{(\mathsf{Enc})} \xleftarrow{\$} \Psi_q$, and $u \xleftarrow{\$} \mathscr{X}_q$. Thus here: $\overrightarrow{\mathscr{B}_{Enc}} = \mathscr{B}_{\mathsf{Enc},q} \times \Psi_q \times \mathscr{X}_q$.*
  *Define $\Lambda^{b,a}_{\mathsf{Enc}} : (\Delta m, u, e_0^{(\mathsf{Enc})}, e_1^{(\mathsf{Enc})}) \to (\Delta m + u \cdot b + e_0^{(\mathsf{Enc})}, u \cdot a + e_1^{(\mathsf{Enc})})$ with $\Delta = \lfloor q/k \rfloor$.*
  *Output $\mathsf{c} \leftarrow \Lambda^{b,a}_{\mathsf{Enc}}(\Delta m, u, e_0^{(\mathsf{Enc})}, e_1^{(\mathsf{Enc})}) \in R_q^2$.*
- $\ell$-BFV.Dec$(\mathsf{sk}, \mathsf{c})$*: Given a ciphertext $\mathsf{c} = (\mathsf{c}[0], \mathsf{c}[1]) \in R_q^2$, define $\Lambda^{\mathsf{c}}_{\mathsf{Dec}} : \mathsf{sk} \to$* $\mathsf{c}[0] + \mathsf{c}[1] \cdot \mathsf{sk}$ *and compute $\mu \leftarrow \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk})$.*
  *Output $m \leftarrow \left[\left\lfloor \frac{k}{q}(\mu) \right\rceil\right]_k := \Omega_{\mathsf{Dec}}(\mu) \in R_k$.*

$\ell$-BFV *is a fully homomorphic* $\ell$-FHE *scheme that serves as our main example for our security proof in Section 5.3 and experiments in Section 7.*

## 3.2 Toy functionality of LSS, and instantiation in 1BC+1 async P2P

The main ingredient in building a robust threshold scheme is a $(n, t)$-linear secret sharing scheme ($(n, t)$-LSS, Definition 9), that divides a secret $s$ into $n$ shares, allowing only authorized subsets of $t+1$ of them to reconstruct the secret. Due to the linearity of the scheme, given shared secrets $m_1, \ldots, m_n$, some linear map $\Lambda$ can be applied to compute the linear combination $\Lambda(\{m_i\}_i)$ on the shared inputs. We now abstract this through a functionality $\mathcal{F}_{\mathsf{LSS}}$ and provide an overview of its implementation. A detailed description and security proof are in Appendix E.

**3.2.1 Functionality $\mathcal{F}_{\mathsf{LSS}}$.** We specify in Fig. 10, an ideal functionality for linear secret sharing, denoted $\mathcal{F}_{\mathsf{LSS}}$. It is parametrized by i) a set $\mathscr{P}$ of $n$ players, ii) a list $\mathscr{S}$ of entities of the senders, where each $\mathcal{S} \in \mathscr{S}$ has a list of inputs: $(x_{\mathcal{S},\alpha})_{\alpha \in X_{\mathcal{S}}}$, identified by input labels $(\overline{x_{\mathcal{S},\alpha}})_{\alpha \in X_{\mathcal{S}}}$. We denote $X_{\mathcal{S}}$ the list of indices $\alpha$ of inputs of sender $\mathcal{S}$. Finally iii), we consider an output-learner $\mathcal{L}$.

*Setup.* Before any sender starts interacting with $\mathcal{F}_{\mathsf{LSS}}$, it needs to wait until $(\mathsf{Setup}, P)$ is stored $\forall P \in \mathscr{P}$. However, the adversary $\mathcal{A}$ can choose to never instruct corrupt players to setup. To remedy this, we use the *fetch-and-delay* mechanism explained in Appendix B.1 and introduce a timeout $T_{\mathcal{A}}$.

*Input.* Upon receiving $(\mathsf{ready})$ from $\mathcal{F}_{\mathsf{LSS}}$, any $\mathcal{S} \in \mathscr{S}$ can then send its inputs $(x_{\mathcal{S},\alpha})_{\alpha \in X_{\mathcal{S}}}$ of labels $(\overline{x_{\mathcal{S},\alpha}})_{\alpha \in X_{\mathcal{S}}}$, after which $\mathcal{F}_{\mathsf{LSS}}$ notifies all the players. These values cannot be subsequently updated; $\mathcal{S}$ is committed to the submitted values.

*Opening.* Let $\mathsf{HOpeners}$ be a set of players, initially empty. Any player $P_i$ can call $\mathsf{LCOpen}$ for some linear map $\Lambda$, and is then included in $\mathsf{HOpeners}$ (for some $\Lambda$). Upon receiving $\mathsf{LCOpen}$ for some linear map $\Lambda$ from $t{+}1$ honest players, i.e. when $|\mathsf{HOpeners}| \geq t+1$, and if $\mathcal{F}_{\mathsf{LSS}}$ has stored all the inputs appearing with nonzero coefficient in $\Lambda$, then it eventually delivers its evaluation.

#### 3.2.2 Implementation of $\mathcal{F}_{\mathsf{LSS}}$.

Our goal is to build a protocol $\Pi_{\mathsf{LSS}}$ that implements $\mathcal{F}_{\mathsf{LSS}}$, i.e. that enables, after the unique round of broadcast, players to have a common view on a set of shared secrets. Subsequently, they can perform the threshold opening of the evaluation of *any* linear map over these shared secrets, using one step of all-to-all asynchronous peer-to-peer messages. $\Pi_{\mathsf{LSS}}$ is detailed in Fig. 12 in Appendix E.2. Overall it can be outlined as follows.

First, each player generates then registers its public key to $\mathsf{bPKI}$. To send a secret $s$ to $\mathcal{F}_{\mathsf{LSS}}$, i.e., to share it, the first step is to generate a (n,t)-linear secret sharing of $s$, yielding shares $\{s^{(i)}\}_{i\in[n]}$. Encrypt each share $s^{(i)}$ under $P_i$'s public key. The resulting $n$-sized vector of ciphertexts is called a public verifiable secret sharing ($\mathsf{PVSS}$, see Def 13). The term *verifiable* is used because when compiling to fully malicious security, NIZKs of knowledge of plaintexts and of a degree $t$ polynomial are appended. For state-of-the-art implementations, see [73,83].

To open a linear map $\Lambda$ over a set of shared secrets $(s_j)_j$: every $P_i$ decrypts its encrypted shares $s_j^{(i)}$, then evaluates $\Lambda$ on them. By linearity of the $(n,t)$-$\mathsf{LSS}$ scheme, the result is a partial opening share $z^{(i)}$ of $\Lambda((s_j)_j)$. Then, it sends $z^{(i)}$ to all via asynchronous P2P channels. Finally, from any $t+1$ partial opening shares, the desired linear combination $\Lambda((s_j)_j)$ is efficiently reconstructible.

We provide more details about the $\mathsf{LSS}$ instantiation in Appendix E and prove, in Proposition 15, that the construction does UC-implement $\mathcal{F}_{\mathsf{LSS}}$.

## 4 Share&Shrink: DKG & Encrypted Input Distribution in 1 BC + 1 Async. P2P

We follow the model and the formalism introduced in Sections 2 and 3 and assume a linear homomorphic encryption scheme as in Definition 3, represented by a tuple of PPT algorithms $\ell\text{-}\mathsf{HE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. Recall in particular that they are built from public fixed linear maps $\Lambda^{a}_{\mathsf{EKeyGen}}, \Lambda^{\mathsf{ek}}_{\mathsf{Enc}}, \Lambda^{\mathsf{c}}_{\mathsf{Dec}}$[6].

We now describe a protocol in the $\mathcal{F}_{\mathsf{LSS}}$-hybrid model, called Share&Shrink and formalized in Fig. 1, which performs a "$\mathsf{DKG}$ & Encrypted Input distribution" in 1 BC+asynch P2P. It allows players to obtain all-at-once: (i) a common threshold encryption key $\mathsf{ek}$[7], (ii) a secret-shared decryption key $\mathsf{sk}$; (formally: in $\mathcal{F}_{\mathsf{LSS}}$); and (iii) a common view on $\ell\text{-}\mathsf{HE}$ ciphertexts of the inputs encrypted

[6] And possibly an extra $\Lambda_{\mathsf{RlkGen}}$ if the specific scheme requires a relinearization key.

[7] Possibly along with a common relinearization key $\mathbf{rlk}$, e.g. in the case of $\ell\text{-}\mathsf{BFV}$.

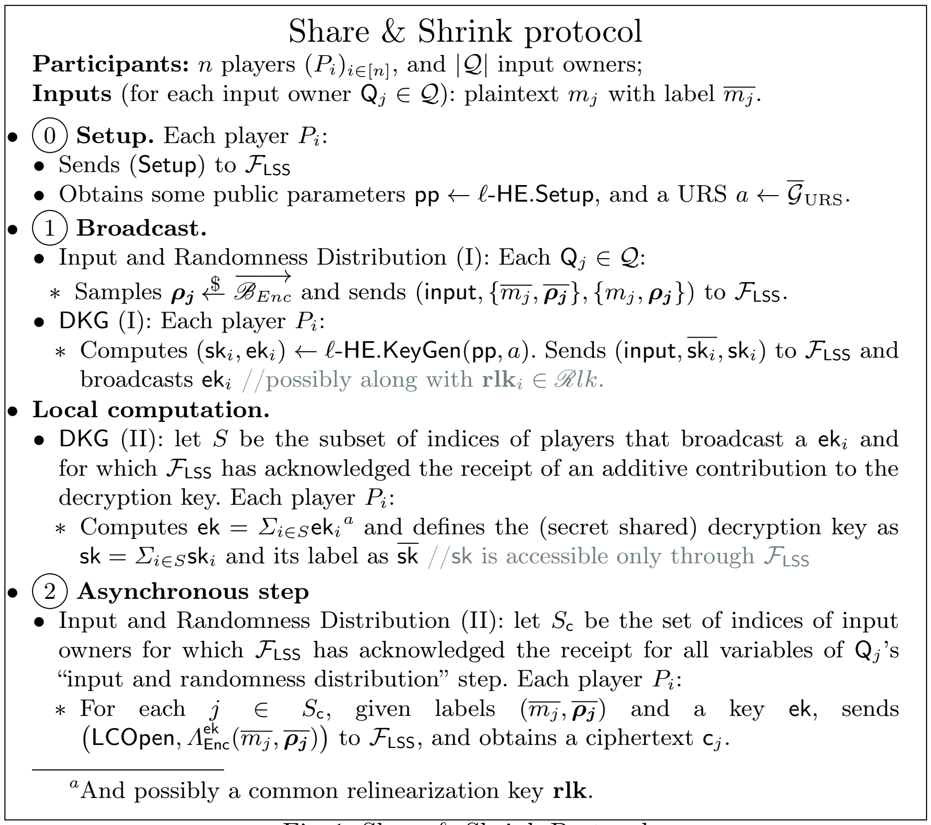
under $\mathsf{ek}$. The challenge is that the input-owners have access to the broadcast to distribute their inputs *only before* the encryption key $\mathsf{ek}$ is known!

$\textcircled{0}$ **Setup:** Players receive some public parameters $\mathsf{pp}$ and a URS $a$.
**In parallel**, they $\mathsf{Setup}$ $\mathcal{F}_{\mathsf{LSS}}$ (concretely: publish $\mathsf{PKE}$ encryption keys).

$\textcircled{1}$ **Broadcast:** Input-owners send (**Share**) their inputs $m$ & encryption randomnesses $\boldsymbol{\rho}_{\mathsf{Enc}}$ to $\mathcal{F}_{\mathsf{LSS}}$ (concretely: broadcast $\mathsf{PVSS}$s of them).
***In parallel***, based on public parameters $\mathsf{pp}$ and URS $a$, players generate additive contributions $\mathsf{sk}_i$ to the $\ell$-$\mathsf{HE}$ decryption key, which they input to $\mathcal{F}_{\mathsf{LSS}}$; and $\mathsf{ek}_i$ to the threshold encryption key (using randomness $\rho_i^{\text{key}}$), which they broadcast (along with possibly extra material, e.g., a relinearization key for $\ell$-$\mathsf{BFV}$).

- **Local computation:** Then, each player locally computes the common threshold encryption key $\mathsf{ek}$[7], out of the contributions from the subset $S$ of indices of non-aborting players. Precisely, $S \subset [n]$ are indices of those that broadcast correct material (including the $\mathsf{PVSS}$, as captured by $\mathcal{F}_{\mathsf{LSS}}$). By linearity of the $\ell$-$\mathsf{HE}$ scheme, this key is merely the sum of the contributions from $S$:

$$\mathsf{ek} = \Sigma_{i\in S}\Lambda^{a}_{\mathsf{EKeyGen}}(\mathsf{sk}_i, \rho_i^{\text{key}}) = \Lambda^{a}_{\mathsf{EKeyGen}}(\Sigma_{i\in S}(\mathsf{sk}_i, \rho_i^{\text{key}})) \tag{1}$$



## Share & Shrink protocol

**Participants:** $n$ players $(P_i)_{i\in[n]}$, and $|\mathcal{Q}|$ input owners;
**Inputs** (for each input owner $\mathsf{Q}_j \in \mathcal{Q}$): plaintext $m_j$ with label $\overline{m_j}$.

- $\textcircled{0}$ **Setup.** Each player $P_i$:
  - Sends $(\mathsf{Setup})$ to $\mathcal{F}_{\mathsf{LSS}}$
  - Obtains some public parameters $\mathsf{pp} \leftarrow \ell\text{-}\mathsf{HE.Setup}$, and a URS $a \leftarrow \overline{\mathcal{G}}_{\mathrm{URS}}$.
- $\textcircled{1}$ **Broadcast.**
  - Input and Randomness Distribution (I): Each $\mathsf{Q}_j \in \mathcal{Q}$:
    - * Samples $\boldsymbol{\rho_j} \overset{\$}{\leftarrow} \overrightarrow{\mathscr{B}_{Enc}}$ and sends $(\mathsf{input}, \{\overline{m_j}, \overline{\boldsymbol{\rho_j}}\}, \{m_j, \boldsymbol{\rho_j}\})$ to $\mathcal{F}_{\mathsf{LSS}}$.
  - $\mathsf{DKG}$ (I): Each player $P_i$:
    - * Computes $(\mathsf{sk}_i, \mathsf{ek}_i) \leftarrow \ell\text{-}\mathsf{HE.KeyGen}(\mathsf{pp}, a)$. Sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSS}}$ and broadcasts $\mathsf{ek}_i$ //possibly along with $\mathbf{rlk}_i \in \mathscr{R}lk$.
- **Local computation.**
  - $\mathsf{DKG}$ (II): let $S$ be the subset of indices of players that broadcast a $\mathsf{ek}_i$ and for which $\mathcal{F}_{\mathsf{LSS}}$ has acknowledged the receipt of an additive contribution to the decryption key. Each player $P_i$:
    - * Computes $\mathsf{ek} = \Sigma_{i\in S}\mathsf{ek}_i$[a] and defines the (secret shared) decryption key as $\mathsf{sk} = \Sigma_{i\in S}\mathsf{sk}_i$ and its label as $\overline{\mathsf{sk}}$ //$\mathsf{sk}$ is accessible only through $\mathcal{F}_{\mathsf{LSS}}$
- $\textcircled{2}$ **Asynchronous step**
  - Input and Randomness Distribution (II): let $S_{\mathsf{c}}$ be the set of indices of input owners for which $\mathcal{F}_{\mathsf{LSS}}$ has acknowledged the receipt for all variables of $\mathsf{Q}_j$'s "input and randomness distribution" step. Each player $P_i$:
    - * For each $j \in S_{\mathsf{c}}$, given labels $(\overline{m_j}, \overline{\boldsymbol{\rho_j}})$ and a key $\mathsf{ek}$, sends $\big(\mathsf{LCOpen}, \Lambda^{\mathsf{ek}}_{\mathsf{Enc}}(\overline{m_j}, \overline{\boldsymbol{\rho_j}})\big)$ to $\mathcal{F}_{\mathsf{LSS}}$, and obtains a ciphertext $\mathsf{c}_j$.

[a] And possibly a common relinearization key $\mathbf{rlk}$.

Fig. 1: Share & Shrink Protocol

**② Asynchronous step (Shrinking of the inputs):** Finally, players jointly compute threshold encryptions under $\mathsf{ek}$ of the shared inputs $m_j$. Concretely, thanks to linearity of the $\ell$-$\mathsf{HE}$ scheme, this can be done as threshold openings of the images of the $m_j$ and of shared encryption randomnesses $\boldsymbol{\rho}_{\mathbf{Enc},j}$ by the linear map $\Lambda^{\mathsf{ek}}_{\mathsf{Enc}}$, in one step of asynchronous P2P messages, as recalled in Section 3.2.

The outlined protocol differs fundamentally from related threshold-$\mathsf{FHE}$ works [88,98] which follow the pattern $\mathsf{DKG}$-then-Input-distribution, since using Share& Shrink the Input Distribution is done without knowing a common key, which allows us *i*) to reduce the number of broadcast rounds to just one while guaranteeing output delivery, and *ii*) to keep short ciphertexts compared to [17].

## 5 MPC Protocol $\Pi^{\mathcal{F}_{\mathsf{LSS}}}_{\mathsf{MPC}}$

We now present our delegated MPC protocol that operates in one single initial $\mathsf{BC}$ followed by asynchronous P2P messages, leveraging our novel Share&Shrink protocol introduced in Section 4. First, we discuss the computation itself. When using a scheme with fully homomorphic capabilities, specifically one that satisfies Definition 3, i.e. a $\ell$-$\mathsf{FHE}$ scheme like $\ell$-$\mathsf{BFV}$ (see Definition 4), players can locally evaluate a circuit on the encrypted inputs and produce an output ciphertext. While our primary focus is on evaluating circuits using $\mathsf{FHE}$, we also explore alternative approaches for generating this output. To formalize this, we introduce a generic evaluation protocol $\mathsf{Eval}$ in Section 5.1, to evaluate a circuit on $\ell$-$\mathsf{HE}$-encrypted ciphertexts in various ways. This protocol, possibly interactive over asynchronous channels, must be simulatable. We then review several known evaluation protocols that meet these criteria. Next in Section 5.2, we detail methods for threshold decryption, and finally describe our generic MPC protocol $\Pi^{\mathcal{F}_{\mathsf{LSS}}}_{\mathsf{MPC}}$ in the $\mathcal{F}_{\mathsf{LSS}}$-hybrid model in Section 5.3, which integrates these components.

### 5.1 (Asynchronous) Evaluation of a Circuit

Consider a linear homomorphic encryption scheme $\ell$-$\mathsf{HE}$ (Def 3), and players with input a set of ciphertexts $\{\mathsf{c}_j\}_{j\in[|\mathcal{Q}|]}$ of plaintexts $\{m_j\}_{j\in[|\mathcal{Q}|]}$ under a common threshold encryption key $\mathsf{ek}$. We assume that there exists an asynchronous evaluation protocol $\mathsf{Eval}$ for any arithmetic circuit $\mathrm{C} : (\mathscr{M} \cup \{\bot\})^{|\mathcal{Q}|} \to \mathscr{M}$ with $|\mathcal{Q}|$ input gates, that outputs a ciphertext of the evaluation. Formally, we require that $\forall\ \mathsf{pp}, \mathbf{a} \leftarrow \ell\text{-}\mathsf{HE.Setup}(1^\lambda)$, there exists a $\mathsf{DKG}$ in one $\mathsf{BC}$ that returns a threshold encryption key $\mathsf{ek}$ and, privately to the players, shares $\mathsf{sk}_i$ of the corresponding decryption key $\mathsf{sk}$; $\forall\,(m_j)_{j\in[|\mathcal{Q}|]}$, $\mathsf{c}_j \leftarrow \ell\text{-}\mathsf{HE.Enc}(\mathsf{pp}, \mathsf{ek}, m_j)$; then $\ell\text{-}\mathsf{HE.Dec}(\mathsf{sk}, \mathsf{Eval}(\mathrm{C}, \mathsf{c}_1, \ldots, \mathsf{c}_{|\mathcal{Q}|}, \mathsf{ek}^{8})) = \mathrm{C}(m_1, \ldots, m_{|\mathcal{Q}|})$. We also require $\mathsf{Eval}$ to be simulatable. In practice, it can be implemented in various ways, including:

- $\ell$-$\mathsf{FHE}$. When using a scheme with fully homomorphic capabilities as a particular kind of $\ell$-$\mathsf{HE}$ scheme (as for $\ell$-$\mathsf{BFV}$, see Section 3.1.1), there exists a built-in non-interactive function $\mathsf{Eval}$, simulatable from the knowledge of the threshold encryption key and of the relinearization key.

[8] Possibly along an extra relinearization key.

- [46]. Choudhury et al. proposed, using pre-processed masks, a protocol for evaluating a circuit based on an efficient interactive multi-party bootstrapping protocol for an encryption scheme that supports a limited number of homomorphic operations. Here, players open threshold decryptions of masked intermediary evaluations; and the simulator simulates the opening of a random value.
- [21]. From a common view of $\ell$-$\mathsf{HE}$ encrypted inputs and decryption key shares assigned to each player, one can apply the asynchronous CDN-like[51] protocol of [21] to evaluate a circuit using a $\ell$-$\mathsf{HE}$ scheme with only partial homomorphic properties. Players open threshold decryptions of masked intermediary results. Since some masks can be inferred from each other by adversarially-chosen (but extractable) offsets, the simulator takes extra care.

## 5.2 Threshold Decryption.

$\ell$-$\mathsf{HE}$ decryption can be seen as a two steps process: (1) the interactive opening of a linear map $\Lambda^{\mathsf{c}}_{\mathsf{Dec}}$ applied to the (secret shared) decryption key $\mathsf{sk}$, with public coefficients equal to the ciphertext $\mathsf{c}$, (2) followed by local computation of a non-linear decoding function $\Omega_{\mathsf{Dec}}$. However, a direct adaptation from this decryption to the threshold setting is not straightforward when $\Omega_{\mathsf{Dec}}$ is nontrivial. This is notably the case for the $\mathsf{RLWE}$-based schemes, such as $\ell$-$\mathsf{BFV}$, that we now consider in the remainder of this section.

In this setting, the output $\mu_{\mathrm{dec}}$ of (1) allows recovering the plaintext, but also reveals a small *decryption noise* term that depends on the given ciphertext and the decryption key. Asharov et al. [14] demonstrated that this noisy output of (1) reveals too much information about the decryption key. To prevent information leakage, [14] introduced the technique of adding additional noise to $\mu_{\mathrm{dec}}$ before it can be reconstructed. This "smudging" noise $e_{\mathrm{sm}}$ is, roughly, sampled uniformly in some large enough interval $[-B_{sm}, B_{sm}]$. Now consider an arithmetic circuit C, and denote $B_{\mathrm{C}}$ the upper-bound on the decryption noise of a ciphertext after evaluation of C. The choice of $B_{sm}$ is crucial for both the security and correctness of our MPC protocol. This translates into the following two requirements:

1. First, the output of (1) $\mu_{\mathrm{dec}} = \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk})$ must be statistically close to the (scaled) plaintext circuit evaluation $\Delta \cdot y$. Then, some level of noise $B_{sm}$ should exist so that adding a uniform noise $e_{\mathrm{sm}} \in [-B_{sm}, B_{sm}]$ to both $\mu_{\mathrm{dec}}$ and $\Delta \cdot y$, makes them indistinguishable, while leaving correct the result: $y = \Omega_{\mathsf{Dec}}(\mu_{\mathrm{dec}} + e_{\mathrm{sm}})$. Lemma 20 imposes a level of noise high enough so that $B_{\mathrm{C}}/B_{sm} \leqslant \mathrm{negl}(\lambda)$.
2. Second, the correctness requirement imposes that $B_{\mathrm{C}}$ added with this smudging noise stays small, e.g. in the case of $\ell$-$\mathsf{BFV}$, we want that $B_{\mathrm{C}} + n \cdot B_{sm} \leq \Delta/2$.

Different methods exist for distributively opening $\mu_{\mathrm{dec}}$ added with such noise:

- Most previous works [14,25,88] let each player $P_i$ locally sample a smudging noise $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$, multiply it by $n!^2$, then add it to its opening share of $\mu_{\mathrm{dec}}$, which it sends. The reason for multiplying by $n!^2$ is to clear-out the denominators of the Lagrange coefficients applied during reconstruction (see [7,25]) when Shamir $\mathsf{LSS}$ is used. This introduces an overhead of $n.n!^3$ on the ciphertext modulus $q$, and therefore, a $n\times$ blowup of the ciphertext size.

- To keep the ciphertext size small, an improved method [61,111] has been proposed, in which players do not anymore smudge their opening share of $\mu_{\text{dec}}$, but open all at once the decryption $\mu_{\text{dec}}$ and a common shared noise $e_{\text{sm}}$, i.e., the linear map defined as:

$$\Lambda^{\mathsf{c}}_{\mathsf{Dec}+sm} : (\mathsf{sk}, e_{\text{sm}}) \to \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk}) + e_{\text{sm}} \tag{2}$$

  The distributed generation of the noise (one for every circuit to be evaluated) is simply by adding secret-shared contributions $e_{\text{sm},i}$ sampled in $[-B_{sm}, B_{sm}]$. As a result, the correctness constraint now imposes that the ciphertext expansion factor $\Delta$ has a dependency in $n$ which is only linear, instead of cubic previously.

**Protocol $\Pi^{\mathcal{F}_{\mathsf{LSS}}}_{\mathsf{MPC}}$**

**Participants:** $n$ players $\{P_i\}_{i\in[n]}$, and $|\mathcal{Q}|$ input-owners;
**Inputs** (for each $\mathsf{Q}_j \in \mathcal{Q}$): a plaintext $m_j$ with label $\overline{m_j}$.

- **Share & Shrink.** Players and Input-Owners play the Share&Shrink protocol of Fig. 1 (with the added smudging noise sharing by players), in which each input-owner $\mathsf{Q}_j \in \mathcal{Q}$ has an input $m_j$.
  Denote $S_{\mathsf{c}} \subset [|\mathcal{Q}|]$ the indices of input-owners (resp. $S \subset [n]$ of players), for which no instance returned $\perp$.
  After Share&Shrink, players have a common view on i) a threshold encryption key $\mathsf{ek}^a$, ii) a set of ciphertexts $\{\mathsf{c}_j\}_{j\in S_{\mathsf{c}}}$ encrypted under $\mathsf{ek}$, iii) a shared decryption key $\mathsf{sk}$ in $\mathcal{F}_{\mathsf{LSS}}$, as well as, iv) a shared smudging noise $e_{\text{sm}}$ in $\mathcal{F}_{\mathsf{LSS}}$.
- **Evaluation.** To evaluate a circuit C, each player runs $\mathsf{c} \leftarrow \mathsf{Eval}(\mathrm{C}, \{\mathsf{c}_j\}_{j\in S_{\mathsf{c}}}, \mathsf{ek}^a)$.
- **Threshold Decryption.** Players play the Threshold Decryption protocol of Section 5.2 with input the ciphertext $\mathsf{c}$ and the shared decryption key $\mathsf{sk}$ in $\mathcal{F}_{\mathsf{LSS}}$. They output the plaintext $m$ obtained.

[a] Possibly along with a common relinearization key **rlk**.

Fig. 2: MPC protocol $\Pi^{\mathcal{F}_{\mathsf{LSS}}}_{\mathsf{MPC}}$

### 5.3 Protocol $\Pi^{\mathcal{F}_{\mathsf{LSS}}}_{\mathsf{MPC}}$ in $(\mathcal{F}_{\mathsf{LSS}}, \mathsf{BC})$-hybrid model, with external resource $\overline{\mathcal{G}}_{\mathbf{URS}}$

Consider a $\ell$-HE scheme satisfying Def 3. Specifically, in this section we consider the RLWE-based $\ell$-BFV scheme defined in Section 3.1.1 as an example (although other $\ell$-HE schemes could have been used). We show in Fig. 2 how to build a delegated MPC protocol $\Pi^{\mathcal{F}_{\mathsf{LSS}}}_{\mathsf{MPC}}$ from the Share&Shrink protocol introduced in Section 4, an evaluation algorithm Eval as discussed in Section 5.1, and a threshold decryption protocol as detailed in Section 5.2.

***Sketch of UC Proof of Theorem 2.*** Following [32], a protocol *UC-implements* the ideal functionality $\mathcal{F}_{\mathrm{C}}$ if there exists a PPT *simulator* Sim, such that for every PPT *environment* Env, that controls a “dummy” adversary $\mathcal{A}$ *and* which may send inputs to honest participants *and* is forwarded their outputs in real-time, has negligible advantage in distinguishing between the following two executions:

- REAL$_{\mathcal{A}}$: an actual execution of the MPC protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$, with adversary $\mathcal{A}$ fully controlled by $\mathsf{Env}$, and functionalities $\mathcal{F}_{\mathsf{LSS}}$, $\mathsf{BC}$;
- IDEAL$_{\mathcal{F}_{\mathrm{C}},\mathsf{Sim}}$: an execution denoted as *ideal*, where $\mathsf{Sim}$ interacts with $\mathsf{Env}$ on behalf of $\mathcal{A}$. On the other side, $\mathsf{Sim}$ interacts with $\mathcal{F}_{\mathrm{C}}$ on behalf of the corrupt participants, and also of $\mathcal{A}$. The honest $\mathsf{dummy}$ participants are connected to $\mathsf{Env}$ as in a real execution. But on the other side, they only interact with $\mathcal{F}_{\mathrm{C}}$.

In more detail, $\mathsf{Sim}$ initiates in its head, sets $\mathscr{P}$ of $n$ players and $\mathcal{Q}$ of inputs-owners, and may initially receive corruption requests from $\mathsf{Env}$ for arbitrarily many owners and up to $t$ players, indexed by $\mathcal{I}$. It simulates functionalities $(\mathsf{BC}, \mathcal{F}_{\mathsf{LSS}})$ following a correct behavior, apart from the value returned by $\mathcal{F}_{\mathsf{LSS}}$ in the *Output Computation* step. $\mathsf{Sim}$ does the following:

- **Setup.** Simulates the setup of $\mathcal{F}_{\mathsf{LSS}}$ and retrieves the URS $a$ from $\overline{\mathcal{G}}_{\mathrm{URS}}$ and sends it to all on behalf of $\overline{\mathcal{G}}_{\mathrm{URS}}$.
- **Distributed Key and Smudging Noise Generation:** Simulates a correct behavior of $\mathcal{F}_{\mathsf{LSS}}$. For every simulated honest player $(P_i)_{i\in\mathcal{H}}$:
  - ⋆ Samples $\mathsf{sk}_i \xleftarrow{\$} \mathscr{X}_q$, and sends it to $\mathcal{F}_{\mathsf{LSS}}$.
  - ⋆ Samples $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$, and sends it to $\mathcal{F}_{\mathsf{LSS}}$.
  - ⋆ Samples $b_i \xleftarrow{\$} U(R_q)$, and sends $\mathsf{ek}_i = (b_i, a)$ over $\mathsf{BC}^{P_i}$[9].
- **Input &Randomness Distribution:** Simulates correct behavior of $\mathcal{F}_{\mathsf{LSS}}$, and:
  - ⋆ $\forall$ simulated honest input-owners $\mathsf{Q}_j \in \mathcal{Q}$: sets $\widetilde{m_j} := 0$ and samples $\boldsymbol{\rho}_{\mathbf{Enc},\boldsymbol{j}} \xleftarrow{\$} \overrightarrow{\mathscr{B}_{Enc}}$. Then sends $(\mathsf{input}, \{\overline{\widetilde{m_j}, \boldsymbol{\rho}_{\mathbf{Enc},\boldsymbol{j}}}\}, \{\widetilde{m_j}, \boldsymbol{\rho}_{\mathbf{Enc},\boldsymbol{j}}\})$ to $\mathcal{F}_{\mathsf{LSS}}$.
  - ⋆ $\forall$ simulated corrupt $\mathsf{Q}_j \in \mathcal{Q}$, upon receiving $(\mathsf{c}_j)$ from $\mathsf{Env}$, uses $\overline{\mathsf{sk}}$ to decrypt $\mathsf{c}_j$ into $m_j$ and sets $\widetilde{m_j} := 0$ if $m_j = \perp$ or $\widetilde{m_j} := m_j$ otherwise, and sends $(\mathsf{input}, \mathsf{Q}_j, m_j)$ to $\mathcal{F}_{\mathrm{C}}$.

  As in the protocol, $\mathsf{Sim}$ sets $S_{\mathsf{c}} \subset [|\mathcal{Q}|]$ the indices of input-owners, resp. $S \subset [n]$ of the players, for which no instance returned $\perp$.
- **Threshold Encryption:** Simulates correct behaviors to compute key $\mathsf{ek} := (\sum_{i\in S} b_i, a)$[9]; to make $\mathcal{F}_{\mathsf{LSS}}$, $\forall j \in S_{\mathsf{c}}$, eventually output: $\big(\Lambda_{\mathsf{Enc}}^{\mathsf{ek}}(\widetilde{m}_j, \boldsymbol{\rho}_{\mathbf{Enc},\boldsymbol{j}})\big)$.
- **Circuit Evaluation:** Simulates $\mathsf{Eval}$ as discussed in Section 5.1.
- **Output Computation:** Upon being leaked the evaluation $y$ from $\mathcal{F}_{\mathrm{C}}$, where by definition $y = \mathrm{C}(\{m_i\}_{i\in S_c})$, then $\mathsf{Sim}$ simulates the following incorrect behavior:
  - ⋆ $\mathcal{F}_{\mathsf{LSS}}$ outputs $\left(\Lambda_{\mathsf{Dec}+sm}^{\mathsf{c}},\ \Delta y + \Sigma_{j\in S} e_{\mathrm{sm},j}\right)$ to $\mathcal{L}$. (See Equation (2))

Recall that the view of $\mathsf{Env}$ consists of its interactions with $\mathcal{A}/\mathsf{Sim}$, and of the outputs of the actual honest players. To show indistinguishability, we go through a series of hybrid games starting from the real execution. The first hybrid, $\mathsf{Hybrid}^{\mathcal{F}_{\mathsf{LSS}}}$, modifies the behavior of $\mathcal{F}_{\mathsf{LSS}}$ in the *Output Computation* step, to incorrectly return a simulated output $\mu^{\mathsf{Sim}} := \Delta.y + \Sigma_{j\in S} e_{\mathrm{sm},j}$, where $y := \mathrm{C}((m_j)_{j\in S_c})$ is the evaluation of the circuit on the actual inputs. Indistinguishability follows from the "smudging" Lemma 20, as detailed in Appendix F. In the second hybrid, $\mathsf{Hybrid}^{\mathsf{key}}$, the additive contributions $(b_i)_{i\in\mathcal{H}}$ of honest players to the encryption key[9] are replaced by a sample in $U(R_q)$. Then, in

[9]And possibly a relinearization key $\mathbf{rlk}_i$ depending on the chosen $\mathsf{Eval}$.

$\mathsf{Hybrid}^{\mathsf{Eval}}$, the *Evaluation* is simulated using techniques that depend on the $\mathsf{Eval}$ instantiation as discussed in Section 5.1. Finally, $\mathsf{Hybrid}^{\mathsf{0Share}}$ replaces the input of simulated honest input-owners by 0. By this point, honest players' decryption keys are no longer in use. Furthermore, since honest players sample their contributions $\mathsf{ek}_i$ to the common threshold encryption key independently, we can assume without loss of generality that corrupt contributions are generated after seeing the honest ones. We can thus apply the classical "IND-CPA under Joint Keys" Lemma [14], adapted in [111, §D] for RLWE-based schemes.

In conclusion, we arrived at a view produced by a machine that interacts only with $\mathsf{Env}$ and $\mathcal{F}_{\mathrm{C}}$. We discuss malicious security in Appendix F.2.

## 6 Proof of Theorem 1

The MPC protocol of [17] proceeds in 3 rounds of broadcasts described as follows:

**Setup:** Players first run a distributed setup to generate public/secret key pairs and contributions to public parameters and to the uniform random strings. These public contributions and the public keys are then broadcast.

**Input Distribution:** Let $S_1 \subseteq [n]$ be the set of indices of players that sent a round 1 message. Players encrypt their inputs using a multikey-FHE scheme, then broadcast the ciphertexts along with PVSSs of their decryption keys.

**Evaluation and Threshold Decryption:** Let $S_2 \subseteq S_1$ be the set of indices of players that sent a round 2 message. Players then evaluate a circuit C on the encrypted inputs and perform a threshold decryption in one BC.

Now let us show how to (*i*) obtain termination in 1 BC-then-1 step of asynchronous P2P messages; and (*ii*) to allow inputs from external owners.

We first note that the first broadcast round is input-independent. Hence, we replace their round 1 by publication of public keys and URSs on bPKI. The modification to obtain (*ii*) is simply to allow any external input-owner to perform their round 2 broadcast, directed to the $n$ players. In more detail, consider the subset $S_2 \subset [|\mathcal{Q}|]$ of owners which correctly shared their inputs and shares of multikey-FHE keys (in [17], $S_2$ is instead a subset of players). From these secret shared keys, it is described in [17] how honest players in round-3 can *emulate* multikey-FHE reconstruction, as if it would have been performed by members of $S_2$ themselves (their participation *as input-owners* is not needed anymore).

Finally, to obtain (i), we need to replace the broadcast in their third round with asynchronous P2P messages, such that the output after round 3 is unchanged: players still obtain the same common threshold decryption. We remark that their round 3 can be performed over asynchronous P2P channels, since the computation step performed by a player $P$ at the end of round 3 is: *choose any* set $\mathcal{U}$ of $t+1$ valid decryption shares received in round-3 messages, then apply the local threshold decryption algorithm on them. So this step does not depend on whether all round-3 messages from honest players were received by $P$ or not, but it may well be that $t$ out-of-the $t+1$ valid decryption shares chosen by $P$, originate from corrupt players, without impacting decryption correctness.

## 7 Experimental Evaluation

We present a proof-of-concept implementation that shows that our approach leads to practical results. We instantiate Share&Shrink from the $\ell$-BFV scheme presented in Section 3.1.1 and, for a fair comparison, we compare it to our Theorem 1 instantiated from an efficient multikey scheme [89] based on BFV, denoted MK-BFV. We consider inputs in $R_k$ with $\log k = 16$, and parameters that achieve at least 128-bit of security level according to LWE-estimator [8]:

- For MK-BFV, we use the candidate parameter set described in [89, Table 2], recalled in Table 3, that supports circuits of depth 6.
- For Share&Shrink instantiated from $\ell$-BFV, we use the same parameters as for MK-BFV in our specific single-key case.

We consider the following delegated setting:

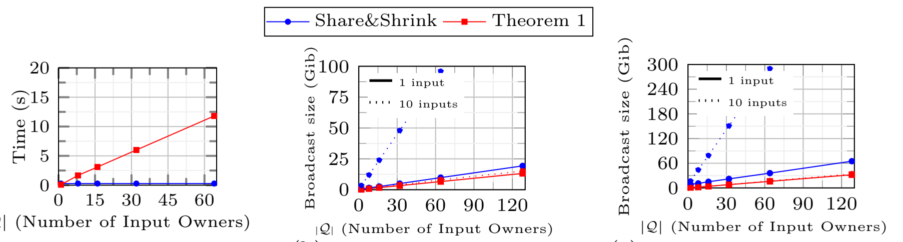
- A number of input-owners ranging from 1 to 128, each owning inputs in $R_k$,
- A cluster of $n = 11$ or 50 players to perform the computation.

All experiments were performed on a MacBookPro with a 3.1GHz Intel i5 processor, using Lattigo[10] as well as the implementation of MK-BFV done by [89][11].

### 7.1 Experiment#1: Mult gate computation

We first compare in Fig. 3a the running time of a Multiplication between two ciphertexts for different numbers of input-owners ranging from 1 to 64 (we can run MK-BFV with up to 64 input-owners, but RAM prevents scaling to 128).

Overall, this verifies that the running time is almost linear with the number of input-owners when using a multikey scheme, while Share&Shrink reduces this duration to a small constant, independent of $|\mathcal{Q}|$ as expected.



(a) Comparison of median Mult gate computation times of the Theorem 1 instantiated from the MK-BFV scheme [89] and Share&Shrink instantiated from $\ell$-BFV.

(b) Comparison of the total broadcast size between the Theorem 1 instantiated from MK-BFV [89] and Share&Shrink for 1 and 10 inputs per owner, for $n = 11$.

(c) Comparison of the total broadcast size between the Theorem 1 instantiated from MK-BFV [89] and Share&Shrink for 1 and 10 inputs per owner, for $n = 50$.

Fig. 3: Experimental Results

### 7.2 Experiment #2: Broadcast size

We compare in Fig. 3b the total broadcast size for $n = 11$ and a different number of input-owners ranging from 1 to 128 that each owns one or ten inputs in $R_k$, when using MK-BFV or Share&Shrink instantiated from $\ell$-BFV. It comprises:

[10]https://github.com/tuneinsight/lattigo

[11]https://github.com/SNUCP/MKHE-KKLSS/

- An input-independent part that consists of the encryption and relinearization keys as well as a $\mathsf{PVSS}$ of some decryption key.
- An input-dependent part that consists either of $\mathsf{MK\text{-}BFV}$ ciphertexts or of $\mathsf{PVSS}$s of inputs and randomnesses $e_0^{(\mathsf{Enc})}, e_1^{(\mathsf{Enc})}$, $u$ over $R_q$ as required in Fig. 1.

Let us first provide some details about Fig. 3b. For the $\mathsf{PVSS}$, we use the class-group-based public-key encryption scheme recently employed in [84], while omitting zero-knowledge proofs in our semi-malicious corruption model. For $n$ players, the total bit-length of ciphertexts is $1752 \cdot (n+1)$ bits for a 256-bit plaintext, resulting in an asymptotic ciphertext expansion factor of 6.8.
Following Section 3.1.1 and denoting |#inputs| the number of inputs per input-owner, the broadcast for Share&Shrink instantiated from $\ell$-$\mathsf{BFV}$ is of size:

$$n \cdot |(3 \cdot l \cdot |R_q| + \mathsf{PVSS}(|R_q|))| + |\mathcal{Q}| \cdot |\#\text{inputs}| \cdot |\mathsf{PVSS}(3 \cdot |R_q| + |R_k|)|. \tag{3}$$

For Theorem 1 instantiated from $\mathsf{MK\text{-}BFV}$, the broadcast is of size:

$$|\mathcal{Q}| \cdot (4 \cdot l \cdot |R_q| + \mathsf{PVSS}(|R_q|)) + |\mathcal{Q}| \cdot |\#\text{inputs}| \cdot 2|R_q|. \tag{4}$$

In Fig. 3b, we remark that for a small number of inputs per owner, Theorem 1 requires broadcasting a comparable amount of data than Share&Shrink. However, the greater the number of inputs per owner, the more the RHS of both Equations (3) and (4) dominates. Consequently, Share&Shrink requires a larger broadcast since a greater number of n-sized $\mathsf{PVSS}$s are sent, which are larger than a $\mathsf{MK\text{-}BFV}$ ciphertext, whose size is $2|R_q|$.

***Impact of the number $n$ of players:*** For completeness, Fig. 3c illustrates the broadcast size for different numbers of input-owners when $n = 50$. In this case, for the $\mathsf{PVSS}$, we use the LWE-based encryption scheme employed in [73]. For $n = 50$, this results in an amortized ciphertext expansion factor of 4.8. Overall, the results exhibit a similar dynamic to the $n = 11$ case shown in Fig. 3b.

In practice, we believe that it is reasonable to consider a small number $n$ of powerful computation players and a very large number of resource-constrained input-owners sending few inputs.

## 8 Impossibility of 1-Broadcast-then-Asynchronous MPC

“Secure channels" is the upgrade of $\mathcal{F}_{\mathrm{AT}}$ which does not leak the content of messages to $\mathcal{A}$. The functionality of simultaneous broadcast ([44]), denoted $\mathsf{SB}$, is parametrized by two senders $P_1$ and $P_n$, and returns to all players a pair $(x_1, x_n)$ such that, for $i \in \{1, n\}$, $x_i$ is the input of $P_i$ if it is honest.

**Theorem 5.** *Consider a hybrid network in which players are* initially *given access to one single round of: synchronous pairwise secure channels and broadcast (non-simultaneous, i.e., the adversary observes the honest broadcasts before rushing the ones from corrupt players);* then: *only access to pairwise secure channels with guaranteed eventual delivery. Then for any $t \geq 3$ and $n \leq 3t - 4$ there is no computationally secure* $\mathsf{SB}$ *protocol. The impossibility holds given any public setup, i.e., a possibly structured random string.*

*Proof.* We assume such a protocol for $n = 3t - 4$. We construct an adversary $\mathcal{A}$ which corrupts $P_n$ but not $P_1$, and still manages that the second output: $x_n$, will be *correlated* with the input of the honest $P_1$. However, in an ideal execution, there is no link between $\mathcal{A}$ and $P_1$ until $\mathsf{SB}$ reveals $(x_1, x_n)$. Thus, $\mathcal{A}$'s influence on the outcome is unachievable in the ideal execution, hence a contradiction.

We define the subsets $\mathcal{Q} := \{P_1, \dots, P_{n-t=2t-4}\}$ and $\mathcal{Q}' := \{P_t, \dots, P_{n-1=3t-5}\}$, denoted as "quorums". The $3t - 4$ comes from the following tight constraints: We managed to have $|\mathcal{Q}| = |\mathcal{Q}'| = n - t$, so that in our proof, for each of these quorums, when all its members behave honestly and do not hear from the outside, then its honest members must output in a finite number of steps. $\mathcal{A}$ corrupts a well-chosen player $P_i \in \mathcal{Q}$, which we will detail. $\mathcal{A}$ selects a player $P_j$ in $\mathcal{Q}'$ at random and corrupts it. $\mathcal{A}$ corrupts the remaining players in $\mathcal{Q} \cap \mathcal{Q}'$, which it can do since they are at most $t - 3$, reaching a total of at most $t$ corruptions.

In the first round, all players act honestly, except $P_n$, which we will detail. Then in the asynchronous phase, $P_n$ is silent. $\mathcal{A}$ cuts the network in two, i.e.: $\mathcal{A}$ does not deliver {any message from honest players in $\mathcal{Q}$ to honest players in $\mathcal{Q}'$}, nor {any message from honest players in $\mathcal{Q}'$ to honest players in $\mathcal{Q}$}. $\mathcal{Q} \cap \mathcal{Q}'$ behaved honestly so far, thus it is meaningful to define the internal states they would have if honest. $\mathcal{A}$ makes a copy of these $\mathsf{States}(\mathcal{Q} \cap \mathcal{Q}')$.

***$\mathcal{A}$ first "probes" the value of $x_1$ as follows.*** It "freezes" $\mathcal{Q}$, i.e., it does not deliver any message to honest players in $\mathcal{Q}$. $\mathcal{A}$ makes the corrupt players in $\mathcal{Q}'$ play honestly, and has messages in $\mathcal{Q}'$ delivered in round-robin order. Thus, the view of honest players in $\mathcal{Q}'$ is indistinguishable from an execution in which {$\mathcal{Q}'$ were the *n–t* honest players, while the remaining ones would be silent}. Thus, there is a finite number of steps after which they all output. In particular, since $P_j$ plays the correct algorithm, it will learn the same output as honest players. With probability $1/(n-t)$, $P_j$ is the *first* in $\mathcal{Q}'$ to learn the output $(x_1, x_n)$.

***Upon $P_j$ being the first to learn, i.e., a "successful probing"***, then (i) $\mathcal{A}$ *learns* the actual input value $x_1$ of $P_1$; (ii) whereas *no* honest player has output yet, thus, the *actual* output of the protocol is *not yet* defined. In this case, $\mathcal{A}$ (immediately) "freezes" $\mathcal{Q}'$, i.e., delivers no more message to honest players in $\mathcal{Q}'$, so none will ever output, and reinitializes corrupt players in $\mathcal{Q} \cap \mathcal{Q}'$ back to their $\mathsf{States}(\mathcal{Q} \cap \mathcal{Q}')$ just after the first round.

The rest of the strategy is that $\mathcal{A}$ "un-freezes" the set of honest players in $\mathcal{Q}$, and has all players in $\mathcal{Q}$ play honestly the protocol, with their messages delivered in round-robin order. $\mathcal{A}$ will manage to correlate their second output to $x_1$, thereby breaking SB. To this end, it chooses carefully which "influential" player $P_i$ to corrupt, i.e., such that the first-round messages from $P_i$ to $\mathcal{Q}$ have a substantial impact on the second output. We carry out the details in Appendix I: they are an adaptation of [71, Thm 2] in our hybrid network setting. Roughly, such $P_i$ exists because otherwise, the first-round broadcasts would be enough to predict the protocol outputs, which would enable to break SB. In Appendix I, we pin down how a bulletin board PKI would defeat such a prediction (which was to be expected since SB is feasible in a hybrid network with a PKI).

□

## References

1. Google apple. exposure notification privacy-preserving analytics (enpa). Online: `https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf` (2021)
2. Compilation of public comments on multi-party threshold cryptography project. Online: `https://csrc.nist.gov/files/pubs/ir/8214/c/ipd/docs/nistir-8214c-ipd-public-feedback.pdf` (2023)
3. Delegated multi-key private matching for compute: Improving match rates and enabling adoption. Online: `https://research.facebook.com/blog/2023/1/delegated-multi-key-private-matching-for-compute-improving-match-rates-and-enabling-adoption/` (2023)
4. Abraham, I., Asharov, G., Patil, S., Patra, A.: Asymptotically free broadcast in constant expected time via packed vss. In: TCC (2022)
5. Abspoel, M., Cramer, R., Damgård, I., Escudero, D., Yuan, C.: Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In: TCC (2019)
6. Acharya, A., Hazay, C., Kolesnikov, V., Prabhakaran, M.: Scales: Mpc with small clients and larger ephemeral servers. TCC (2022)
7. Agrawal, S., Boyen, X., Vaikuntanathan, V., Voulgaris, P., Wee, H.: Functional encryption for threshold functions (or fuzzy ibe) from lattices. In: PKC (2012)
8. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic Encryption Standard (2021)
9. Alon, B., Naor, M., Omri, E., Stemmer, U.: Mpc for tech giants (gmpc): enabling gulliver and the lilliputians to cooperate amicably. CRYPTO (2024)
10. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: CRYPTO (2014)
11. Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Round-optimal secure multiparty computation with honest majority. In: CRYPTO (2018)
12. Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Two round information-theoretic mpc with malicious security. In: EUROCRYPT (2019)
13. Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Towards efficiency-preserving round compression in mpc. In: ASIACRYPT (2020)
14. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold fhe. In: EUROCRYPT (2012)
15. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing. STOC'91 (1991)
16. Badertscher, C., Hesse, J., Zikas, V.: On the (ir)replaceability of global setups, or how (not) to use a global ledger. In: TCC (2021)
17. Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: Secure mpc: Laziness leads to god. In: ASIACRYPT (2020)
18. Badrinarayanan, S., Miao, P., Mukherjee, P., Ravi, D.: On the round complexity of fully secure solitary mpc with honest majority. In: TCC (2023)
19. Barak, A., Hirt, M., Koskas, L., Lindell, Y.: An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants. In: CCS (2018)

20. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing. pp. 503–513 (1990)
21. Beerliová-Trubíniová, Z., Hirt, M., Nielsen, J.B.: Almost-asynchronous mpc with faulty minority. PODC'10 (2010), we refer to eprint 2008/416
22. Bell, J.H., Bonawitz, K.A., Gascón, A., Lepoint, T., Raykova, M.: Secure single-server aggregation with (poly) logarithmic overhead. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (2020)
23. Blum, E., Liu-Zhang, C.D., Loss, J.: Always have a backup plan: fully secure synchronous mpc with asynchronous fallback. In: CRYPTO (2020)
24. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: CCS (2017)
25. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: CRYPTO (2018)
26. Boneh, D., Goh, E., Nissim, K.: Evaluating encryption schemes for pattern matching. In: EUROCRYPT 2005. pp. 40–62. Springer (2005)
27. Borcherding, M.: Levels of authentication in distributed agreement. In: WDAG (1996)
28. Boudgoust, K., Scholl, P.: Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In: ASIACRYPT (2023)
29. Boyle, E., Cohen, R., Goel, A.: Breaking the $o(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. In: PODC (2021)
30. Brakerski, Z., Halevi, S., Polychroniadou, A.: Four round secure computation without setup. In: TCC (2017)
31. Braun, L., Damgård, I., Orlandi, C.: Secure multiparty computation from threshold encryption based on class groups. In: CRYPTO (2023)
32. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS (2001), we refer to eprint 2000/067 version 02/20/2020
33. Canetti, R.: Universally composable signature, certification, and authentication. In: CSFW. p. 219. IEEE Computer Society (2004)
34. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: TCC (2007)
35. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: STOC (2000)
36. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC (2002)
37. Cascudo, I., David, B.: Albatross: Publicly attestable batched randomness based on secret sharing. In: ASIACRYPT (2020)
38. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-efficient threshold ec-dsa. In: PKC (2020)
39. Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from ddh. In: CT RSA (2015)
40. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: CCS (2019)
41. Chen, S., Cheon, J.H., Kim, D., Park, D.: Verifiable computing for approximate computation. Cryptology ePrint Archive (2019)
42. Cheon, J.H., Cho, W., Kim, J.: Improved universal thresholdizer from threshold fully homomorphic encryption. ePrint 2023/545 (2023)

43. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: ASIACRYPT (2017)
44. Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults. In: FOCS (1985)
45. Choudhury, A.: Brief announcement: Almost-surely terminating asynchronous byzantine agreement protocols with a constant expected running time. In: PODC (2020)
46. Choudhury, A., Loftus, J., Orsini, E., Patra, A., Smart, N.P.: Between a rock and a hard place: Interpolating between mpc and fhe. In: ASIACRYPT (2013)
47. Choudhury, A., Patra, A.: On the communication efficiency of statistically-secure asynchronous mpc with optimal resilience. Iacr ePrint 2022/913 (2022), long version of ICITS 2009 and INDOCRYPT 2020
48. Cohen, R.: Asynchronous secure multiparty computation in constant time. In: PKC (2016)
49. Cohen, R., Garay, J., Zikas, V.: Broadcast-optimal two-round mpc. In: EUROCRYPT (2020)
50. Coretti, S., Garay, J., Hirt, M., Zikas, V.: Constant-round asynchronous multiparty computation based on one-way functions. In: ASIACRYPT (2016)
51. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: EUROCRYPT (2001)
52. Cramer, R., Damgård, I.B., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press (2015)
53. Dachman-Soled, D., Gong, H., Kulkarni, M., Shahverdi, A.: Towards a ring analogue of the leftover hash lemma. Journal of Mathematical Cryptology (2021)
54. Dahl, M., Demmler, D., Elkazdadi, S., Meyre, A., Orfila, J.B., Rotaru, D., Smart, N.P., Tap, S., Walter, M.: Noah's ark: Efficient threshold-fhe using noise flooding. ePrint 2023/815 (2023)
55. Damgård, I., Geisler, M., Kroigaard, M., Nielsen, J.: Asynchronous multiparty computation: Theory and implementation. In: PKC (2009)
56. Damgård, I., Magri, B., Ravi, D., Siniscalchi, L., Yakoubov, S.: Broadcast-optimal two round mpc with an honest majority. In: CRYPTO (2021)
57. Damgård, I., Ravi, D., Siniscalchi, L., Yakoubov, S.: Minimizing Setup in Broadcast-Optimal Two Round MPC (2023)
58. Dao, Q., Grubbs, P.: Spartan and bulletproofs are simulation-extractable (for free!). In: EUROCRYPT (2023)
59. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: CRYPTO 2001 (2001)
60. Dolev, D., Reischuk, R.: Bounds on information exchange for byzantine agreement. Journal of the ACM (JACM) **32**(1), 191–204 (1985)
61. Dov Gordon, S., Liu, F.H., Shi, E.: Constant-round mpc with fairness and guarantee of output delivery. In: CRYPTO (2015)
62. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR ePrint (2012)
63. Fehr, S.: Span Programs over Rings and How to Share a Secret from a Module. Master's thesis, ETH Zurich (1998)
64. Fiore, D., Nitulescu, A., Pointcheval, D.: Boosting verifiable computation on encrypted data. In: Public-Key Cryptography–PKC (2020)
65. Fitzi, M., Liu-Zhang, C.D., Loss, J.: A new way to achieve round-efficient byzantine agreement. In: PODC (2021)
66. Fitzi, M., Nielsen, J.B.: On the number of synchronous rounds sufficient for authenticated byzantine agreement. In: DISC (2009)

67. Fouque, P.A., Stern, J.: One round threshold discrete-log key generation without private channels. In: PKC (2001)
68. Garay, J., Givens, C., Ostrovsky, R., Raykov, P.: Broadcast (and round) efficient verifiable secret sharing. In: ITC (2013)
69. Garay, J., Kiayias, A., Ostrovsky, R.M., Panagiotakos, G., Zikas, V.: Resource-restricted cryptography: Revisiting mpc bounds in the proof-of-work era. In: EUROCRYPT (2020)
70. Garay, J.A., Kiayias, A., Leonardos, N., Panagiotakos, G.: Bootstrapping the blockchain, with applications to consensus and fast pki setup. In: PKC (2016)
71. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: On 2-round secure multiparty computation. In: CRYPTO (2002)
72. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: Yoso: You only speak once / secure mpc with stateless ephemeral roles. In: CRYPTO (2021)
73. Gentry, C., Halevi, S., Vadim, L.: Practical non-interactive publicly verifiable secret sharing with thousands of parties. In: EUROCRYPT (2022)
74. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: CRYPTO (2013)
75. Goel, A., Jain, A., Prabhakaran, M., Raghunath, R.: On communication models and best-achievable security in two-round mpc. In: TCC (2021)
76. Groth, J., Ostrovsky, R.: Cryptography in the multi-string model. J. of Cryptol. (2014)
77. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for NP. In: EUROCRYPT (2006)
78. Guo, Y., Pass, R., Shi, E.: Synchronous, with a chance of partition tolerance. In: CRYPTO (2019)
79. Hirt, M., Nielsen, J.B., Przydatek, B.: Cryptographic asynchronous multi-party computation with optimal resilience. In: EUROCRYPT (2005)
80. Impagliazzo, R., Levin, L.A., Luby, M.: Pseudo-random generation from one-way functions. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing. STOC '89 (1989)
81. ISO/IEC: Iso/iec 18033-6:2019 it security techniques-encryption algorithms-part 6: Homomorphic encryption (2019), `https://www.iso.org/obp/ui/#iso:std:iso-iec:18033:-6`
82. Jain, A., Rasmussen, P.M.R., Sahai, A.: Threshold fully homomorphic encryption. ePrint 2017/257 (2017)
83. Kate, A., Mangipudi, E.V., Mukherjee, P., Saleem, H., Thyagarajan, S.A.K.: Non-interactive vss using class groups and application to dkg. ePrint 2023/451 (2023)
84. Kate, A., Mangipudi, E.V., Mukherjee, P., Saleem, H., Thyagarajan, S.A.K.: Non-interactive VSS using class groups and application to DKG. CCS (2024)
85. Katz, J., Koo, C.Y.: On expected constant-round protocols for byzantine agreement. In: Annual International Cryptology Conference (2006)
86. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: TCC (2011)
87. Kiayias, A., Moore, C., Quader, S., Russell, A.: Efficient random beacons with adaptive security for ungrindable blockchains. IACR ePrint 2021/1698 (2021), `https://ia.cr/2021/1698`
88. Kim, E., Jeong, J., Yoon, H., Kim, Y., Cho, J., Cheon, J.H.: How to securely collaborate on data: Decentralized threshold he and secure key update. IEEE Access (2020)

89. Kim, T., Kwak, H., Lee, D., Seo, J., Song, Y.: Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition. In: CCS (2023)
90. Klemsa, J., Önen, M., Akin, Y.: A practical tfhe-based multi-key homomorphic encryption with linear complexity and low noise growth. In: ESORICS (2023)
91. Kolby, S., Ravi, D., Yakoubov, S.: Constant-round YOSO MPC without setup. Cryptology ePrint Archive, Paper 2022/187 (2022)
92. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. (1982)
93. Lindell, Y., Pinkas, B., Smart, N.P., Yanai, A.: Efficient constant-round multi-party computation combining bmr and spdz. Journal of Cryptology **32**, 1026–1069 (2019)
94. Lindell, Y., Smart, N.P., Soria-Vazquez, E.: More efficient constant-round multi-party computation from bmr and she. In: TCC (2016)
95. Liu-Zhang, C., Loss, J., Maurer, U., Moran, T., Tschudi, D.: MPC with synchronous security and asynchronous responsiveness. In: ASIACRYPT (2020)
96. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM (2013)
97. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-lwe cryptography. In: EUROCRYPT (2013)
98. Mouchet, C., Bertrand, E., Hubaux, J.P.: An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. Journal of Cryptology **36**(2), 10 (2023)
99. Mouchet, C., Troncoso-Pastoriza, J., Bossuat, J.P., Hubaux, J.P.: Multiparty homomorphic encryption from ring-learning-with-errors. PoPETS (2021)
100. Mouris, D., Masny, D., Trieu, N., Sengupta, S., Buddhavarapu, P., Case, B.: Delegated private matching for compute. Proceedings on Privacy Enhancing Technologies (2024)
101. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key fhe. In: EUROCRYPT (2016)
102. Park, J.: Homomorphic encryption for multiple users with less communications. IEEE Access (2021)
103. Parno, B., Raykova, M., Vaikuntanathan, V.: How to delegate and verify in public: Verifiable computation from attribute-based encryption. In: TCC (2012)
104. Patra, A., Ravi, D.: On the power of hybrid networks in multi-party computation. IEEE Transactions on Information Theory (2018)
105. Rachuri, R., Scholl, P.: Le mans: dynamic and fluid mpc for dishonest majority. In: CRYPTO (2022)
106. Reyzin, L., Smith, A.D., Yakoubov, S.: Turning HATE into LOVE: compact homomorphic ad hoc threshold encryption for scalable MPC. In: CSCML (2021)
107. Schoenmakers, B., Veeningen, M., de Vreede, N.: Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In: ACNS (2016)
108. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (nov 1979), `https://doi.org/10.1145/359168.359176`
109. Shrestha, N., Bhat, A., Kate, A., Nayak, K.: Synchronous distributed key generation without broadcasts. ePrint 2021/1635 (2021)
110. Tang, F., Ling, G., Cai, C., Shan, J., Liu, X., Tang, P., Qiu, W.: Solving small exponential ecdlp in ec-based additively homomorphic encryption and applications. IEEE Transactions on Information Forensics and Security **18**, 3517–3530 (2023)
111. Urban, A., Rambaud, M.: Robust multiparty computation from threshold encryption based on rlwe. ISC (2024)

112. Wan, J., Momose, A., Ren, L., Shi, E., Xiang, Z.: On the amortized communication complexity of byzantine broadcast. In: Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing. pp. 253–261 (2023)
113. Wu, W., Homsi, S., Zhang, Y.: Confidential and verifiable machine learning delegations on the cloud. Cryptology ePrint, Paper 2024/537 (2024)
114. Yao, A.C.: Protocols for secure computations. In: 23rd annual symposium on foundations of computer science (sfcs 1982). pp. 160–164. IEEE (1982)

## A Further Details on Related Works

### A.1 Details on the Approach of [61]

Let us recap their construction. Players first receive a uniform random string (denoted $\mathbf{B}$), and, ⓪ generate and publish GSW encryption keys on the PKI. To encrypt its input $m_i$, each player $P_i$ ① generates a ciphertext $\mathsf{c}_{i,i}$ of it under its own GSW encryption key and concatenates to it, encryptions of 0 under the $n-1$ keys of other players, all with the same encryption randomness (denoted $\mathbf{R}$). Such a vector of ciphertexts $\hat{\mathsf{c}}_i = (\mathsf{c}_{i,1}, \mathsf{c}_{i,2}, \ldots, \mathsf{c}_{i,n})$ is denoted as a *flexible ciphertext*, and is broadcast. *In parallel* ①, players perform the second event of a DKG, establishing a common threshold encryption key. Because a flexible ciphertext is generated using the same secret randomness for all the $n$ GSW ciphertexts contained in it, players are able to Transform, locally and deterministically, a flexible ciphertext into a GSW ciphertext under the common threshold encryption key by linearity. Then, players proceed with the local evaluation of the circuit, and finally ② with the threshold decryption, which can be done over asynchronous P2P channels, by our observation. Because the same encryption randomness is used, and given that $t$ of these ciphertexts are encrypted under GSW encryption keys which were generated by the adversary $\mathcal{A}$, this a priori gives $\mathcal{A}$ an extra advantage to guess the plaintext of $\mathsf{c}_{i,i}$. For the security of GSW to hold, their encryption keys are thus scaled slightly larger ($m = \Omega((d+n)\log(q))$ vs $m = \Omega(d\log(q))$ in GSW), in order to apply the leftover-hash-lemma (LHL [61, lemma 1]).

Overall, this protocol has two main limitations:

1. ***It is not delegable*!** The generation of the flexible ciphertext $\hat{\mathsf{c}}_i$ has, by construction the decryption key $\mathsf{sk}_i$ known by the corresponding player $P_i$ (i.e. $\hat{\mathsf{c}}_i$ reveals the encrypted input $m_i$ to the player $P_i$ that owns the GSW decryption key $\mathsf{sk}_i$). It is therefore impossible to accommodate external input-owners without losing privacy, which prevents delegation and a fortiori lightweight input-owners.
2. ***It is not generic*!** A second limitation is that, since this technique relies on the Leftover Hash Lemma (LHL) [80], it is unknown how to port this construction over other FHE schemes. Let us illustrate this issue by taking the BFV scheme detailed in Section 3.1.1 as an example. Overall, their technique is not easily transposable to efficient RLWE-based cryptosystems, for the following reasons. Suppose that the adversary is given one (or several) BFV encryptions of 0 under semi-maliciously generated key(s) $(a, b_i)$, i.e., $\left(u \cdot b_i + e_{0,i}^{(\mathsf{Enc})}, u \cdot a + e_{1,i}^{(\mathsf{Enc})}\right)$, which would all be generated with the same secret randomness $u \leftarrow \mathscr{X}_q$. Then this may provide it with a distinguishing advantage when given a BFV encryption of some $m$ under some honest key $(a, b_j)$ which would re-use the same randomness $u$. One could possibly think of a fix, e.g., adapting BFV by specifying that the first component of the encryption key, $a := \mathbf{a}[0]$, would be instead vectors with coordinates in $R_q$, and encryption randomness $\mathbf{u}$ equal to a random vector with entries in $R_q$. But this fails since [53, §1] has evidenced a counterexample showing that the LHL does not hold in general (a leakage of $1/d$ of the

secret randomness which would make the outcome far from indistinguishable from uniform). They also point that [97, Cor 7.5] showed that a weaker version of LHL, denoted "regularity", does apply in this setting, nonwithstanding the previous leakage issue, in the case where the distribution of secret randomness would be Discrete Gaussian with a sufficiently large parameter. Concretely, we would apply "regularity" to $\mathbf{A}.\mathbf{u}$, where $\mathbf{A}$ would be a matrix with $n$ rows encoding all encryption keys. The problem is that for the applicability of "regularity", it is required that $\mathbf{A}$ be sampled uniformly, whereas in our setting we have $t$ keys in $\mathbf{A}$ which are semi-maliciously generated, furthermore possibly depending on the other $t+1$ honest keys. Thus, this situation could potentially leak substantial information on $\mathbf{u}$, and thus potentially enable to distinguish the outcome from random uniform, such as mentioned above [53, §1].

*Remark 1.* As a final remark on [61], notice that, on the face of it, their DKG is specified over pairwise channels, thus a player that would abort after sending only part of the messages it is meant to send, would leave honest players with inconsistent views on which contributions to the threshold key should be taken into account. However this is handled by [61, Remark 4.1], who observe that players can be instructed to broadcast the set of their messages, encrypted under each recipient's public key. From this perspective, their DKG, as well as the one of [67], can be seen as a particular case of $\mathcal{F}_{\mathsf{LSS}}$ to do a sum of contributions to the secret threshold decryption key, for the particular parameter of a modulus $q$ equal to a prime larger than $n+1$ [61, §B,§C].

## A.2 Related Works based on $(n,t)$-threshold FHE

The work of Asharov et al. [14] was the first to introduce a $(n,t)$-threshold FHE scheme with thresholds $t$ lower than $n-1$. Unfortunately, their construction adds 2 additional rounds to the MPC protocol as soon as one player aborts. Indeed, in this case, it is required that the honest majority reconstructs the player's state and resume the protocol.

The work [88, §IV B] implements a DKG for CKKS with reconstruction from $t+1$-out-of-$n$ Shamir shares of the decryption key. It is implemented by having players reshare the key from $n$-out-of-$n$, into $t+1$-out-of-$n$. A variant of the DKG of [88] is proposed in [98]. Since [88,98] follow the DKG-then-input distribution approach, they require at least two broadcasts, which is incompatible with our requirement. Finally, the MPC protocols suggested in [88,98] are also non-robust, by lack of a robust distributed generation of relinearization & bootstrapping keys.

The work [25] also proposed $(n,t)$-threshold FHE schemes. However, their §5 leaves unspecified the DKG, while their §6.2 has the drawback that the encryption and decryption key pair is generated by one entity (typically, one secret owner, before encrypting its secrets), thus it would be unsafe to have other secret owners also encrypt their secret under this same key, which prevents MPC.

### A.3 Other synchronous / asynchronous hybrid models

The question of minimizing the number of initial synchronous rounds or broadcast(s) in an execution of MPC was initiated in [79]. The broadcast of [66] uses a few synchronous rounds, then guarantees eventual output delivery, under an honest majority. The consensus of [45] with $t < n/3$ tolerance uses one initial round of synchrony. In information-theoretic MPC with $t < n/3$, [104] reached an optimal 3 initial synchronous rounds. In the setting of MPC with non-constant latency, then [21] exhibited a protocol with only one all-to-all broadcast of encrypted inputs, followed by asynchronous peer-to-peer messages. However, they assume for granted a $\mathsf{DKG}$ setup, i.e., a $(n, t)$-threshold additively homomorphic encryption scheme. Establishing such a $\mathsf{DKG}$ setup with Paillier, as they suggest, would cost a number of more broadcast rounds, so is incompatible with our goal.

The protocol [23] proceeds by intervals of fixed duration, denoted rounds, of which the number grows with the depth of the circuit. Since their model is free of primitives such as Byzantine Agreement under honest majority, if the network is asynchronous and more than $t_a < n/3$ players are corrupt, then they cannot guarantee input provision, nor agreement on the output.

The protocols [50,48] are purely asynchronous, i.e., responsive, and thus do not withstand more than $t < n/3$ corruptions. Even if the latter has a trusted setup, withstanding more than $t < n/3$ corruptions is impossible since the protocol is purely responsive.

## B Model: Further Formalism and Discussion

### B.1 Formalizing Eventual Delivery in UC.

We now explain the high-level idea of the mechanism, denoted $\mathsf{fetch}$-*and*-$\mathsf{delay}$, used to formalize eventual delivery following [86,50]. Every ideal functionality $\mathcal{F}$, when it needs to eventually deliver $(\mathsf{ssid}, v)$ to some entity $P$, engages in the following interaction. It notifies $\mathcal{A}$ of the output id $(\mathsf{ssid})$, and initializes a counter $D_{\mathsf{ssid}} \leftarrow 1$, which captures the delivery delay. Upon receiving $(\mathsf{delay})$ from $\mathcal{A}$ , it sets $D_{\mathsf{ssid}} \leftarrow D_{\mathsf{ssid}} + 1$. Upon receiving $(\mathsf{fetch})$ from P, it sets $D_{\mathsf{ssid}} \leftarrow D_{\mathsf{ssid}} - 1$, as well as for all other counters related to pending outputs for $P$. In addition, we specify that it leaks $(\mathsf{fetch})$ to $\mathcal{A}$[12]. It is left implicit that entities $\mathsf{fetch}$ as much as they can all. Since $\mathcal{A}$ is $\mathsf{PPT}$, at some point, it gets exhausted from pressing the button $\mathsf{delay}$. So, after sufficiently many $\mathsf{fetch}$es more, the counter drops down to 0. Then $\mathcal{F}$ can deliver $(\mathsf{ssid}, v)$ to $P$.

### B.2 More on $\mathcal{F}_{\mathrm{C}}$

We describe in Fig. 4 the ideal functionality of secure circuit evaluation, as introduced in Section 2.3. In the presentation, C is hardcoded in $\mathcal{F}_{\mathrm{C}}$. But our MPC protocol allows players to *adaptively* choose C based on the list of non-$\perp$ inputs received.

[12]This precision is not present in previous works [86,50,95], since that way, the adversary knows at any moment in time when an output will be delivered.

$\mathcal{F}_{\mathrm{C}}$

**Output format** Initialize an empty vector $\mathbf{m} = \{\top\}^n$.

- Initialize $\mathsf{outpout{-}available} \leftarrow \mathsf{false}$.
- When all $m_i \neq \top$, $\forall i \in [n]$, set $\mathsf{outpout{-}available} = \mathsf{true}$. //it is now frozen

**Formalizing timeout for inputs of corrupt players**

- Initialize a counter $T_{\mathcal{A}} \leftarrow 1$ // the timeout.
- Upon receiving $\mathsf{delay{-}inputs}$ from $\mathcal{A}$, $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} + 1$.
- Upon receiving $\mathsf{fetch}$ from $\mathcal{L}$, $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} - 1$.
- When $T_{\mathcal{A}} = 0$ for the first time, freeze forever $T_{\mathcal{A}} = 0$. Then, for all $i \in \mathcal{I}$: if $m_i = \top$, then set $m_i \leftarrow \perp$.

**Input** (Only accessible while $\mathsf{outpout{-}available} = \mathsf{false}$) On input $(\mathsf{input}, \widetilde{m_i} \in \mathscr{M})$ from any $\mathsf{Q}_i \in \mathcal{Q}$ for the first time, or possibly from $\mathcal{A}$ if $\mathsf{Q}_i$ is corrupt, set $m_i \leftarrow \widetilde{m_i}$, then store $(\mathsf{input}, \mathcal{S}, m_i)$, and *eventually-deliver* $(\mathsf{stored}, \overline{m_i})$ to each player $P \in \mathscr{P}$. //eventually-delivers" consists of the fetch-and-delay mechanism explained in Appendix B.1.

**Formalizing eventual delivery**

- Initialize a counter $D_R \leftarrow 0$ // the delivery delay.
- Upon receiving $\mathsf{fetch}$ from $\mathcal{L}$, $D_R \leftarrow D_R - 1$.
- When $D_R = 0$ for the first time, if no output was delivered yet to $\mathcal{L}$, wait until $\mathsf{outpout{-}available} = \mathsf{true}$, then deliver $y = \mathrm{C}(\mathbf{m})$ to $\mathcal{L}$.

Fig. 4: Functionality of secure circuit evaluation. Each input $m_i$ is identified by a public label $\overline{m_i}$.

## B.3 More on Broadcast $\mathsf{BC}$ with Possibly External Senders

### B.3.1 Broadcast functionality

***Definition 6.*** *A broadcast protocol [65, Definition 1] involves a sender $\mathcal{S}$ and a set of receivers $\mathcal{R}$. It requires the following properties:*

**(Termination):** *all honest receivers eventually output;*
**(Consistency):** *any two honest receivers output the same value;*
**(Validity):** *if the sender $\mathcal{S}$ is honest and input value $x$, all honest receivers output the same value $x$.*

*We dub it as* $\mathsf{BC}$ *and formalize it in Fig. 5.*

Overall, it simply proceeds as follows. On receiving a message $s$ from the sender $\mathcal{S}$, it sends $s$ to each receiver $R \in \mathcal{R}$ by using the $\mathsf{fetch}$-and-$\mathsf{delay}$ mechanism introduced in Appendix B.1.

$\mathsf{BC}^{\mathcal{S}\to\mathcal{R}}$

- Upon receiving a message $(\text{send}, m)$ from $\mathcal{S}$, for each $R \in \mathcal{R}$, do the following. Initialize $D_{mid} \leftarrow 1$, where $mid$ is a unique message ID, store $(mid, D_{mid}, m, R)$ and leak $(mid, D_{mid}, R, m)$ to $\mathcal{A}$.
- Upon receiving a message $(\text{fetch})$ from $R$:
  1. Set $D_{mid} \leftarrow D_{mid} - 1$ for all $(mid, D_{mid}, R, m)$ stored, and leak $(\mathsf{fetch}, R, m)$ to $\mathcal{A}$.
  2. If $D_{mid} = 0$ for some stored $(mid, D_{mid}, R, m)$, deliver the message $m$ to $R$ and delete $(mid, R, m)$ from the memory.
- Upon receiving a message $(\mathsf{delay}, mid, R)$ from $\mathcal{A}$, for some stored $(mid, D_{mid}, R, m)$, set $D_{mid} \leftarrow D_{mid} + 1$.

Fig. 5: Ideal functionality of reliable broadcast. It is parametrized by a sender $\mathcal{S}$ and a set of receivers $\mathcal{R}$.

*Remark 2.* There exists another functionality for broadcast in [47,4], which is denoted as $\mathcal{F}_{\text{ACast}}$. The difference is that for every given $R \in \mathcal{R}$, $\mathcal{F}_{\text{ACast}}$ may *never* deliver $s$ to $R$ if the adversary does not allow to, nonwithstanding other honest players could have been delivered $s$. The reason is that they use the classical UC framework of *delayed output* of Canetti, in which the delivery of every single output from a functionality needs to be allowed by the adversary.

**B.3.2 Sub-sessions** For the sake of UC analysis, in our MPC protocol, we specify multiple broadcast instances per sender. This is why we formalize $\mathsf{BC}^{\mathcal{S}\to\mathcal{R}}$ with sub-session identifiers denoted $\mathsf{ssid}$. Importantly, we force $\mathsf{BC}^{\mathcal{S}\to\mathcal{R}}$ into allowing at most one $\mathsf{ssid}$ per sender, to prevent two players from receiving different outputs from a corrupt sender for the same $\mathsf{ssid}$. In practice, in our protocol, the ssid is the label of the variable that is broadcast. Furthermore, we make the abuse of notation, in our protocol, to have one sender concatenate multiple broadcast instances of several variables at once, likewise for the input command of $\mathcal{F}_{\mathsf{LSS}}$. Indeed this is how the protocol would be efficiently implemented. In a model allowing that, an input owner can possibly be logically identified to some player, and thus both would either be simultaneously honest or corrupt. One could furthermore have them concatenate all their broadcasts in $\textcircled{1}$.

## B.4 $\mathcal{F}_{\mathbf{AT}}$

In Fig. 6, we present our Authenticated Message Transmitting functionality $\mathcal{F}_{\mathrm{AT}}$. Our baseline for $\mathcal{F}_{\mathrm{ST}}$ is the functionality denoted $\mathcal{F}_{\text{ed-smt}}$ [86]. For $\mathcal{F}_{\mathrm{AT}}$, we made the addition to leak the contents of the messages to $\mathcal{A}$. We also incorporated two other additions, borrowed from the $\mathcal{F}_{\mathrm{NET}}$ in [95]. The first consists in attaching a unique identifier to each message and counter, in order to give to $\mathcal{A}$ a control on the delay of each message individually. Notice that [50] model this individual control by, instead, giving the power to $\mathcal{A}$ to re-order messages not delivered yet. The second addition consists in forcing explicitely $\mathcal{A}$ to press (delay) to augment the delay by $+1$, instead of the (equivalent) formalization in which $\mathcal{A}$ enters the additional delay in unary notation.

$\mathcal{F}_{\mathrm{AT}}$ $/\mathcal{F}_{\mathrm{ST}}$

- Upon receiving a message $(\mathsf{send}, m)$ from $\mathcal{S}$, initialize $D_{mid} \leftarrow 1$, where $mid$ is a unique message ID, store $(mid, D_{mid}, m)$ and leak $(mid, D_{mid}, m)$ to $\mathcal{A}$. $\mathcal{F}_{\mathrm{ST}}$ leaks only $(mid, D_{mid}, |m|)$.
- Upon receiving a message (fetch) from $R$:
  1. Set $D_{mid} \leftarrow D_{mid} - 1$ for all $mid$ stored, and leak $(\mathsf{fetch})$ to $\mathcal{A}$.
  2. If $D_{mid} = 0$ for some stored $(mid, D_{mid}, m)$, deliver the message $m$ to $R$ and delete ($mid$,m) from the memory.
- Upon receiving a message ($\mathsf{delay}$,$mid$) from $\mathcal{A}$, for some stored $mid$, set $D_{mid} \leftarrow D_{mid} + 1$.
- (Adaptive message replacement) Upon receiving a message $((mid, m) \rightarrow m')$ from $\mathcal{A}$, if $\mathcal{S}$ is corrupt and the tuple $(mid, D_{mid} > 0, m)$ is stored, then replace the stored tuple by $(mid, D_{mid}, m')$.

Fig. 6: Ideal functionality of asynchronous *public authenticated* message transmitting with eventual delivery delay, parametrized by sender $\mathcal{S}$ and receiver $R$. The straightforward upgrade to obtain asynchronous *secure* message transmitting $\mathcal{F}_{\mathrm{ST}}$ is described inline.

*Remark 3.* In the last (asynchronous) steps of our MPC protocol, when $\mathcal{F}_{\mathsf{LSS}}$ is instantiated with $\Pi_{\mathsf{LSS}}$, each player is instructed to send the same opening share to all using $\mathcal{F}_{\mathrm{AT}}$. Notice that nothing prevents corrupt players from sending different opening shares to different players. However, when sending a share, semi-maliciously corrupt players must exhibit an input tape containing a pair of: a decryption key and a key noise, compatible with the $\mathbf{b_i}$ which they broadcast in $\textcircled{1}$. Thus, for whatever compatible pair which they could exhibit, the decryption share coming with it is necessarily correct.

*Remark 4.* Notice that, in our protocol, players are instructed to publish their public key on bPKI at the beginning, thus $\mathcal{F}_{\mathrm{AT}}$ could actually have been implemented from non-authenticated channels.

### B.5 More on our bPKI, and the { Bulletin board / Untrusted / bare } PKI model

In Fig. 7, we present our bulletin board PKI functionality bPKI. For our usage, bPKI is limited to the publication of public keys, and could be traded by the assumption denoted {Bare/Untrusted/Bulletin board} public key setup (PKI)" [11,29,75]. Importantly, bPKI does not perform any checks on the written strings, it displays them to all players.

**B.5.1 PKI assumptions in the literature** The bPKI functionality, for our usage limited to publication of public keys, could be traded by the assumption denoted {Bare/Untrusted/Bulletin board} public key setup (PKI)" [11,29,75]. The PKI model was first sketched in [35, §6], then denoted as "bare PKI" in [11]. It is renamed as "untrusted PKI" in [75]. As specified in [35, §6.1][78,56], the PKI model is slightly stronger than ours, since it abstracts-out all the implementation constraints discussed in Appendix B.3, in a way which could be phrased as: (i) all instances of bPKI are assumed to terminate before a public time denoted $t = 0$, (ii) players (and owners) are then able to reset their clocks synchronously at $t = 0$, which, e.g., enables them to subsequently run implementations of BC. By contrast, the implementation of BC in [66] does not guarantee delivery within a fixed delay (the "$t = 0$"), only eventually. Notice that, formalized like this, the "$t = 0$" is related to the notion of "synchronization point" in [55,95].

Likewise, in the specific case of protocols for Byzantine agreement, [29] also assumes access to the board only before players are assigned their inputs, which is the same assumption in our case. We refer to [29] for a comparison of bulletin-board PKI with more demanding setup assumptions. Notice that our generalization, where inputs are formally assigned to owners instead of players, parallels the regime of state machine replication in which commands originate from external lightweight "clients".

**B.5.2 The power of { Bulletin board / Untrusted / bare } PKI in MPC** By construction, bPKI does not perform any checks on the written strings, it displays them to all players. Thus, it is strictly weaker than the setup denoted as "registered PKI", or KRK, in [34], where it is proven to have strictly more power. Notice that implementing KRK without $\overline{\mathcal{G}}_{\mathrm{URS}}$ would, in turn, require

**bPKI**

**Output format** Initialize an empty vector $\mathsf{pk} = \{\top\}^n$.
When all $\mathsf{pk}_i \neq \top$, $\forall i \in [n]$, set $\mathsf{outpout{-}available} = \mathsf{true}$.

**Formalizing eventual delivery** For every $R \in \mathcal{R}$, initialize a counter $D_R \leftarrow 0$ // the delivery delay.
Initialize $\mathsf{outpout{-}available} \leftarrow \mathsf{false}$ //the flag telling if the output can be delivered.
Upon receiving $\mathsf{fetch}$ from any $R \in \mathcal{R}$, $D_R \leftarrow D_R - 1$. When $D_R = 0$ for the first time, if no output was delivered yet to $R$, wait until $\mathsf{outpout{-}available} = \mathsf{true}$ then deliver $\mathsf{pk}$ to $R$.

**Formalizing timeout for keys of corrupt players** Initialize a counter $T_{\mathcal{A}} \leftarrow 1$ // the timeout.
Upon receiving $\mathsf{delay{-}keys}$ from $\mathcal{A}$, $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} + 1$.
Upon receiving $\mathsf{fetch}$ from any $R \in \mathcal{R}$, $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} - 1$.
When $T_{\mathcal{A}} = 0$ for the first time, freeze forever $T_{\mathcal{A}} = 0$. Then, for all $i \in \mathcal{I}$: if $\mathsf{pk}_i = \top$, then set $\mathsf{pk}_i \leftarrow \perp$.

**Honest keys registration** Upon receiving the first message $(\mathsf{Register}, \widetilde{\mathsf{pk}_i})$ from an honest key-holder $P_i$, send $(\mathsf{Registered}, P_i, \widetilde{\mathsf{pk}_i})$ to $\mathcal{A}$ and set $\mathsf{pk}_i \leftarrow \widetilde{\mathsf{pk}_i}$.

**Corrupt keys registration** Upon receiving a message $(\mathsf{Register}, (\widetilde{\mathsf{pk}_i} \neq \top)_{i \in \mathcal{I}})$ from $\mathcal{A}$, set $\mathsf{pk}_i \leftarrow \widetilde{\mathsf{pk}_i}$ $\forall i \in \mathcal{I}$.

Fig. 7: The bulletin board of public keys functionality bPKI, parametrized by a set of $n$ key-holders, of which the corrupt ones are indexed by $\mathcal{I} \subset [n]$, and by a set of receivers $\mathcal{R}$. It does not perform any checks on the keys received. A published key that is not in the correct distribution is automatically considered as $\perp$ by honest players.

an extra event before bPKI in which players would publish multi-string CRS. Fortunately, KRK is not required in our protocol, only plaintexts are extracted in our UC proof, not secret keys.

In [75] it is proven that MPC in two rounds is impossible in the plain model, even with identifiable abort. However, it is feasible assuming the bare/untrusted PKI Model.

To the best of our knowledge, bPKI is the minimal setup necessary in all implementations of synchronous broadcast for $t \geq n/3$, since the seminal [92]. Also, [27, Thm 1] shows that the relaxation of bPKI denoted as "local setup", precludes synchronous broadcast for $t \geq n/3$. [ Local setup means that a corrupt player can possibly make bPKI display different strings to different players. However, it cannot claim for itself a string previously published by an honest player. ]

To be complete, let us mention that Borcherding's impossibility is circumvented with the additional assumption that the majority of a restricted resource, e.g. the computing power (or, alternatively, storage space) is in the hands of honest players, which we may denote as the "proof of work (PoW) model". There, [70] implement what is denoted as a "pseudonymous" PKI, i.e., a mechanism that outputs to all honest players a single set of public keys, with possibly several keys assigned to the same players, while guaranteeing that the majority of keys were issued by, and thus are owned by, honest players. In the same PoW model, [69] implement an actual untrusted PKI, i.e., the $\mathcal{F}_{CA}$ of [33], which they denote $\mathcal{F}_{REG}$.

## B.6 $\overline{\mathcal{G}}_{\mathbf{URS}}$

$\overline{\mathcal{G}}_{\mathrm{URS}}$ is a particular case of $\mathcal{F}_{crs}$ in [36].

$\overline{\mathcal{G}}^{\kappa}_{\mathrm{URS}}$

On input query from all honest players in $\mathscr{P}$, then samples a sequence of $\kappa$ bits uniformly at random then outputs it to each player $P \in \mathscr{P}$, then halts.

Fig. 8: Uniform Random String.

On the one hand, MPC under honest majority enables to UC implement fair coin tossing. Thus $\overline{\mathcal{G}}_{\mathrm{URS}}$ could have been implemented, at the cost of extra preliminary steps. Notice that optimized implementations exist, but at the cost of more assumptions. E.g., [37] requires a CRS for the generation of NIZKs, while [87] requires an initial seed.

On the other hand, when upgrading $\overline{\mathcal{G}}_{\mathrm{URS}}$ to a global setup, then it is not proven in general if one can safely implement a global setup by using any protocol proven UC secure, e.g., see [16].

### B.7 UC Non-Interactive Zero-knowledge (NIZK) functionality

***Definition 7.*** *(Zero-Knowledge Proof) A pair of probabilistic polynomial time interactive programs $P, V$ is a zero-knowledge proof if the following properties are satisfied:*

**Soundness** *: If the statement is false, a cheating prover $P$ cannot convince the honest verifier $V$ that it is true, except with negligible probability.*

**Completeness:** *If the statement is true, then an honest verifier $V$ will be convinced by an honest prover $P$.*

**Zero-knowledge** *: If the statement is true, no verifier $V$ learns anything beyond the statement being true.*

We present in Fig. 9 the ideal functionality of *non-interactive zero-knowledge arguments of knowledge*, denoted as $\mathcal{F}_{\mathsf{NIZK}}$ and mainly borrowed from [77]. It is parametrized by an NP relation $\mathcal{R}$. Upon request of a prover $P$ exhibiting some public input $x$ and knowledge of some secret witness $w$, it verifies if $(x, w) \in \mathcal{R}$ then deletes $w$ from its memory. If the verification passes, then $\mathcal{F}_{\mathsf{NIZK}}$ *outputs* to $P$ a string $\pi$. We denote $\Pi$ the space of such strings $\pi$. During the delay of output, $\mathcal{A}$ has the power to set the value of $\pi$. If it does not use this power, then $\mathcal{F}_{\mathsf{NIZK}}$ sets $\pi$ to a default value $\pi_0$. Upon subsequent input the same string $\pi$ and $x$ from any verifier, $\mathcal{F}_{\mathsf{NIZK}}$ then confirms to the verifier that $P$ knows some witness for $x$.

*Remark 5.* The main difference with [77], in which $\mathcal{A}$ could delay forever the delivery of $\pi$, is the introduction of a time-out, based on the fetch-and-delay mechanism. Sticking to this model would have prevented us from specifying a protocol with guaranteed output delivery (GOD) in case of honest majority.

UC implementations of $\mathcal{F}_{\mathsf{NIZK}}$ exist, which do not require honest majority [59], but at the cost of requiring a uniform random string (URS). Nonwithstanding that [59] allows the same URS to be reused in concurrent executions, the bottom-line is that the URS needs to be part of a local setup in their implementation. Without an honest majority assumption, then [34] prove that UC NIZK is non-implementable in the global common random string model, i.e., which we formalized as $\overline{\mathcal{G}}_{\mathrm{URS}}$ in the particular case where the string is uniform.

*Remark 6.* The need for a URS can be escaped under honest majority, provided access to bPKI, thanks to the technique denoted multi-string CRS [76,17].

### B.8 Reminder of the UC model

Consider a protocol $\Pi$, a functionality $\mathcal{F}$ and any PPT environment Env which can interact with either one or the other of the following protocols, without being informed which of them. In every execution, Env may provide inputs to honest players, may provide instructions to the adversary, and may observe the outputs of players. At some point in every execution, Env must output a bit. In

$\mathcal{F}_{\mathsf{NIZK}}$

The functionality is parametrized with an NP relation $\mathcal{R}$ of an NP language $L$ and a prover $P$.

**Proof:** On input $(\mathsf{prove}, \mathsf{sid}, \mathsf{ssid}, x, w)$ from $P$, ignore if $(x, w) \notin \mathcal{R}$. Request $(\mathsf{proof}, x)$ to $\mathcal{A}$ then go to the next step.

**Reception of the NIZK** Initialize a counter $T_\mathcal{A} \leftarrow 1$.
Upon receiving $\mathsf{delay}$ from $\mathcal{A}$, $T_\mathcal{A} \leftarrow T_\mathcal{A} + 1$.
Upon receiving $\mathsf{fetch}$ from $P$, $T_\mathcal{A} \leftarrow T_\mathcal{A} - 1$.
Upon receiving $(\mathsf{activate}, \pi)$ from $\mathcal{A}$ *and* if $T_\mathcal{A} > 0$, then: freeze forever store $(x, \pi)$ and deliver $(\mathsf{proof}, \mathsf{sid}, \mathsf{ssid}, \pi)$ to P.
If $T_\mathcal{A} = 0$, then: freeze forever $T_\mathcal{A} = 0$, store $(x, \pi)$ and deliver $(\mathsf{proof}, \mathsf{sid}, \mathsf{ssid}, \pi_0)$ to P.

**Verification:** On input $(\mathsf{verify}, \mathsf{sid}, \mathsf{ssid}, x, \pi)$ from any verifier $V$, check whether $(x, \pi)$ is stored. If not, then do the following instructions:

- request $(\mathsf{verify}, x, \pi)$ to $\mathcal{A}$;
- initiate a counter $D_{\text{verif}}$ which $\mathcal{A}$ can increase by $+1$ steps, and $V$ by $-1$ steps;
- upon receiving an answer $(\mathsf{activate}, \mathsf{witness}, w)$ from $\mathcal{A}$ *and* if $D_{verif} > 0$ *and* if $(x, w) \in \mathcal{R}$, then: store $(x, \pi)$;
- when $D_{\text{verif}} = 0$, halt those instructions and go to the next (and last) step.

If $(x, \pi)$ is stored, return $(\mathsf{verification}, \mathsf{sid}, \mathsf{ssid}, 1)$ to $V$, else return $(\mathsf{verification}, \mathsf{sid}, \mathsf{ssid}, 0)$.

Fig. 9: Non-interactive zero-knowledge functionality

the first, denoted "real" $\mathrm{REAL}_{\Pi}$, $\mathsf{Env}$ interacts with the dummy adversary $\mathcal{A}$, and honest players follow the actual protocol $\Pi$. In the second, denoted "ideal" $\mathrm{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathsf{Env}}$, $\mathsf{Env}$ interacts with an adversary $\mathsf{Sim}$, while honest players are connected only to $\mathcal{F}$. Following [32], we say that protocol $\Pi$ UC emulates $\mathcal{F}$ if there exists a PPT machine, denoted as the simulator $\mathsf{Sim}$, such that for any such $\mathsf{Env}$, the gap of probabilities of outputting 1 when faced with an execution of the first protocol, and when faced with an execution of the second, is negligible. Notice that this definition, with only the dummy adversary, is easily seen, and proven in [32, §4.3.1], to be equivalent to UC emulation against any adversary.

## B.9 Semi-Malicious corruptions.

In our protocols and proofs, we will consider what we define as *Semi-Malicious Corruptions*, following [14, §A.2] [30,61,17]. Semi-maliciously corrupt players and input-owners continuously forward to $\mathcal{A}$ their outputs received from ideal functionalities, and act arbitrarily as instructed by $\mathcal{A}$. E.g., they can possibly not send some message although the protocol instructs them to. However, when a corrupt entity $M$ inputs a message $m$ to $\mathcal{F}_{\mathrm{AT}}$ or $\mathsf{BC}$, then the sending of $m$ must be compatible with the requirements of the protocol, with respect to: (i) all outputs of instances of $\overline{\mathcal{G}}_{\mathrm{URS}}$, $\mathsf{bPKI}$, $\mathsf{BC}$, and also $\mathcal{F}_{\mathsf{LSS}}$ in the case of our $\Pi$, required for sending $m$, and (ii) an internal witness tape that $M$ must have, of the form $(x, r)$ with $x$ an input and $r$ of the same length as all random coins that an honest player would have been meant to have tossed upon sending $m$. $M$ can however use conflicting $(x, r)$ when sending different messages $m$, $m'$. Moreover, we also require that the semi-malicious adversary can only make $\mathsf{BC}^M$ to output $v$ for some corrupt $M$ only if: either $v$ could have been input to $\mathsf{BC}^M$ by $M$ itself according to the above rule, or, if $v = \perp$.

In summary, the adversary must have a witness tape containing: an input value of $P$, coins explaining the random choices of $P$, and the set of all broadcast values so far (possibly from other senders), including in our case the encryption keys, and, for each previous asynchronous step, a set of $t + 1$ received messages. The latter requirement is new and specific to our asynchronous context. Recall that in [14], semi-malicious players simply had to explain their behavior based on previous broadcast messages. By contrast, in our model, nothing prevents them from arbitrarily picking the sets of $t + 1$ messages. The observation which we make is that this new choosing power is useless, since our MPC protocol is made only of threshold openings of linear maps. Precisely, the choice of the $t+1$ messages *does not modify* the reconstructed value, from which semi-malicious players must build their next message.

Finally, notice that we *do not* impose any condition for the sending of some $m$ on $\mathsf{bPKI}$. Concretely, this is because in the UC proof of Prop. 15 of our implementation of $\mathcal{F}_{\mathsf{LSS}}$, we do not need extractable secret keys. The consequence is that our protocol can be instantiated with NIZKs with *non-necessary straight line* simulation extractability, so this allows more efficient NIZKs. Recent such examples are Bulleproofs and Spartan [58], of which the former are the ones used in [73] for our purpose. See Appendix F.2 for more details on the consequences of this relaxation.

### B.10 Reminder of IND-CPA security

***Definition 8.*** PKE *A public-key encryption scheme consists of the following algorithms:*

- ***Key Generation*** $(\mathsf{dk}, \mathsf{pk}) \leftarrow \mathsf{EKeyGen}(1^\lambda)$*: Given a security parameter* $\lambda$*, the key generation algorithm outputs the public key* $\mathsf{pk} \in \mathscr{P}k$ *with the associated private key* $\mathsf{dk}$*;*

- ***Encryption*** $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*: Given a message* $m$ *and a public key* $\mathsf{pk}$*, it outputs the ciphertext* $\mathsf{c}$*;*

- ***Decryption*** $m' \leftarrow \mathsf{Dec}(\mathsf{dk}, \mathsf{c})$*: Given a ciphertext* $\mathsf{c}$ *and a private key* $\mathsf{dk}$*, it outputs a message* $m'$*.*

1. ***Security:*** *For subsequent use later in this work, we now describe in more detail the semantic security of a* PKE *scheme* $\mathscr{E} = (\mathsf{EKeyGen}, \mathsf{Enc}, \mathsf{Dec})$*. Consider the following* IND-CPA *game:*

```
$\mathsf{Game}^{\mathcal{A}}_{\mathsf{IND\text{-}CPA}}(1^\lambda)$
1 : $b \xleftarrow{\$} \{0,1\}$
2 : $(\mathsf{dk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{EKeyGen}(1^\lambda)$
3 : $(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk})$
4 : $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$
5 : $b' \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}, \mathsf{c}, \mathsf{state})$
6 : return $b = b'$
```

*The advantage of* $\mathcal{A}$ *in this game is defined as* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Enc}} = |Pr[b = b']|$*.*
*We say that* $\mathscr{E}$ *is* IND-CPA *secure if, for any* PPT *adversary* $\mathcal{A}$*, it holds that:*

$$|2.Adv^{\mathcal{A}}_{\mathsf{Enc}} - 1| \leq negl(\lambda) \tag{5}$$

2. ***Correctness:*** *A public-key encryption scheme if said* correct *is for all message* $m$ *and* $(\mathsf{dk}, \mathsf{pk}) \leftarrow \mathsf{EKeyGen}(1^\lambda)$*,*

$$\mathsf{Dec}(\mathsf{dk}, \mathsf{Enc}(\mathsf{pk}, m)) = m\,. \tag{6}$$

## C On Reusability and Comparison with [75]

In the MPC protocols we present in Theorems 1 and 2, we point out that they verify the *reusability* property of [17], defined as follows:

- **Reusability** (from [17]): Given the transcript of the input distribution phase of the protocol, the computation phase of the protocol should be able to be reused across an unbounded polynomial number of executions to compute different functions on the same fixed joint inputs of all the players.

The latter is to be compared with the weaker *delayed-function* property used until now, e.g. in [11], which roughly states that the first round messages of the honest players are computed independent of the function and the number of players.

Let us note that the concurrent work of [75] presented a result close to our Theorem 1, discovered independently, which affirms the feasibility of MPC with GOD under honest majority with a bulletin board PKI. However, we believe that our result is stronger for several reasons. Notably, we identified a miscitation [61] for their feasibility result (p10). Upon notification, they confirmed that their result was derived from [11], which carries several important implications:

1. First, their MPC protocol does not have the *reusability* property (as observed by [17])
2. Second, the communication complexity of their protocol is, as observed in [17], linear in the circuit size, when it is only proportional to the circuit depth and the number of inputs in our Theorem 1.
3. Third, broadcast is assumed in both rounds in [75, Section 6], while our Theorem 1 requires only an asynchronous $2^{nd}$ round.
4. Finally, we believe that their protocol, in its current form, is not delegable. Indeed, the computation of both round 1 and round 2 messages by a player $P_i$ requires the player's own input $x_i$. As a result, the only way an external input-owner could delegate a computation would be to reveal its input in clear text to one of the players, which compromises security. However, we do not rule out the possibility of a more sophisticated solution that enables delegation, similar to approaches like [17] or [25], albeit with the additional overhead such methods entail.

## D Examples of LHE schemes

We now give more examples of $\ell$-$\mathsf{HE}$ schemes following Definition 3.

### D.1 CL [39].

The $\ell$-$\mathsf{HE}$ of Castagnos-Laguillaumie ($\mathsf{CL}$) [39] has plaintexts in $\mathbb{Z}/p\mathbb{Z}$ but ciphertexts in a group of hidden order, hence the operations are seen as $\mathbb{Z}$-linear (the law $*$ in the target group being multiplication). In [38, §3.2] it is described how to set-up the parameters for a public common prime $p$.

We refer to [31, Fig.4] for details about how to perform a $\mathsf{DKG}$ for $\mathsf{CL}$, including a suitable secret sharing over $\mathbb{Z}$. Note that the $\mathsf{DKG}$ can be made non-interactive in one round of $\mathsf{BC}$, using $\mathsf{PVSS}$ (Section 3.2).

### D.2 GSW.

From a remote perspective, the original GSW [74] public key FHE scheme falls short from our linearity requirements. Indeed the encryptor needs to secretly compute a *non-linear* function, which takes as input the public key and some encryption randomness (namely: $\mathsf{BitDecomp}(A.R)$). Then, [10] introduced a variation of GSW which is compatible with our syntax, since encryption is now a linear map. Furthermore, they observe that their variation is lossless, i.e., a ciphertext under their variation can be transformed into a GSW ciphertext without knowing the secret decryption key. This GSW-AP variation is explicitly spelled-out in [101] ([10] described only a symmetric-key simplification) and used in [25, Appendix B]. Then, [30] and [17] used a dual version of the GSW-AP variation, which we call "GSW*". It differs from GSW only from the choices of dimensions and distributions. Below we recall the GSW-AP[13], where $\mathscr{E}$ is a distribution over $\mathbb{Z}$, $m$ an integer, $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ a fixed efficiently computable matrix and $\mathbf{G}^{-1}(.)$ an efficiently computable deterministic "short preimage" function as defined in [101, Lemma 2.1]. Moreover recall that, unlike $\ell$-BFV, there is not need for a relinearization key for performing homomorphic operations. As in Section 3.1.1, for additivity reasons we consider that the public uniform randomness $\mathbf{A} \in \mathbb{Z}_q^{(n-1)m}$ is fixed and drawn from a common URS.

- $\mathsf{GSW.KeyGen}(\mathsf{pp} = (\mathbf{A} \in \mathbb{Z}_q^{(n-1)m}))$: Sample $\mathbf{e}^{(\mathbf{pk})} \xleftarrow{\$} \mathscr{E}^m$ and $s \xleftarrow{\$} \mathbb{Z}_q^{n-1}$ and set $\mathsf{sk} = (-s, 1) \in \mathbb{Z}_q^n$, and define the linear map $\Lambda^{\mathbf{A}}_{\mathsf{EKeyGen}} : (\mathsf{sk}, \mathbf{e}^{(\mathbf{pk})}) \to (s.\mathbf{A} + \mathbf{e}^{(\mathbf{pk})}, \mathbf{A})$.

  Output $\mathsf{ek} \leftarrow \Lambda^{\mathbf{A}}_{\mathsf{EKeyGen}}(s, \mathbf{e}^{(\mathbf{pk})}) = (\mathsf{sk}.\mathbf{A} + \mathbf{e}^{(\mathbf{pk})}, \mathbf{A}) = (\mathbf{b}, \mathbf{A})$.
- $\mathsf{GSW.Enc}(\mathsf{ek} = (\mathbf{b}, \mathbf{A}),\ m \in \mathbb{Z})$: Sample $\mathbf{R} \xleftarrow{\$} \{0,1\}^{m\times m}$, and define the linear map $\Lambda^{\mathbf{A},\mathbf{b}}_{\mathsf{Enc}} : (\mathbf{R}, m) \to \left( \begin{bmatrix} \mathbf{A} \\ \mathbf{b} \end{bmatrix} \mathbf{R} + m\mathbf{G} \right)$, where $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$.

  Output $\mathsf{c} \leftarrow \Lambda^{\mathbf{A},\mathbf{b}}_{\mathsf{Enc}}(\mathbf{R}, m) \in \mathbb{Z}_q^{n\times m}$.
- $\mathsf{GSW.Dec}(\mathsf{sk}, \mathsf{c})$: Given a ciphertext $\mathsf{c} \in \mathbb{Z}_q^{n\times m}$, define a vector $\mathbf{w} = [0, \ldots, 0, \lceil q/2 \rfloor] \in \mathbb{Z}_q^n$, and $\Lambda^{\mathsf{c}}_{\mathsf{Dec}} : (sk) \to \mathsf{sk}.\mathsf{c}\mathbf{G}^{-1}(\mathbf{w}^T)$ and compute $\mu \leftarrow \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk})$.

  Output $m := \left| \left\lfloor \frac{\mu}{q/2} \right\rceil \right| = \Omega_{\mathsf{Dec}}(\mu)$.

## E More on $\mathcal{F}_{\mathsf{LSS}}$ and Secret Sharing over Rings

In this section, we first detail in Appendix E.1, what is a $(n, t)$-Linear Secret Sharing scheme (LSS) and how it can be used to design a Publicly Verifiable Secret Sharing scheme (PVSS). Then in Appendix E.2, we detail the implementation of $\mathcal{F}_{\mathsf{LSS}}$ and its security.

[13] That we denote simply as GSW for simplicity.

$\mathcal{F}_{\mathsf{LSS}}$

**Participants:** A set $\mathscr{S}$ of senders, an output learner $\mathcal{L}$, and a set $\mathscr{P}$ of players.
**Inputs** (For each $S \in \mathscr{S}$): a list $(x_{S,\alpha})_{\alpha \in X_{\mathcal{S}}}$, where each input $x_{S,\alpha}$ is identified by a unique predefined 'label' $\overline{x_{S,\alpha}}$.

**Setup**

- On input $(\mathsf{Setup})$ from any $P \in \mathscr{P}$ for the first time, or possibly from $\mathcal{A}$ if $P$ is corrupt: stores $(\mathsf{Setup}, P)$, and *eventually-delivers* $(\mathsf{Setup}, P)$ to each player $P \in \mathscr{P}$ . //eventually-delivers" consists of the same fetch-and-delay mechanism as explained in Appendix B.1.
- Initialize a counter $T_{\mathcal{A}} \leftarrow 1$ //Formalizing timing for setup of corrupt players
  - Upon receiving $\mathsf{delay{-}Setup}$ from $\mathcal{A}$, $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} + 1$.
  - Upon receiving $\mathsf{fetch}$ from any $P \in \mathscr{P}$, $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} - 1$.
  - When $T_{\mathcal{A}} = 0$ for the first time, freeze forever $T_{\mathcal{A}} = 0$. Then, send $\mathsf{ready}$ to every $\mathcal{S} \in \mathscr{S}$.

**Input** On input $(\mathsf{input}, \overline{x_{\mathcal{S},\alpha}}, x_{\mathcal{S},\alpha} \in R_q)$ from any $\mathcal{S} \in \mathscr{S}$ for the first time[a], or possibly from $\mathcal{A}$ if $\mathcal{S}$ is corrupt: first, if $x_{\mathcal{S},\alpha} = \perp$ then set it to 0, store $(\mathsf{input}, \mathcal{S}, x_{\mathcal{S},\alpha})$, and *eventually-deliver* $(\mathsf{stored}, \overline{x_{\mathcal{S},\alpha}})$[b] to each player $P \in \mathscr{P}$ .

**$\mathcal{A}$ delaying eventual delivery**

- Initialize $D \leftarrow 1$ // Delivery delay
- Upon receiving $\mathsf{delay}$ from $\mathcal{A}$, set $D \leftarrow D + 1$

**Bookkeeping requests from honest players**

- Initialize $\mathsf{HOpeners} \leftarrow \{\}$[c]
- Upon receiving $(\mathsf{LCOpen}, \mathsf{ssid} = \Lambda)$ from any *honest* player $P_i \in \mathscr{P}$, set $\mathsf{HOpeners} \leftarrow \mathsf{HOpeners} \cup \{P_i\}$, set $D \leftarrow D - 1$ and leak $(\mathsf{LCOpen}, \mathsf{ssid} = \Lambda, P_i)$ to $\mathcal{A}$.

**LCOpen**

- [Early Opening] If $|\mathsf{HOpeners}| \geq 1$ *and* if all $\overline{x_{\mathcal{S},\alpha}}$ appearing with nonzero coefficient in $\Lambda$ are stored, then,
  1. if $\mathcal{L}$ is corrupt, leak $y := \Lambda((x_{\mathcal{S},\alpha})_{\mathcal{S},\alpha})$ to $\mathcal{A}$;
  2. if $\mathcal{L}$ is honest, upon receiving $(\mathsf{open{-}order}, \Lambda)$ from $\mathcal{A}$, if no output was delivered yet to $\mathcal{L}$, then send $(\mathsf{ssid} = \Lambda, y := \Lambda((x_{\mathcal{S},\alpha})_{\mathcal{S},\alpha}))$ to $\mathcal{L}$.
- [Collective Opening] If $|\mathsf{HOpeners}| \geq t + 1$ *and* $D \leq 0$ *and* no output was delivered yet to $\mathcal{L}$, *and* if all $\overline{x_{\mathcal{S},\alpha}}$ appearing with nonzero coefficient in $\Lambda$ are stored, then send $(\mathsf{ssid} = \Lambda, y := \Lambda((x_{\mathcal{S},\alpha})_{\mathcal{S},\alpha})$ to $\mathcal{L}$.

[a]Once a sender $\mathcal{S}$ (or $\mathcal{A}$) send an input $x_{\mathcal{S},\alpha}$ with label $\overline{x_{\mathcal{S},\alpha}}$, the former cannot be subsequently updated.

[b]Appended with "$x_{\mathcal{S},\alpha} = \perp$" when this is the case.

[c]Recall that we consider in this description an unique $\Lambda$. If multiple are considered, then several sets $\mathsf{HOpeners}_{\Lambda}$ must also be considered.

Fig. 10: Sharing with Linear Combination functionality for one single linear map $\Lambda$. $\mathsf{sid}$ omitted

## E.1 Linear Secret Sharing

We now introduce the concept of linear secret sharing that will prove useful throughout this work to design multiparty schemes.

***Definition 9.*** *($(n,t)$-LSS) Let $\mathbb{R}$ be a ring. A $(n,t)$-Linear Secret Sharing Scheme is defined by the following two algorithms:*

- $\mathsf{LSS.Share}(s \in \mathbb{R}, n, t)$[14] $\rightarrow (\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)})$*: For a given secret $s \in \mathbb{R}$, the sharing algorithm generates a vector $(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)})$ of shares, where $\mathbf{s}^{(i)}$ is the share of player $P_i$.*
- $\mathsf{LSS.Reco}(\{\mathbf{s}^{(i)}\}_{i\in\mathcal{U}}, \mathcal{U}) \rightarrow s$*: For any set $\mathcal{U}$ of size $t+1$ and shares $\{\mathbf{s}^{(i)}\}_{i\in\mathcal{U}}$, the reconstruction algorithm outputs a secret $s \in \mathbb{R}$.*

*To ease notations, we define a* sharing *of some secret $s \in \mathbb{R}$ as $[s] = \{\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)}\}$.*

*Furthermore, it must satisfy the following properties:*

1. ***Correctness**: For any set $\mathcal{U}$ of size $t+1$, the value $s$ can be efficiently reconstructed from the set of shares $\{\mathbf{s}^{(i)}\}_{i\in\mathcal{U}}$, i.e. for any projection $S_{\mathcal{U}}$ of $S \leftarrow \mathsf{LSS.Share}(s,n,t)$, it holds that $\mathsf{LSS.Reco}(S_{\mathcal{U}}, \mathcal{U}) = s$ with probability 1.*

2. ***Privacy**: For all $\mathcal{V}$ such that $|\mathcal{V}| \leq t$, and secrets $s_L, s_R \in \mathbb{R}$, then the shares of $s_L$ and $s_R$ output by $\mathsf{LSS.Share}$ follow the same distribution. More formally, we have:*

$$\left\{ \{\mathbf{s}_R^{(i)}\}_{i\in\mathcal{V}} \approx \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{V}} \middle| \begin{array}{l} \wedge \{\mathbf{s}_R^{(i)}\}_{i\in[n]} \leftarrow \mathsf{Share}(s_R, n, t) \\ \wedge \{\mathbf{s}_L^{(i)}\}_{i\in[n]} \leftarrow \mathsf{Share}(s_L, n, t) \end{array} \right\} \tag{7}$$

*In other words, the set of shares $\{\mathbf{s}^{(i)}\}_{i\in\mathcal{V}}$ does not leak anything about the value $s$.*

3. ***Linearity**: Linear operations (namely additions and subtractions) can be applied on the shares of different secrets to obtain the shares of the corresponding operations applied on these secrets. Specifically, when considering two sharings $[x] = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$ and $[y] = \{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n)}\}$ of some values $x, y \in \mathbb{R}$, then $\{\mathbf{x}^{(1)} + \mathbf{y}^{(1)}, \ldots, \mathbf{x}^{(n)} + \mathbf{y}^{(n)}\}$ (resp - for the subtraction) is a sharing of $x + y$ (resp $x - y$).*

   *This notion can be generalized to any set of secret values. Consider a linear map $\Lambda$ and a set of sharings $\{[x_i]\}_{i\in S}$ of some secrets $\{x_i\}_{i\in S} \in \mathbb{R}$. Then, we have that $\{\Lambda(\{\mathbf{x}_i^{(1)}\}_{i\in S}), \ldots, \Lambda(\{\mathbf{x}_i^{(n)}\}_{i\in S})\} = [\Lambda(\{x_i\}_{i\in S})]$.*

*Moreover, for our UC proofs to go through, we require in addition the following two properties (4) and (5), which enable the simulation of shares. They are satisfied by all linear secret sharings used in practice, e.g., Shamir[108] and the $\{0,1\}$-LSSD of [82] (renamed $\{0,1\}$-LSS in [28]).*

[14]We leave implicit the randomness used for the sharing.

4. ***Simulatability****: Additionally, we require the existence of an efficient function* $\mathsf{ShSim}$ *such that for every* $\mathsf{PPT}$ *adversary* $\mathcal{A}$*, for any set* $\mathcal{V}$ *such that* $|\mathcal{V}| \le t$ *(and* $\mathcal{U} = [n] \setminus \mathcal{V}$*), and any two secrets* $s_L, s_R \in \mathbb{R}$*, for* $(\mathbf{s}_L^{(1)}, \ldots, \mathbf{s}_L^{(n)}) \leftarrow \mathsf{LSS.Share}(s_L, n, t)$, $(\mathbf{s}_R^{(1)}, \ldots, \mathbf{s}_R^{(n)}) \leftarrow \mathsf{LSS.Share}(s_R, n, t)$ *and* $\{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{U}} \leftarrow \mathsf{ShSim}(\{\mathbf{s}_R^{(i)}\}_{i\in\mathcal{V}}, s_L)$,

   (8)
   $$\left|Pr[\mathcal{A}(\{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{V}}, \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] - Pr[\mathcal{A}(\{\mathbf{s}_R^{(i)}\}_{i\in\mathcal{V}}, \{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{U}}) = 1]\right| \le negl(\lambda)$$

5. ***Inference of Shares****: Finally, we require the existence of an efficient function* $\mathsf{ShInfer}$ *such that for every* $\mathsf{PPT}$ *adversary* $\mathcal{A}$*, for any set* $(t+1)$*-sized set* $\mathcal{U}$ *(and* $\mathcal{V} = [n] \setminus \mathcal{U}$*), and any two secrets* $s_L, s_R \in \mathbb{R}$*, for* $(\mathbf{s}_L^{(1)}, \ldots, \mathbf{s}_L^{(n)}) \leftarrow \mathsf{LSS.Share}(s_L, n, t)$, $(\mathbf{s}_R^{(1)}, \ldots, \mathbf{s}_R^{(n)}) \leftarrow \mathsf{LSS.Share}(s_R, n, t)$ *and* $\{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{V}} \leftarrow \mathsf{ShInfer}(\{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}})$,

   (9)
   $$\left|Pr[\mathcal{A}(\{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{V}}, \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] - Pr[\mathcal{A}(\{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{V}}, \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}}) = 1]\right| \le negl(\lambda)$$

We now discuss a classical example of a linear secret-sharing scheme.

***Example: Shamir Secret Sharing.*** For the purpose of this section, we consider a finite field $\mathbb{F}$ and assume that each player $P_i \in \mathscr{P}$ is associated with a non-zero element $\alpha_i \in \mathbb{F}$ such that if $i \neq j$ then $\alpha_i \neq \alpha_j$. We recall the secret-sharing scheme of Shamir [108] that implements a $(n, t)$-$\mathsf{LSS}$ scheme based on polynomial interpolation in a finite field. As a reminder, let us first define what are Lagrange coefficients.

***Definition 10.*** *Given* $\mathcal{U} \subseteq [n]$ *with* $|\mathcal{U}| = t + 1$*, we denote as Lagrange coefficients the values* $\{\lambda_i^{\mathcal{U}}\}_{i\in\mathcal{U}}$ *computed as*

$$\lambda_i^{\mathcal{U}} = \prod_{j\in\mathcal{U}, j\neq i} \frac{\alpha_0 - \alpha_j}{\alpha_i - \alpha_j} \qquad (10)$$

Intuitively, Shamir uses polynomial evaluation to share a secret and interpolation to reconstruct it from shares. In more detail, to share a value $s \in \mathbb{F}$ using Shamir, the dealer samples at random a polynomial $f(Y) \in F_{\le t}[Y]$ of degree at most $t$, such that $f(0) = s$. The shares corresponding to each player $P_i$ are then define as the evaluation of $f$ in $\alpha_i$, i.e. $s^{(i)} = f(\alpha_i)$. The reconstruction of the secret is done by doing a Lagrange interpolation at $\alpha_0$ from any set of $t+1$ shares.

Formally, the scheme can be defined by the following two algorithms:

$\mathsf{Shamir.Share}(s, n, t)$: To secret-share a value $s \in \mathbb{F}$, sample $f_1, \ldots, f_t \xleftarrow{\$} \mathbb{F}$ and output $s^{(i)} = s + \Sigma_{j=1}^{t} f_j \alpha_i^j$ for all $i \in [n]$.

$\mathsf{Shamir.Reco}(\{s^{(i)}\}_{i\in\mathcal{U}},\mathcal{U})$: To reconstruct $s$ from shares $\{s^{(i)}\}_{i\in\mathcal{U}}$, compute

$$s = \sum_{i\in\mathcal{U}} \lambda_i^{\mathcal{U}} s^{(i)} \tag{11}$$

Correctness of the scheme follows from polynomial evaluation and reconstruction, while privacy intuitively follows from the fact that any set of $t$ shares does not leak anything about the secret $s$.

***Instantiation of Linear Secret Sharing over Polynomial Rings:*** Since we are working with efficient linear homomorphic encryption schemes in which the ciphertext space can be a polynomial ring $R_q$, we require an efficient $\mathsf{LSS}$ scheme over polynomial rings. We now briefly discuss possible instantiations:

- First, one can consider the $\{0,1\}-\mathsf{LSSD}$ scheme of [82] over $R_q$. [111] showed it fulfills all the properties desired in Definition 9, in particular the simulatability and the inference of shares. Note that in a similar way, the recent $\mathsf{TreeSS}$ scheme presented in [42] also satisfies the properties we seek.
- For Shamir, let us recall that it is a known result since [63] that using a ring is possible, as long as the set of Shamir public-points forms an *exceptional sequence* [5,52] as defined in Definition 11. We then consider two cases:
  - First, let us recall that such an exceptional sequence exists when the prime factor $q$ is of size at least $n+1$.
  - Second, let us note that following [5], an exceptional sequence can be built for an arbitrary modulus $q$, and therefore by Theorem 12, a Shamir secret-sharing scheme. We refer to [111] for details about this construction.

  All in all, provided with an exceptional sequence over $\mathbb{R}$, we then have the following Theorem 12.

***Definition 11.*** *From [5] For a ring $\mathbb{R}$, the sequence $\alpha_1,\ldots,\alpha_n$ of elements of $\mathbb{R}$ is an* exceptional sequence *if $\alpha_i-\alpha_j$ is a unit in $\mathbb{R}$ for all $i\neq j$.*

**Theorem 12.** *From [5] Let $\mathbb{R}$ be a commutative ring and $\alpha_1,\ldots,\alpha_n$ be an exceptional sequence in $\mathbb{R}$. Then, a Shamir secret-sharing scheme instantiated in $\mathbb{R}$ with Shamir public-points, $\alpha_1,\ldots,\alpha_n$, is correct and secure.*

### E.1.1 Publicly Verifiable Secret Sharing (PVSS)

Let $\mathsf{PKE} = (\mathsf{EKeyGen},\mathsf{Enc},\mathsf{Dec})$ be any public key encryption scheme satisfying $\mathsf{IND\text{-}CPA}$. We introduce the following definition:

***Definition 13.*** *(Publicly Verifiable Secret Sharing ($\mathsf{PVSS}$)) Let us consider the following randomized function $\mathsf{PVSS}$, parametrized by $n$ strings $(\mathsf{pk}_i^{\mathsf{PKE}})_{i\in[n]}$. On input $s\in R_q$ compute $(\mathbf{s}^{(1)},\ldots,\mathbf{s}^{(n)}) \leftarrow \mathsf{LSS.Share}(s)$, and output $\left[\mathsf{PKE.Enc}(\mathsf{pk}_i^{\mathsf{PKE}},\mathbf{s}^{(i)})\right]_{i\in[n]}$ along with a NIZK proof $\pi$ of correct sharing for*

*the following relation:*
(12)

$$\mathscr{R}_{\mathsf{Share}} = \left\{ \begin{array}{l|l} x = (\{\mathsf{pk}_i^{\mathsf{PKE}}\}_{j\in[n]}, \mathsf{enc{-}shares}) & \wedge \{\mathbf{s}^{(i)}\}_{i\in[n]} \leftarrow \mathsf{LSS.Share}(s;\mathbf{r}) \\ w = (s, \mathbf{r}, \{\rho_i\}_{i\in[n]}) & \wedge \mathsf{enc{-}shares} \leftarrow [\mathsf{Enc}(\mathsf{pk}_i^{\mathsf{PKE}}, \mathbf{s}^{(i)}; \rho_i)]_{i\in[n]} \end{array} \right\}$$

PVSS *is* IND-CPA *for any $\mathcal{A}$ being given at most $t$ secret keys* $\left(\mathsf{dk}_i^{\mathsf{PKE}}\right)_{i\in\mathcal{I}\subset[n],|\mathcal{I}|\le t}$*, as will be shown in Proposition 14.*

*Remark 7.* By convention, encryption under an incorrectly formatted public key $\mathsf{pk}^{\mathsf{PKE}}$, e.g., $\perp$, returns the plaintext itself.

*Remark 8.* We describe the MPC protocols in Section 5 in the semi-malicious model, for which the NIZK proof can be dropped. This leads to the manipulation of new structures, denoted as Public Secret Sharing, namely a PVSS without a proof of correctness.

***Proof of* IND-CPA *of Publicly Verifiable Secret Sharing.*** Proposition 14 states that any PPT adversary $\mathcal{A}$ corrupting at most $t$ players, has negligible advantage in distinguishing between the encrypted $(n,t)-$LSS sharings of any two chosen secrets $(s_L, s_R) \in R_q^2$.

***Proposition 14 (*IND-CPA *of encrypted sharing).*** *For any integers $0 \le t \le n$, we consider the following game between an adversary $\mathcal{A}_{\mathsf{PVSS}}$ and an oracle $\mathscr{O}$. $\mathscr{O}$ is parametrized by a secret $b \in \{L, R\}$ (left or right oracle).*

$\underline{\mathsf{Game}^{\mathcal{A}}_{\mathsf{IND-PVSS}}}$

**Setup.** *$\mathcal{A}_{\mathsf{PVSS}}$ gives to $\mathscr{O}$: a subset of $t$ indices $\mathcal{I} \subset [n]$, and a list of $t$ public keys $(\mathsf{pk}_i)_{i\in\mathcal{I}} \in (\mathscr{P}k \sqcup \perp)^t$. For each $i \in [n]\backslash\mathcal{I}$, $\mathscr{O}$ generates $(\mathsf{dk}_i, \mathsf{pk}_i) \leftarrow \mathsf{EKeyGen}(1^\lambda)$ and shows $\mathsf{pk}_i$ to $\mathcal{A}_{\mathsf{PVSS}}$.*

**Challenge.** *$\mathcal{A}_{\mathsf{PVSS}}$ is allowed to query $\mathscr{O}$ an unlimited number of times as follows. $\mathcal{A}_{\mathsf{PVSS}}$ gives to $\mathscr{O}$ a pair $(s_L, s_R) \in R_q^2$. Depending on $b \in \{L, R\}$, $\mathscr{O}$ replies as either $\mathscr{O}^L$ or $\mathscr{O}^R$:*

- $\mathscr{O}^{\boldsymbol{L}}$**:** *computes $(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)}) \leftarrow \mathsf{Share}(s_L)$ and returns $(\mathsf{Enc}_{\mathsf{pk}}(\mathbf{s}^{(i)}))_{i\in[n]}$*
- $\mathscr{O}^{\boldsymbol{R}}$**:** *computes $(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)}) \leftarrow \mathsf{Share}(s_R)$ and returns $(\mathsf{Enc}_{\mathsf{pk}}(\mathbf{s}^{(i)}))_{i\in[n]}$.*

**Guess.** *$\mathcal{A}_{\mathsf{PVSS}}$ gets some $(\mathsf{Enc}_{\mathsf{pk}}(\mathbf{s}^{(i)}))_{i\in[n]}$ and outputs $b' \in \{L, R\}$. It wins if $b' = b$.*

Fig. 11: IND-CPA of encrypted sharing

*At some point $\mathcal{A}_{\mathsf{PVSS}}$ may output a string, e.g., a bit. Then for any* PPT *machine $\mathcal{A}_{\mathsf{PVSS}}$, we want to show that the* distinguishing advantage $\mathsf{Adv}_{L,R} = |Pr(1 \leftarrow \mathcal{A}^{\mathscr{O}}_{\mathsf{PVSS}}) - Pr(1 \leftarrow \mathcal{A}^{\mathscr{O}}_{\mathsf{PVSS}})|$ *is negligible.*

*Proof.* Overall, the intuition of proof is that, under the idealized assumption that $\mathsf{PKE}$ ciphertexts under the unknown $t+1$ public keys would perfectly hide their content, then, the view of the adversary is the vector of $t$ plaintext shares $\{\mathbf{s}^{(i)}\}_{i\in\mathcal{I}}$. In particular, for any value $s \in R_q$, we have that, in a $\mathsf{PVSS}$ of $s$, the $t+1$ coordinates $\left[\mathsf{Enc}(\mathsf{pk}_i^{\mathsf{PKE}}, \mathsf{s}^{(i)}),\ i \in [n]\backslash\mathcal{I}\right]$, which are encrypted under the honest keys $\mathsf{pk}_i^{\mathsf{PKE}}$, perfectly hide the plaintext coordinates $s^{(i)}$ to the adversary.

We refer to [111] for a detailed proof of this proposition. □

## E.2 Implementation of $\mathcal{F}_{\mathsf{LSS}}$.

We now detail in Fig. 12 protocol $\Pi_{\mathsf{LSS}}$ that instantiates $\mathcal{F}_{\mathsf{LSS}}$ in the $(\mathsf{BC}, \mathcal{F}_{\mathrm{AT}}, \mathsf{bPKI})$-hybrid model. Recall from Section 3.2 that we consider a set $\mathscr{P}$ of $n$ players, a set $\mathscr{S}$ of senders, and an output learner $\mathcal{L}$.

***Proposition 15.*** *Protocol $\Pi_{\mathsf{LSS}}$ UC implements $\mathcal{F}_{\mathsf{LSS}}$*

$\Pi_{\mathsf{LSS}}$

**Parameters:** Any $\mathsf{PKE} = (\mathsf{EKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and $\mathsf{LSS} = (\mathsf{Share}, \mathsf{Reco})$ and, from them, the $\mathsf{PVSS}$ algorithm detailed in Definition 13.
**Participants:** A set $\mathscr{S}$ of senders, an output learner $\mathcal{L}$, and a set $\mathscr{P}$ of $n$ players.

$\Pi_{\mathsf{LSS}}$**.Setup** : $\forall P \in \mathscr{P}$: $(\mathsf{dk}_P^{\mathsf{PKE}}, \mathsf{pk}_P^{\mathsf{PKE}}) \leftarrow \mathsf{PKE.EKeyGen}(1^\lambda)$, send $(\mathsf{Register}, \mathsf{pk}_P^{\mathsf{PKE}})$ to $\mathsf{bPKI}$.

$\Pi_{\mathsf{LSS}}$**.Input** :

- Each sender $\mathcal{S} \in \mathscr{S}$ sets $(\mathsf{pk}_P^{\mathsf{PKE}})_{P\in\mathscr{P}}$ as the output delivered by $\mathsf{bPKI}$. For each $\alpha \in X_{\mathcal{S}}$:
  - Compute $\mathsf{enc{-}shares}_{\mathcal{S},\alpha}, \_ := \mathsf{PVSS}\big((\mathsf{pk}_P^{\mathsf{PKE}})_{P\in\mathscr{P}}, x_{\mathcal{S},\alpha}\big)$.
  - Broadcast $(\mathsf{input}, \mathsf{ssid} := \overline{x_{\mathcal{S},\alpha}}, \mathsf{enc{-}shares}_{\mathcal{S},\alpha})$ over $\mathsf{BC}^{\mathcal{S}}$.
- $\forall P_j \in \mathscr{P}$, upon receiving outputs from all sub-instances of all $\mathsf{BC}^{\mathcal{S}}$ whose label $\mathsf{ssid} = \overline{x_{\mathcal{S},\alpha}}$ has nonzero coefficient in $\Lambda$: for each output $(\overline{x_{\mathcal{S},\alpha}}, *)$, if $* = \bot$ then set $x^{(j)} := 0$; else if $* = [c_{\mathcal{S},\alpha}^{(1)}, \ldots, c_{\mathcal{S},\alpha}^{(n)}]$ then set $x_{\mathcal{S},\alpha}^{(j)} := \mathsf{PKE.Dec}(\mathsf{dk}_j^{\mathsf{PKE}}, c_{\mathcal{S},\alpha}^{(j)})$.

$\Pi_{\mathsf{LSS}}$**.LCOpen($\Lambda$)** :

- Upon calling $\mathsf{LCOpen}$, each player $P_j \in \mathscr{P}$ evaluates $\mu^{(j)} := \Lambda\big((x_{\mathcal{S},\alpha}^{(j)})_{\mathcal{S},\alpha}\big)$ and sends it over $\mathcal{F}_{\mathrm{AT}}^{P,\mathcal{L}}$ to $\mathcal{L}$.
- Upon receiving opening shares $(\mu^{(i)})_{i\in\mathcal{U}}$ from any $(t+1)$-set $\mathcal{U} \subset [n]$ of players, outputs $\mu := \mathsf{LSS.Reco}\big(\mu^{(i)})_{i\in\mathcal{U}}, \mathcal{U}\big)$.

Fig. 12: Protocol for secret-sharing then linear combination

*Proof.* For simplicity, we construct a simulator for an honest $\mathcal{L}$[15] and the opening of only one evaluation of one linear map. The case of multiple openings is handled as in [52, p127], when they simulate each new $\mathsf{Open}$.

[15] The case where the output learner is corrupt is easy. Namely, the simulator plays $\Pi_{\mathsf{LSS}}$ honestly, then indistinguishability follows from the correctness of $\Pi_{\mathsf{LSS}}$.

The UC security property follows from four hybrids. The first two, $\mathsf{Hybrid}^{\mathsf{ShSim}}$ and $\mathsf{Hybrid}^{\mathcal{F}}$, replace the opening shares of honest players of the output of the protocol $\Pi_{\mathsf{LSS}}$, by ones simulated out of the actual evaluation of a linear map $\Lambda$. They are indistinguishable from the real execution, by simulatability of openings and by correctness of $\Pi_{\mathsf{LSS}}$. Then, $\mathsf{Hybrid}^{\mathsf{0Share}}$ replaces the input of a simulated honest senders by 0. Finally, $\mathsf{Hybrid}^{\mathsf{ShInfer}}$ changes the way opening shares of corrupt players are obtained.

***Game $REAL_{\mathcal{A}}$.*** This is the actual execution of the protocol $\Pi_{\mathsf{LSS}}$ with adversary $\mathcal{A}$ fully controlled by $\mathsf{Env}$ (and ideal functionalities $\mathsf{bPKI}, \mathcal{F}_{\mathrm{AT}}, \mathsf{BC}$).

***Game $\mathsf{Hybrid}^{\mathsf{ShSim}}$.*** (Skipped if $\mathcal{L}$ is honest.)

In this hybrid, we change the method of computation of the opening shares of honest players. To do so, we first define quantities denoted *Inferred Corrupt Opening Shares* $(\mu^{(i)})_{i\in\mathcal{I}}$, nonwithstanding corrupt players may not have any opening shares on their witness tapes, since they may not send any.

For every input $x_{\mathcal{S},\alpha}$ of some honest $\mathcal{S}$, we simply define $\big(x^{(i)}_{\mathcal{S},\alpha}\big)_{i\in\mathcal{I}}$ as the actual shares produced by $\mathcal{S}$ when it computes the $\mathsf{PVSS}$ of $x_{\mathcal{S},\alpha}$.

For each output $(\overline{x_{\mathcal{S},\alpha}}, *)$ of $\mathsf{BC}^{\mathcal{S}}$ from some corrupt $\mathcal{S}$: (i) if $* = \perp$ then we define $\big(x^{(i)}_{\mathcal{S},\alpha} := 0\big)_{i\in\mathcal{I}}$, otherwise (ii) this implies that $*$ is a correctly formed $\mathsf{PVSS}$. Thus in this case, we define as $\big(x^{(i)}_{\mathcal{S},\alpha}\big)_{i\in\mathcal{I}}$ the plaintext shares read on the witness tape of $\mathcal{S}$.

For all $i \in \mathcal{I}$ we set $\mu^{(i)} := \Lambda\big((x^{(i)}_{\mathcal{S},\alpha})_{\alpha\in X_{\mathcal{S}},\mathcal{S}\in\mathscr{S}}\big)$. By linearity of the $\mathsf{LSS}$ scheme, they are equal to the opening shares of $\widetilde{y}$ that the $(P_i)_{i\in\mathcal{I}}$ would have sent if they were honest. Finally, we generate the opening shares of honest players as $\mathsf{ShSim}(\widetilde{y}, (\mu^{(i)})_{i\in\mathcal{I}})$.

***Claim 16.*** $REAL_{\mathcal{A}} \equiv \mathsf{Hybrid}^{\mathsf{0Share}}$.
<u>Proof</u>: *Since $\mathsf{ShSim}^{\mathcal{I}}$ simulates perfectly, they are identical to the ones of the Real execution.*

***Game $\mathsf{Hybrid}^{\mathcal{F}_{\mathsf{LSS}}}$.*** (Skipped if $\mathcal{L}$ is honest.) This game differs from $\mathsf{Hybrid}^{\mathsf{ShSim}}$ in that the input $\widetilde{y}$ to $\mathsf{ShSim}$ is replaced by the actual $y$ leaked by $\mathcal{F}_{\mathsf{LSS}}$.

***Claim 17.*** $\mathsf{Hybrid}^{\mathsf{0Share}} \equiv \mathsf{Hybrid}^{\mathcal{F}}$.
<u>Proof</u>: *By correctness of $\Pi_{\mathsf{LSS}}$, $y = \widetilde{y}$ so the view of $\mathsf{Env}$ is unchanged.*

***Game $\mathsf{Hybrid}^{\mathsf{0Share}}$.*** We modify $\mathsf{Hybrid}^{\mathcal{F}}$ in that each simulated honest sender plays the protocol as if it had input 0 instead of $x$.

***Claim 18.*** $\mathsf{Hybrid}^{\mathcal{F}} \equiv \mathsf{Hybrid}^{\mathsf{0Share}}$.
<u>Proof</u>: *Since the private decryption keys $\mathsf{dk}_h$ of all honest players $h \in \mathcal{H}$ are not used anymore, we have that the $\mathsf{IND\text{-}CPA}$ property of $\mathsf{PVSS}$ stated in Proposition 14 applies. Thus the view of $\mathsf{Env}$ is indistinguishable from the one in the previous game.*

***Game*** $\mathsf{Hybrid}^{\mathsf{ShInfer}}$***.*** If $\mathcal{L}$ is honest, this game is identical to $\mathsf{Hybrid}^{\mathsf{0Share}}$. Else (if $\mathcal{L}$ is corrupt), we now modify the method to *Infer* the corrupt shares of the $\mathsf{enc{-}shares}_{\mathcal{S},\alpha}$ broadcast by corrupt senders $\mathcal{S}$. First, decrypt the honest shares of $\mathsf{enc{-}shares}_{\mathcal{S},\alpha}$ using, again, the honest private keys $(\mathsf{dk}_h)_{h\in\mathcal{H}}$. From them, compute the opening shares $\{\widetilde{\mu^{(i)}}\}_{i\in\mathcal{I}}$ and use them to infer the corrupt shares using $\mathsf{ShInfer}^{\mathcal{H}}$.

***Claim 19.*** $\mathsf{Hybrid}^{\mathsf{0Share}} \equiv \mathsf{Hybrid}^{\mathsf{ShInfer}}$.
*Proof: The inferred shares are identical to the ones in the previous game, by the property of* $\mathsf{ShInfer}^{\mathcal{H}}$.

What we have achieved is a simulator that interacts only with the environment and with the ideal functionality of linear combination computation, so this concludes the proof.

□

## F Further Details on the Proof of Theorem 2

### F.1 Smudging Lemma

The following lemma states that two distributions differing by a small noise, can be made indistinguishable by adding an exponentially larger "smudging" noise to both. Its parameters were recently improved in [54, Lemma 2.3], in our use-case where the smudging noise comes itself as the sum of several contributions (sampled uniformly by honest players).

***Lemma 20 (Smudging lemma [14]).*** *For $B_1, B_2$ positive integers and $e_1 \in [-B_1, B_1]$ a fixed integer, sample $e_2$ uniformly at random in $[-B_2, B_2]$. Then the distribution of $e_2$ is statistically indistinguishable from that of $e_2 + e_1$ if $B_1/B_2 = \epsilon$, where $\epsilon = \epsilon(\lambda)$ is a negligible function.*

### F.2 Semi-malicious Security to Malicious Security

At a high level, malicious security can be achieved by applying the compiler of [14, §E], i.e., by instructing players and owners to append NIZK to their messages, to prove knowledge of a witness explaining them. But the compiler of [14] is designed for broadcast-based protocols, whereas in ours, players in $\textcircled{2}$ and $\textcircled{3}$ also act based on previous outputs of $\mathcal{F}_{\mathsf{LSS}}$. This is why we also required semi-malicious players to explain their messages based on these outputs, in our adapted model in Appendix B.9. We can also simplify their compiler by allowing players to prove their statements with UC NIZKs, recalled in Appendix B.7, since these can be set-up under honest majority from one initial call to $\mathsf{bPKI}$, thanks to the technique denoted Multi-String CRS [76,17]. On the face of it, this call pre-pends one more step before the publication of keys on $\mathsf{bPKI}$. However, we can actually have players publish multi-strings *in parallel*. Indeed in our semi-malicious model Appendix B.9 we did not impose any condition when publishing on bPKI. Multi-Strings instead of $\overline{\mathcal{G}}_{\mathrm{URS}}$-based NIZKs have the merits (i) to relieve Owners from the need to access $\overline{\mathcal{G}}_{\mathrm{URS}}$ when constructing their NIZKs, and (ii) to preserve $\overline{\mathcal{G}}_{\mathrm{URS}}$ as a global resource, which would otherwise have needed to be simulated if used to produce NIZKs: see Appendix B.7.

## G YOSO MPC

A recent line of research [72,105,6] studies MPC with specialized computation models, that support a *dynamically* evolving set of players, i.e. where participants can join and leave the computation as desired, without interrupting the protocol. The rationale is that players may devote only a limited amount of time (and computational resources) to a computation that can last a long period. This computation model was captured by Gentry et al. [72] under the name **YOSO** (You Only Speak Once), where the computation is divided into a number of successive steps $d = 1, \ldots$. For each one of them, a committee $\mathcal{C}^{(d)}$ of *ephemeral players* carries out some local computation, each of them publishes a single message on a bulletin board, and then vanishes from the system. An attractive consequence of this model is the drastic reduction of the window for adaptive corruption of these players due to the unpredictable selection of committees of players for the computation. Unfortunately, this model inherently requires some form of consistent terminating broadcast (BC) at the end of each computation step, often modeled as a public ledger [72]. Roughly, this is needed for players to perform their computation on the same intermediary values.

It is not difficult to imagine building a YOSO MPC protocol from our Share&Shrink protocol in one single BC instantiated from an $\ell$-FHE scheme presented in Section 5.3. Indeed, we have already dissociated the input-owners from the players, so the former can distribute their inputs in only one message without knowing the threshold encryption key. In order to fully move to the YOSO model, we can further dissociate the participants in the 3 rounds of the original protocol, by introducing the following committees:

- a key generation committee $\mathcal{C}^{(\mathsf{DKG})}$, whose members broadcast their contributions to the encryption key and PVSSs of their decryption keys;
- an input-owner committee $\mathcal{C}^{(\mathsf{owners})}$, whose members broadcast *in parallel* with $\mathcal{C}^{(\mathsf{DKG})}$, PVSSs of their inputs and encryption randomnesses;
- a shrinking committee $\mathcal{C}^{(\mathsf{Shrink})}$, which encrypts the distributed inputs under a common threshold encryption key, by sending opening shares for some linear form via asynchronous P2P channels to the next committee;
- and finally, a decryption committee $\mathcal{C}^{(\mathsf{Dec})}$, whose members locally perform the evaluation and the threshold decryption in one step of P2P asynchronous messages.

We refer to [91] for further discussion about adaptions to be made to move to the YOSO setting.

## H Experimental Parameters

In Table 3, we recall the candidate parameter set described in [89, Table 2], that achieves at least 128-bit of security level according to LWE-estimator [8]. We also use the error distribution $\Psi_q$ with $\sigma = 3.2$.

| $\log k$ | $\log d$ | $\log q$ | l |
|---|---|---|---|
| 16 | 14 | 438 | 8 |

Table 3: Experimental cryptographic parameters: Overview

## I End of the proof of Theorem 5

***Choosing $P_i$ and its joint action with $\mathcal{A}$.*** By $(B, M)$, where $M = (M_1, \dots, M_n)$, we denote the joint distribution of broadcast and messages sent by $P_n$ in round 1. By $(B^0, M^0)$ and $(B^1, M^1)$ we denote the *honest* distributions for inputs $x_n = 0$ and $x_n = 1$. For a distribution $(B, M)$, and $\sigma \in \{0, 1\}$, let $q_\sigma(B, M)$ be the probability that the protocol's output second entry is 1, given that:
(1) $x_1 = \sigma$,
(2) $P_n$'s messages in round 1 are $(B, M)$, then $P_n$ becomes silent,
(3) all players in $\mathcal{Q}$ follow the protocol, with their messages delivered in round-robin order, while players in $\mathcal{Q}' \backslash \mathcal{Q}$ play honestly in round 1 and then become silent.

Finally, let $q(B, M) := (q_0(B, M), q_1(B, M))$

***Lemma 21.*** *$(B^0, M_1^0)$ and $(B^1, M_1^1)$ are computationally indistinguishable.*

*Proof.* Assume a distinguisher $\mathcal{D}$ with non-negligible advantage. Then we could construct an adversary $\mathcal{A}_1$ corrupting $P_1$, but not $P_n$, rushing to learn $(B^0, M_1^0)$ before other players, submit it to $\mathcal{D}$, obtain an estimate of $x_n$, and choose $P_1$'s input equal to this estimate, obtaining a non-negligible correlation. □

***Corollary 22.*** $\forall \sigma \in \{0, 1\}, \quad q_\sigma(B^0, M_1^0) - q_\sigma(B^1, M_1^1) = \mathsf{negl}.$

*Proof.* For each, we build a distinguisher $\mathsf{Env}$ for the distributions of Lemma 21 as follows. Simulate a set $\mathscr{P}$ of players, all starting with a blank state excepted $P_1$ with input $\sigma$. Upon receiving a challenge $(B^b, M_1^b)$, make $P_n$ broadcast $B^b$ and send $M_1^b$, while the other players play the first round honestly. Then silence all players not in $\mathcal{Q}$, simulate an execution complete for $\mathcal{Q}$ with messages delivered in round-robin order, and output the same $b := x_n$ as players. □

*Remark 9.* This leverages the lack of private correlated randomness setup. Otherwise, with a bulletin board PKI, $\mathsf{Env}$ would need to initialize players with an internal state compatible with their public keys, somehow guessing their secret keys. Such an $\mathsf{Env}$ is impossible since *there exists* a secure $n$-$\mathsf{SB}$ under a bulletin board PKI setup. Namely: players would broadcast PVSSs of their inputs, then threshold-decrypt them over asynchronous channels).

Consider now the following four pairs of probabilities:

$$\begin{aligned} Q_1 &= q(B^1, M_1^1, \ldots, M_{n-t}^1, 0, \ldots, 0) \\ Q_2 &= q(B^1, M_1^1, 0, \ldots, 0, \ldots, 0) \\ Q_3 &= q(B^0, M_1^0, 0, \ldots, 0, \ldots, 0) \\ Q_4 &= q(B^0, M_1^0, \ldots, M_{n-t}^0, 0, \ldots, 0) \end{aligned}$$

**Claim:** By the protocol's correctness, we have that $Q_1 \geq 1 - \mathsf{negl}$, where the latter notation means that both entries of the $Q_1$ are $\geq 1 - \mathsf{negl}$.
[*Proof of the Claim* The view of honest players in $\mathcal{Q}$ under $(B^1, M_1^1, \ldots, M_{n-t}^1, 0, \ldots, 0)$ is indistinguishable from $(B^1, M_1^1, \ldots, M_n^1)$. Recall that in the latter case, they must ultimately output by what was observed above: "for each of these quorums (in our case: $\mathcal{Q}$), when all its members behave honestly and do not hear from the outside (in our case: $(Q' \setminus Q) \cup P_n$), then its honest members must output in a finite number of steps.". Furthermore, honest players in $\mathcal{Q}$ cannot tell apart if $P_n$ is corrupt and silent after the first round; or, honest with input 1 and its messages delayed for very long. Because of the latter possibility, by the correctness of SB, their output must be 1.]

**Claim bis: Symmetrically we have $Q_4 \leq \mathsf{negl}$.**

But by Corollary 22, $Q_2 - Q_3 = \mathsf{negl}$. Hence, there is a substantial difference either between $Q_1$ and $Q_2$ or $Q_3$ and $Q_4$. W.l.o.g., we assume the former, i.e., that $|Q_1 - Q_2| \geq 1/3$ (the other cases are similar). By a hybrid argument, we have existence of a $i \in [2, \ldots, n-t]$ such that $|q_0(B^1, M_1^1, \ldots, M_i^1, 0, \ldots, 0) - q_0(B^1, M_1^1, \ldots, M_{i-1}^1, 0, \ldots, 0)| \geq 1/(3(n-t))$. It follows, that one of the two $q_0$ probabilities above must be different by at least $1/(6(n-t))$ from one of the two corresponding $q_1$ probabilities, e.g., w.l.o.g.:

$$|q_0(B^1, M_1^1, \ldots, M_i^1, 0, \ldots, 0) - q_1(B^1, M_1^1, \ldots, M_{i-1}^1, 0, \ldots, 0)| > \frac{1}{6(n-t)} \tag{13}$$

***Back to the main strategy.*** In round 1, $\mathcal{A}$ sends $(B^1, M_1^1, \ldots, M_i^1, 0, \ldots, 0)$. In the asynchronous phase, after having learned $x_1$, once $\mathcal{Q}'$ is frozen, and upon unfreezing $\mathcal{Q}$, then if $x_1 = 0$, $\mathcal{A}$ instructs $P_i$ to play honestly; else if $x_1 = 1$, $P_i$ is instructed to play as if it had not received $M_i$ but otherwise honestly. In the first case, the view of honest players follows the distribution which defines the $q_0$ on the left-hand in (13), while in the latter case, follows the distribution which defines the $q_1$ on the right-hand in (13). Thus, the joint action of $P_i$ and $P_n$ will have for effect to substantially correlate the second output $x_n$ with $x_1$.