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

Antoine Urban and Matthieu Rambaud

Télécom Paris, Institut Polytechnique de Paris

Version 2 - 9 June 2023[1]

**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 which operate in 1 single initial round of broadcast (BC), followed by some steps of asynchronous peer-to-peer (P2P) messages. The power of closely related "hybrid networks" was studied in [Fitzi-Nielsen, Disc'09], [Beerliova-Hirt-Nielsen, Podc'10], [Patra-Ravi, IEEE Trans. Inf. Theory'18] and [Choudhury, Podc'20]. Interest of such protocols is that they go at the actual speed of the network, and their security is preserved under arbitrary network conditions (past the initial broadcast).

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 on multi-key fully homomorphic encryption [BJMS, Asiacrypt'20], to state feasibility of MPC in a tight 1 BC then 1 single step of asynchronous P2P.

We then consider efficiency. The only protocols which can be adapted to tolerate such network model and setup are [Gordon-Liu-Shi, Crypto'15] and [BJMS, Asiacrypt'20]. The former does not allow inputs from external lightweight owners and is inherently limited to the GSW FHE, while the sizes of the ciphertexts of the latter are quadratic in the number of input owners.

Our main contribution is a very simple and generic design which enables MPC in 1BC-then-asynchronous P2P. It operates over ciphertexts encrypted over a (threshold) single-key encryption scheme. Hence, they have the smallest sizes expectable. It operates from any public key encryption scheme with a key generation, encryption and decryption which are built from linear maps (such as GSW, BFV, CL). Our main building block is the squishing in the BC of both the publicly verifiable sharing of the inputs ("Share"), in parallel with distributed key generation (DKG), then followed by threshold encryption ("Shrink") in one step of asynchronous P2P. As a bonus, this design allows inputs from possibly lightweight external owners.

We then aim at instantiating the design from the BFV FHE, but surprisingly there exists no robust threshold BFV scheme. Precisely, all existing protocols for generating a common relinearisation key can abort as soon as one player deviates. We solve this issue, with a relinearisation key (adapted from [CDKS, CCS'19]) which we show how to securely generate in parallel of the threshold key, in the same broadcast. We thus obtain the first robust threshold BFV. We believe that this contribution is of independent interest.

Of independent interest, as an optional alternative, we propose the first threshold FHE decryption enabling simultaneously: (i) robustness under asynchrony with honest majority; (ii) tolerating a power-of-small-prime ciphertext modulus, e.g., $2^e$; and (iii) secret shares of sizes quasi-independent of $N$.

1 Introduction . . . . . 2
2 Model . . . . . 7
3 Cryptographic Ingredients . . . . . 9
4 Share & Shrink: DKG & Encrypted Input Distribution in 1 BC + 1 Async. P2P . . . . . 12
5 MPC Protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ . . . . . 14
6 $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ from $\mathsf{BFV} + \mathsf{CDKS}^*$: the first robust threshold $\mathsf{BFV}$ scheme. . . . . 16

[1] Framework enlarged to various linearly homomorphic encryption schemes (GSW, CL) in addition to BFV. The Theorem 1 (feasibility of MPC in 1BC + 1 async P2P) has been imported from the Section 4.4.1 of eprint 2021/503 (8 November 2021 version).

7 Proofs of Theorems 1 to 3 . . . . . 18
8 Impossibility of 1-Broadcast-then-Asynchronous MPC . . . . . 19
A Further Details on Related Works . . . . . 24
B Model: Further Formalism and Discussion . . . . . 26
C More on $\mathcal{F}_{\mathsf{LSSS}}$ and Secret Sharing over Rings . . . . . 31
D Complements on $\mathsf{BFV} + \mathsf{CDKS}^*$ . . . . . 35
E $\mathsf{BFV} + \mathsf{CDKS}^*$ Bootstrapping . . . . . 42
F Detailed Protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ when instantiated from $\mathsf{BFV} + \mathsf{CDKS}^*$ . . . . . 43
G Practical Parameters Estimation . . . . . 43
H Further Details on the Proof of Theorem 3 . . . . . 45

## 1 Introduction

Byzantine broadcast (BC) and round-by-round synchronous communication, are handy abstractions for designing simple multi-party computation protocols. However both of them are costly to implement. For instance, due to the loss of security when messages are not delivered within a round, protocol implementations must set very high the duration of a round (typically [AMN+20] 50× larger than the actual network delay, in order to tolerate slowdowns). Implementing BC from such protocols fails beyond $t < n/3$ corruptions if the network cannot deliver a number of consecutive synchronous rounds. An example is the protocol of [SBKN21] for the closely related primitive of consensus, without distributed key generation (DKG) setup, which requires an expected 48 rounds, as shown in their Table 2. Thus significant effort is currently put to remove or minimize the use of BC and synchronous rounds of communications in MPC [FN09; BHN10; GGOR13; PR18; CGZ20; GJPR21; DMR+21; DRSY23]. We push further this line of work by asking for only one initial call to BC without any 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* [CGHZ16; PR18]. 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 [BHN10].

A mainstream tool for round-efficient MPC is based on $(N, t)$-*Threshold Fully Encryption schemes* (ThFHE). The following generic approach, or close variations ([BGG+18, §6.2]), is highly mainstream. It may be called "DKG-then-Input-distribution", and consists of the following steps:

**0. Setup,** i.e., the generation of common public parameters;

**1. DKG,** i.e., the generation of a common public encryption key ek, denoted *threshold key*, along with the private assignment to each player of a secret *decryption key share* such that ciphertexts have IND-CPA for any adversary controlling up to $t$ key shares;

**2. Input distribution,** i.e., the broadcast of encrypted inputs under the threshold key;

**3. Evaluation & Partial decryption,** i.e., the evaluation of a circuit through an algorithm Eval on the encrypted inputs, that outputs a ciphertext. Either Eval is non-interactive (FHE), or interactive gate-by-gate in a CDN ([CDN01]) manner [BHN10]. The ciphertext output is then used along with a key share to produce a *partial decryption*.

Finally an algorithm that takes any $t+1$ correct partial decryptions of any ciphertext and combines them into the plaintext. However this approach does not match our goal, since implementing DKG would require at the very least one more BC [FS01], hence a total of at least two BC instead of one. Of course it would be possible to further open the box of DKG under honest majority [SBKN21]. But inside there are several synchronous rounds, due to the primitive called consensus (or MVBA), which is essentially equivalent to broadcast. So we aim at a different approach. Let us further illustrate that sometimes the DKG is not addressed, as in [BHN10; AJL+12][BGG+18, §5], or, is established by a single trusted entity, as in [BGG+18, §6.2], thereby preventing security under honest majority. Even when a threshold key is set-up, aborting players can delay [AJL+12] or completely prevent output delivery [KJY+20; MTBH21; Par21] in some protocols (see Table 2, §1.2 and §A). An orthogonal technique denoted multi-key (MFHE) [CDKS19; BJMS20], enables joint

evaluation of encryptions under different keys without a DKG. But the sizes of ciphertext are at least linear in the number $|\mathscr{L}|$ of input owners, as well as their homomorphic evaluation complexity (since [KKL+22; KÖA23]).

What we see from the previous DKG-then-Input-distribution approach, is that broadcasts are paid for the goal of providing players with *a common view on threshold-encrypted inputs*. In a breakthrough approach, [GLS15], achieved this task in one single BC (plus preliminary publication of input-independent material on a bulletin board PKI). Then, the very original approach of [BJMS20] achieves the same, and furthermore, as we are going to state in Theorem 1, they allow to incorporate in the computation the inputs of external lightweight input owners not participating to the MPC. However both these approaches suffer from a number of limitations, which impacts the works using them, e.g., [GPS19]. The sizes of the (multi-key) ciphertexts of [BJMS20] are quadratic in the number $|\mathscr{L}|$ of input owners. The sizes of the ciphertexts of [GLS15] are $n\times$ larger than those under the BFV FHE [FV12] ($n$ the lattice dimension, see Table 2). These sizes are inherited from [GSW13]. To achieve better sizes would require a RLWE-based encryption. But the approach of [GLS15] is not portable to RLWE, since it relies on the subset-sum Lemma of [Reg05] (see Appendix A). Moreoever, [GLS15] cannot enable inputs from lightweight owners (see Section 1.2).

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

### 1.1 Results

Before we move to our main contributions, we 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 Section 1.1.1.

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

**Theorem 1.** *There exists a MPC protocol with guaranteed output delivery (GOD) under honest majority ($N = 2t + 1$), under the sole setup of a bulletin board PKI; which furthermore enjoys (i) termination in* 1BC*-then-*1 *step of asynchronous P2P messages; (ii) allows inputs from external owners, i.e., which do not take part in the computation.*

The baseline is the protocol of [BJMS20]. As stated, it does not have properties (i) nor (ii). We show in Section 7.4 how to modify it to obtain these claimed properties.

| Comm. \ Setup | DKG [2] | bPKI + URS | bPKI | No setup |
|---|---|---|---|---|
| 1 BC + 1 Sync P2P | | [GLS15]+[DMR+21] ✓ ← | ✓ [BJMS20] + **Thm. 1** | [GIKR02] ✗ |
| 1 BC + $\infty$ Asynch P2P | [BHN10] ✓ | → ✓ | | **Thm. 6** ✗ |

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

[2] Threshold encryption key, secret-shared decryption key

***Impossibility of* 1*-Broadcast-then-Asynchronous MPC without setup*** In Theorem 6 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 [PR18], which dealt with perfect security. The strategy adapts [GIKR02, §4.1].

**1.1.2 Main Result: Share & Shrink method.** We answer positively the main question by proposing a new generic protocol in the bulletin-board PKI ($\mathsf{bPKI}$) model, called Share & Shrink. It performs *in parallel*: a $\mathsf{DKG}$, and a distribution of ciphertexts of inputs under the single threshold key (thus of sizes independent of $|\mathscr{L}|$). The broadcast $\mathsf{BC}$ is used **only once** simultaneously by both players and input owners. The second (and last) step is performed over asynchronous point-to-point channels. The first ingredient for Share & Shrink is any (linear) publicly verifiable secret sharing (PVSS) scheme [GV22; KMM+23]. Recall that a PVSS sharing algorithm, on input a secret $s$, roughly consists in: generate a $t$-out-of-$N$ linear secret sharing of $s$, generate an encryption of each share under the key of one player, and output the $N$ ciphertexts obtained. The second ingredient is any linearly homomorphic encryption scheme, in a precise sense to be defined in Section 3.1. This includes the schemes known as CL [CL15], GSW [GSW13] and BFV [FV12], along with most of their variations. Share & Shrink can be described as follows.

**0. Setup.** Each player non-interactively generates then publishes a public key, for any public-key encryption scheme (PKE). Players also retrieve a uniform random string (URS), as needed in most of the practical cases considered.

**1. Share.** Players run a $\mathsf{DKG}$ protocol in one round of broadcast. The pattern is the same as in [FS01]. Namely, each player $P_i$ generates an additive contribution $\mathsf{sk}_i$ to the decryption key, and $\mathsf{ek}_i$ to the threshold encryption key, which is the image of $\mathsf{sk}_i$ (and possibly some noise) by a *fixed public linear map* (e.g., for BFV: $\mathsf{ek}_i \leftarrow (-a.\mathsf{sk}_i + e_i^{(\mathsf{pk})}, a)$, where $a$ is a URS). It broadcasts $\mathsf{ek}_i$ and a PVSS of $\mathsf{sk}_i$. We leave here implicit the necessary NIZK's proving that the public $\mathsf{ek}_i$ is derived from the shared $\mathsf{sk}_i$ (with possibly some noise $e_i^{(\mathsf{pk})}$).

***In parallel,*** input owners also broadcast PVSS's of their inputs and of encryption randomnesses.

**2. Shrink,** each player locally sets the *threshold encryption key* $\mathsf{ek}$ as roughly the sum of the $\mathsf{ek}_i$'s for which the $\mathsf{sk}_i$'s were correctly shared. Players perform threshold encryption of the shared inputs under $\mathsf{ek}$, thereby "shrinking" them down to the sizes to ciphertext encrypted under the (threshold) single-key $\mathsf{ek}$. What makes threshold encryption work in *one step of point-to-point asynchronous messages*, is that it simply consists in the opening of a *linear map*, parametrized by $\mathsf{ek}$, evaluated over the shared secret inputs (and the shared encryption randomnesses).

In conclusion, players obtain a common view on inputs encrypted with a (threshold) single-key $\mathsf{ek}$. They can thus proceed as in the remaining of the $\mathsf{DKG}$-then-Input-distribution method (which requires no further broadcast).

**Theorem 2 (Share & Shrink).** *For any linearly homomorphic encryption scheme in the sense of Definition 4, and evaluation algorithm (or protocol) over encrypted inputs in the sense of Section 5.1, there exists a MPC protocol under honest majority with GOD, in 1* $\mathsf{BC}$ *followed by asynchronous point to point messages. It furthermore allows inputs of* external input owners*. It operates on ciphertexts of inputs under a (threshold) single-key encryption, in particular their sizes are* independent *of the number of input owners* $|\mathscr{L}|$*, and of* $N$*.*

It has furthermore the *delayed function property*, i.e., messages from owners are independent of the circuit to be evaluated. [BHKL18] point out that the lack of handling inputs from external lightweight owners, like mobile phones or web browsers, is one of the main obstacles to the deployment of MPC in practice. We thus believe that enabling input owners, and not having a complexity which grows with their number, is a significant advantage of our scheme over previous ones in 1BC then asynchrony [GLS15; BJMS20].

**1.1.3 Of independent interest: the *first robust threshold BFV* (and instantiating Share & Shrink with it).** To obtain the smallest possible ciphertexts, we take the example of the RLWE-based

| Protocol | 1 BC + asynch P2P | GOD for $t < N/2$ | Size of ciphertexts | External Inputs |
|---|---|---|---|---|
| [AJL+12][KJY+20] [MTBH21][Par21][MBH23] | ✗ | ✗ | $O(n \log q)$ | ✗ |
| [CDKS19] | ✗ | ✗ | $O(nN \log q)$ | ✗ |
| [GLS15] | ✓ | ✓ | $O(n^2 \log(q)^3)$ | ✗ |
| [ACGJ18] | ✓ | ✓ | $O(N^\tau \lvert C\rvert + N^{\tau+1} d)$ [(a)] | ✗ |
| [BJMS20] + **Thm. 1** | ✓ | ✓ | $O\big((\lvert\mathscr{L}\rvert n)^2 \log(q)^3\big)$ | ✓ |
| **Thm. 2** | ✓ | ✓ | Same size as single-key encryption [(b)] | ✓ |
| **Thm. 3** | ✓ | ✓ | $O(n \log q)$ | ✓ |

[(a)] $\tau > 2$ and $d$ is the depth of $C$.
[(b)] Size independent of $N$ and of $\mathscr{L}$. However a minimum modulus $q$ exponentially larger than the noise of the evaluated ciphertext is required: see Section 1.1.4.

Table 2: MPC for $N = 2t+1$ players and $|\mathscr{L}|$ input owners, using FHE with lattice dimension $n$ and modulus $q$, and assuming a URS and a bulletin-board PKI. The last column indicates whether or not the protocol allows inputs from lightweight external owners, which do not take part in the computation. We did not mark as GOD the protocols which must be restarted when one player drops-out in the middle (detailed in Appendix A.1.3). The "Size" is the one of the ciphertexts which undergo homomorphic evaluation (called "Transformed" ciphertexts, in [GLS15; BJMS20]) (possibly with asynchronous interactions, cf Section 5.1).

FHE known as BFV. Let us denote by $R_q$ its ciphertext space, where elements are encoded in size $O(n \log q)$ bits, with $n$ the dimension of the ring and $q$ the modulus, see Appendix D for details. Then, we have:

**Theorem 3 (Share & Shrink instantiated from BFV (§7.1)).** *In the model of $(\overline{\mathcal{G}}_{URS}, \mathsf{bPKI})$, consider $N = 2t + 1$ players, of which $t$ are maliciously corrupt. There exists a protocol that UC implements secure evaluation of any arithmetic circuit, with GOD, in 1 broadcast of size of $O(Nn \log q)$ bits for each player and owner, followed by 2 asynchronous steps of peer-to-peer messages, comprising non-interactive homomorphic evaluation of the circuit on ciphertexts of size $O(n \log q)$.*

Applying the Share&Shrink method to BFV is not straightforward, especially the task of generating a common "relinearization key" **rlk** (also known as "evaluation key"). A protocol for distributed generation of the **rlk** of the RLWE-based FHE "CKKS" was cleverly carried out in [KJY+20, p. III]. However their protocol requires one more BC, so is not compatible with our model. This additional BC is because **rlk** (both in CKKS and [FV12]) is an encryption of the *square* of the secret threshold key sk. There is actually one more issue in the generation protocols of **rlk** in both [KJY+20; Par21]. It is the fact that, since their sk is additively shared (which makes computations easier), if one player aborts in the subsequent broadcast for generating **rlk**, then the whole DKG needed to be restarted. The same squaring-of-sk issue shows up in the bootstrapping key (of which no robust distributed generation was ever carried out, to our knowledge). In Section 6 we overcome these issues by introducing an alternative relinearization key (adapted from [CDKS19]), along with a robust distributed generation of it. More precisely, its generation operates *in parallel* of the DKG, and is guaranteed to terminate under honest majority. We do the same for generation of the bootstrapping key (§6.1.2). So this makes the whole robust, since, unlike in previous works ([KJY+20; Par21] & appendix A), the protocol needs not be restarted if players subsequently deviate.

**1.1.4 Of independent interest: alternative threshold decryption enabling $q$ power of a small prime (+ robustness under asynchrony & honest majority).** Most previous threshold FHE schemes

used the following mainstream approach. To decrypt a ciphertext $\mathsf{c}$, each player added some locally generated noise $e_{sm}$ to its decryption share of $\mathsf{c}$, then sent the whole. As a result, when the secret sharing was instantiated with Shamir, these noises were subsequently multiplied by the $N!$-sized Lagrange coefficients. Worse, as explained in [BGG+18, §2.1], players actually multiplied their local noises by an extra $N!^2$ factor, in order to later clear-out the denominators of the Lagrange coefficients. All-in-all, this is why [BGG+18, §5.3.1] imposed an extra-multiplicative overhead of $N.N!^3$ on the ciphertext modulus $q$. This resulted in a $N\times$ blowup of the ciphertext length, as observed in [GLS15, §1.2] and [BGG+18, §2.1]. They also imposed the modulus $q$ to be a prime ([BGG+18, Appendix B]) in order for the multiplied noise to be uniformly distributed modulo $q$.

Although this maintream approach is compatible with Share & Shrink, we now propose an alternative optional approach for threshold decryption. It enables simultaneously (i) a $N!^3\times$ smaller total smudging noise, (ii) and a modulus $q$ which is possibly a power of a small prime, e.g., $2^e$, thereby allowing efficient implementations [CH18; GIKV23]. It is obtained by the novel combination of two existing ingredients. First, players *pre-generate common secret-shared smudging noises*, one for every subsequent opening to be done. To open $\mathsf{c}$, players perform all-at-once the opening of the evaluation of the linear form of (raw) decryption, added with one secret-shared smudging noise. This first ingredient was introduced by [GLS15], but was never later used to our knowledge. Second, in order to enable $q$ of small size $2^e$, we use Shamir sharing over $\mathbb{Z}/2^e\mathbb{Z}$, i.e., embed polynomials into Galois rings extensions [Feh98; ACD+19]. The shares size overhead of the latter is only of $\log(N)$. Notice that this last ingredient, alone, would *not* have been applicable. Indeed, without the first ingredient, i.e., with the mainstream approach, then it would have been required that $q$ has no factor in common with $N!$.

**Related approaches.** Another technique was proposed in [BGG+18, §8.4] to remove the linear dependency in $N$ of sizes of ciphertexts. Roughly, players pre-generate a common threshold FHE ciphertext of the shared decryption key. It is roughly contained in what they denote as $\mathsf{utpp}$. Then, to do distributed decryption of a ciphertext $\mathsf{c}$, players homomorphically evaluate the decryption circuit of $\mathsf{c}$ over $\mathsf{utpp}$, then threshold-decrypt the output. On the one hand, we observe that Share & Shrink would enable to do the generation of $\mathsf{utpp}$ with no further broadcast. On the other hand, their method requires an extra-layer of homomorphic evaluation. Another way around both the linear dependency in $N$ and the requirement for a prime modulus, is proposed in [BGG+18, §5.2]. However, it comes at the cost of replacing Shamir sharing by a $\{0,1\}-$LSSSD secret sharing[3] , of which the shares have large size $\Omega(N^{4.3})$ [BS23, Table 1]. It was recently announced [CCK23] that a special purpose secret-sharing can reduce this size to $\Omega(N^{2+o(1)})$. By comparison, the threshold decryption which we propose as an optional alternative, has amortized constant overhead. Indeed, the communication complexity of an $\ell$-sized broadcast, when amortized over $\ell$, is $O(N\ell)$ ([NRS+20]). Then, the cost of the $N$ to $N$ broadcasts can be amortized over the generation of $N-t$ smudging noises in parallel, using well-known linear randomnss extraction techniques [BH08]. Finally, the somewhat follow-up work [DDE+23, Fig. 8] introduces a very nice optimization of the smudging lemma. They propose a threshold decryption using Canetti's "online error-correction" ([Can95, p. 4.4.4]). As a result, they tolerate up to $t < n/4$ corruptions under asynchrony. We observe that they could be upgraded into robustness under asynchrony and honest majority, which we have. Following their [DDE+23, Fig. 14], this would have been at the cost of players proving in ZK their correct evaluations of AES.

Finally, some works [CSS+22; BS23] address an orthogonal size dependency. It is that the smudging noise, hence, the modulus, should be exponentially larger than the decryption noise of the ciphertext of the output. Denoting $B_{\mathrm{C}}$ an upper-bound on the latter noise, they propose the smudging noise to be only polynomial in $B_{\mathrm{C}}$. However, as they stress, they cannot anymore achieve that simulated decryption shares would be indistinguishable from actual shares [4]. Hence, it is not proven if their alternative can be used to do UC-secure MPC (since all existing proof strategies since [CDN01] require indistinguishable shares). What they achieve is a weaker property, i.e., IND-CPA in presence of partial decryption queries.

[3] D stands for "derived" [JRS17], which is stated equivalently as "with strong reconstruction" in [BS23]. As pointed in [BS23, footnote 6], the specification $\{0,1\}-$LSSS in the merged paper [BGG+18] is slightly too weak.

[4] In [BS23, Footnote 4], and in [CSS+22]: "On the contrary, it is seemingly hard to achieve the original notion of simulation security proposed in [MW16; BGG+18; CCK23] without a superpolynomial modulus-to-noise ratio."

#### 1.1.5 Minor contribution: an asynchronous semi-malicious model compilable into malicious security

We consider the well-known paradigm of *semi-malicious-to-malicious security* [AJL+12, §A.2] [BHP17; GLS15; BJMS20]. Recall that in these prior works, semi-malicious players were meant to explain the messages which they sent based on the broadcasts in prior rounds. So, contrary to our asynchronous setting, they had no freedom to pretend that they did not receive such or such message. In Sections 2.2 and 7.2 we extend this model to the asynchronous phase of our MPC protocol, in a way which can still be compiled into malicious security.

### 1.2 More Related Works

Further related works and details can be found in Appendix A.

**In the Approach of [GLS15]**, players initially generate and publish GSW public keys, using the same common randomness (from a URS). To encrypts its input, a player generates a ciphertext of it under its own public 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 vector of ciphertext is denoted as a *flexible ciphertext*. They are subsequently transformed into threshold GSW ciphertexts (by summing the coordinates of nonaborting players). A first limitation is that, since this technique relies on the leftover hash Lemma (see §A.1.1), it is unknown how to port it over RLWE. So if we are aiming at shorter sizes of ciphertexts (in $O(n \log q)$ for BFV vs $O(n^2 \log(q)^3)$ for [GSW13]), we need completely different techniques, which we do in this work.

**The Approach of [BJMS20]** is described in Section 7.4. Its drawback is that MFHE ciphertexts unavoidably undergo a processing expanding their size in $|\mathscr{L}|$, even in $|\mathscr{L}|^2$ in their construction that uses GSW.

**DKG-then-Input distribution approach.** [KJY+20] carried-out the DKG-then-Input distribution approach, with a thresholdization of the RLWE-based FHE "CKKS". However, their protocol for distributed generation of a relinearization key, in III, fails as soon as there exists one player which participated in the DKG, but not in this subsequent protocol. The same issue appears in the $(N,N)$-ThFHE of [Par21]. Finally, an inherent limitation of the DKG-then-Input distribution approach is that the encryption key ek produced by a DKG is not published explicitly: it is the result of a local computation made by each player, which furthermore involves checking NIZKs of correctness of broadcasts in the DKG. So to allow external input owners would require another intermediary step after the DKG, in which players would notify ek to the external input owners.

**Other approaches.** [ACGJ18, §6.1] achieve MPC with GOD and delayed function property in 3 rounds. Contrary to TFHE-based approaches, the per-player communication cost depends on $|C|$ the size of the circuit evaluated, since it is in $O(N^\tau |C| + N^{\tau+1} d)$, where $\tau > 2$ and $d$ is the depth of $C$. Provision of external inputs is prevented by the fact that messages contain hardcoded information combining both material specific to the player $P$ sending them, and its secret input. We discuss further related works using Garbled circuits in §A.2, and how they benefit from being combined with TFHE, e.g., in use-cases of deep neural networks. The protocols [DHL21; GPS19] proceed by intervals of fixed duration, denoted rounds, so are not responsive. When cast under a malicious dishonest minority: $N = 2t+1$, then the latter is not secure under asynchrony (it is however more network-tolerant when the malicious corruptions tolerance is lowered). The protocols [DHL21; LLM+20] do not tolerate more than $t_a < N/3$ corruptions under asynchrony.

## 2 Model

More details and comments can be found in §B.

**General notations.** All logarithms are in base two, excepted in §C.2. We denote $x \xleftarrow{\$} \mathscr{D}$ the sampling of $x$ according to distribution $\mathscr{D}$. Cardinality of a set $X$ is denoted as $|X|$. For a finite set $E$, we denote $U(E)$ the uniform distribution on $E$. The set of positive integers $[1, \ldots, N]$ is denoted $[N]$. We denote by $\lambda$ the security parameter throughout the paper. $\{0,1\}^*$ denotes bitstrings of arbitrary lengths.

**2.1 Players, Input Owners and Corruptions.** We consider $N = 2t+1$ players $\mathcal{P} = (P_i)_{i\in[N]}$, which are probabilistic polynomial-time (PPT) machines, of public identities. We also consider PPT machines denoted *input owners* $(Q_\ell)_{\ell\in\mathscr{L}}$, which are logically disjunct from players. We consider the Universal Composability (UC) model [Can01] with static corruptions, recalled in §B.7. We consider a PPT machine, denoted as the Environment $\mathscr{E}$. It fully controls an entity denoted the "dummy adversary" $\mathcal{A}$. At the beginning of the execution, $\mathcal{A}$ may corrupt up to $t$ players of its choice, along with an arbitrary number of owners. They behave as arbitrarily instructed by $\mathcal{A}$. Our model needs not counting in the corruption budget $t$ the honest players whose hardware is hosting a corrupt owner. Without loss of generality we assume that $\mathcal{A}$ corrupts exactly $t$ players, of which we denote the indices by $\mathcal{I} \subset [N]$. The remaining ones are called *honest* and indexed by $\mathcal{H} = [N] \setminus \mathcal{I}$. $\mathcal{A}$ notifies $\mathscr{E}$ of every message received by corrupt players and from (simulated) functionalities. The adversary $\mathcal{A}$ can *rush*, in the sense that all messenging functionalities ($\mathsf{BC}$, $\mathsf{bPKI}$, $\mathcal{F}_{\mathsf{AUTH}}$) will let $\mathcal{A}$ learn the messages sent by honest players before letting $\mathcal{A}$ choose those sent by corrupt players.

**2.2 Extending the Semi-Malicious Model to Asynchrony** We broadly define a *semi-maliciously corrupt* player, as one which, if it sends a message, then must be able to exhibit a view of the execution explaining this message. Precisely, it must be able to exhibit a consistent witness tape containing: values of coins explaining its random choices, the set of all broadcasted values so far (possibly from other senders) 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 [AJL+12, §A.2], semi-malicious players simply had to explain their behavior based on previous broadcasted 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. Then, we compile our results into malicious security, roughly, by having players append NIZKs to all their messages. Further subtleties are discussed in Appendix B.9 and section 7.2.

**2.3 Formalizing Guaranteed Output Delivery (GOD) in UC.** Informally, our MPC protocols guarantee output delivery (GOD) in executions where all messages sent by honest players are eventually delivered. To formalize GOD, we use the following mainstream UC formalism called public *delayed output*, from [Can01]. It is still used for its simplicity, e.g., [AAPP22; CP23]. We refer to appendix B.5 for more complex approaches. We say that an ideal functionality $\mathcal{F}$ sends a *public delayed output* $v$ to $R$, if it, first, makes a request to $\mathcal{A}$, which we denote $\mathsf{ReqDeliv}$, for permission to output $v$ to $R$. The adjective "public" denotes that the value $v$ is given to the adversary in the request. Since all our functionalities have public output, we leave it implicit in what follows. When $\mathcal{A}$ allows, then $\mathcal{F}$ outputs $v$ to $R$. Also, the broadcast functionality $\mathsf{BC}$ (and also $\mathsf{bPKI}$) waits for $\mathcal{A}$ to provide the input values of corrupt senders, which we formalize as $\mathsf{ReqInput}$. Hence, the adversary can artificially block the MPC protocol by not giving inputs to $\mathsf{BC}$, and/or, by not allowing the delivery of some messages (by $\mathsf{BC}$ or asynchronous P2P channels $\mathcal{F}_{\mathsf{AUTH}}$). Of course in actual synchronous implementations of $\mathsf{BC}$, players always obtain an output, e.g., $\perp$, even if the sender is corrupt.

Briefly, we say that a MPC protocol has GOD if, in every execution in which the adversary provides the inputs required by $\mathsf{BC}$, and eventually allows delivery of their ouptput by $\mathsf{BC}$ and $\mathcal{F}_{\mathsf{AUTH}}$, then all honest players obtain an output. We provide more formalism in Appendix B.5. There, we argue why our definition of GOD coincides with the classical definition of GOD, as soon as $\mathsf{BC}$ and $\mathcal{F}_{\mathsf{AUTH}}$ are implemented with protocols or ressources which eventually deliver messages.

The session identifier of the MPC protocol, $\mathsf{sid}$, is left implicit in all calls to functionalities. Some calls to functionalities are parametrized by sub-session identifiers $\mathsf{ssid}$. In $\mathcal{F}_{\mathsf{LSSS}}$ and $\mathsf{BC}$, $\mathsf{ssid}$s are encoded by labels of variables.

**2.4 BC Broadcast with eventual termination,** also known as Byzantine Agreement. It is formalized in Fig. 1 (adapted from [GO14]). It is parametrized by a sender $\mathcal{S}$, which in this work will be in $\mathscr{P} \sqcup \mathscr{L}$, and by a set of receivers, which in this work will be $\mathscr{P}$ (except for $\mathsf{bPKI}$, for which it will be $\mathscr{P} \sqcup \mathscr{L}$).

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

Upon receiving $\{(\mathsf{input}, \mathsf{ssid}, v \in \{0,1\}^*)$ from $\mathcal{S}$ if $\mathcal{S}$ honest$\}$ OR $\{(\mathsf{ssid}, v \in \{0,1\}^* \sqcup \{\perp\})$ from $\mathcal{A}$ if $\mathcal{S}$ corrupt$\}$ then: if ssid is not stored, store it and delay-output $(\mathsf{ssid}, v)$ to every $R \in \mathcal{R}$.

Fig. 1: Broadcast for sender $\mathcal{S}$ and receivers $\mathcal{R}$.

**2.5 "Bulletin Board PKI": bPKI.** We consider the ideal functionality denoted as bPKI. It is simply an instance of BC accessible by players before the protocol starts, and with set of receivers equal to $\mathscr{P} \sqcup \mathscr{L}$. We refer to Appendix B.2 for a detailed survey of related notions in the literature.

**2.6 (Asynchronous) Authenticated Message Transmitting $\mathcal{F}_{\mathsf{AUTH}}$** as defined in [Can01], is formalized in Fig. 7 of §B. It is parametrized by a sender $\mathcal{S}$ and receiver $R$. On input $v$ from $\mathcal{S}$, it provides $R$ with delayed output $v$.

**2.7 Global Public Uniform Random String (URS) $\overline{\mathcal{G}}_{\mathbf{URS}}$.** 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 Figure 8 of §B, along with discussions on implementation. The following discussion is optional, as the reader may simply consider $\overline{\mathcal{G}}_{\mathrm{URS}}$ in the plain UC model. We strictly upgrade the security of our model in that we allow the string produced by $\overline{\mathcal{G}}_{\mathrm{URS}}$ to be directly observed by the Environment. In particular, our simulator will not have the choice but to use the URS provided by $\overline{\mathcal{G}}_{\mathrm{URS}}$ when simulating the contributions of honest players to the keys ($\mathbf{b}_i$ and $\mathbf{rlk}_i$). Furthermore, it will be clear from our UC proof in §7.1, i.e., $\mathrm{Hybrid}_2$, based on Corollary 14, that the same URS can possibly re-used in multiple concurrent executions. Which, by the "EUC⇒GUC" in [CDPW07], implies that $\overline{\mathcal{G}}_{\mathrm{URS}}$ can be treated as a global resource.

**2.8 Ideal Functionality of MPC $\mathcal{F}_{\mathbf{C}}$.** We refer to appendix B.7 for reminders on the UC model. The ideal functionality of MPC that we aim to UC implement, is formalized as $\mathcal{F}_{\mathrm{C}}$ in Fig. 2. It returns to players the evaluation of a public arithmetic circuit C over their inputs. For simplicity: C has $|\mathscr{L}|$ input gates, one single output gate, $\mathcal{F}_{\mathrm{C}}$ expects exactly one single input from each input owner, and delivers the output to all players and only them. For simplicity, C is hardcoded in $\mathcal{F}_{\mathrm{C}}$. Inputs of owners which come as $\perp$ are arbitrarily set to 0. But our protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ actually allows players to *adaptively* choose C based on the list of non-$\perp$ inputs received: see §B.6. This is actually what will do $\mathcal{F}_{\mathsf{LSSS}}$, for the case of linear combinations.

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

**Input** Upon receiving $\{(\mathsf{input}, m_\ell \in R_k)$ from any $\ell \in \mathscr{L}\}$ OR $\{$for a corrupt $\ell$, $(\mathsf{input}, \ell, m_\ell \in R_k \cup \{\perp\})$ from $\mathcal{A}$ $\}$, if no value is stored for $\ell$, then: (i) send to each player the delayed output $(\mathsf{input}, \ell, \overline{m_\ell})$; (ii) if $m_\ell = \perp$ then set $m_\ell := 0 \in R_k$; (iii) store $(\mathsf{input}, \ell, m_\ell)$.

**Circuit evaluation** Wait until for all $\ell \in \mathscr{L}$, there is some $(\mathsf{input}, \ell, m_\ell \in R_k)$ which is stored. Compute $y := \mathrm{C}((m_\ell)_{\ell \in \mathscr{L}})$. Send to each player the delayed output $m$.

Fig. 2: Functionality of secure circuit evaluation. Each input $m_\ell$ is identified by a public label $\overline{m_\ell}$.

## 3 Cryptographic Ingredients

### 3.1 Toy model of linear homomorphic encryption scheme

We observe that in a number of encryption schemes [FV12; CL15; CKKS17] (and more specifically in most FHE schemes), key generation, encryption and decryption are essentially *linear maps*. In a general sense, a linear map between abelian groups $f : (E, +) \to (F, *)$ is such that $f(e_1 + e_2) = f(e_1) * f(e_2)$, with some further compatibility to multiplication by scalar constants (either polynomials, or integers, possibly modulo some prime $p$ or prime power $p^e$, depending on the cases). Specifically, the key generation is a linear

function in a secret key $\mathsf{sk}$ ad some randomness, the encryption is a linear function in a message and some randomnesses, while the decryption is, roughly, linear in $\mathsf{sk}$. The following Definition 4 provides a wrapper for all such schemes, formalized using generic linear maps ($\Lambda^{\mathsf{pp}}_{\mathsf{EKeyGen}}$, $\Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk})$ and $\Lambda^{\mathsf{c}}_{\mathsf{Dec}}$).

***Definition 4 (Linear Homomorphic Encryption (LHE)).*** *A linear homomorphic encryption scheme (LHE) consists in a message space $\mathscr{M}$; spaces of secret and public keys $\mathscr{X}$, $\mathscr{E}k$; randomness spaces $\mathscr{B}_{Key}$ and $\overrightarrow{\mathscr{B}_{Enc}}$ (for key generation, and encryption) and a cipherspace $\mathscr{C}$; along with the following probabilistic polynomial-time algorithms:*

- $\mathsf{Setup}(1^\lambda)$*: On input the security parameter $\lambda$, the setup algorithm outputs a set of public parameters* $\mathsf{pp}$*. Some schemes furthermore assume a uniform random string (URS), noted $a$, as part of the public parameters.*
- $\mathsf{KeyGen}(\mathsf{pp})$*: On input some public parameters* $\mathsf{pp}$*, the key generation algorithm samples a secret key $\mathsf{sk} \xleftarrow{\$} \mathscr{X}$, a key randomness $\rho_{Key} \xleftarrow{\$} \mathscr{B}_{Key}$ and outputs an encryption (public) key $\mathsf{ek} \leftarrow \Lambda^{\mathsf{pp}}_{\mathsf{EKeyGen}}(\mathsf{sk}, \rho_{key}) \in \mathscr{E}k$, where $\Lambda^{\mathsf{pp}}_{\mathsf{EKeyGen}}$ is a public fixed* linear map *with coefficients determined by the public parameters* $\mathsf{pp}$*.*
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{ek} \in \mathscr{E}k, m \in \mathscr{M})$*: On input public parameters* $\mathsf{pp}$*, a public 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 $\mathsf{c} \leftarrow \Lambda^{\mathsf{pp},\mathsf{ek}}_{Enc}(m, \rho_{\mathsf{Enc}}) \in \mathscr{C}$, where $\Lambda^{\mathsf{pp},\mathsf{ek}}_{Enc}$ is a public fixed* linear map*.*
- $\mathsf{Dec}(\mathsf{sk} \in \mathscr{X}, \mathsf{c} \in \mathscr{C})$*: On input a ciphertext* $\mathsf{c}$ *and a secret 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 the symbol $\perp$, where $\Omega_{\mathsf{Dec}}$ denotes a non-linear decoding function.*
- *IND-CPA, as reminded in Appendix B.10;*
- *and bounds on the "error noise" obtained by raw decryption of the ciphertexts after they underwent circuit evaluation, as sketched in Section 5.2 and analyzed in Appendix D.7.*

**3.1.1 CL** The LHE of Castagnos-Laguillaumie (CL) 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 [CCL+20, §3.2] it is described how to set-up the parameters for a public common prime $p$. A DKG for CL is provided in [BDO23, Fig.4], including a suitable secret sharing over $\mathbb{Z}$. The DKG can be made non-interactive in one round of BC, using PVSS (Section 3.2).

**3.1.2 BFV [FV12]** We consider a positive integer $n$, denoted as *lattice dimension*; a monic polynomial $f$ of degree $n$; $k < q$ positive integers denoted plaintext and ciphertext moduli; and $R := \mathbb{Z}[X]/f(X)$. They will be further specified in Appendix D.1. We denote $R_k = R/(k.R)$ and $R_q = R/(q.R)$ the residue rings of $R$ modulo $k$ and $q$. Let $\Psi_q$, $\mathscr{B}_{Enc,q}$ and $\mathscr{X}_q$ be distributions over $R_q$. We denote $\lceil . \rfloor, \lfloor . \rfloor, \lfloor . \rceil$ the rounding to the next, previous, and nearest integer respectively, and $[.]_k$ the reduction of an integer modulo $k$ into $R_k$. When applied to polynomials or vectors, these operations are performed coefficient-wise. Let $\Delta = \lfloor q/k \rfloor$ be the integer division of $q$ by $k$. We now describe BFV seen as a mere $\mathsf{LHE}$ scheme, following Definition 4. Departing from BFV, we specify that the key generation algorithm takes a fixed public uniform random string (URS) denoted $a$ as input, while $a$ is sampled locally in BFV. The reason is that, for our distributed key generation ($\mathsf{DKG}$) to operate, some form of additivity will be required between the keys. Intuitively, this specification of $a$ as a URS is harmless, since $t + 1 = N - t$ honest public keys ($b_i$) generated with this same $a$ are $t + 1$ instances of RLWE with same public $a$, hence, are indistinguishable from $t + 1$ uniform randomnesses. This is exactly what is done in the UC proof of MPC (in Section 7.1 $\mathsf{Hybrid}_2$).

- $\mathsf{BFV.KeyGen}(\mathsf{pp} = (a \in R_q))$: Sample $e^{(\mathsf{pk})} \xleftarrow{\$} \Psi_q$ and $\mathsf{sk} \xleftarrow{\$} \mathscr{X}_q$, and define the linear form $\Lambda^a_{\mathsf{EKeyGen}} : (\mathsf{sk}, e^{(\mathsf{pk})}) \to (-a.\mathsf{sk} + e^{(\mathsf{pk})}, a)$.

  Output $\mathsf{ek} \leftarrow \Lambda^a_{\mathsf{EKeyGen}}(\mathsf{sk}, e^{(\mathsf{pk})}) = (-a\,\mathsf{sk} + e^{(\mathsf{pk})}, a) = (b, a)$ and $\mathsf{sk}$.
- $\mathsf{BFV.Enc}(\mathsf{ek} = (b, a), m \in R_k)$: Sample the encryption randomnesses $e_0^{(Enc)} \xleftarrow{\$} \mathscr{B}_{Enc,q}$, $e_1^{(Enc)} \xleftarrow{\$} \Psi_q$, and $u \xleftarrow{\$} \mathscr{X}_q$. Define the linear map $\Lambda^{b,a}_{Enc} : (\Delta m, u, e_0^{(Enc)}, e_1^{(Enc)}) \to \left(\Delta m + ub + e_0^{(Enc)},\ u\,a + e_1^{(Enc)}\right)$.

Output $\mathsf{c} \leftarrow \Lambda^{b,a}_{Enc}\big(\Delta m, u, e_0^{(Enc)}, e_1^{(Enc)}\big) \in R_q^2$.

[In the formalism of Definition 4, the space of encryption randomness is thus $\overrightarrow{\mathscr{B}_{Enc}} = \mathscr{B}_{Enc,q} \times \Psi_q \times \mathscr{X}_q$.]

- $\mathsf{BFV.Dec}(\mathsf{sk}, \mathsf{c})$: Given a ciphertext $\mathsf{c} = (\mathsf{c}[0], \mathsf{c}[1]) \in R_q^2$, define the linear form $\Lambda^{\mathsf{c}}_{Dec} : \mathsf{sk} \to \mathsf{c}[0] + \mathsf{c}[1].\mathsf{sk}$ and compute $\mu \leftarrow \Lambda^{\mathsf{c}}_{Dec}(\mathsf{sk})$.

  Output $m \leftarrow [\left\lfloor \frac{k}{q}(\mu) \right\rceil]_k := \Omega_{\mathsf{Dec}}(\mu) \in R_k$.

**3.1.3 GSW.** From a remote perspective, the original GSW [GSW13] public key fully homomorphic encryption scheme (FHE) 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, [AP14] 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 explicitely spelled-out in [MW16] ([AP14] described only a symmetric-key simplification) and used in [BGG+18, Appendix B]. Then, [BHP17] and [BJMS20] 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, 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 [MW16, Lemma 2.1]. As in Section 3.1.2, for additivity reasons we consider that the public uniform randomness $\mathbf{A}$ 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 form $\Lambda^{\mathbf{A}}_{\mathsf{EKeyGen}} : (\mathsf{sk}, \mathbf{e}^{(\mathbf{pk})}) \to (s.\mathbf{A} + \mathbf{e}^{(\mathbf{pk})}, \mathbf{A})$. 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 form $\Lambda^{\mathbf{A},\mathbf{b}}_{Enc} : (\mathbf{R}, m) \to \big(\begin{bmatrix}\mathbf{A}\\ \mathbf{b}\end{bmatrix}\mathbf{R} + m\mathbf{G}\big)$, where $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$. Output $\mathsf{c} \leftarrow \Lambda^{\mathbf{A},\mathbf{b}}_{Enc}\big(\mathbf{R}, m\big) \in \mathbb{Z}_q^{n \times m}$.
- $\mathsf{GSW.Dec}(\mathsf{sk}, \mathsf{c})$: 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 $\Lambda^{\mathsf{c}}_{Dec} : (sk) \to \mathsf{sk}.\mathsf{c}\mathbf{G}^{-1}(\mathbf{w}^T)$ and compute $\mu \leftarrow \Lambda^{\mathsf{c}}_{Dec}(\mathsf{sk})$. Output $m := \left\lfloor \left\lfloor \frac{\mu}{q/2} \right\rceil \right\rfloor = \Omega_{\mathsf{Dec}}(\mu)$.

### 3.2 Toy functionality of LSSS, instantiation in 1BC (share) + 1 async P2P (open)

We now specify, in Fig. 3, a generic ideal functionality for linear secret sharing, denoted $\mathcal{F}_{\mathsf{LSSS}}$. It parametrized by a list of entities, denoted $\mathscr{S}$ the Senders, and for each of them by a predetermined list $X_{\mathcal{S}}$ of input labels $(\overline{x_{\mathcal{S},\alpha}})_{\alpha \in X_{\mathcal{S}}}$. Here, all senders can be corrupt. Upon receiving all these inputs from Senders, $\mathcal{F}_{\mathsf{LSSS}}$ accepts subsequent requests from players to open to them the evaluations of *any* linear maps over these inputs.

We now outline its implementation, which is one of the main ingredients of Share & Shrink. Its main features are that, after the unique round of broadcast, players have a common view on the set of shared secrets. Subsequently, they can perform the threshold opening of the evaluation of *any* linear map over the shared secrets, using only one step of all-to-all asynchronous peer-to-peer messages. To send a secret $s$ to $\mathcal{F}_{\mathsf{LSSS}}$, i.e., to share it, the first step is to generate a (N,t)-linear secret sharing (LSSS) of $s$. Let $[s^{(i)} : i \in [N]]$ the vector of shares obtained. Encrypt each share $s^{(i)}$ under $P_i$'s public key. The $N$-sized vector of ciphertexts obtained is called a Public Secret Sharing (PSS). Notice that the usual terminology is PVSS, where V stands for verifiable, because of the additional inclusion of NIZKs proving that the vector is well-formed. These NIZKs will be added when compiling from semi-malicious to malicious security. State of the art implementations of PVSS can be found in [GV22; KMM+23]. The former includes NIZKs of smallness of the secret, which will be needed, e.g., for sharing noises.

To open a linear map $\Lambda$ over a set of shared secrets $(s_j)_j$: every player $i$ decrypts its encrypted shares $s_j^{(i)}$, then evaluates $\Lambda$ on them. By linearity of the LSSS, the result is a partial opening share $z^{(i)}$ of $\Lambda((s_j)_j)$.

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

**Participants:** A set $\mathscr{S}$ of senders.
**Inputs** (For each $S \in \mathscr{S}$): $x_{S,\alpha}$ identified by a unique 'label' $\overline{x_{S,\alpha}}$.

**Setup** For each corrupt $P \in \mathscr{P}$: send $(\mathsf{ReqInput}, P)$ to $\mathcal{A}$. Upon receiving $\{(\mathsf{Setup})$ from any $P \in \mathscr{P}\}$ OR {if $P$ corrupt, upon receiving $(\mathsf{activate}, \mathsf{Setup}, P)$ from $\mathcal{A}$ }, then: if no $(\mathsf{Setup}, P)$ is stored yet, then store it and provide every player with delayed output $(\mathsf{Setup}, P)$.
Wait until $(\mathsf{Setup}, P)$ is stored $\forall P \in \mathscr{P}$, then send $\mathsf{ready}$ to every $\mathscr{S}$.

**Input** For each corrupt $\mathcal{S} \in \mathscr{S}$ and $\alpha \in X_{\mathcal{S}}$: send $(\mathsf{ReqInput}, \overline{x_{\mathcal{S},\alpha}})$ to $\mathcal{A}$.
Upon receiving $\{(\mathrm{input}, \overline{x_{\mathcal{S},\alpha}}, x_{\mathcal{S},\alpha} \in R_q)$, from any $S \in \mathscr{S}\}$ OR $\{$for some corrupt $\mathcal{S}$: $(\mathsf{activate}, \overline{x_{\mathcal{S},\alpha}}, x_{\mathcal{S},\alpha} \in R_q \cup \perp)$ from $\mathcal{A}$ }, then: (i) delay-output to every $P \in \mathscr{P}$: $(\mathsf{stored}, \overline{x_{\mathcal{S},\alpha}})$ appended with "$x_{\mathcal{S},\alpha} = \perp$" when the case; (ii) if $x_{\mathcal{S},\alpha} = \perp$ then set it to 0; (iii) store $(\mathsf{input}, \mathcal{S}, x_{\mathcal{S},\alpha})$.

**LCOpen** Upon receiving input $(\mathsf{LCOpen}, \mathsf{ssid} = \Lambda)$ from $t+1$ players, such that all $\overline{x_{\mathcal{S},\alpha}}$ appearing with nonzero coefficient in $\Lambda$ are stored, then delay output $(\mathsf{ssid} = \Lambda, y := \Lambda((x_{\mathcal{S},\alpha})_{\mathcal{S},\alpha})$ to every player.

Fig. 3: Sharing with Delayed Linear Combination functionality. sid omitted

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.

Let us first examplify when the ciphertext space is a polynomial ring $R_q$, of which the modulus $q$ is chosen as a prime (larger than $N+1$). There, the LSSS can simply be implemented as Shamir-sharing separately every coordinate of the secret polynomial $s$ (see also [BS23, §2.2]). Then, to multiply a shared secret by a fixed polynomial $a$, simply apply the *linear map* (w.r.t. $q$) of polynomial multiplication by $a$ to the vector of the coordinates of shares. So we see that no extra structure is needed in the sharing. Actually, this construction is equivalent to the one described in [KJY+20, IV. A] as "extended Shamir sharing". Now, we introduce the full generalization to any $q$, including the useful case where $q$ is a power $q = p^e$ of a prime, itself possibly small $p \leqslant N$ ([CH18; GIKV23]). For this, two options are left. Either a $\{0,1\}$-LSSSD ([JRS17], mistakenly weakened in [BGG+18], as noticed in [BS23]) with $\Omega(N^{4.3})$-sized shares (recently improved in [CCK23]). The second option is a secret sharing over Galois extensions of polynomial rings which we introduce in Appendices C.2 and C.4 (which is a mere extention of [Feh98; ACD+19]). The overhead in the size of shares is only of $\lceil \log_p N \rceil$. In this appendix we also prove, in Proposition 11, that the construction does UC-implement $\mathcal{F}_{\mathsf{LSSS}}$. The construction is much simpler than it sounds. For instance, it boils down to an elementary variation of Shamir in the case where the modulus $q = p_1 \times \cdots \times p_\alpha$ has prime factors which are *all larger* than $N+1$. In this case, simply apply Shamir sharing over a CRT decomposition of $q$.

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

We follow the model and the formalism introduced in Section 3 and assume a linear homomorphic encryption scheme as in Definition 4, represented by a tuple of PPT algorithms $\mathsf{LHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. Recall in particular that they are built from public fixed linear forms $\Lambda^{\mathsf{pp}}_{\mathsf{EKeyGen}}, \Lambda^{\mathsf{pp},\mathsf{ek}}_{Enc}, \Lambda^{\mathsf{c}}_{Dec}$ (as well as a non-linear decoding function $\Omega_{\mathsf{Dec}}$).

We now describe a protocol, called Share & Shrink and formalized in Fig. 4, which performs a "DKG & Encrypted Input distribution" in 1 BC + asynch P2P. Precisely, it allows players to obtain all-at-once: (i) a common threshold encryption key ek, (ii) a secret-shared secret key sk (formally: in $\mathcal{F}_{\mathsf{LSSS}}$), and (iii) a common view on LHE ciphertexts of the inputs encrypted under ek. The challenge is that the input owners have access to the broadcast, to distribute their inputs, *only before* ek is known! Events are gathered by the strict ordering relation, e.g., steps denoted (1) are performed by honest players after they did the ones denoted (0), but on the other hand, all the steps (1) can be done concurrently by all entities, in arbitrary order.

(0) **Setup:** Players receive some public parameters pp (which optionally include a uniform random string). **In parallel**, they Setup $\mathcal{F}_{\mathsf{LSSS}}$ (concretely: generate and publish encryption keys).

(1) **Broadcast:** Input owners send (**Share**) their inputs & encryption randomnesses to $\mathcal{F}_{\mathsf{LSSS}}$ (concretely: broadcast PSS of them).
***In parallel***, based on public parameters $\mathsf{pp}$, players generate additive contributions $\mathsf{sk}_i$ to the threshold $\mathsf{LHE}$ private key, which they input to $\mathcal{F}_{\mathsf{LSSS}}$ (concretely: broadcast PSS of them); and $\mathsf{ek}_i$ to the public key, which they broadcast (along with possibly extra material, e.g., Section 6.1.1 for BFV).
**Local computation:** Then each player locally computes the common threshold key $\mathsf{ek}$, out of the contributions of the subset $S$ of non-aborting players. Precisely, $S \subset [N]$ are those which broadcasted correct material (including the PSS, as captured by $\mathcal{F}_{\mathsf{LSSS}}$). By linearity of the $\mathsf{LHE}$ scheme, this key is merely the sum of the contributions from $S$:

$$(\mathbf{sk} = \Sigma_{i\in S}\mathsf{sk}_i, \mathsf{ek} = \Sigma_{i\in S}\Lambda^{\mathsf{pp}}_{\mathsf{EKeyGen}}(\mathsf{sk}_i, \rho_i^{\text{key}}))^5 \tag{1}$$

(2) **Asynchronous step (Shrinking of the inputs):** Then players jointly compute threshold encryptions under $\mathsf{ek}$ of the shared inputs $m_\ell$. Concretely, thanks to linearity of the $\mathsf{LHE}$ scheme, this can be done as simple threshold openings of the images of the $m_\ell$ (and of the shared encryption randomnesses) by the linear map $\Lambda^{\mathsf{ek}}_{Enc}$. So this is expedited in one step of asynchronous P2P messages, as recalled in Section 3.2.

The outlined protocol is fundamentally different from related threshold-$\mathsf{LHE}$ works [CDKS19; KJY+20; MTBH21] which roughly follow the pattern $\mathsf{DKG}$-then-Input-distribution, since using our method 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 [GLS15; BJMS20].

### Share & Shrink protocol

**Participants:** $N$ players $P_i$, $i = 0, \ldots, N$, and $|\mathscr{L}|$ input owners;
**Inputs** (for each input owner $\ell \in \mathscr{L}$): plaintext $m_\ell$ with label $\overline{m_\ell}$. // we assume only one plaintext per input owner.

- (0) **Setup.** Each player $P_i$:
  - Sends $(\mathsf{Setup})$ to $\mathcal{F}_{\mathsf{LSSS}}$
  - Obtains some public parameters $\mathsf{pp} \leftarrow \mathsf{LHE.Setup}$.
- (1) **Broadcast.**
  - Input and Randomness Distribution (part I): Each input owner $\ell$:
    - Samples $\rho_\ell \xleftarrow{\$} \mathscr{B}_{Enc}$ and sends $(\mathsf{input}, \{\overline{m_\ell}, \overline{\rho_\ell}\}, \{m_\ell, \rho_\ell\})$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  - $\mathsf{DKG}$ (part I): Each player $P_i$:
    - Computes $(\mathsf{sk}_i, \mathsf{ek}_i) \leftarrow \mathsf{LHE.KeyGen}(\mathsf{pp})$. Sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSSS}}$ and broadcasts $\mathsf{ek}_i$, denoted "additive contribution to the encryption key".
- **Local computation.**
  - $\mathsf{DKG}$ (part II): let $S$ be the subset of players which broadcast a $\mathsf{ek}_i$ and for which $\mathcal{F}_{\mathsf{LSSS}}$ has acknowledged the receipt of an additive contribution to the secret key. Each player $P_i$:
    - Computes $\mathsf{ek} = \Sigma_{i\in S}\mathsf{ek}_i$ and define the (secret shared) secret key as $\mathsf{sk} = \Sigma_{i\in S}\mathsf{sk}_i$ and its label as $\overline{\mathsf{sk}}$ //sk is accessible only through $\mathcal{F}_{\mathsf{LSSS}}$
- (2) **Asynchronous step**
  - Input and Randomness Distribution (part II): let $S_{\mathsf{c}}$ be the set of input owners for which $\mathcal{F}_{\mathsf{LSSS}}$ has acknowledged the receipt for all variables of $\ell$'s "input and randomness distribution". Each player $P_i$:
    - For each $\ell \in S_{\mathsf{c}}$, given labels $(\overline{m_\ell}, \overline{\rho_\ell})$ and a key $\mathsf{ek}$, sends $\big(\mathsf{LCOpen}, \Lambda^{\mathsf{ek}}_{Enc}(\overline{m_\ell}, \overline{\rho_\ell})\big)$ to $\mathcal{F}_{\mathsf{LSSS}}$, and obtains a ciphertext $\mathsf{c}_\ell$.

Fig. 4: Share & Shrink Protocol

[5]Note that in the case of $\mathsf{BFV}$ presented in section 3.1.2, the public key has another component with is the public parameter a, which is left outside of the sum. The same caveat applies to GSW (Section 3.1.3).

Note that the security properties will come as a byproduct of the proof of MPC in section 7.1, which is a strictly harder setting, due to adversarial influence on the public key (Lemma 23), re-use of random public parameters (Corollary 14), and public openings of smudged raw decryptions (§5.2 and Lem. 21).

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

When instantiating Share & Shrink from an FHE scheme, players can locally evaluate the ciphertexts of the inputs, into a ciphertext of the output. We now aim at allowing alternative methods to obtain such an output. To this end, in Section 5.1 we introduce the following generic ingredient. We specify an evaluation protocol, possibly interactive over asynchronous channels, of a circuit over threshold-encrypted ciphertexts. We require it to be simulatable. We then review a number of known such evaluation protocols. Then in section 5.2, we detail a method for distributed decryption with improved complexity. In section 5.3 we provide our generic MPC protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ in the $\mathcal{F}_{\mathsf{LSSS}}$-hybrid model.

## 5.1 Asynchronous Evaluation of a Circuit

Consider a generic linear homomorphic encryption scheme $\mathsf{LHE}$ (Def 4), $N$ players with inputs a common set of $\mathsf{LHE}$ ciphertexts $\{\mathsf{c}_\ell\}_{\ell\in\mathscr{L}}$ of plaintexts $\{m_\ell\}_{\ell\in\mathscr{L}}$ under a common key $\mathsf{ek}$. We assume that there exists an asynchronous evaluation protocol $\mathsf{Eval}$ for (any) arithmetic circuit $\mathrm{C} : \mathscr{M}^{|\mathscr{L}|} \to \mathscr{M}$ with $|\mathscr{L}|$ input gates, that outputs a ciphertext of the evaluation of the circuit. Formally, we require that $\forall\ \mathsf{pp} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^d)$, there exists a distributed key generation in one broadcast that returns a public encryption key $\mathsf{ek}$ and, privately to the players, shares $\mathsf{sk}_i$ of the corresponding secret decryption key $\mathsf{sk}$; $\forall\,(m_\ell)_{\ell\in\mathscr{L}}$, $\mathsf{c}_\ell \leftarrow \mathsf{LHE.Enc}(\mathsf{pp}, \mathsf{ek}, m_\ell)$ ; then $\mathsf{LHE.Dec}(\mathsf{sk}, \mathsf{Eval}(\mathrm{C}, \mathsf{c}_1, \ldots, \mathsf{c}_{|\mathscr{L}|}), \mathsf{ek}) = \mathrm{C}(m_1, \ldots, m_{|\mathscr{L}|})$. We furthermore require that $\mathsf{Eval}$ is simulatable, as examplified below. In practice, there are different ways to implement $\mathsf{Eval}$, among others:

- $\mathsf{FHE}$. When using a fully homomorphic scheme as a particular kind of $\mathsf{LHE}$ (as seen for BFV or GSW in Section 3.1), there exist a built-in non-interactive function $\mathsf{Eval}$ that comes as a property of the scheme. In particular it is simulatable from the knowledge of the public encryption key (& evaluation/relinearization key: see Section 6.1 for these further issues in MPC).
- [CLO+13]. Choudhury et al. proposed, through a clever use of pre-processed masks, a protocol for evaluating a circuit based on an efficient interactive multi-party bootstrapping protocol for a somewhat homomorphic encryption (SHE) scheme. Players open threshold decryptions of masked intermediary evaluations, the simulator simply simulates the opening of a random value.
- [BHN10]. From a common view of $\mathsf{LHE}$ encrypted inputs under a common key, and from decryption key shares assigned to each player, it is possible to apply that asynchronous CDN-like ([CDN01]) protocol of [BHN10] to evaluate a circuit. Players open threshold decryptions of masked intermediary results. Since some masks can be deduced from each other by adversarially-chosen (but extractable) offsets, some extra care is paid by their simulator.

## 5.2 Distributed decryption, Alternative enabling $N$ times shorter ciphertexts

$\mathsf{LHE}$ decryption can be seen as a two steps process: (1) interactive opening of a linear map $\Lambda^{\mathsf{c}}_{Dec}$ 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 trivial, when $\Omega_{\mathsf{Dec}}$ is nontrivial, as the case in fully homomorphic encryption (FHE). Indeed, the raw output $\mu_{\mathrm{raw}}$ of (1) actually leaks information about the secret (threshold) decryption key (see also, e.g., [BGG+18, §2.1]), which ruins the security of the $\mathsf{LHE}$. To circumvent this issue, Asharov et al. [AJL+12] introduced the technique of adding additional noise to the raw output of (1) before it can be opened. This noise $e_{\mathrm{sm}}$ is, roughly, sampled uniformly in some large enough interval $[-B_{sm}, B_{sm}]$. Concretely, we add as a combined requirement to Def. 4 and $\mathsf{Eval}$, that the raw output of (1) $\mu_{\mathrm{raw}} = \Lambda^{\mathsf{c}}_{Dec}(\mathsf{sk})$ is

statistically close enough to the (scaled) plaintext circuit evaluation $\Delta.y$ (e.g., in GSW we have $\Delta = \lfloor q/2 \rfloor$). Then, there should exist some level of noise $B_{sm}$, so that adding a uniform noise $e_{\mathrm{sm}} \in [-B_{sm}, B_{sm}]$ to both $\mu$ and $y$, makes them indistinguishable, while leaving correct the result: $y = \Omega_{\mathsf{Dec}}(\mu_{\mathrm{raw}} + e_{\mathrm{sm}})$. As stated in Lemma 21, the indistinguishability requirement imposes a level of noise high enough so that $B_{\mathrm{C}}/B_{sm} \leqslant \mathrm{negl}(\lambda)$ (Equation (12)), where $B_{\mathrm{C}}$ is a fixed upper-bound on the decryption noise of a ciphertext after evaluation of circuit C (given by Equation (11), in the case of BFV), and $\mathrm{negl}(\lambda)$ a fixed negligible function in the security parameter $\lambda$. On the other hand, the correctness requirement imposes that $B_{\mathrm{C}}$ added with this noise stays small (in GSW: smaller than $q/2$, where $q$ is the ciphertext modulus; in BFV: smaller than $\Delta/2$, as stated in Equation (10)). In Fig. 5, we give two methods for distributively opening $\mu_{\mathrm{raw}}$ added with such noise.

## Distributed Decryption protocol

**Participants:** $N$ players $P_i$, $i = 1, \ldots, N$;
**Inputs:** a public ciphertext $\mathsf{c}$; shared decryption key $\mathsf{sk}$ in $\mathcal{F}_{\mathsf{LSSS}}$ with label $\overline{\mathsf{sk}}$ // concretely, in the form of a PSS of $\mathsf{sk}$.
**Outputs:** decryption $\mathsf{LHE.Dec}(\mathsf{sk}, \mathsf{c})$
**Additional input for $2^{nd}$ method**: secret shared smudging noise $e_{sm} \in [-N.B_{sm}, N.B_{sm}]$ in $\mathcal{F}_{\mathsf{LSSS}}$ with label $\overline{e_{\mathrm{sm}}}$ (usable only for one decryption).
**($1^{st}$ method - mainstream):** Each player $P_i$, given a ciphertext $\mathsf{c}$ and a secret share $\mathsf{sk}_i$ of the secret key:

- Generates its opening share of the raw decryption (1), i.e. $\mu_{\mathrm{raw},i} = \Lambda^{\mathsf{c}}_{Dec}(\mathsf{sk}_i)$. Samples a noise $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$. Then, multicasts over P2P channels its "smudged opening share" $\mu_i = \mu_{\mathrm{raw},i} + N!^2 e_{\mathrm{sm},i}$. // The $N!^2$ factor is not needed if $\{0,1\}$-LSSSD is used [JRS17; BS23; CCK23];
- Each player $P_i$ waits until it receives smudged opening shares from a subset $\mathcal{U} \subset [N]$ of $t+1$ players: $(\mu_{j\to i})_{j\in\mathcal{U}}$. Denote $(\lambda_j^{\mathcal{U}})_{j\in\mathcal{U}}$ the Lagrange reconstruction coefficients corresponding to the subset $\mathcal{U} \subset [N]$. It sets $\mu = \sum_{j\in\mathcal{U}} \lambda_j^{\mathcal{U}} \mu_{i\to j}$ the smudged raw decryption. Then it outputs $\Omega_{Dec}(\mu)$.

**($2^{nd}$ method, with smaller noise)**: Each player $P_i$:

- Given labels $(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})$, and a ciphertext $\mathsf{c}$, sends $\left(\mathsf{LCOpen}, \Lambda^{\mathsf{c}}_{Dec+sm}(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})\right)$ [a] to $\mathcal{F}_{\mathsf{LSSS}}$, obtains $\widetilde{\mu}$, and outputs $m = \Omega_{Dec}(\mu)$.

[a] Where $\Lambda^{\mathsf{c}}_{Dec+sm}$ is the linear form $(\mathsf{sk}, e_{\mathrm{sm}}) \to \Lambda^{\mathsf{c}}_{Dec}(\mathsf{sk}) + e_{\mathrm{sm}}$

Fig. 5: Distributed Decryption Protocol

*The first mainstream method* Follows the approach of [AJL+12] and has been used in most other works [AJL+12; BGG+18; KJY+20]. Each player $P_i$ locally samples a so-called smudging noise $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ uniformly in some interval to be specified, multiplies it by $N!^2$ ([BGG+18, Construction 5.11]), then adds it to its decryption share of $\mathsf{c}$, which it multicasts. The reason for multiplying by $N!^2$ is to clear-out the denominators of the Lagrange coefficients applied at reconstruction (see [ABV+12] [BGG+18, §2.1]). Following the previous notation and explanations, the bound $B_{sm}$ is chosen in [BGG+18, §5.3.1] such that: $B_C/B_{sm} = \mathrm{negl}(\lambda)$ (for indistinguishability), and such that $B_{\mathrm{C}} + N.N!^3.B_{sm} < q/4$ (for correctness of Lagrange reconstruction-then-rounding, in their instantiation with GSW). Notice that under our instantiation with BFV, the RHS would then be $\Delta/2$ instead of $q/4$.

*The second method* circumvents this issue. It is the following forgotten approach, which we credit to [GLS15]. Players do not anymore blurr their opening share of $\mu_{\mathrm{raw}}$, which we recall is the linear combination (1). Instead, they now open all at once the sum of $\mu_{\mathrm{raw}}$ and a common shared noise $e_{\mathrm{sm}}$, i.e., they open $\Lambda^{\mathsf{c}}_{Dec}(\mathsf{sk}) + e_{\mathrm{sm}}$. The distributed generation of the noise is simply by adding secret-shared contributions $e_{\mathrm{sm},i}$, each sampled in $[-B_{sm}, B_{sm}]$. As a result, the correctness constraint now imposes only $B_{\mathrm{C}} + N.B_{sm} < \Delta/4$ (Equation (10)). Hence, the ciphertext expansion factor $\Delta$ has dependency in $N$ which is only linear ($N$), instead of $N.N!^3$ in the previous method. These formulas show up in the proof of MPC in Section 7.1, as

further detailed in Lemma 22. Since the noise can be used only for one distributed decryption, players must precompute as much noises as many circuits to be subsequently evaluated. We will formalize this simple Distributed Noise Generation protocol, denoted $\mathsf{DNG}$, in the MPC protocol. Concretely, each player $P_i$ secret-shares a contribution $e_{\mathrm{sm},i} \leftarrow [-B_{sm}, B_{sm}]$ in the form of a PSS in the broadcast step. Then, players define the common shared smudging noise as the sum over the contributions of the players which did not abort: $e_{\mathrm{sm}} = \sum_{i \in S} e_{\mathrm{sm},i}$.

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

Consider any linear homomorphic encryption scheme satisfying Def 4, represented by a tuple of PPT algorithms $\mathsf{LHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ formalized by a set of linear forms $\Lambda^{\mathsf{pp}}_{\mathsf{EKeyGen}}, \Lambda^{\mathsf{ek}}_{Enc}, \Lambda^{\mathsf{c}}_{Dec}$ as well as a non-linear decoding function $\Omega_{\mathsf{Dec}}$. Then, we show in Fig. 6 how to build a MPC protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ from the Share&Shrink protocol instantiated from $\mathsf{LHE}$ as introduced in Section 4, an evaluation algorithm $\mathsf{Eval}$ as discussed in Section 5.1, and a distributed decryption protocol as detailed in Section 5.2. Note that the number of shared smudging noises to be generated in parallel, for use in the $2^{nd}$ method of distributed decryption, is equal to the number of distinct decryptions to be performed, i.e., of circuits to be evaluated.

Note that we push back the problems related to the generation of the evaluation keys in section 6.1, since they depend on the $\mathsf{LHE}$ scheme used. It will be done, along with a concrete instantiation of $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ from $\mathsf{BFV}$, and a summarized proof of security in Section 7.1, fully detailed in Appendix H.3.

Protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$

**Participants:** $N$ players $P_1, \ldots, P_N$, and $|\mathscr{L}|$ input owners;
**Inputs** (for each input owner $\ell \in \mathscr{L}$): a plaintext $m_\ell$ with label $\overline{m_\ell}$.

- **Share** & **Shrink.** Players and Input Owners play the Share&Shrink protocol of Fig. 4, in which each input owner $\ell \in \mathscr{L}$ has an input $m_\ell$.
  Denote $S_{\mathsf{c}} \subset \mathscr{L}$ the owners (resp. $S \subset \mathscr{P}$ the players), for which no instance returned $\perp$ i.e players which broadcast a $\mathsf{ek}_i$ and for which $\mathcal{F}_{\mathsf{LSSS}}$ has acknowledged the receipt of an additive contribution to the secret key, and input owners for which $\mathcal{F}_{\mathsf{LSSS}}$ has acknowledged the receipt for all variables of $\ell$'s "input and randomness distribution".
  After Share & Shrink, players have a common view on i) a set of ciphertexts $\{\mathsf{c}_j\}_{j \in S_{\mathsf{c}}}$ under a common key $\mathsf{ek}$, ii) a shared decryption key $\mathsf{sk}$ in $\mathcal{F}_{\mathsf{LSSS}}$, as well as iii) a shared smudging noise $e_{\mathrm{sm}}$ in $\mathcal{F}_{\mathsf{LSSS}}$ for threshold decryption (if the $2^{nd}$ method is used for decryption).
- **Evaluation.** To evaluate a circuit C, each player $P_i$ runs $\mathsf{c} \leftarrow \mathsf{Eval}(\mathrm{C}, \{\mathsf{c}_j\}_{j \in S_{\mathsf{c}}}, \mathsf{ek})$.
- **Distributed Decryption.** Players play the Distributed Decryption of Fig. 5 with input the ciphertext $\mathsf{c}$ and the shared decryption key $\mathsf{sk}$ in $\mathcal{F}_{\mathsf{LSSS}}$. They output the plaintext $m$ obtained.

Fig. 6: MPC protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$

## 6 $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ from $\mathsf{BFV} + \mathsf{CDKS}^*$: the first robust threshold $\mathsf{BFV}$ scheme.

Our goal in this section is to propose an instantiation of our MPC protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ from $\mathsf{BFV}$. The choice of this scheme is motivated by the small size of its ciphertexts, which are only of size $O(n \log q)$, although we remind that other schemes (such as GSW) or other evaluation method (as seen in section 5.1) can be chosen.

Unfortunately, this poses some challenges, which we detail in section 6.1. Along the line, we describe a new variant of $\mathsf{BFV}$ (introduced in section 3.1.2) that we call $\mathsf{BFV} + \mathsf{CDKS}^*$. This will serve as our baseline scheme, and we will show how to thresholdize it. Finally, in section 6.2 we detail an instantiation of $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ from $\mathsf{BFV}$ that we prove in sections 7.1 and 7.2.

### 6.1 Challenges when implementing $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ from BFV

We now highlight the specific challenges posed by BFV to implement $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$. Note that we provide a comprehensive description of the scheme in Appendix D.

**6.1.1 Challenge 1: Robust Relinearization Key Generation** Homomorphic multiplication in BFV involves two steps: the first, denoted "tensoring" produces a *degree two* ciphertext consisting of three elements $\hat{\mathsf{c}} = (\hat{\mathsf{c}}[0], \hat{\mathsf{c}}[1], \hat{\mathsf{c}}[2]) \in R_q^3$. To reduce the degree back to one, a second step, denoted *relinearization*, must be carried out to turn $\hat{\mathsf{c}}$ into a size 2 ciphertext $\mathsf{c}' = (\mathsf{c}'[0], \mathsf{c}'[1])$ which can be decrypted as the product of the plaintexts. The latter requires what is denoted as a *relinearization key* (**rlk**) that takes the form of an encryption of the square of the secret key. In our context of secret-shared key, computing this squaring in an MPC manner would take 2 consecutive broadcasts, which is done in other works [AJL+12; MTBH21; KJY+20] but which we want to avoid.

*Our solution.* Chen et al [CDKS19] introduced a new **rlk** for BFV which depends *linearly* on the secret key and on some noises. Their context, denoted *multi*-key FHE (MFHE), is a priori different, since each player $P_i$ generates its own secret key, from which it generates then publishes its own $\mathbf{rlk}_i$, the global relinearization key then consisting in the vector of all $\mathbf{rlk}_i$. Our simple but seemingly new observation is that their method can be particularized to the single key case, and thus, applied to our setting. In this particularization, their vector collapses into a single $\mathbf{rlk} = (\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2) \in (R_q^l)^3$ (the choice of the integer $l$ is detailed in [CDKS19]).

Notice that our description of the generation of a standalone $\mathbf{rlk} \in (R_q^l)^3$ takes as argument common uniform random strings $(\mathbf{a}, \mathbf{d}_1) \in (R_q^l)^2$. In particular, the $a$ of the public key (Section 3.1.2) is the first coordinate $a = \mathbf{a}[0]$. The modification with respect to [CDKS19] is that they specified, instead, that $\mathbf{d}_1$ is sampled locally by the owner of a public key $(b, a)$ to generate the corresponding relinearization key **rlk**. Our reason for this modification is that, in our MPC protocol, players will need to generate **rlk** distributively, by providing additive contributions denoted $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i})$. Additivity between these would not be possibly if they had not been generated using the same common $\mathbf{d}_1$. In Appendix D.4 we formalize this modification to [CDKS19] as a function: CDKS which generates a relinearization key, and which has the desired additivity. We call $\mathsf{BFV} + \mathsf{CDKS}^*$ our new LHE derived from BFV with this new relinearization key generation function. In order to apply Share&Shrink, at setup (0), we let players obtain common uniform strings $(\mathbf{a}, \mathbf{d}_1)$ from $\overline{\mathcal{G}}_{\mathrm{URS}}$, and let (1) each $P_i$ compute and broadcast $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$. Denoting $S$ the subset of players of which the broadcasts in (1) were correctly formed, a common relinearization key $\mathbf{rlk} := (\Sigma_{i\in S}\mathbf{d}_{0,i}, \mathbf{d}_1, \Sigma_{i\in S}\mathbf{d}_{2,i})$ can later be computed in (2).

In Corollary 14 we prove that, despite our specification of a common $\mathbf{d}_1$, the concatenation of all the honestly generated contributions $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$ to the common relinearization key, as well as the contributions $b_i$ to the public key, is indistinguishable from a large uniform random string, under the *same* circular security assumption as implicitely made in [CDKS19]. For completeness we recall in Appendix D.6.2 the relinearization algorithm *Relin* of [CDKS19, §3.3.2], when particularized into our single-key setting.

**6.1.2 Challenge 2: Enabling Bootstrapping** The "Bootstrapping" of a single-key BFV ciphertext is a local computation that homomorphically brings back the size of its decryption noise, roughly to the one of a fresh ciphertext, in order to evaluate deeper circuit. We show in Appendix E how to add a robust bootstrapping key generation, by particularizing, as we did for relinearization, the one described in [CDKS19, §5] in their context of multikey BFV. In brief, we detail in Appendix E a function BkKG for bootstrapping key generation.

In order to apply Share&Shrink, at setup (0), we let players obtain a common uniform string $\mathbf{h}_1$ from $\overline{\mathcal{G}}_{\mathrm{URS}}$, and let (1) each $P_i$ compute and broadcast $\mathbf{bk}_i(j) \leftarrow \mathsf{BkKG}(\mathbf{h}_1, \mathsf{sk}_i, j)$. Denoting $S$ the subset of players of which the broadcasts in (1) were correctly formed, each player locally sets the common ($j$-th component of) the bootstrapping key as $\mathbf{bk}(j) := (\Sigma_{i\in S}\mathbf{h}_{0,i}(j), \mathbf{h}_1)$.

### 6.2 Protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ instantiated from $\mathsf{BFV} + \mathsf{CDKS}^*$

We give in Fig. 11 a complete breakdown of protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ when instantiated from $\mathsf{BFV} + \mathsf{CDKS}^*$. Thanks to the generic definition provided in Section 4, we can easily instantiate the Share&Shrink protocol from $\mathsf{BFV} + \mathsf{CDKS}^*$ using the public parameters and distributions $\mathsf{pp} = (R_q, l, \mathscr{X}_q, R_k, \Psi_q, \mathscr{B}_{Enc,q}, B_{sm}, \mathbf{a})$ and linear forms $\Lambda^{\mathbf{a}}_{\mathsf{EKeyGen}}$, $\Lambda^{b,a}_{Enc}, \Lambda^{\mathbf{c}}_{Dec+sm}$ introduced in section 3.1.2. Parameters $(R_q, l, \mathscr{X}_q, \Psi_q)$ are chosen so that Assumption 13 is verified. Notice that this assumption implies RLWE (cf Appendix D.3).

More details about the distributions selection and a comprehensive noise analysis are provided in appendix D.

#### 6.2.1 Communication complexity of $\Pi^{\mathcal{F}_{\mathsf{LSSS}} \to \Pi_{\mathsf{LSSS}}}$, i.e., when $\mathcal{F}_{\mathsf{LSSS}}$ is instantiated with $\Pi_{\mathsf{LSSS}}$.

Each player initially broadcasts $\mathsf{PSS}$s of its key and noise shares, each of size $O(N\,n\,\log q)$ bits, as well as its public and relinearization keys of size $O(n\,\log q)$ bits. Thus overall, $O(N^2\,n\,\log q)$ bits are broadcast. In parallel, each input owner initially broadcasts an $\mathsf{PSS}$ of size $O(N\,n \log q)$ bits. Ciphertexts obtained by threshold encryption (the "Shrink" step) are of size $2.n \log q$ and, by construction of the $\mathsf{BFV} + \mathsf{CDKS}^*$ scheme, so are the evaluated ciphertexts and the partial decryptions.

## 7 Proofs of Theorems 1 to 3

**7.1 UC Proof of Theorem 3** By Proposition 11, $\Pi_{\mathsf{LSSS}}$ UC implements $\mathcal{F}_{\mathsf{LSSS}}$. Thus, the following Theorem 5 implies Theorem 3.

**Theorem 5.** *$\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ implemented from $\mathsf{BFV} + \mathsf{CDKS}^*$ UC implements the ideal functionality $\mathcal{F}_{\mathrm{C}}$ for any semi-malicious adversary, in the $(\mathcal{F}_{\mathsf{LSSS}}, \mathsf{BC})$-hybrid model with external resource $\overline{\mathcal{G}}_{URS}$.*

To prove Theorem 5, we describe in Appendix H.3.1 a simulator $\mathcal{S}$ of $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ that simulates honest players following the protocol, and ideal functionalities behaving as specified.

We now convey the main ideas of $\mathcal{S}$ by describing it via a sequence of incremental changes, starting from a real execution. In the last hybrid obtained, the view of $\mathscr{E}$ is generated solely by interaction with $\mathcal{F}_{\mathrm{C}}$, hence what we are describing is a simulator. The full details of the hybrids and the proofs of indistinguishability are in appendix H.3.

We first simulate decryption by modifying the behavior of $\mathcal{F}_{\mathsf{LSSS}}$ in Output computation. There it, incorrectly, outputs $\mu^{\mathcal{S}} := \Delta.y + \Sigma_{j \in S} e_{\mathrm{sm},j}$, where $y := \mathrm{C}((m_\ell)_{\ell \in S_{\mathrm{c}}})$ is the evaluation in clear of the circuit on the actual inputs. Indistinguishability follows from the "smudging" Lemma 21, as detailed in Lemma 22.

Then, in $\mathsf{Hybrid}_2$, the additive contributions $(\mathbf{b}_i, (\mathbf{d}_{0,i}, \mathbf{d}_{2,i}))_{i \in \mathcal{H}}$ of honest players to the public and relinearization keys, are replaced by a sample in $U(R_q^{l \times 3})$. Indistinguishability from $\mathsf{Hybrid}_1$ follows from Corollary 14.

Finally, in $\mathsf{Hybrid}_3$, we replace the actual inputs $m_\ell$ of simulated honest owners by $\widetilde{m_\ell} := 0$. Importantly, the behavior of $\mathcal{F}_{\mathsf{LSSS}}$ is unchanged, i.e., correct until (3) included, then outputs $\mu^{\mathcal{S}} := \Delta y + \Sigma_{j \in S} e_{\mathrm{sm},j}$, where $y := \mathrm{C}((m_\ell)_{\ell \in S_{\mathrm{c}}})$ is still the evaluation of the circuit on the *actual* inputs. Thanks to the modifications so far, the secret keys of the honest players are no longer used in any computation. Furthermore, since honest players sample their contributions $\mathbf{b}_i$ to the common public key independently (uniformly at random), we can assume without loss of generality that corrupt contributions are generated after having seen the honest ones. We can thus apply Lemma 20 "IND-CPA under Joint Keys", which adapts the one of [AJL+12, Lemma 3.4] in the RLWE setting. This enables to conclude that the distributions are indistinguishable. In conclusion, we arrived at a view produced by a machine which interacts only with $\mathscr{E}$ and $\mathcal{F}_{\mathrm{C}}$.

**7.2 Semi-malicious Security to Malicious Security** At a high level, malicious security can be achieved by applying the compiler of [AJL+12, §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 [AJL+12] is designed for broadcast-based protocols, whereas in ours, players in (2) and (3) also act based on previous outputs of

$\mathcal{F}_{\mathsf{LSSS}}$. This is why we also required semi-malicious players to explain their messages also based on these outputs, in our adapted model in section 2.2. We can also simplify their compiler by allowing players to prove their statements with UC NIZKs, recalled in §B.8, since these can be set-up under honest majority from one initial call to $\mathsf{bPKI}$, thanks to the technique denoted Multi-String CRS [GO14; BJMS20]. 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 §2.2 we did not impose any condition when publishing on $\mathsf{bPKI}$. 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 §B.8.

**7.3 Proving Theorem 2 for other instantiations.** We believe that the previous proof of Theorem 3 conveys all the ideas of a generic proof of Theorem 2, since it is arguably the most complex instantiation. For instance, the instantiation with GSW is a strict simplification, since no relinearization key is needed. How to adapt our proof of Theorem 2 to interactive evaluation protocols, follows directly (in black box) from our simulatability requirement, examplified in Section 5.1.

Let us finally quickly review a minor alternative for a specific part of the proof. For the encryption scheme of CL, the IND-CPA property under a distributively generated key is not argued as in our lattice-based context (Lemma 20 "IND-CPA under Joint Keys"). Instead, in [BDO23], they provide a biased-but-simulatable $\mathsf{DKG}$, which also returns a pair of encryptions: of 1 and of 0. Their simulator of [CDN01] uses the latter to replace the actual inputs by 0.

**7.4 Proof of Theorem 1** The MPC protocol of [BJMS20] proceeds in 3 rounds of broadcast, of which the first is input-independent. Hence, we replace their round 1 by publication on the PKI. Then, the broadcast in their round 2 consists in broadcasting one's input encrypted with multikey FHE (MFHE), along with a publicly verifiable secret sharing (PVSS) of one's secret decryption key. The $3^{rd}$ broadcast is used to send one's decryption share of the evaluated MFHE ciphertext of the output (with a suitable NIZK of correct decryption, as in [BJMS20], when compiled from semi-malicious to malicious security).

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 $S_2 \subset \mathscr{L}$ of owners which correctly shared their input and key (in [BJMS20] $S_2$ is instead a subset of players). From these secret shared keys, it is described in [BJMS20] how the players still-honest-in-round-3 can *emulate* MFHE-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 round 3 by one step of asynchronous P2P messages. The output after round 3 is unchanged: players still obtain the same common threshold decryption. In case, here is one more low-level explanation why their round 3 can be performed over asynchronous P2P channels. It is because the computation step performed by a player $P$ at the end of round 3 is: *choose any* set $S$ 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 where received by $P$ or not (it could be that $t$ out-of the $t+1$ valid decryption shares chosen by $P$, originate from corrupt players)

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

"Secure channels" is the upgrade of $\mathcal{F}_{\mathsf{AUTH}}$ which does not leak the content of messages to $\mathcal{A}$. The functionality of simultaneous broadcast ([CGMA85]), 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 6.** *Consider a hybrid network in which players are* initially *given access to one single round of: synchronous pairwise secure channels and broadcast;* 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.*

*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 so that the second output, $x_N$, will be *correlated* with the input of the honest $P_1$. However, in an ideal execution, then there is no link between the adversary and $P_1$ until the moment when $\mathsf{SB}$ reveals $(x_1, x_N)$. Thus, the influence of $\mathcal{A}$ on the outcome is unachievable in the ideal execution, hence a contradiction.

We define the subsets $\mathcal{Q} := \{P_1, \ldots, P_{N-t=2t-4}\}$ and $\mathcal{Q}' := \{P_t, \ldots, P_{N-1=3t-5}\}$, which we may denote 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, each of these quorums, respectively, when all its members behave honestly and do not hear from the outside, then they must output in a noninfinite number of steps. $\mathcal{A}$ corrupts $P_2$. $\mathcal{A}$ 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 that they would have if they were 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., from now 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, they all output after some non-infinite number of steps. 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 $\mathcal{Q}'$ which learns the output $(x_1, x_N)$.

***If the probing is successful, i.e., upon this event***, then we have that (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, then $\mathcal{A}$ (immediately) "freezes" $\mathcal{Q}'$, i.e., delivers no more message to honest players in $\mathcal{Q}'$, so that none of them will ever output. Then $\mathcal{A}$ reinitializes corrupt players in $\mathcal{Q} \cap \mathcal{Q}'$ back to their $\mathsf{States}(\mathcal{Q} \cap \mathcal{Q}')$ just after the first round. Then $\mathcal{A}$ "un-freezes" the set of honest players in $\mathcal{Q}$, and have all players in $\mathcal{Q}$ play honestly the protocol, with their messages delivered in round-robin order. Let us now specify the behavior of $P_i$.

***Choosing $P_i$ and its joint action with $\mathcal{A}$.*** By $(B, M)$, where $M = (M_1, \ldots, M_N)$ we denote 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 corresponding to 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 be 1, given that

(1) $x_1 = \sigma$,
(2) $P_N$'s messages in round 1 are $(B, M)$ then $P_N$ becomes silent,
(3) and 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 then become silent.

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

***Lemma 7.*** *$(B^0, M_1^0)$ and $(B^1, M_1^1)$ are computationally indistinguishable.*

*Proof.* Assume that there exists 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 all 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 8.*** $\forall \sigma \in \{0, 1\}, \quad q_\sigma(B^0, M_1^0) - q_\sigma(B^1, M_1^1) = \mathsf{negl}$.

*Proof.* For each $\sigma$, we build a distinguisher $\mathscr{E}$ for the distributions of Lemma 7 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.* This is where we used that there is no setup. Otherwise, if there had been a bulletin board PKI, then $\mathscr{E}$ would have had to initialize players with an internal state compatible with their public keys, thereby somehow guessing their secret keys. Actually, such $\mathscr{E}$ could not exist since *there exists* a secure $N$-SB under a bulletin board PKI setup. Namely: players broadcast PVSSs of their inputs, then threshold-decrypt the PVSSs over asynchronous channels.

Consider now the following four pairs of probabilities:

$$Q_1 = q(B^1, M_1^1, \ldots, M_{N-t}^1, 0, \ldots, 0)$$
$$Q_2 = q(B^1, M_1^1, 0, \ldots, 0, \ldots, 0)$$
$$Q_3 = q(B^0, M_1^0, 0, \ldots, 0, \ldots, 0)$$
$$Q_4 = q(B^0, M_1^0, \ldots, M_{N-t}^0, 0, \ldots, 0)$$

By the protocol's correctness, we have that $Q_1 \geq 1 - \mathsf{negl}$. Indeed, 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 $(B^1, M_1^1, \ldots, M_N^1)$, in which case their output must be 1 since $P_N$ acts honestly. Symmetrically we have $Q_4 \leq \mathsf{negl}$. By Corollary 8, $Q_2 - Q_3 = \mathsf{negl}$. Thus, there is a substantial difference either between $Q_1$ and $Q_2$ or $Q_3$ and $Q_4$. Without loss of generality 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.:

$$|q_0(B^1, M_1^1, \ldots, M_i^1, 0, \ldots, 0) - q_1(B^1, M_1^1, \ldots, M_{i-1}^1, 0, \ldots, 0)| > \frac{1}{6(n-t)} \tag{2}$$

***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 $x_1 = 0$, $P_i$ is instructed to play as if it did not have 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 (2), while in the latter case, follows the distribution which defines the $q_1$ on the right-hand in (2). Thus, the joint action of $P_i$ and $P_N$ will has for effect to substantially correlate the second output $x_N$ with $x_1$. □

## References


[AAPP22] I. Abraham, G. Asharov, S. Patil, and A. Patra. "Asymptotically Free Broadcast in Constant Expected Time via Packed VSS". In: *TCC*. 2022.

[ABV+12] S. Agrawal, X. Boyen, V. Vaikuntanathan, P. Voulgaris, and H. Wee. "Functional encryption for threshold functions (or fuzzy IBE) from lattices". In: *PKC*. 2012.

[ACC+21] M. Albrecht et al. "Homomorphic Encryption Standard". In: *Protecting Privacy through Homomorphic Encryption*. 2021.

[ACD+19] M. Abspoel, R. Cramer, I. Damgård, D. Escudero, and C. Yuan. "Efficient Information-Theoretic Secure Multiparty Computation over $\mathbb{Z}/p^k\mathbb{Z}$ via Galois Rings". In: *TCC*. 2019.

[ACGJ18] P. Ananth, A. R. Choudhuri, A. Goel, and A. Jain. "Round-optimal secure multiparty computation with honest majority". In: *CRYPTO*. 2018.

[AJL+12] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. "Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE". In: *EUROCRYPT*. 2012.

[AMN+20] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin. "Sync HotStuff: Simple and Practical Synchronous State Machine Replication". In: *IEEE S&P*. 2020.

[AP14] J. Alperin-Sheriff and C. Peikert. "Faster Bootstrapping with Polynomial Error". In: *CRYPTO*. 2014.

[BDO23] L. Braun, I. Damgård, and C. Orlandi. "Secure Multiparty Computation from Threshold Encryption based on Class Groups". In: *CRYPTO*. 2023.

[BGG+18] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. Rasmussen, and A. Sahai. "Threshold Cryptosystems from Threshold Fully Homomorphic Encryption". In: *CRYPTO*. 2018.

[BHKL18] A. Barak, M. Hirt, L. Koskas, and Y. Lindell. "An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants". In: *CCS*. 2018.

[BHN10] Z. Beerliová-Trubíniová, M. Hirt, and J. B. Nielsen. *Almost-Asynchronous MPC with Faulty Minority*. PODC'10. we refer to eprint 2008/416. 2010.

[BHP17] Z. Brakerski, S. Halevi, and A. Polychroniadou. "Four round secure computation without setup". In: *TCC*. 2017.

[BJMS20] S. Badrinarayanan, A. Jain, N. Manohar, and A. Sahai. "Secure MPC: Laziness Leads to GOD". In: *ASIACRYPT*. 2020.

[BLL20] E. Blum, C.-D. Liu-Zhang, and J. Loss. "Always have a backup plan: fully secure synchronous MPC with asynchronous fallback". In: *CRYPTO*. 2020.

[BS23] K. Boudgoust and P. Scholl. "Simple Threshold (Fully Homomorphic) Encryption From LWE With Polynomial Modulus". In: (2023).

[CCK23] J. H. Cheon, W. Cho, and J. Kim. *Improved Universal Thresholdizer from Threshold Fully Homomorphic Encryption*. ePrint 2023/545. 2023.

[CCL+20] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. "Bandwidth-efficient threshold EC-DSA". In: *PKC*. 2020.

[CDKS19] H. Chen, W. Dai, M. Kim, and Y. Song. "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference". In: *CCS*. 2019.

[CDN01] R. Cramer, I. Damgård, and J. B. Nielsen. "Multiparty Computation from Threshold Homomorphic Encryption". In: *EUROCRYPT*. 2001.

[CGJ+17] A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers. "Fairness in an Unfair World: Fair Multiparty Computation from Public Bulletin Boards". In: *CCS*. 2017.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. "Verifiable secret sharing and achieving simultaneity in the presence of faults". In: *FOCS*. 1985.

[CGZ20] R. Cohen, J. Garay, and V. Zikas. "Broadcast-Optimal Two-Round MPC". In: *EUROCRYPT*. 2020.

[CH18] H. Chen and K. Han. "Homomorphic Lower Digits Removal and Improved FHE Bootstrapping". In: *EUROCRYPT*. 2018.

[Cho20] A. Choudhury. "Brief Announcement: Almost-surely Terminating Asynchronous Byzantine Agreement Protocols with a Constant Expected Running Time". In: *PODC*. 2020.

[CL15] G. Castagnos and F. Laguillaumie. "Linearly Homomorphic Encryption from DDH". In: *CT RSA*. 2015.

[CLO+13] A. Choudhury, J. Loftus, E. Orsini, A. Patra, and N. P. Smart. "Between a Rock and a Hard Place: Interpolating between MPC and FHE". In: *ASIACRYPT*. 2013.

[CP23] A. Choudhury and A. Patra. "On the Communication Efficiency of Statistically-Secure Asynchronous MPC with Optimal Resilience". In: *J. Cryptol.* (2023).

[CSS+22] S. Chowdhury, S. Sinha, A. Singh, S. Mishra, C. Chaudhary, S. Patranabis, P. Mukherjee, A. Chatterjee, and D. Mukhopadhyay. *Efficient Threshold FHE with Application to Real-Time Systems*. ePrint 2022/1625. 2022.

[DDE+23] M. Dahl, D. Demmler, S. Elkazdadi, A. Meyre, J.-B. Orfila, D. Rotaru, N. P. Smart, S. Tap, and M. Walter. *Noah's Ark: Efficient Threshold-FHE Using Noise Flooding*. ePrint 2023/815. 2023.

[DHL21] G. Deligios, M. Hirt, and C.-D. Liu-Zhang. "Round-efficient byzantine agreement and multiparty computation with asynchronous fallback". In: *TCC*. 2021.

[DMR+21] I. Damgård, B. Magri, D. Ravi, L. Siniscalchi, and S. Yakoubov. "Broadcast-Optimal Two Round MPC with an Honest Majority". In: *CRYPTO*. 2021.

[DRSY23] I. Damgård, D. Ravi, L. Siniscalchi, and S. Yakoubov. "Minimizing Setup in Broadcast-Optimal Two Round MPC". In: *EUROCRYPT*. 2023.

[Feh98] S. Fehr. "Span Programs over Rings and How to Share a Secret from a Module". MA thesis. ETH Zurich, 1998.

[FS01] P.-A. Fouque and J. Stern. "One Round Threshold Discrete-Log Key Generation without Private Channels". In: *PKC*. 2001.

[FV12] J. Fan and F. Vercauteren. "Somewhat practical fully homomorphic encryption." In: *IACR ePrint* (2012).

[GGOR13] J. Garay, C. Givens, R. Ostrovsky, and P. Raykov. "Broadcast (and Round) Efficient Verifiable Secret Sharing". In: *ITC*. 2013.

[GIKR02] R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. "On 2-Round Secure Multiparty Computation". In: *CRYPTO*. 2002.

[GIKV23] R. Geelen, I. Iliashenko, J. Kang, and F. Vercauteren. "On Polynomial Functions Modulo $p^e$ and Faster Bootstrapping for Homomorphic Encryption". In: *EUROCRYPT*. 2023.

[GLS15] S. Dov Gordon, F.-H. Liu, and E. Shi. "Constant-Round MPC with Fairness and Guarantee of Output Delivery". In: *CRYPTO*. 2015.

[GMP19] N. Genise, D. Micciancio, and Y. Polyakov. "Building an Efficient Lattice Gadget Toolkit: Subgaussian Sampling and More". In: *EUROCRYPT*. 2019.

[GMPS21] V. Goyal, E. Masserova, B. Parno, and Y. Song. "Blockchains Enable Non-Interactive MPC". In: *TCC*. 2021.

[GO14] J. Groth and R. Ostrovsky. "Cryptography in the multi-string model". In: *J. of Cryptol.* (2014).

[GPS19] Y. Guo, R. Pass, and E. Shi. "Synchronous, with a Chance of Partition Tolerance". In: *CRYPTO*. 2019.

[GSW13] C. Gentry, A. Sahai, and B. Waters. "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based". In: *CRYPTO*. 2013.

[GV22] S. Gentry Craig and Halevi and L. Vadim. "Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties". In: *EUROCRYPT*. 2022.

[HNP05] M. Hirt, J. B. Nielsen, and B. Przydatek. "Cryptographic Asynchronous Multi-party Computation with Optimal Resilience". In: *EUROCRYPT*. 2005.

[JRS17] A. Jain, P. M. R. Rasmussen, and A. Sahai. *Threshold Fully Homomorphic Encryption*. ePrint 2017/257. 2017.

[KJY+20] E. Kim, J. Jeong, H. Yoon, Y. Kim, J. Cho, and J. H. Cheon. "How to Securely Collaborate on Data: Decentralized Threshold HE and Secure Key Update". In: *IEEE Access* (2020).

[KKL+22] T. Kim, H. Kwak, D. Lee, J. Seo, and Y. Song. *Asymptotically Faster Multi-Key Homomorphic Encryption from Homomorphic Gadget Decomposition*. eprint 2022/347. 2022.

[KMM+23] A. Kate, E. V. Mangipudi, P. Mukherjee, H. Saleem, and S. A. K. Thyagarajan. *Non-interactive VSS using Class Groups and Application to DKG*. ePrint 2023/451. 2023.

[KÖA23] J. Klemsa, M. Önen, and Y. Akin. "A Practical TFHE-Based Multi-Key Homomorphic Encryption with Linear Complexity and Low Noise Growth". In: *ESORICS*. 2023.

[LPR13a] V. Lyubashevsky, C. Peikert, and O. Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *J. ACM* (2013).

[MTBH21] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux. "Multiparty homomorphic encryption from ring-learning-with-errors". In: *PoPETS* (2021).

[MW16] P. Mukherjee and D. Wichs. "Two round multiparty computation via multi-key FHE". In: *EUROCRYPT*. 2016.

[NRS+20] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang. "Improved Extension Protocols for Byzantine Broadcast and Agreement". In: *DISC*. 2020.

[Par21] J. Park. "Homomorphic Encryption for Multiple Users with Less Communications". In: *IEEE Access* (2021).

[PR18] A. Patra and D. Ravi. "On the Power of Hybrid Networks in Multi-Party Computation". In: *IEEE Transactions on Information Theory* (2018).

[Reg05] O. Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *STOC*. 2005.

[SBKN21] N. Shrestha, A. Bhat, A. Kate, and K. Nayak. *Synchronous Distributed Key Generation without Broadcasts*. ePrint 2021/1635. 2021.

# A Further Details on Related Works

## A.1 Related Approaches with FHE

**A.1.1 Details on the Approach of [GLS15]** (-1) players receive a uniform random string. (0) based on the received string, players generate and publish GSW public keys. (1) each player $P_i$ encrypts its input. The encryption consists of a ciphertext of it under its own GSW public key, appended with $N-1$ encryptions of 0 under the keys of the other players. The whole vector output by $P_i$ is denoted as a Flexible$_i$ ciphertext (more details below). $P_i$ broadcasts it. (1) *In parallel*, players perform the second event of a $\mathsf{DKG}$, establishing a common $(N,t)$-TFHE public key. (2) Upon learning this key, players are able to Transform, locally and deterministically, the Flexible ciphertexts into TFHE ciphertexts under the common key. Then players proceed with local evaluation of the circuit; then finally threshold decryption, which can done over asynchronous P2P channels, by our observation.

More precisely, Flexible$_i$ of player $i$ goes as follows: on input $m_i$, output the vector $\hat{\mathsf{c}}_i = (\mathsf{c}_{i,1}, \mathsf{c}_{i,2}, \ldots, \mathsf{c}_{i,N})$, where $\mathsf{c}_{i,i}$ is a ciphertext of input $m_i$ under $P_i$'s GSW public key, while the other $\mathsf{c}_{i,j}$s are GSW ciphertexts of 0 under each $P_j$'s public key. By construction, since every Flexible$_i$ scheme has its secret key known by the corresponding $P_i$, it has threshold 0 with respect to the players, which prevents it from being used by external input owners. Another particularity of Flexible ciphertexts is that all the $N$ GSW ciphertexts contained are actually generated with the same secret randomness. Thus, given that half of these ciphertexts are encrypted under GSW public keys which were generated by the adversary $\mathcal{A}$, they a priori give $\mathcal{A}$ an extra advantage to guess the plaintext of $\mathsf{c}_{i,i}$. For the security of GSW to hold, their public keys are thus scaled slightly larger ($m = \Omega((n+N)\log(q))$ vs $m = \Omega((n)\log(q))$ in GSW), in order to apply the leftover-hash-lemma (LHL [GLS15, lemma 1]).

However this technique is not efficiently transposable to our setting, for the following reasons. To start with, referring to §D, suppose that the adversary is given one (or several) BFV encryptions of 0 under semi-maliciously generated key(s) $(a, b_i)$, i.e., $\left(u\, b_i + e_{0,i}^{(Enc)}, u\, a + e_{1,i}^{(Enc)}\right)$, which would all be generated with the same secret randomness $u \leftarrow \mathscr{X}_q$. Then this may provide it with 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 public keys, $a := \mathbf{a}[0]$, 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 [DGKS21, §1] evidenced a counterexample showing that the LHL does not hold in general (a leakage of $1/n$ of the secret randomness which would make the outcome far from indistinguishable from uniform). They also point that [LPR13b, 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 sufficiently large parameter. Concretely, we would apply "regularity" to $\mathbf{A}.\mathbf{u}$, where $A$ would be a matrix with $N$ rows encoding all public 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 [DGKS21, §1].

As a final remark on [GLS15], notice that, on the face of it, their DKG is specified over pairwise channels, thus a player which 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 [GLS15, 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 $\mathsf{DKG}$, as well as the one of [FS01], can be seen as a particular case of $\mathcal{F}_{\mathsf{LSSS}}$ 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 $N+1$ [GLS15, §B, §C].

**A.1.2 Based on Multi-key FHE and on $(N, N)$-Threshold FHE** The recent work [CDKS19] constructs a multikey variant (MFHE) of the BFV [FV12] and CKKS [CKKS17; LM21] schemes, with ciphertexts of linear size in the number $N$ of players. So this is $N\times$ larger than single-key FHE (it was even quadratic before: [MW16; BHP17]). Their homomorphic evaluation complexity is quadratic in $N$ [CDKS19, Table 1]. Recently, [KKL+22; KOA23] reduced the overhead in homomorphic evaluation complexity to linear in $N$, which is still $N\times$ larger than under single-key.

Turning to $(N, N)$-threshold FHE, this particular threshold $t := N$ yields a simplification over general $(N, t)$-$\mathsf{ThFHE}$, which is that establishing the threshold key can be done straight from the PKI: each player generates a key pair and publishes the public key, the threshold key is then defined as the sum of public keys. The secret decryption key is equal to the sum of the secret keys, which by definition constitutes an additive secret sharing.

However, both MFHE and $(N, N)$-threshold FHE suffer from nonconstant worst case latencies. Indeed threshold decryption fails as soon as one player aborts, by definition of $N$-out-of-$N$ access structure. *Assuming synchrony*, then one could imagine a compilation of such protocols into ones guaranteeing GOD, at the cost of $\Omega(t)$ consecutive broadcasts and re-evaluations of the circuit. Namely: require players to send their decryption shares via terminating reliable broadcast. Then, if some instances of broadcast for some players returned $\perp$, discard them and restart the whole protocol with the remaining players. Recall that we have another non-responsive event at the beginning of every execution, which is the time-out after which players which did not broadcast an encryption of their input are discarded from the protocol. Thus, in the case of $t$ consecutively aborting players in the distributed decryption, the execution is restart $t$ times.

There are moreover other sources of potential slowdown in specific approaches, such as the $(N, N)$-TFHE of [Par21]. There, generation of the relinearization key is done after the threshold key is known, and completes only if all players who contributed to generation of the threshold key, also participate. Notice that there is the same issue for the distributed generation of the bootstrapping key. Thus, if one had wanted to enable GOD in [Par21], one would need to also require that players send their contribution to the relinearization key via broadcast. Let us observe that this broadcast could be done in parallel of the broadcast of encrypted inputs. However, if some instances of broadcast for some players returned $\perp$, then these players need to be discarded and the whole $\mathsf{DKG}$ restarted with the remaining players.

**A.1.3 Based on $(N, t)$-threshold FHE** The work [AJL+12] (in their paragraph "Fairness") was the first to introduce $(N, t)$-$\mathsf{ThFHE}$ with thresholds $t$ lower than $N-1$. Unfortunately, their construction adds 2 additional rounds to the 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 [KJY+20, §IV B] implement a $\mathsf{DKG}$ for CKKS with reconstruction from $t+1$-out-of-$N$ Shamir shares of the secret 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 $\mathsf{DKG}$ of [KJY+20, §IV B] is proposed in [MBH23]. Since [KJY+20; MBH23] follow the $\mathsf{DKG}$-then-input distribution approach, they require at least two broadcasts, which is incompatible with our requirement. Finally, the MPC protocols suggested in [KJY+20; MBH23] are also non-robust, by lack of a robust distributed generation of relinearization & bootstrapping keys. Recall that this is an issue which we also solve in this work (in Section 6).

The work [BGG+18] also proposed $(N, t)$-$\mathsf{ThFHE}$ schemes. However, their §5 leaves unspecified the $\mathsf{DKG}$, 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.2 Garbled circuits, and Joint Use with BFV for Numerical Computation

Note that the communication size in [ACGJ18] is improved by [ACGJ20] to $O(|C| + N^4)$, but with a suboptimal corruption threshold $\frac{1}{2} - \epsilon$, this is why they recommend to use [ACGJ18] for small values of $N$. [AZ21], which provides security with abort, allows external inputs.

Garbled circuits and TFHE benefit from being used together, e.g., in use-cases of machine learning. Indeed, each layer of a neural network has a large circuit of linear gates over arithmetic values, where homomorphic encryption or secret-sharing techniques are very fast, whereas conversion of these linear operations to a boolean circuit is prohibitively expensive. Thus, the typical approach is hybrid: on the one hand, computations of linear functions (FC/CNNs) use either homomorphic encryption (Gazelle [JVC18]) or secret-sharing (MiniONN [LJLA17]). Then, non-linear functions are computed with garbled circuits (except possibly matrix products [CKR+20]).

### 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 [HNP05]. The broadcast of [FN09] uses a few synchronous rounds, then guarantees responsive eventual output delivery, under a honest majority. The consensus of [Cho20] with $t < N/3$ tolerance uses one initial round of synchrony. In information-theoretic MPC with $t < N/3$, [PR18] reached an optimal 3 initial synchronous rounds. In the setting of MPC with non-constant latency, then [BHN10] 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. If they were instantiated with CL (Section 3.1.1), the $\mathsf{DKG}$ would add at least 1 other broadcast round (in the bulletin board PKI model).

The protocol [BLL20] proceeds by intervals of fixed duration, denoted rounds, of which the number grows with the depth of the circuit. In particular it is not responsive. 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 [CGHZ16; Coh16] are purely asynchronous, i.e., responsive, 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 More on Broadcast $\mathsf{BC}$ with Possibly External Senders

**B.1.1 Comments on Implementations** Implementations of $\mathsf{BC}^{P,\mathscr{P}}$ for $P \in \mathscr{P}$ are mainstream since [LSP82]. They necessarily assume that $P$ and $\mathscr{P}$ are able to start the protocol synchronously, and that $P$ is provided with authenticated $\Delta$-synchronous channels to $\mathscr{P}$. Otherwise, one would need to relax the specification into allowing that the output be not always the one of $\mathcal{S}$ whenever honest, as the case in weak/partial synchronous models [GPS19; ANRX21]. Indeed, a $\mathcal{S}$ which lags behind could not be told apart from a dishonest $\mathcal{S}$ which remains silent. In our generalized context with possibly external receivers, implementation of $\mathsf{bPKI}^P$ can be obtained directly from $\mathsf{BC}^{P,\mathscr{P}}$, as: players run $\mathsf{BC}^{P,\mathscr{P}}$, then, upon receiving their output, each player sends it to all external receivers, i.e., in $\mathscr{L}$. Each $\ell \in \mathscr{L}$ waits to receive $t + 1$ times the same value, then outputs it. In our generalized context with a possibly external sender $\ell \in \mathscr{L}$ : an implementation of $\mathsf{BC}^{\ell}$ can be obtained assuming (i) channels from $\ell$ to $\mathscr{P}$ with guaranteed delay $\delta$, and that (ii) $\ell$ and $\mathscr{P}$ are able to start synchronously, by: $\ell$ sends its input to all $\mathscr{P}$, then after $\delta$, $\mathscr{P}$ perfom Byzantine consensus with inputs what they received from $\ell$ ($\perp$ if none). Notice that our protocol will instruct all players and owners to act as senders in $\mathsf{BC}$ upon completion of all instances of $\mathsf{bPKI}$. As discussed above, implementation of $\mathsf{BC}$ thus necessarily requires senders and $\mathscr{P}$ to start synchronously at a point in time after completion of these instances.

**B.1.2 Sub-sessions** For sake of UC analysis, in our MPC protocol we specify multiple broadcast instances per sender. This is why we formalize $\mathsf{BC}^{\mathcal{S},\mathcal{R}}$ with sub-session identifiers denoted $\mathsf{ssid}$. Importantly, we force $\mathsf{BC}^{\mathcal{S},\mathcal{R}}$ 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 $\mathsf{ssid}$ is the label of the variable which is broadcast. Furthermore, we make the abuse of notation, in our protocol, to have one sender concatenate multiple broadcasts instances of several variables at once, likewise for the $\mathsf{input}$ command of $\mathcal{F}_{\mathsf{LSSS}}$. 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 (1).

### B.2 More on our bPKI, and the { Bulletin board / Untrusted / bare } PKI model

**B.2.1 Comparison with Canetti's certification authority** Recall that we formalized $\mathsf{bPKI}$ as a mere broadcast, with possibly external receivers. The closest UC definition of $\mathsf{bPKI}$ known to us is the "certification authority" $\mathcal{F}_{CA}$ of Canetti [Can04]. There, it is presented in the same use-case, namely, a service enabling players to record their public keys. In appearance, its formalization is different than ours since it provides delayed output only to players who requested it. But it actually seems to implement our $\mathsf{bPKI}$, by having players repeatedly query an output. Notice the conflict of terminology in [Can04], in which the symbol $\perp$ means that the instance did not terminate yet, instead of, in our definition, did terminate with adversarially chosen output $\perp$.

**B.2.2 PKI assumptions in the literature** The $\mathsf{bPKI}$ 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)" [ACGJ18; BCG21; GJPR21]. The PKI model was first sketched in [CGGM00, §6], then denoted as "bare PKI" in [ACGJ18]. It is renamed as "untrusted PKI" in [GJPR21]. As specified in [CGGM00, §6.1][GPS19; DMR+21], the PKI model is slightly stronger than ours, since it abstracts-out all the implementation constraints discussed in §B.1, in a way which could be phrased as: (i) all instances of $\mathsf{bPKI}$ 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 $\mathsf{BC}$. By contrast, the implementation of $\mathsf{BC}$ in [FN09] 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 [DGKN09; LLM+20].

Likewise, in the specific case of protocols for Byzantine agreement, [BCG21] also assume access to the board only before players are assigned their inputs, which is the same assumption in our case. We refer to [BCG21] 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.2.3 The power of { Bulletin board / Untrusted / bare } PKI in MPC** By construction, $\mathsf{bPKI}$ does not perform any check 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 [CDPW07], where it is proven to have strictly more power. Notice that implementing KRK without $\overline{\mathcal{G}}_{\mathrm{URS}}$ would, in turn, require an extra event before $\mathsf{bPKI}$ 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 [GJPR21] 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, $\mathsf{bPKI}$ is the minimal setup necessary in all implementations of synchronous broadcast for $t \geq N/3$, since the seminal [LSP82]. Also, [Bor96, Thm 1] shows that the relaxation of $\mathsf{bPKI}$ 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 a 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, [GKLP16] 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, [GKOPZ20] implement an actual untrusted PKI, i.e., the $\mathcal{F}_{CA}$ of [Can04], which they denote $\mathcal{F}_{REG}$.

**B.2.4 A Related Notion of Bulletin Board** The terminology "bulletin board" is introduced in [CGJ+17], where it is used as a support of communication for MPC, as well as in [GMPS21]. Therefore, this usage contrasts with the mainstream usage of a "bulletin board PKI", which is also ours, in which players cannot write on the board after the time $t = 0$ when owners receive their inputs. However, the formalization of [CGJ+17; GMPS21] seems identical to our bPKI. The only formal difference is that they specify synchrony, with possibly offline players. Thus, in their terminology, our "eventual delivery" is replaced by something which could be phrase as "provides output even to players which were offline while the bulletin-board was written on".

### B.3 $\mathcal{F}_{\mathsf{AUTH}}$

$\mathcal{F}_{\mathsf{AUTH}}^{\mathcal{S},R}$

On input $(\mathsf{input}, \mathsf{ssid}, v)$ from $\mathcal{S}$: then provide $R$ with delayed output $(\mathsf{ssid}, v)$.

Fig. 7: (Asynchronous, Public) Authenticated message transmitting for $\mathcal{S}$ and $R$.

In the two last (asynchronous) steps of our MPC protocol, when $\mathcal{F}_{\mathsf{LSSS}}$ is instantiated with $\Pi_{\mathsf{LSSS}}$, each player is instructed to send the same opening share to all using $\mathcal{F}_{\mathsf{AUTH}}$. 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: secret key and a key noise, compatible with the $\mathbf{b_i}$ which they broadcast in (1). Thus, for whatever compatible pair which they could exhibit, the decryption share coming with it is necessarily correct.

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

Notice that, contrary to BC and bPKI, there is no activate command for the output from a corrupt sender in $\mathcal{F}_{\mathsf{AUTH}}$. Hence, our definition of "complete execution" does not require termination of $\mathcal{F}_{\mathsf{AUTH}}$ instances for corrupt senders. Requiring so would have lead to a definition of "guaranteed output delivery" which would not even have withstanded fail-stop adversaries.

On the other hand, in order to guarantee GOD in our MPC protocol, we need implementations of $\mathcal{F}_{\mathsf{AUTH}}$ with guaranteed eventual delivery from a honest sender. Notice that without this guarantee, then no asynchronous protocol could have guaranteed termination.

### B.4 $\overline{\mathcal{G}}_{\mathbf{URS}}$

$\overline{\mathcal{G}}_{\mathrm{URS}}$ is a particular case of $\mathcal{F}_{crs}$ in [CLOS02].

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

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 $\overline{\mathcal{G}}_{\text{URS}}$ could have been implemented, at the cost of extra preliminary non-responsive steps. Notice that optimized implementations exist, but at the cost of more assumptions. E.g., [CD20] requires a CRS for generation of NIZKs, while [KMQR21] requires an initial seed.

On the other hand, when upgrading $\overline{\mathcal{G}}_{\text{URS}}$ 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 [BHZ21].

### B.5 More on Guaranteed Eventual Output Delivery (GOD)

Our formalism of UC functionalities which deliver their outputs only when allowed by the adversary, is the classical one [Can01]. It is still in use for its simplicity, e.g., in [AAPP22; CP23].

Subsequent works [CGHZ16; LLM+20] formalised eventual delivery in UC. To delay a message, the adversary somehow repeatedly presses a button (technically: it enters a delay in unary notation). Since it is polynomial machine, it is exhausted at some point so the message gets delivered. We did not choose this approach to keep the presentation simple.

Consider a protocol $\Pi$, in the ($\overline{\mathcal{G}}_{\text{URS}}$,$\mathcal{F}_{\mathsf{AUTH}}$, $\mathsf{BC}$, $\mathsf{bPKI}$)-hybrid model. Extending [Lam06], we denote as *complete* an execution of $\Pi$ in which: honest players took all the steps they could, the adversary responded to all requests from $\mathsf{BC}$ by providing input messages on behalf of all corrupt senders, allowed all delivery requests of $\mathsf{BC}$ and $\mathcal{F}_{\mathsf{AUTH}}$ for honest senders to honest receivers and all delivery requests from $\overline{\mathcal{G}}_{\text{URS}}$ for honest receivers. We say that a protocol $\Pi$ has GOD if, in every *complete* execution, then every honest player outputs. [Although our MPC protocol has finite executions, this definition also applies to infinite ones [DLS88, Remark 2].]

**Claim:** *If $\Pi$ has GOD in the ($\overline{\mathcal{G}}_{URS}$,$\mathcal{F}_{\mathsf{AUTH}}$,$\mathsf{BC}$, $\mathsf{bPKI}$) model, then, when ($\mathsf{BC}$, $\mathsf{bPKI}$, $\mathcal{F}_{\mathsf{AUTH}}$) are instantiated by protocols with guaranteed eventual output delivery, then $\Pi$ has GOD in the usual sense.*

*Proof:* From the specifications of the previous functionalities, an execution of $\Pi$ is complete if and only if: (i) all instances of $\mathsf{BC}$ and $\mathsf{bPKI}$ delivered an output to every honest player, whatever the sender, and (ii) every input to $\mathcal{F}_{\mathsf{AUTH}}$ from a honest sender to a honest receiver was delivered. Thus, when (i) $\mathsf{BC}$ and $\mathsf{bPKI}$ are instantiated by Broadcast protocols, and (ii) $\mathcal{F}_{\mathsf{AUTH}}$ by channels with eventual delivery, then every execution of $\Pi$ is complete.

### B.6 Straightforward Generalizations of $\mathcal{F}_{\mathsf{C}}$

To start with, in our MPC protocol, messages of input owners do not depend on the actual circuit to be computed. Thus, our protocol actually achieves the "delayed function" property ([ACGJ18]), i.e., it implements a more general functionality to which honest players can possibly provide the circuit to be computed *after* the broadcast instances from all owners terminated, possibly with some $\perp$ or badly formed messages for some of them. Such a functionality is also known as *reactive* [CDN15, §5]. We do formalize and implement such a functionality in §3.2 for the specific case of linear forms, because we need this possibility in the implementation of MPC protocol.

As such, our definition of $\mathcal{F}_{\mathrm{C}}$ imposes that every honest player, and only them, outputs. However, it straightforwardly generalizes to capture protocols delivering outputs to arbitrary receivers, be them included, overlapping, or disjunct, from the set $\mathscr{P}$ of players. Interestingly, different feasibility bounds hold for general receivers, e.g., the case of Solitary MPC [BMMR21].

### B.7 Reminder of the UC model, and our Simulators

Consider a protocol $\Pi$, a functionality $\mathcal{F}$ and any PPT environment $\mathscr{E}$ which can interact with either one or the other of the following protocols, without being informed which of them. In every execution, $\mathscr{E}$ may provide inputs to honest players, may provide instructions to the adversary, and may observe the outputs of players. At some point of every execution, $\mathscr{E}$ must output a bit. In the first, denoted "real" $REAL_{\Pi}$, $\mathscr{E}$ interacts with the dummy adversary $\mathcal{A}$, and honest players follow the actual protocol $\Pi$. In the second,

denoted "ideal" $IDEAL_{\mathcal{F},\mathcal{S},\mathscr{E}}$, $\mathscr{E}$ interacts with an adversary $\mathcal{S}$, while honest players are connected only to $\mathcal{F}$. Following [Can01], we say that protocol $\Pi$ UC emulates $\mathcal{F}$ if there exists a PPT machine, denoted as the simulator $\mathcal{S}$, such that for any such $\mathscr{E}$, the gap of probabilities of outputing 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 [Can01, §4.3.1], to be equivalent to UC emulation against any adversary.

**B.7.1 Simulators in our UC Proofs** Our simulators $\mathcal{S}$, unless specified, have the following high level behavior. Initiate in its head: a set of $N$ players and $|\mathscr{L}|$ Owners, along with the ideal functionalities with which they are meant to interact in the actual protocol, excepted $\overline{\mathcal{G}}_{\text{URS}}$, since it is an external resource. Upon corruption requests from $\mathscr{E}$, $\mathcal{S}$ labels in turn the corresponding simulated players or owners as corrupt. Upon every output from a simulated functionality to a simulated corrupt player, or, upon every $\mathsf{ReqInput}$ request from a simulated functionality to $\mathcal{S}$, then $\mathcal{S}$ immediately updates $\mathscr{E}$ with it, at would have done the actual dummy adversary.

We will abuse notations such as: "honest $P$ sends message to corrupt $Q$", where we actually mean that, in the simulated execution, $\mathcal{F}_{\mathsf{AUTH}}^{P,Q}$ requests ($\mathsf{ReqInput}$,$m$) to $\mathcal{S}$, then, upon receiving an instruction to $\mathsf{activate}$ from $\mathscr{E}$, then the simulated $\mathcal{F}_{\mathsf{AUTH}}^{P,Q}$ outputs to simulated corrupt $Q$, then immediately updates $\mathscr{E}$ with this reception by $Q$, as the dummy adversary would have done.

The same abuses of notation apply to other interactions of simulated corrupt players with other simulated functionalities, such as $\mathcal{F}_{\mathsf{LSSS}}$ in our proof in Appendix H. In particular, when we say that "$\mathcal{F}_{\mathsf{LSSS}}$ delivers some delayed output $c$ to some corrupt player $Q$", then we actually mean that: $\mathcal{S}$ creates a value $c$; simulates towards $\mathscr{E}$ that it received a request (a "$\mathsf{ReqDeliv}$") from $\mathcal{F}_{\mathsf{LSSS}}$ (which does not exist) for delayed output of $c$. Then, upon receiving an instruction to $\mathsf{activate}$ from the Environment, then $\mathcal{S}$ simulates towards $\mathscr{E}$ that it was notified by $Q$ (which does not exist) its reception of $c$ from $\mathcal{F}_{\mathsf{LSSS}}$.

### B.8 UC Non-Interactive Zero-knowledge (NIZK) functionality

**Non-Interactive Zero-knowledge** $\mathcal{F}_{\mathsf{NIZK}}$, following exactly [GOS06], upon request of a prover $P$ exhibiting knowledge of some witness $w$ verifying a public statement $(x, *) \in R$, delivers to $P$ the *delayed output* of a string $\pi$ which can subsequently be checked by any verifier to verify that $P$ does know some $w$ verifying $(x, w) \in R$.

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

The functionality is parameterized with an NP relation $R$ of an NP language $L$, and a prover $P$.

**Proof:** On input $(prove, \mathsf{sid}, \mathsf{ssid}, x, w)$ from $P$, ignore if $(x, w) \notin R$. Send $(\mathsf{ReqInput}, proof, x)$ to $\mathcal{A}$ and wait for answer $(\mathsf{activate}, \pi)$. Upon receiving the answer store $(x, \pi)$ and send $(proof, \mathsf{sid}, \mathsf{ssid}, \pi)$ to P.

**Verification:** On input $(verify, sid, ssid, x, \pi)$ from $V \in \mathscr{P}$, check whether $(x, \pi)$ is stored. If not send $(\mathsf{ReqInput}, verify, x, \pi)$ to and wait for an answer $(\mathsf{activate}, witness, w)$. Upon receiving of the answer, check whether $(x, w) \in R$ and in that case, store $(x, \pi)$. 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

UC implementations of $\mathcal{F}_{\mathsf{NIZK}}$ exist, which do not require honest majority [DDOPS01], but at the cost of requiring a uniform random string (URS). Nonwithstanding that [DDOPS01] allow 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 a honest majority assumption, then [CDPW07] prove that UC NIZK are non-implementable in the global common random string model, i.e., which we formalized as $\overline{\mathcal{G}}_{\text{URS}}$ in the particular case where the string is uniform.

Fortunately, the need of a URS can be escaped under honest majority, provided access to $\mathsf{bPKI}$, thanks to the technique denoted multi-string CRS [GO14; BJMS20].

### B.9 Semi-Malicious corruptions.

In our protocols and proofs we will consider what we define as *Semi-Malicious Corruptions*, following [AJL+12, §A.2] [BHP17; GLS15; BJMS20]. Semi-maliciously corrupt players and 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 $\mathcal{F}_{\mathsf{AUTH}}$ 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 $\mathcal{F}_{\mathsf{LSSS}}$ in the case of our $\Pi$, required for sending $m$ (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 a honest player would have been meant to have tossed upon sending $m$. $M$ can however use conflicting $(x, r)$ when sending different messages $m$, $m'$. Finally, we also require that the semi-malicious adversary can only $\mathsf{activate}$ an output $v$ to $\mathsf{BC}^M$ 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 $v = \perp$. 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. §11 of our implementation of $\mathcal{F}_{\mathsf{LSSS}}$, we do not need extractable secret keys. See §7.2 for the favorable consequence of this relaxation.

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

For $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$ for any $m_0, m_1 \in \mathscr{M}$, for any $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{pp})$ and for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, letting $\mathsf{c}_0 \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{ek}, m_0)$ and $\mathsf{c}_1 \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{ek}, m_1)$, we have

$$|\mathbb{P}[\mathcal{A}(\mathsf{ek}, m_0, m_1, \mathsf{c}_0) = 1] - \mathbb{P}[\mathcal{A}(\mathsf{ek}, m_0, m_1, \mathsf{c}_1) = 1]| \leq \mathrm{negl}(\lambda).$$

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

Notice that the specification of $\mathcal{F}_{\mathsf{LSSS}}$, done in Section 3.2, is related to the one denoted $F_{com}$ in [CDN15, p. V].

The minor difference is that we merged in one single command $\mathsf{LCOpen}$ the three separated commands of $F_{com}$ which were Addition, scalar Multiplication and Open.

The major difference is that $\mathcal{F}_{\mathsf{LSSS}}.\mathsf{LCOpen}$ produces an output as soon as it receives request from any $t+1$ players, i.e., is responsive. By contrast, $F_{com}$ needs request from all $t+1$ honest players to open a value.

### C.1 High Level Implementation of $\mathcal{F}_{\mathsf{LSSS}}$ from Public Secret Sharing (PSS).

To implement $\mathcal{F}_{\mathsf{LSSS}}$, we use the following (non interactive) algorithms as main ingredients.

***Definition 9 ((N, t)-LSSS).*** *A (N, t)-linear secret sharing scheme is defined by two algorithms* $(\mathsf{LS.Share}, \mathsf{LS.Reco})$ *for sharing and reconstructing a secret.* $\mathsf{LS.Share}$ *takes a secret* $s \in R_k$ *and outputs a set* $\{s^{(1)}, ..., s^{(N)}\}$ *of shares.* $\mathsf{LS.Reco}$ *takes as input a set of shares* $\{s^{(1)}, ..., s^{(d)}\}$ *and returns an integer* $s \in R_k$ *if* $d \geq t$*, or* $\perp$ *if* $d < t$*.*

Furthermore they must satisfy correctness and privacy, as follows.

*Correctness:* any $s \in, \mathcal{U} \subset [n]$ with $|\mathcal{U}| > t$ and any projection $S_{\mathcal{U}}$ of $S \leftarrow \mathsf{LS.Share}(s)$, it holds that $\mathsf{LS.Reco}(S_{\mathcal{U}}) = s$ with probability 1.

*Privacy:* For privacy we demand that for any $s, s' \in$ and set $\mathcal{U} \subset [n]$ with $|\mathcal{U}| \leq t$, the projections $S_{\mathcal{U}}$ of $S \leftarrow \mathsf{LS.Share}(s)$ and $S'_{\mathcal{U}}$ of $S' \leftarrow \mathsf{LS.Share}(s')$ are identically distributed.

*High level implementation* To send a secret $s$ to $\mathcal{F}_{\mathsf{LSSS}}$, generate a (N,t)-secret sharing of $s$ using $\mathsf{LS.Share}$. Then for all $i \in [N]$, encrypt each share $s^{(i)}$ under $P_i$'s public key: this constitutes a Public Secret Sharing. Later, to compute a linear combination $\Lambda$ of some inputs $(s_j)_j$: each player $P$ decrypts its share $s_j^{(P)}$ of each $s_j$, then evaluates $\Lambda$ on these shares: this constitutes a partial opening share. Then from any $t+1$ partial opening shares, the desired linear combination $\Lambda((s_j)_j)$ is efficiently reconstructible.

More formally, let $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be *any* public key encryption scheme satisfying IND-CPA. By convention, encryption under an incorrectly formatted public key $\mathsf{pk}^{\mathsf{PKE}}$, e.g., $\perp$, returns the plaintext itself.

***Def-Prop 10 (Public Secret Sharing).*** *Let us consider the following randomized function* $\mathsf{PSS}$*, parametrized by* $N$ *strings* $(\mathsf{pk}_i^{\mathsf{PKE}})_{i\in[N]}$*. On input* $s \in R_q$*: compute* $(s^{(1)}, ..., s^{(N)}) \leftarrow \mathsf{LS}_{R_q}.\mathsf{Share}(s)$*, output* $\left[\mathsf{Enc}(\mathsf{pk}_i^{\mathsf{PKE}}, s^{(i)}), i \in [N]\right]$*.* $\mathsf{PSS}$ *is IND-CPA for any* $\mathcal{A}$ *being given at most* $t$ *secret keys* $\left(\mathsf{sk}_i^{\mathsf{PKE}}\right)_{i\in\mathcal{I}\subset[N]}$*.*

An 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 $[s^{(i)},\ i \in \mathcal{I}]$. But by Property 12 in §C, for any chosen plaintext $s$, it varies in $U(R_q^t)$. We carry out this idea in §C, by lack of reference, with a reduction to multi-messages multi-keys IND-CPA of $\mathsf{PKE}$.

### C.2 Linear Secret Sharing over Polynomial Rings Modulo $p^e$.

For any commutative ring $A$ with unit 1, we denote as $A[Y]_t$ the polynomials of degree $\leq t$. For any $s \in A$, we denote $A[Y]_t^{(s)}$ the affine subspace evaluating at 0 to $s$, i.e., with constant coefficient $s$. Suppose that we are given a sequence $\{\alpha_0 := 0, \ldots, \alpha_N\} \in A^{N+1}$, such that all pairwise differences $\alpha_i - \alpha_j$ for $i \neq j$ are invertible in $A$. Such a sequence is denoted as "exceptional" in [ACD+19].

For $\mathcal{U} \subset (\alpha_i)_{i\in[0,\ldots,N]}$ we denote $\mathsf{Eval}_{\mathcal{U}} : P \in A[Y]_t \to [P(\alpha_i), i \in \mathcal{U}]$ the map returning the evaluations at points of $\mathcal{U}$. By [ACD+19, Thm 3], for every $(t+1)$-sized $\mathcal{U} \subset [0, \ldots, N]$, we have that $\mathsf{Eval}_{\mathcal{U}}$ is an *isomorphism*. For such $\mathcal{U}$, we denote the inverse of $\mathsf{Eval}_{\mathcal{U}}$ as "Polynomial Reconstruction" $\mathsf{PolReco}^{\mathcal{U}} : A^{t+1} \longrightarrow A[Y]_t$. We can construct is as: for each $i \in \mathcal{U}$, define the Lagrange polynomial as $\lambda_i^{\mathcal{U}}(Y) := \prod_{j\in\mathcal{U}\setminus\{i\}} \frac{Y-j}{i-j}$; then $\mathsf{PolReco}^{\mathcal{U}}([s^{(i)},\ i \in \mathcal{U}]) := \sum_{i\in\mathcal{U}} s^{(i)}.\lambda_i^{\mathcal{U}}(Y)$.

This isomorphism yields the following [ACD+19, Construction 1] of *Linear Sharing over A*. Define the randomized function $\mathsf{LS}_A.\mathsf{Share} : A \longrightarrow A^N$, as: on input a secret $s \in A$, sample $P \leftarrow U\left(A[Y]_t^{(s)}\right)$ then return $\mathsf{Eval}_{(\alpha_i)_{i\in[N]}}(P) \in A^N$ denoted "shares" of $s$. We denote "$(N,t)$-secret sharing of $s$" any such possible output of $\mathsf{LS}_A.\mathsf{Share}(s)$. In the other direction, for a $(t+1)$-sized $\mathcal{H} \subset [N]$, we have the map $\mathsf{SReco}^{\mathcal{H}} := \mathsf{Eval}_{\{0\}} \circ \mathsf{PolReco}^{\mathcal{H}}$ which takes as input any $t+1$ shares and reconstructs the secret. Also, surjectivity of $\mathsf{Eval}_{\{0\}\cup\mathcal{I}}$ for any $t$-sized $\mathcal{I}$, implies Uniformity of any $t$ shares of any fixed secret, cf. Property 12 of §C. In addition, let us introduce two useful $A$-linear maps:

- ***Perfect Simulation of (honest) Shares.*** For any $t$-sized $\mathcal{I} \subset [N]$, denote $\mathsf{ShSim}^{\mathcal{I}} := \mathsf{Eval}_{[N]\setminus\mathcal{I}} \circ \mathsf{PolReco}^{\{0\}\sqcup\mathcal{I}}$. It takes as input any $t$ (typically corrupt) shares $(s^{(i)})_{i\in\mathcal{I}}$ with some secret $s$, and reconstructs the *unique* set of shares $(s^{(h)})_{h\in\mathcal{H}}$, with indices $\mathcal{H} := [N]\setminus\mathcal{I}$, such that the whole $(s^{(j)})_{j\in[N]}$ forms a $(N,t)$-secret sharing of $s$;
- ***Inference of (Corrupt) Shares.*** For any $t+1$-sized $\mathcal{H} \subset [N]$, denote $\mathsf{ShInfer}^{\mathcal{H}} := \mathsf{Eval}_{[N]\setminus\mathcal{H}} \circ \mathsf{PolReco}^{\mathcal{H}}$. It takes as input any $t+1$ shares $(s^{(h)})_{h\in\mathcal{H}}$, and reconstructs the *unique* set of shares $(s^{(i)})_{i\in\mathcal{I}}$ , with (corrupt) indices $\mathcal{I} := [N]\setminus\mathcal{H}$, such that the whole $(s^{(j)})_{j\in[N]}$ forms a $(N,t)$-secret sharing of some $s$.

***Construction of an Exceptional Sequence for*** $R_q$***.*** The easy case is when all prime factors of $q$ are of size at least $N+1$. Then we have that $[0, \ldots, N] \subset R_q$ forms an exceptional sequence. Indeed, all $i-j$ for $\left\{(i,j) \in [0,\ldots,N]^2,\ i \neq j\right\}$ are invertible modulo all the prime factors of $q$, thus are invertible modulo $q$ by the Chinese remainders theorem (CRT), and thus in $R_q$. In the general case, we need to enlarge $R_q$. We do the construction for $q = p^e$ a prime power, then the case of composite $q$ follows from the CRT. Denote $d := \lceil \log_p(N+1) \rceil$. The construction is conceptually as follows, and made explicit in full details in §C.4.

Consider the $\mathbb{Z}/q\mathbb{Z}$-algebra $B := (\mathbb{Z}/q\mathbb{Z})[Y]/g(Y)$, where $g$ is monic of degree $d$, and irreducible modulo $p$, denoted "Galois ring extension". In [ACD+19] they observe that $B$ contains a $p^d$-sized exceptional sequence, from which they deduce linear secret-sharing over $B$, that we denote $\mathsf{LS}_B$. By tensorisation of $\mathsf{LS}_B$, over $\mathbb{Z}/q\mathbb{Z}$, with any inclusion of $\mathbb{Z}/q\mathbb{Z}$-algebras, e.g. $\mathbb{Z}/q\mathbb{Z} \hookrightarrow R_q$, we obtain a $S$-linear secret-sharing scheme $\mathsf{LS}_S$ over $S := R_q \otimes_{\mathbb{Z}/q\mathbb{Z}} B$. $R_q$ being a sub-ring of $S$, we have that $\mathsf{LS}_S$ particularizes to a $R_q$-linear sharing over $R_q$, denoted $\mathsf{LS}_{R_q}$, as desired. Notice that, since each share is in $S \cong R_q^d$, we have a size overhead of $d$. But for simplicity, in the remaining we do as if shares were in $R_q$.

### C.3 Implementation of $\mathcal{F}_{\mathsf{LSSS}}$.

$\Pi_{\mathsf{LSSS}}$

$\Pi_{\mathsf{LSSS}}$.**Setup** $\forall P \in \mathscr{P}$: $(\mathsf{sk}_P^{\mathsf{PKE}}, \mathsf{pk}_P^{\mathsf{PKE}}) \leftarrow \mathsf{KeyGen}$, publish $\mathsf{pk}_P^{\mathsf{PKE}}$ on $\mathsf{bPKI}^P$.

$\Pi_{\mathsf{LSSS}}$.**Input** Each sender $\mathcal{S} \in \mathscr{S}$, upon output of all instances of $\mathsf{bPKI}^P$ $\forall P \in \mathscr{P}$, set $\mathsf{pk}_P^{\mathsf{PKE}}$ as the output of $\mathsf{bPKI}^P$, possibly $\perp$. For each $\alpha \in X_{\mathcal{S}}$:

(i) Compute $\boldsymbol{pss}_{\mathcal{S},\boldsymbol{\alpha}} := \mathsf{PSS}\big((\mathsf{pk}_P^{\mathsf{PKE}})_{P\in\mathscr{P}},\, x_{\mathcal{S},\alpha}\big)$.

(ii) Broadcast $(\mathsf{input}, \mathsf{ssid} := \overline{x_{\mathcal{S},\alpha}},\, \boldsymbol{pss}_{\mathcal{S},\boldsymbol{\alpha}})$ over $\mathsf{BC}^{\mathcal{S}}$.

$\Pi_{\mathsf{LSSS}}$.**LCOpen($\Lambda$)** (i) $\forall P_j \in \mathscr{P}$, upon receiving output 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 $* = \perp$ then set $x^{(j)} := 0$; else if $* = [c_{\mathcal{S},\alpha}^{(1)}, \ldots, c_{\mathcal{S},\alpha}^{(n)}]$ then set $x_{\mathcal{S},\alpha}^{(j)} := \mathsf{PKE.Dec}(\mathsf{sk}_j, c_{\mathcal{S},\alpha}^{(j)})$. Evaluate $\mu^{(j)} := \Lambda\big((x_{\mathcal{S},\alpha}^{(j)})_{\mathcal{S},\alpha}\big)$ and send it over $\mathcal{F}_{\mathsf{AUTH}}^{P_j,R}$ to every player $R \in \mathscr{P}$.

(ii) Upon receiving opening shares $(\mu^{(i)})_{i\in\mathcal{U}}$ from any $(t+1)$-set $\mathcal{U} \subset [N]$ of players, outputs $\mu := \mathsf{LS}_{R_q}.\mathsf{SReco}^{\mathcal{U}}\big([\mu^{(i)},\ i \in \mathcal{U}]\big)$.

Fig. 10: Protocol for secret-sharing then delayed linear combination

***Proposition 11.*** *Protocol $\Pi_{\mathsf{LSSS}}$ (Fig. 10 UC implements $\mathcal{F}_{\mathsf{LSSS}}$*

*Proof.* For simplicity we construct a simulator for the opening of only one evaluation of one linear map. The case of multiple openings is handled as in [CDN15, p127], when they simulate each new $\mathsf{Open}$.

***- Game 1.*** We modify the Real execution in that the opening shares of $\mu$ sent by honest players, are replaced as follows. Firstly, we define quantities denoted *Infered 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 $\big(x_{\mathcal{S},\alpha}^{(i)}\big)_{i\in\mathcal{I}}$ as the actual shares produced by $\mathcal{S}$ when it computes the $\mathsf{PSS}$ 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 $\big(x_{\mathcal{S},\alpha}^{(i)} := 0\big)_{i\in\mathcal{I}}$, otherwise (ii) this implies that $*$ is a correctly formed $\mathsf{PSS}$. Thus in this case, we define as $\big(x_{\mathcal{S},\alpha}^{(i)}\big)_{i\in\mathcal{I}}$ the plaintext shares read on the witness tape of $\mathcal{S}$. Finally, for all $i \in \mathcal{I}$ we set $\mu^{(i)} := \Lambda\big((x_{\mathcal{S},\alpha}^{(i)})_{\alpha\in X_{\mathcal{S}},\mathcal{S}\in\mathscr{S}}\big)$. By linearity of $\mathsf{LS}_{R_q}$, they are equal to the opening shares of $\mu$ that the $(P_i)_{i\in\mathcal{I}}$ would have sent if they were honest.

We now replace the honest opening shares by $\mathsf{ShSim}^{\mathcal{I}}\big(\mu, (\mu^{(i)})_{i\in\mathcal{I}}\big)$. Since $\mathsf{ShSim}^{\mathcal{I}}$ simulates perfectly, they are identical to the ones of the Real execution.

***- Game 2.*** All the inputs $x_{\ell,\alpha}$ of honest $\mathcal{S} \in \mathscr{S}$ are replaced by 0. Since the secret decryption keys $\mathsf{sk}_h$ of all honest players $h \in \mathcal{H}$ are not used anymore, we have the IND-CPA property of $\mathsf{PSS}$ which applies, thus the view of $\mathscr{E}$ is indistinguishable from the one in the previous game.

***- Game 3.*** We now modify the method to *Infer* the corrupt shares of the $\boldsymbol{pss}_{\boldsymbol{\ell},\boldsymbol{\alpha}}$ broadcast by corrupt senders $\ell$. First, decrypt the honest shares of $\boldsymbol{pss}_{\boldsymbol{\ell},\boldsymbol{\alpha}}$ using, again, the honest secret keys $(\mathsf{sk}_h)_{h\in\mathcal{H}}$. From them, infer the corrupt shares using $\mathsf{ShInfer}^{\mathcal{H}}$. The infered shares are identical to the ones in the previous game, by the property of $\mathsf{ShInfer}^{\mathcal{H}}$.

***- Game 4.*** Honest players are now simulated, including generation of their their $(\mathsf{pk}_h, \mathsf{sk}_h)_{h\in\mathcal{H}}$, as well as all other honest senders, i.e., in $\mathscr{L}$. Their behavior is the same as in the previous game, thus both views have the same distribution.

But, in Game 4, the view is produced only from what is received from $\mathcal{F}_{\mathsf{LSSS}}$ and from the adversary, without reading on the tapes of corrupt entities, i.e. without rewinding. Thus we are describing a simulator in the Ideal execution. □

### C.4 Detailed construction of the linear secret sharing over $R_q$

Now for the details: consider an irreducible polynomial $\overline{Q(T)} \in \mathbb{F}_p[T]$ of degree $d := \lceil \log(N+1) \rceil$, then an arbitrary lift $Q$ in $\mathbb{Z}/q\mathbb{Z}$. Embed $R_q$ in the $R_q$-algebra $S := R_q[T]/Q$, which we may also denote as $\mathrm{Gal}(R_q, d)$. Now, in $S = \mathrm{Gal}(R_q, d)$ we have the sub-ring $B := \mathbb{Z}/q\mathbb{Z}[T]/Q$, denoted $\mathrm{Gal}(\mathbb{Z}/q\mathbb{Z}, d)$ the "Galois ring extension of degree $d$ of $\mathbb{Z}/q\mathbb{Z}$" [ACD+19]. [If it is not clear yet that $B$ is a sub-ring: apply [AM69, ex 6 p32] to $A := \mathbb{Z}/q\mathbb{Z}$, $M := R_q$, $B = \mathrm{Gal}(\mathbb{Z}/q\mathbb{Z}, d)$]. But, recall from [ACD+19] that $\mathrm{Gal}(T/qT, d)$ has the desired property to contain at least $N+1 = 2^{\log(N+1)}$ elements, denoted $(\alpha_0 := 0, \alpha_1, ..., \alpha_N)$ such that all pair-wise differences $\alpha_i - \alpha_j$ for $i \neq j$ are invertible [Concretely: choose the $(\alpha_i)_{i=0,...,N}$ as arbitrary lifts modulo $q$ of distinct elements of the finite field $\mathbb{F}_{p^d}$]. Thus we can apply to $S$ and these evaluation points $(\alpha_i)_{i=0,...,N}$ the same previous construction as for $\mathsf{LS}_{R_q}$. We obtain a $S$-linear secret sharing scheme over $S$: $LS_S$, in particular by the ring inclusion $R_q \hookrightarrow S$, a $R_q$-linear secret sharing over $R_q$. Each share, which is in $S$, can be encoded as $d$ elements of $R_q$. [Concretely, $LS_S(s)$ is defined as: sample $P$ at random in $S[Y]_t^{(s)}$, then output $[P(\alpha_i)]_{i\in[N]}$. Then for reconstruction use the Lagrange polynomials $\Pi_{j\neq i}(X-\alpha_j)/(\alpha_i-\alpha_j)$.]

### C.5 Detailed Proofs

Recall that, although in general each share is in $R_q^d$, where $d := \lceil \log_p(N+1) \rceil$ when $q$ is some power of $p$, for simplicity, in the remaining we do as if shares were in $R_q$.

#### C.5.1 Uniformity of any $t$ shares of any given secret.

***Property 12.*** *For every $s \in R_q$, for any subset of $t$ indices $\mathcal{I} \subset [N]$, the distribution of shares $(s^{(i)})_{i\in\mathcal{I}}$ output by $\mathsf{LS}_{R_q}.\mathsf{Share}(s)$ is $U(R_q^t)$.*

*Proof.* By surjectivity (isomorphism) of $\mathsf{Eval}_{\{0\}\cup\mathcal{I}} : R_q[Y]_t \to R_q^{t+1}$ for any $t$-sized $\mathcal{I}$, we have surjectivity (isomorphism) of $\mathsf{Eval}_{\mathcal{I}} : R_q[Y]_t^{(s)} \to R_q^t$ for any fixed $s \in R_q$. Furthermore, the map $\mathsf{Eval}_{\mathcal{I}}$ being also linear, we have in conclusion that it maps the uniform distribution onto the uniform distribution. □

### C.6 Proof of IND-CPA of Public Secret Sharing

We are going to show a bit more than IND-CPA. We consider a game in which the adversary $\mathcal{A}_{\mathsf{PSS}}$ can select $t$ indices $\mathcal{I} \subset [N]$ to corrupt, without any further information given to it at this stage. Then the oracle $\mathscr{O}_{\mathsf{PSS}}$ generates $[N]$ public keys for $\mathsf{PKE}$ and shows them to $\mathcal{A}_{\mathsf{PSS}}$. Furthermore, it reveals to $\mathcal{A}_{\mathsf{PSS}}$ the $t$ secret keys with indices in $\mathcal{I}$. Then $\mathscr{O}_{\mathsf{PSS}}$ tosses a bit $b$ and subsequently responds to encryption requests from $\mathcal{A}_{\mathsf{PSS}}$ as follows. Either $(b = 1)$ then $\mathscr{O}_{\mathsf{PSS}}$ returns to $\mathcal{A}_{\mathsf{PSS}}$, upon each plaintext $s$ chosen by $\mathcal{A}_{\mathsf{PSS}}$, a correctly generated $\mathsf{PSS}$ of $s$ of its choice, or, if $(b = 0)$, a sample of the following distribution, which we denote $\mathcal{V}$ such that:

- the entries in $\mathcal{I}$ are $\mathsf{PKE}$ encryptions of uniform independent values in $R_q$;
- the remaining entries are $\mathsf{PKE}$ encryptions of 0.

Since this distribution $\mathcal{V}$ is independent from $s$, the indistinguishability that we claim indeed implies IND-CPA.

We are going to bound the advantage by any adversary $\mathcal{A}_{\mathsf{PSS}}$, by the maximum advantage of an adversary $\mathcal{A}_E$ against oracle $\mathscr{O}_E$ of the following $(N-t)$-keys variant indistinguishability game for $\mathsf{PKE}$. The latter is upper-bounded by $(N-t)$ times the advantage for one-message indistinguishability, see e.g. [BS20, Thm 5.1] or our proof of Corollary 14, which is identical. $\mathscr{O}_E$ samples $(N-t)$ $\mathsf{PKE}$ public keys $(\mathsf{pk}_h^{\mathsf{PKE}})_{h\in\mathcal{H}}$ which it gives to $\mathcal{A}_E$. $\mathscr{O}_E$ tosses a bit $b \in \{0,1\}$ and subsequently has the following behavior: When $\mathcal{A}_E$ submits $(N-t)$ chosen plaintexts $(s^h)_{h\in\mathcal{H}}$ to $\mathscr{O}_E$ either $(b=0)$ then $\mathscr{O}_E$ returns $(N-t)$ encryptions of 0: $(\mathsf{Enc}(\mathsf{pk}_h^{\mathsf{PKE}}, 0))_{h\in\mathcal{H}}$ , or: $(b=1)$ then $\mathscr{O}_E$ returns actual encryptions of the plaintexts $(\mathsf{Enc}(\mathsf{pk}_h^{\mathsf{PKE}}, s^h))_{h\in\mathcal{H}}$.

The reduction is as follows. Upon receiving a set of keys $(\mathsf{pk}_h^{\mathsf{PKE}})_{h\in\mathcal{H}}$ from $\mathscr{O}_E$, then $\mathcal{A}_E$ samples itself $t$ key pairs $(\mathsf{sk}_i^{\mathsf{PKE}}, \mathsf{pk}_i^{\mathsf{PKE}})_{i\in\mathcal{I}}$, initiates $\mathcal{A}_{\mathsf{PSS}}$, reorganizes the indices so that the indices chosen by $\mathcal{A}_{\mathsf{PSS}}$ correspond to $\mathcal{I}$, gives to $\mathcal{A}_{\mathsf{PSS}}$ the total $N = |\mathcal{H}| + |\mathcal{I}|$ public keys and furthermore gives to $\mathcal{A}_E$ the $t$ secret keys $(\mathsf{sk}_i^{\mathsf{PKE}})_{i\in\mathcal{I}}$.

Upon receiving one challenge plaintexts $s$ from $\mathcal{A}_{\mathsf{PSS}}$, $\mathcal{A}_E$ computes the first step of $\mathsf{PSS}$ on it, namely: $(s^{(1)}, ..., s^{(N)}) \leftarrow \mathsf{LS}_{R_q}.\mathsf{Share}(s)$. It then sends the challenge $(N-t)$ plaintext messages: $\left(s^{(h)}\right)_{h\in\mathcal{H}}$ to $\mathscr{O}_E$. Upon receiving the response ciphertexts $(c_h)_{h\in\mathcal{H}}$ from $\mathscr{O}_E$, it then computes the $N$-sized vector $V$ consisting of:

- The entries in $\mathcal{I}$ equal to correct encryptions $\left\{\mathsf{Enc}(\mathsf{pk}_i^{\mathsf{PKE}}, s^{(i)})_{i\in\mathcal{I}}\right\}$ that $\mathcal{A}_E$ generates itself.
- The remaining entries are set to the $\{c_h\}_{h\in\mathcal{H}}$ received from $\mathscr{O}_E$.
  And sends it to $\mathcal{A}_{\mathsf{PSS}}$ as response to its challenge. Upon answer a bit $b$ from $\mathcal{A}_{\mathsf{PSS}}$, then $\mathcal{A}_E$ outputs the same bit $b$ to $\mathscr{O}_E$.

- in the case where the ciphertexts $\{c_h\}_{h\in\mathcal{H}}$ are encryptions of the actual $N-t$ shares $\{s^{(h)}\}_{h\in\mathcal{H}}$, then $\mathcal{A}_{\mathsf{PSS}}$ receives from $\mathcal{A}_E$ a correctly generated $\mathsf{PSS}$ of $s$.
- in the case where the ciphertexts $\{c_h\}_{h\in\mathcal{H}}$ are encryptions of 0, then, by Property 12 of uniform independence of the $t$ plaintext shares $\left(s^{(i)}\right)_{i\in\mathcal{I}}$, we have that what $\mathcal{A}_{\mathsf{PSS}}$ receives from $\mathcal{A}_E$ is undistinguishable from a sample in the distribution $\mathcal{V}$.

Thus in both cases $b \in \{0,1\}$ , $\mathcal{A}_{\mathsf{PSS}}$ is faced with the same distribution as would have been generated by oracle $\mathscr{O}_{\mathsf{PSS}}$ for the same $b$, thus the distinguishing advantage of $\mathcal{A}_E$ is the same as the one of $\mathcal{A}_{\mathsf{PSS}}$.

*Handling semi-adaptive security* Notice that, although not necessary for our MPC model, where corruptions are done ahead of publications of keys of honest players, one could have imagined a game in which $\mathcal{A}_{\mathsf{PSS}}$ first sees $n$ public keys, then subsequently chooses for which $t$ ones it wants to be revealed the secret keys. However we do not consider corruptions after decryption shares are issued, thus the terminology "semi-adaptive".

We can compile the reduction above to this semi-adaptive setting, although with an exponential loss in $n$, as follows. $\mathcal{A}_E$, upon having received the $(\mathsf{pk}_h^{\mathsf{PKE}})_{h\in\mathcal{H}}$ from $\mathscr{O}_E$, and sampled itself the $t$ key pairs $(\mathsf{sk}_i^{\mathsf{PKE}}, \mathsf{pk}_i^{\mathsf{PKE}})_{i\in\mathcal{I}}$, shuffles the indices at random. Then it gives the $N$ public keys to $\mathcal{A}_{\mathsf{PSS}}$, which queries $t$ indices of which it wants to be revealed the secret keys. In the case where $\mathcal{A}_E$ would not known at least one secret key of these $t$ out of $N$ indices then $\mathcal{A}_E$ simply outputs a bit $b$ at random to $\mathscr{O}_E$.

Otherwise, it means that $\mathcal{A}_{\mathsf{PSS}}$ queried exactly the $t$ indices which were previously the ones, denoted $\mathcal{I}$, which $\mathcal{A}$ generated itself. Thus in this case $\mathcal{A}_E$ opens the secret keys to $\mathcal{A}_{\mathsf{PSS}}$ and we are exactly in the same situation as in the non-adaptive game. Notice however that this situation happens with probability in $1/\binom{N}{t}$,

# D Complements on BFV + CDKS*

*Roadmap:* This section builds on the description of BFV as a standalone public key encryption, done in Section 3.1.2. First, in Appendices D.1 to D.3, we recall some generalities and cryptographic tools, that we will use throughout this section. In Appendix D.4, we: detail our alternative relinearization key generation

algorithm, prove its security under a circular security assumption formalizing [CDKS19], and prove its security when used to distributively generate a common relinearization key. As a result, it provides the solution to the challenge of robust distributed generation of a relinarization key, as summarized in Section 6.1.1. Then in Appendix D.5 we introduce the *decryption noise* and prove, as a warmup, correctness of the decryption of a fresh encryption. In Appendix D.6, we detail the homomorphic properties that can be added to the standalone scheme. Finally in Appendix D.7, we perform a complete noise analysis after homomorphic evaluation of a circuit. Of course, security of MPC will be proven as a whole (in Section 7.1 and appendix H.3), re-using all the results of this section.

*Opening Remark:* As in Section 3.1.2, we specify our algorithms as taking a uniform random string as input. It comes in the form of two vectors $(\mathbf{a}, \mathbf{d}_1) \in (R_q^l)^2$, of which $a = \mathbf{a}[0]$ is, as described in Section 3.1.2, used to generate public encryption keys. The other components of the vectors will be used to generate the relinearization keys. As a result, all functions that took some parameter $a \in R_q$ as input variable, are naturally extended to handle input $\mathbf{a} \in R_q^l$. In particular, $\mathsf{BFV.KeyGen}$ now returns $(\mathbf{b}, \mathbf{a}) \in R_q^{l\times 2}$. Although in [FV12] the uniform random string $a$ is sampled locally uniformly in $R_q$ by $\mathsf{KeyGen}$, our $\mathsf{DKG}$ requires that $\mathbf{a}$ is common and sampled in $R_q^l$ by $\overline{\mathcal{G}}_{\mathrm{URS}}$, to enable some form of additivity. The same goes for $\mathbf{d}_1$: while in the relinearization key of [CDKS19] it is sampled locally by each key owner, we require instead that it is equal to a fixed public uniform random string (URS). We refer to Corollary 14 then Appendix H.1 for why IND-CPA is preserved even if these parameters are drawn from the URS, as we specify. This analysis will be an ingredient in the proof of the MPC in Section 7.1.

### D.1 Generalities

*Notation.* For any element $\widetilde{r} = \Sigma_{i=0}^{n-1} \widetilde{r_i} X^i \in R$, we define its infinity norm as $\|\widetilde{r}\| := \max_i |\widetilde{r_i}|$. For $r \in R_q$, let us consider the unique representative $\widetilde{r} = \Sigma_{i=0}^{n-1} \widetilde{r_i} X^i \in R$ such that $\widetilde{r_i} \in [-(q-1)/2, ..., (q-1)/2]$ for all $i$. Then we define $\|r\| := \|\widetilde{r}\|$. For two vectors $\mathbf{u}, \mathbf{v}$ we denote $\langle \mathbf{u}, \mathbf{v} \rangle$ the dot product and, for a third vector $\mathbf{w}$, we denote $\mathbf{u}^{\langle \cdot \rangle}(\mathbf{v}, \mathbf{w}) := \big(\langle \mathbf{u}, \mathbf{v} \rangle, \langle \mathbf{u}, \mathbf{w} \rangle\big)$. Recall that we denote $\Delta = \lfloor q/k \rfloor$,the integer division of $q$ by $k$. We denote vectors of some length $l$ (see §D.2) in bold, e.g. $\mathbf{a}$. For such vector $\mathbf{r} = (r_1, \ldots, r_l) \in R_q^l$, we define $\|\mathbf{r}\| := \max_i |\widetilde{r_i}|$. Finally, $\otimes : R_q^2 \times R_q^2 \to R_q^3$ denotes the tensor product.

For two polynomials $p$ and $d$ in $R_q$ whose polynomial modulus s a degree-n power of 2 cyclotomic, we have

$$\|pd\| \leq n\|p\|\|d\|. \tag{3}$$

The proof of this inequality is straightforward and it can be found in [BCN18, Lemma 2].

*Distributions.* Motivated by computational optimizations discussed in [FV12; CH18; CDKS19], we specify $n$ as a power of two, $f := X^n + 1$ the $2n$-th cyclotomic polynomial, and thus $q$ odd. According to [ACC+21, p. 2.1.5], this enables to sample $\Psi_q$ as: univariate polynomials with coefficients following a discrete Gaussian distribution centered at zero, then reduced modulo $q$. From the variance $\sigma^2$ of the distribution, one can derive an integer $B$ such that the norms of coefficients are lower than $B$ with overwhelming probability. Similarly, we consider an encryption distribution $\mathscr{B}_{Enc,q}$ bounded by some value $B_{Enc}$, i.e. for an element sampled in $\mathscr{B}_{Enc,q}$ the norms of coefficients are lower than $B_{Enc}$ with overwhelming probability. We specify $\mathscr{X}_q$ as in [FV12, p5], [CDKS19, §6] and [MTBH21], as: polynomials in $\mathbb{Z}[X]/(X^n + 1)$ with coefficients varying uniformly in $\{-1, 0, 1\}$, then reduced modulo $q$.

Semi-malicious adversaries are not required to sample correctly, they are only required to have values of samples on their witness tapes, which are within the essential bounds on norms of the respective distributions, i.e., 1, $B$, $B_{Enc}$ and $B_{sm}$. Thus, the definitions and bounds which we now provide are worst cases over the following choices:

- $\mathbf{a}$ and $\mathbf{d}_1$, both in $R_q^l$;

- for all $i \in [N]$: $\mathsf{sk}_i$ and $r_i$, in $R_q$ and both of norm $\leq 1$; $\mathbf{e}_i^{(\mathbf{pk})}$, $\mathbf{e}_0^{(\mathbf{rlk})}$ and $\mathbf{e}_2^{(\mathbf{rlk})}$; in $R_q^l$ and all of norms $\leq B$;
- deduce from these, for all $i \in [N]$ the additive contributions to encryption and relinearization keys, namely: $(\mathsf{sk}_i, \mathsf{ek}_i)$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i})$;
- Sum them into the threshold key $(\mathsf{sk}, \mathsf{ek})$ and the relinearization key $\mathbf{rlk}$.

### D.2 Gadget Decomposition.

Let us define the widely used, e.g., [GSW13; CDKS19; GMP19], *gadget toolkit*:

1. Gadget vector: $\mathbf{g} = (g_0, g_1, ..., g_{l-1}) \in R_q^l$ ; and integers $l$ and (small) $B_g$;
2. The gadget decomposition denoted $\mathbf{g}^{-1}(.)$: on input any $x \in R_q$, decomposes it into a vector $\mathbf{u} = (u_0, ..., u_{l-1}) \in R^l$ of (small) coordinates, i.e, $\|u_i\| \leq B_g$ for all $0 \leq i \leq l-1$, such that $\Sigma_{i=0}^{l-1} u_i.g_i = x \pmod q$.

### D.3 Ring Learning with Errors.

Let $\Psi_q$ and $\mathscr{X}_q$ be distributions over $R_q$. The *decisional*-Ring Learning with Errors (RLWE) [LPR13a] assumption with parameter $(R_q, \mathscr{X}_q, \Psi_q)$ can be stated as follows: for a fixed secret sample $s \xleftarrow{\$} \mathscr{X}_q$, then any polynomially long sequence of samples in $R_q^2$ of the form $(a_i, b_i = s\,a_i + e_i)_i$, where $a_i \leftarrow U(R_q)$, and $e_i \leftarrow \Psi_q$, is computationally indistinguishable from a uniform random sequence of elements of $R_q^2$.

### D.4 Relinearization Key Generation

$\mathsf{BFV} + \mathsf{CDKS}^*$ differs from $\mathsf{BFV}$ in its relinearization key generation algorithm that we now describe.

**D.4.1 Alternative Relinearization Key Generation, adapted from [CDKS19]** Recall from Section 6.1.1 that we now consider two uniform random strings $(\mathbf{a}, \mathbf{d}_1) \in (R_q^l)^2$. To generate a relinearization key $\mathbf{rlk}$, we use the following $\mathsf{CDKS}$ function:

$(\mathbf{d}_0, \mathbf{d}_2) \in R_q^{l\times 2} \leftarrow \mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk})$: Given $\mathsf{sk} \in R_q$ and $(\mathbf{a}, \mathbf{d}_1) \in R_q^{l\times 2}$:

1. Sample $r \xleftarrow{\$} \mathscr{X}_q$.
2. Sample $\mathbf{e}_0^{(\mathbf{rlk})} \xleftarrow{\$} \Psi_q^l$, and set $\mathbf{d}_0 = -\mathsf{sk}\,\mathbf{d}_1 + \mathbf{e}_0^{(\mathbf{rlk})} + r\,\mathbf{g}$
3. Sample $\mathbf{e}_2^{(\mathbf{rlk})} \xleftarrow{\$} \Psi_q^l$ and set $\mathbf{d}_2 = r\,\mathbf{a} + \mathbf{e}_2^{(\mathbf{rlk})} + \mathsf{sk}\,\mathbf{g}$

and set $\mathbf{rlk} = (\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2)$.

**D.4.2 Circular Security Hardness Assumption of [CDKS19].** The multikey FHE (MFHE) scheme of [CDKS19] has its security based on the hardness of RLWE with parameter $(n, q, \mathscr{X}_q, \Psi_q)$ since it uses the same encryption algorithm as $\mathsf{BFV}$. In addition, they make a circular security assumption under which their MFHE remains secure even if $(\mathbf{b}, \mathbf{rlk})$ is given to the adversary. Precisely, this assumption implies that $(\mathbf{b}, \mathbf{rlk})$ is computationally indistinguishable from the uniform distribution over $R_q^4$. We now show that our modified relinearization key generation, i.e., with a common public randomness $\mathbf{d}_1$, remains secure under their assumption.

We detail in §D.4.4 how this circular security shows up in [CDKS19] with their notations. For our usage, we now state this assumption under a more concrete equivalent form, called Assumption 13. Consider an oracle $\mathscr{O}_{\mathcal{D}_0}$ which samples $\mathbf{a} \xleftarrow{\$} U(R_q^l)$ then $\mathsf{KeyGen}$erates one $\mathsf{BFV}$ key pair $(\mathsf{sk}, \mathsf{ek})$, then samples $\mathbf{d}_1 \xleftarrow{\$} U(R_q^l)$, then, using $\mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk})$, computes from it one public relinearization key $\mathbf{rlk} = (\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2)$ then outputs the pair $(\mathsf{ek}, \mathbf{rlk})$. Then, any adversary has negligible advantage in distinguishing this *single* output from a single sampling in $U(R_q^{l\times 5})$.

***Assumption 13.*** *Define the distribution:*

$$\mathcal{D}_0 := \Big\{(\mathbf{b},\mathbf{a},\mathbf{d}_0,\mathbf{d}_1,\mathbf{d}_2) : (\mathbf{a},\mathbf{d}_1) \leftarrow U(R_q^l)^2,\ \mathsf{sk} \leftarrow \mathscr{X}_q, (\mathbf{e}^{(\mathbf{pk})},\mathbf{e}_0^{(\mathbf{rlk})},\mathbf{e}_2^{(\mathbf{rlk})}) \leftarrow (\Psi_q^l)^3,$$
$$r \leftarrow \mathscr{X}_q,\ \mathbf{b} := -\mathbf{a}\,\mathsf{sk} + \mathbf{e}^{(\mathbf{pk})},\ \mathbf{d}_0 := -\mathsf{sk}\,\mathbf{d}_1 + \mathbf{e}_0^{(\mathbf{rlk})} + r\,\mathbf{g},\ \mathbf{d}_2 := r\,\mathbf{a} + \mathbf{e}_2^{(\mathbf{rlk})} + \mathsf{sk}\,\mathbf{g}\Big\}$$

*Then the maximum distinguishing advantage* $\mathrm{Adv}^{\lambda}_{\mathcal{D}_0}$ *between a* single sample *in* $\mathcal{D}_0$ *and in* $U(R_q^{l\times 5})$*, is* $negl(\lambda)$.

Very briefly, they first define a RLWE-based symmetric encryption scheme denoted UniEnc, for which they state (p7) and prove (§B.1) indistinguishability from uniform randomness of any pair {BFV public key; encryption of some chosen plaintext encrypted with UniEnc using the BFV secret key}, then they make the circular security assumption that indistinguishability still holds if one replaces the chosen plaintext by the BFV secret key itself.

**D.4.3 Distributed Relinearization Key Generation** To distributively generate a common relinearization key **rlk**, we can leverage the additional linearity given by the common $\mathbf{d}_1$ (recall that in [CDKS19], $\mathbf{d}_1$ is sampled locally). In short, we let each player $P_i$ compute additive contributions to the relinearization key $(\mathbf{d}_{0,i},\mathbf{d}_{2,i}) \leftarrow \mathsf{CDKS}(\mathbf{a},\mathbf{d}_1,\mathsf{sk}_i)$ and broadcast them. From a set $S$ of players who have correctly broadcast their additive contributions to the key $(\mathbf{d}_{0,i},\mathbf{d}_{2,i})_{i\in S}$, we can then compute a common

$$\mathbf{rlk} := (\Sigma_{i\in S}\mathbf{d}_{0,i}, \mathbf{d}_1, \Sigma_{i\in S}\mathbf{d}_{2,i})$$

For our use-case of distributed generation of a common **rlk**, let us show that Assumption 13 extends to the setting where several players, which are performing the distributed relinearization key generation, use the same URS to do this generation, just as we specified. We formalize it as the following Corollary 14. This statement, and the kind of reduction used to obtain it, is standard in lattice-based cryptography since at least [BPR11]. We first state a more concrete equivalent statement, for later use in the proof of MPC. Consider a public sampling of a uniform string $(\mathbf{a},\mathbf{d}_1) \in U(R_q^{l\times 2})$, e.g., by $\overline{\mathcal{G}}_{\mathrm{URS}}$. Consider a polynomial number $M$ of independent machines, each of them generates a key pair $(\mathsf{sk}_m,\mathsf{ek}_m)$ by using $\mathsf{BFV.KeyGen}$, all using the common public $\mathbf{a}$. Likewise, each machine $m$ generates $[\mathbf{d}_{0,m},\mathbf{d}_{2,m}] \leftarrow \mathsf{CDKS}(\mathbf{a},\mathbf{d}_1,\mathsf{sk}_m)$. Then the collection of the public data issued by these machines $\{\mathbf{b}_m,\mathbf{d}_{0,m},\mathbf{d}_{2,m}\}_{m\in[M]}$, jointly with the public $(\mathbf{a},\mathbf{d}_1)$, is still indistinguishable from one sample in $U(R_q^{(l\times 3)M} \times R_q^{l\times 2})$.

***Corollary 14 (Security with Common Public Randomness).*** *Consider:*

$$\mathcal{D}_0^M := \Big\{\big\{\mathbf{b}_m,\mathbf{d}_{0,m},\mathbf{d}_{2,m}\big\}_{m\in M},\mathbf{a},\mathbf{d}_1 : (\mathbf{a},\mathbf{d}_1) \leftarrow U(R_q^l)^2,\ \ \textit{and}$$
$$\forall m\in[M]:\ \mathsf{sk}_m \leftarrow \mathscr{X}_q,\ (\mathbf{e}_m^{(\mathbf{pk})},\mathbf{e}_{0,m}^{(\mathbf{rlk})},\mathbf{e}_{2,m}^{(\mathbf{rlk})}) \leftarrow (\Psi_q^l)^3, r_m \leftarrow \mathscr{X}_q,\ \mathbf{b}_m := -\mathbf{a}\,\mathsf{sk}_m + \mathbf{e}_m^{(\mathbf{pk})},$$
$$\mathbf{d}_{0,m} := -\mathsf{sk}_m\,\mathbf{d}_1 + \mathbf{e}_{0,m}^{(\mathbf{rlk})} + r_m\,\mathbf{g},\ \mathbf{d}_{2,m} := r_m\,\mathbf{a} + \mathbf{e}_{2,m}^{(\mathbf{rlk})} + \mathsf{sk}_m\,\mathbf{g}\Big\}$$

Then the maximum distinguishing advantage $\mathrm{Adv}^{\lambda}_{\mathcal{D}_0^M}$ between a single sample in $\mathcal{D}_0^M$ and in $U(R_q^{(l\times 3)M} \times R_q^{l\times 2})$, is bounded by $M\mathrm{Adv}^{\lambda}_{\mathcal{D}_0}$.

*Proof.* Consider a cascade of oracles $\mathscr{O}_0 := \mathscr{O}_{\mathcal{D}_0^M}, \mathscr{O}_1, \ldots, \mathscr{O}_M$ such that each $\mathscr{O}_i$ returns the first $i$ components of $R_q^{(l\times 3)M}$ in $U(R_q^{(l\times 3)i})$ and the remaining ones as in $\mathcal{D}_0^M$. Then the distinguishing advantage between two consecutive $\mathscr{O}_i$ is at most $\mathrm{Adv}_{\mathcal{D}_0}$, as a straightforward reduction shows. □

**D.4.4 How Assumption 13 appears in [CDKS19]** Assumption 13 appears in [CDKS19] with the following notations. They define a RLWE-based symmetric one-time encryption scheme with plaintexts in $R_q$ and ciphertexts in $R_q^{3\times l}$, denoted $UniEnc_{\mathbf{a}}$, parametrized by $\mathbf{a} \in R_q^l$. In their use case, $\mathbf{a} \in R_q^l$ is the URS which is also used to generate $(\mathsf{sk},(\mathbf{b},\mathbf{a})) \leftarrow \mathsf{BFV.KeyGen}(\mathbf{a})$, exactly as in our MPC setting. Then, they state in their (Security) formula p7, and prove in §B.1 that for any (chosen plaintext) $\mu$, we have that:

for a sampling $\mathbf{a} \leftarrow U(R_q^l)$, followed by a sampling $(\mathsf{sk}, (\mathbf{b}, \mathbf{a})) \leftarrow \mathsf{BFV.KeyGen}(\mathbf{a})$, followed by *one single* randomized encryption $UniEnc_{\mathbf{a}}(\mathsf{sk}, \mu)$, then the *single output* $(\mathbf{b}, UniEnc_{\mathbf{a}}(\mathsf{sk}, \mu))$ is indistinguishable from a single sample in $U(R_q^{l\times 5})$. Next, they assume that (Security) also holds when the chosen $\mu$ is replaced by the secret key $\mathsf{sk}$ itself, which is exactly what we spelled-out in Assumption 13. Concretely, in their $UniEnc_{\mathbf{a}}$, the $r$ in our $\mathcal{D}_0$ shows up as the secret encryption randomness, while the $\mathbf{d}_1$ is specified in UniEnc to be sampled uniformly when encrypting.

### D.5 Warmup: Correctness & Decryption Noise of a Fresh Encryption

We first introduce some definitions:

***Def-Prop 15 (Decryption noise).*** *Let $\mathsf{c} \in \mathscr{C}$, $m \in R_k$ and $\mathsf{sk} \in \mathscr{X}_q$. We define the "decryption noise" as $e^{(Dec)}(\mathsf{c}, \mathsf{sk}, m) := \Lambda_{Dec}^{\mathsf{c}}(\mathsf{sk}) - \Delta.m$.*

***Proposition 16 (Correctness).*** *Let $\mathsf{c} = (\mathsf{c}[0], \mathsf{c}[1]) \in R_q^2$, $m \in R_k$ and $\mathsf{sk} \in \mathscr{X}_q$. It satisfies the trivial property that if $|e^{(Dec)}(\mathsf{c}, \mathsf{sk}, m)| < \frac{\Delta}{2}$, then, $\mathsf{BFV.Dec}(\mathsf{sk}, \mathsf{c}) = m$.*

We now formalize the set in which belong the outputs of $\mathsf{BFV.Enc}$. For any $m \in R_k$, we denote as a "Fresh BFV Encryption of $m$", any element of $R_q^2$ of the form: $\mathsf{c} = (\Delta m + u.\, b + e_0^{(Enc)}, u\, a + e_1^{(Enc)})$, where $\|u\| \le 1$, $\left\|e_0^{(Enc)}\right\| \le B_{Enc}$ and $\left\|e_1^{(Enc)}\right\| \le B$. Let us denote $e^{(fresh)} := e^{(Dec)}(\mathsf{c}, \mathsf{sk}, m) := \mathsf{c}[0] + \mathsf{c}[1]\,\mathsf{sk} - \Delta m$ its decryption noise (Def-Prop 16).

Recall that by definition we have that $\mathsf{c}[0] + \mathsf{c}[1]\,\mathsf{sk} = \Delta m + e^{(fresh)}$. With $e^{(pk)} = \mathbf{e}^{(\mathbf{pk})}[0]$, we have

$$\begin{aligned}
\mathsf{c}[0] + \mathsf{c}[1]\,\mathsf{sk} &= \Delta m + ub + e_0^{(Enc)} + \mathsf{sk}\, a\, u + \mathsf{sk}\; e_1^{(Enc)} \\
&= \Delta m + (\cancel{-\mathsf{sk}\, a} + e^{(pk)})\, u + e_0^{(Enc)} + \cancel{\mathsf{sk}\, a\, u} + \mathsf{sk}\, e_1^{(Enc)} \\
&= \Delta m \underbrace{+ u\, e^{(pk)} + e_0^{(Enc)} + \mathsf{sk}\, e_1^{(Enc)}}_{e^{(fresh)}}
\end{aligned}$$

$$\left\|e^{(fresh)}\right\| \le B_{Enc} + n\|e^{(pk)}\| + nB\,\|\mathsf{sk}\| := B_{fresh} \tag{4}$$

where $\Delta = \left\lfloor \frac{q}{k} \right\rfloor$.

### D.6 Homomorphic Properties

We add homomorphic capabilities to $\mathsf{BFV} + \mathsf{CDKS}^*$ and perform a complete noise analysis in §D.6.1 and §D.6.2

- (**Addition**) $\mathsf{BFV.Add}(\mathsf{c}_1, \mathsf{c}_2)$: Return $\mathsf{c} = \mathsf{c}_1 + \mathsf{c}_2 \in R_q^2$.
- (**Multiplication**) $\mathsf{BFV.Mult}(\mathsf{c}_1, \mathsf{c}_2, \mathbf{rlk}, \mathbf{b})$: Compute $\hat{\mathsf{c}} = \left\lfloor \frac{k}{q} \mathsf{c}_1 \otimes \mathsf{c}_2 \right\rceil \in R_q^3$ and return $\mathsf{c}' \leftarrow Relin(\hat{\mathsf{c}}, \mathbf{rlk}, \mathbf{b})$ (cf Appendix D.6.2).
- $\mathsf{BFV.Eval}(C, (\mathsf{c}_\ell \in R_q^2)_{\ell\in\mathscr{L}}, \mathbf{rlk}, \mathbf{b})$, for a circuit $C$ with $|\mathscr{L}|$ input gates: return the evaluation obtained by applying $\mathsf{BFV.Add}$ and $\mathsf{BFV.Mult}$ gate by gate, with inputs the $(\mathsf{c}_\ell)_{\ell\in\mathscr{L}}$.

In §D.6.2, we upper-bound the additional noise introduced by $\mathsf{BFV.Mult}$ as the sum of the bounds given by eqns (7) and (9). These bounds are obtained by particularizing the analysis of [CDKS19, §C.1] in the single key setting, and turning their variances into essential upper-bounds.

From them we deduce:

***Proposition 17 (Decryption noise of a product).*** *Consider two $\mathsf{BFV}$ ciphertexts $\mathsf{c}_1$ and $\mathsf{c}_2$ of $m_1$ and $m_2$ respectively under a key $\left(\mathbf{b} = -\mathbf{a}\,\mathsf{sk} + \mathbf{b}\,, \mathbf{a}\right) \in R_q^{2\times l}$, with decryption noises (Def-prop 16) denoted $e_i^{(Dec)} := \mathsf{c}_i[0] + \mathsf{c}_i[1]\,\mathsf{sk} - \Delta m_i$, $i \in \{1, 2\}$. Consider any $(\mathbf{d}_0, \mathbf{d}_2) \leftarrow \mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk})$, $\mathbf{rlk} = (\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2)$, denote $\mathsf{c}' := \mathsf{Mult}(\mathsf{c}_1, \mathsf{c}_2, \mathbf{rlk}, \boldsymbol{b})$, then $e^{(Dec)}(\mathsf{c}', \mathsf{sk}, m_1 m_2)$ is dominated by $k.n^2.\|\mathsf{sk}\|(\|e_1^{(Dec)}\| + \|e_2^{(Dec)}\|) + n^2.l.B_g.\|\mathsf{sk}\|(\|\mathbf{e}^{(\mathbf{pk})}\| + \|\mathbf{e}_2^{(\mathbf{rlk})}\|)$.*

**D.6.1 Noise Analysis of Addition** Let us consider two ciphertexts $\mathsf{c}_1$ and $\mathsf{c}_2$ such that $\mathsf{c}_1[0]+\mathsf{c}_1[1]\,\mathsf{sk} = \Delta m_1 + e_1^{(Dec)}$ and $\mathsf{c}_2[0]+\mathsf{c}_2[1]\,\mathsf{sk} = \Delta m_2 + e_2^{(Dec)}$. Let $\mathsf{c}_{add} = \mathsf{BFV.Add}(\mathsf{c}_1, \mathsf{c}_2)$ be the homomorphic sum of $\mathsf{c}_1$ and $\mathsf{c}_2$, and let us define the "decryption noise of an addition" as $e^{(add)} := e^{(Dec)}(\mathsf{c}_{add}, \mathsf{sk}, m_1+m_2)$. Thus we have $\mathsf{c}_{add}[0]+\mathsf{c}_{add}[1]\,\mathsf{sk} = \Delta[m_1+m_2]_k + e^{(add)}$, with $m_1+m_2 = [m_1+m_2]_k + k.r$ for $\|r\| \leq 1$ and

$$\|e^{(add)}\| = \|e_1^{(Dec)} + e_2^{(Dec)} + r_k(q)\,r\| \leq \|e_1^{(Dec)}\| + \|e_2^{(Dec)}\| + r_k(q) \tag{5}$$

where $r_k(q)$ denotes the remainder of the integer division of $q$ by $k$.

**D.6.2 Noise Analysis of Multiplication & Relinearization** Let us consider two ciphertexts $\mathsf{c}_1$ and $\mathsf{c}_2$ such that $\mathsf{c}_1[0]+\mathsf{c}_1[0].\mathsf{sk} = \Delta m_1 + e_1^{(Dec)}$ and $\mathsf{c}_2[0]+\mathsf{c}_2[1].\mathsf{sk} = \Delta m_2 + e_2^{(Dec)}$. Recall from §6.1.1 that the multiplication of two BFV ciphertexts involves two steps that introduce noise: a *tensoring* operation followed by a *relinearization*.

***Tensoring*** First, let $\hat{\mathsf{c}} = \left\lfloor \frac{k}{q}\mathsf{c}_1 \otimes \mathsf{c}_2 \right\rceil = (\hat{\mathsf{c}}[0], \hat{\mathsf{c}}[1], \hat{\mathsf{c}}[2])$. Let us define the "decryption noise of a three-terms ciphertext $\hat{\mathsf{c}}$ with respect to secret key $\mathsf{sk}$ and plaintext $m_1\,m_2$", and denote it $e^{(tens)}$, as:

$$\hat{\mathsf{c}}[0] + \hat{\mathsf{c}}[1]\,\mathsf{sk} + \hat{\mathsf{c}}[2]\,\mathsf{sk}^2 = \Delta[m_1\,m_2]_k + e^{(tens)} \tag{6}$$

Using [FV12, Lemma 2], we conclude that

$$\|e^{(tens)}\| \leq n\,k\,(\|e_1^{(Dec)}\| + \|e_2^{(Dec)}\|)\,(n\,\|\mathsf{sk}\| + 1) + 2k^2\,n^2\,(\|\mathsf{sk}\| + 1)^2 \tag{7}$$

which is dominated by the first term. This shows that the noise is roughly multiplied by the factor $2\,k\,n^2\,\|\mathsf{sk}\|$.

***Relinearization*** Second, a relinearization is performed using a key, denoted $\mathbf{rlk}$, generated by the CDKS algorithm detailed in §D.4. Recall that $\mathbf{rlk} = (\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2)$ where $(\mathbf{d}_0, \mathbf{d}_2) \leftarrow \mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk})$, the latter being defined as:
for $(\mathbf{e}_0^{(\mathbf{rlk})}, \mathbf{e}_2^{(\mathbf{rlk})}) \xleftarrow{\$} (\Psi_q^l)^2$ and $r \xleftarrow{\$} \mathscr{X}_q$; output $(\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2) = \big(-\mathsf{sk}.\mathbf{d}_1 + \mathbf{e}_0^{(\mathbf{rlk})} + r.\mathbf{g}, \mathbf{d}_1,\ r.\mathbf{a} + \mathbf{e}_2^{(\mathbf{rlk})} + \mathsf{sk}.\mathbf{g}\big)$.
Consider a three terms ciphertext $\hat{\mathsf{c}}$ with decryption noise $e^{(tens)}$ with respect to plaintext $m$ ($m_1 m_2$ in our context) and secret key $\mathsf{sk}$.

Let us now define an algorithm *Relin*, that takes as input $\hat{\mathsf{c}} = (\hat{\mathsf{c}}[0], \hat{\mathsf{c}}[1], \hat{\mathsf{c}}[2]) \in R_q^3$, $\mathbf{rlk} = \big(\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2\big) \in (R_q^l)^3, \mathbf{b} \in R_q^l$, and outputs $\mathsf{c}' = (\mathsf{c}'[0], \mathsf{c}'[1]) \in R_q^2$.

1 $\mathsf{c}'[0] \leftarrow \hat{\mathsf{c}}[0]$
2 $\mathsf{c}'[1] \leftarrow \hat{\mathsf{c}}[1]$
3 $\mathsf{c}'[2] \leftarrow \big\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathbf{b} \big\rangle$
4 $(\mathsf{c}'[0], \mathsf{c}'[1]) \leftarrow (\mathsf{c}'[0], \mathsf{c}'[1]) + \mathbf{g}^{-1}(\mathsf{c}'[2])^{\langle\cdot\rangle}(\mathbf{d}_0, \mathbf{d}_1)$
5 $\mathsf{c}'[1] \leftarrow \mathsf{c}'[1] + \big\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathbf{d}_2 \big\rangle$

Let us denote $e^{(relin)}$ the additional decryption noise of $\mathsf{c}'$, namely:

$$\hat{\mathsf{c}}[0] + \hat{\mathsf{c}}[1]\,\mathsf{sk} + \hat{\mathsf{c}}[2]\,\mathsf{sk}^2 = \mathsf{c}'[0] + \mathsf{c}'[1]\,\mathsf{sk} + e^{(relin)} \tag{8}$$

Let us estimate the noise introduced by the relinearization. Recall that $\mathsf{c}'[2] = \big\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathbf{b} \big\rangle$. Let us denote $err_1 = \Big\langle \mathbf{g}^{-1}(\mathsf{c}'[2]), \mathbf{e}_0^{(\mathbf{rlk})} \Big\rangle$ and $err_2 = \Big\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2])\,,\, \mathsf{sk}\,\mathbf{e}^{(\mathbf{pk})} + \mathbf{e}_2^{(\mathbf{rlk})}\,\mathsf{sk} \Big\rangle$. We have

$$\Big\langle \mathbf{g}^{-1}(\mathsf{c}'[2])^{\langle\cdot\rangle}(\mathbf{d}_0, \mathbf{d}_1),\ (1, \mathsf{sk}) \Big\rangle = r.\mathsf{c}'[2] + \Big\langle \mathbf{g}^{-1}(\mathsf{c}'[2]), \mathbf{e}_0^{(\mathbf{rlk})} \Big\rangle = r.\mathsf{c}'[2] + err_1 \text{ and}$$

$$\begin{aligned}\left\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathbf{d}_2 \right\rangle \mathsf{sk} &= \left\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), -r\,\mathbf{b} + \mathbf{e}^{(\mathbf{pk})}\,\mathsf{sk} + \mathbf{e}_2^{(\mathbf{rlk})}\,\mathsf{sk} + \mathsf{sk}^2.\mathbf{g} \right\rangle \\ &= -r\,\mathsf{c}'[2] + \hat{\mathsf{c}}[2].\mathsf{sk}^2 + \left\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathsf{sk}.\mathbf{e}^{(\mathbf{pk})} + \mathbf{e}_2^{(\mathbf{rlk})}.\mathsf{sk} \right\rangle \\ &= -r\,\mathsf{c}'[2] + \hat{\mathsf{c}}[2].\mathsf{sk}^2 + err_2.\end{aligned}$$

We can now analyze the noise introduced by the relinearization. First, recall that $\mathbf{g^{-1}}(.)$ decompose an element $x \in R_q$ into a *short* vector $\mathbf{u} = (u_0, ..., u_{l-1}) \in R^l$ such that $\langle \mathbf{u}, \mathbf{g} \rangle = x \mod q$ with $\|u_i\| \leq B_g$ for $i = 0, 1, ..., l-1$. In details, we can write

$$\begin{aligned}\mathsf{c}'[0] + \mathsf{c}'[1].\mathsf{sk} &= \hat{\mathsf{c}}[0] + \mathbf{g}^{-1}(\mathsf{c}'[2])^{<\cdot>}(\mathbf{d}_0, \mathbf{d}_1) + \Big(\hat{\mathsf{c}}[1] + \mathbf{g}^{-1}(\mathsf{c}'[2])^{<\cdot>}(\mathbf{d}_0, \mathbf{d}_1) \\ &\quad + \left\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathbf{d}_2 \right\rangle \Big) \mathsf{sk} \\ &= \hat{\mathsf{c}}[0] + \hat{\mathsf{c}}[1].\mathsf{sk} + \left\langle \mathbf{g}^{-1}(\mathsf{c}'[2])^{<\cdot>}(\mathbf{d}_0, \mathbf{d}_1), (1, \mathsf{sk}) \right\rangle + \left\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathbf{d}_2 \right\rangle \mathsf{sk} \\ &= \hat{\mathsf{c}}[0] + \hat{\mathsf{c}}[1].\mathsf{sk} + \hat{\mathsf{c}}[2].\mathsf{sk}^2 + err_1 + err_2 \\ &= \Delta m + e^{(tens)} + e^{(relin)}\end{aligned}$$

with $e^{(tens)}$ introduced above and

$$\|e^{(relin)}\| \leq \|err_1\| + \|err_2\| \leq n\,B_g \|\mathbf{e}_0^{(\mathbf{rlk})}\| + n^2\,l\,B_g \|sk\| (\|\mathbf{e}^{(\mathbf{pk})}\| + \|\mathbf{e}_2^{(\mathbf{rlk})}\|) \tag{9}$$

### D.7 Correctness of Threshold Decryption after Homomorphic evaluation of a Circuit and Noise Analysis

Let us now define then estimate the noise $B_C$ introduced during the evaluation of a circuit C, and formalize at which condition the threshold decryption of a homomorphically evaluated ciphertext, does return the correctly evaluated plaintext.

***Def-Prop 18 (Decryption noise of a circuit: $B_C$).*** *For any arithmetic circuit* $\mathrm{C} : R_k^{\mathscr{L}} \to R_k$*, with input gates indexed by* $\mathscr{L}$*, we consider the largest norm of the decryption noise* $e^{(Dec)}(\mathsf{c}, \mathsf{sk}, y)$ *of a ciphertext* $\mathsf{c}$*, over the previous choices, and over the choices: of elements* $(m_\ell \in R_k)_{\ell \in \mathscr{L}}$*, and of arbitrary fresh BFV Encryptions of them following Share & Shrink (Fig. 4)* $(\mathsf{c}_\ell)_{\ell \in \mathscr{L}}$*; denoting* $\mathsf{c} := \mathsf{Eval}(C, (\mathsf{c}_\ell)_\ell, \mathbf{rlk}, \mathbf{b})$ *and* $y := C((m_\ell)_\ell)$*. From Def-Prop 15 and Fig. 5, it follows that, for any* $y$ *and* $\mathsf{c}$ *as above,* If *the second distributed decryption method is used with a level of noise* $B_{sm}$ *such that:*

$$B_C + N.B_{sm} < \frac{\Delta}{4} \tag{10}$$

Then: $\Omega_{\mathsf{Dec}}\big(\mathsf{c}[0] + \mathsf{c}[1].\mathsf{sk}\big) = y$.

The noise introduced by evaluating $C$ is dominated by the one introduced by multiplications rather than additions, unless the width is much larger than $L$, which we do not assume in this estimation. Thus we neglect, comparatively, the impact of $|\mathscr{L}|$. Using Proposition 17, we estimate an upper bound on the decryption noise of the evaluated ciphertext as:

$$C_1^L.B_{fresh} + C_2 \Sigma_{i=0}^{L-1} C_1^i \leq C_1^L.B_{fresh} + L.C_2.C_1^{L-1} \tag{11}$$

with $C_1 = 2.k.n^2.N$ and $C_2 = 2.n^2.l^2.N^2.B.B_g$.

# E $\mathsf{BFV} + \mathsf{CDKS}^*$ Bootstrapping

## E.1 Bootstrapping Key Generation

The "Bootstrapping" of a single-key FHE ciphertext is a local computation that homomorphically brings back the size of its decryption noise, roughly to the one of a fresh ciphertext. Our starting point is the one described in [CDKS19, §5] in their context of multikey $\mathsf{BFV}$. As we did for relinearization, we particularize it to our single-key context. Leaving all details in §E, let us just mention that two elementary homomorphic operations in bootstrapping require an auxiliary key. The latter consists of a collection of keys, indexed by $j \in (\mathbb{Z}/2n)^*$, constructed as follows in the single key setting. Let $\mathsf{sk}$ be the decryption key, considering $\tau_j$ over $R_q$, sample $\mathbf{h}_1 \leftarrow U(R_q^l)$ and $\mathbf{e}^{(\mathbf{bk})} \leftarrow \Psi_q^l$ and output $\mathbf{bk}(j) := \big(\mathbf{h}_1,\ \mathbf{h}_0(j) = -\mathsf{sk}\,\mathbf{h}_1 + \mathbf{e}^{(\mathbf{bk})} + \tau_j(\mathsf{sk})\,\mathbf{g}\big)$.

We enrich the MPC protocol with generation of this bootstrapping key without changing the communication complexity, as follows. Receive $\mathbf{h}_1 \leftarrow U(R_q^l)$ from $\overline{\mathcal{G}}_{\mathrm{URS}}$. For each player $i$, let $\mathsf{sk}_i$ be its additive contribution to the secret key, used in the $\mathsf{DKG}$.

- $\mathsf{BkKG}(\mathbf{h}_1, \mathsf{sk}_i, j)$: Given a secret key contribution $\mathsf{sk}_i$, sample $\mathbf{e}_i^{(\mathbf{bk})} \leftarrow \Psi_q^l$ and compute $\mathbf{h}_{0,i}(j) = -\mathsf{sk}_i\,\mathbf{h}_1 + \mathbf{e}_i^{(\mathbf{bk})} + \tau_j(\mathsf{sk}_i)\,\mathbf{g}$.

*Remark 1.* To add bootstrapping in the UC simulation, we must enlarge the assumption 13 in order to take into account these new keys. This is done implicitly in [CDKS19]. From such enlarged assumption, we deduce the analogous of Corollary 14, w.r.t. $\mathbf{h}_1 \leftarrow U(R_q^l)$.

## E.2 Bootstrapping Pipeline

We follow the bootstrapping of [CDKS19] for multikey FHE and particularize it in our single-key context. In short, they follow the algorithm improved by [CH18], that consists in four steps, denoted as (1) Modulus raise, (2) Linear Transformation, (3) Extraction and (4) the Inverse Linear Transformation.

Technically, the linear transformation requires the homomorphic evaluation of the rotation of plaintext slots. In addition [CDKS19] also require the homomorphic evaluation of "Galois elements slot-by-slot". In [GHS12, p23] it is explained how these two evaluations, i.e., operations on plaintexts, can be decomposed into additions, scalar multiplications and applications of automorphisms of $R_k = \mathbb{Z}_k[X]/(X^n+1)$, denoted $\{\tau_j,\ j \in (\mathbb{Z}/2n)^*\}$, each being defined by: $X \to X^j$. In [CDKS19], it is observed that homomorphic evaluation of each $\tau_j$ can be realized with the auxiliary key, which we denoted as $\big(\mathbf{h}_0(j),\ \mathbf{h}_1\big)$ whose technical purpose is "key-switching".

In some more details, given a ciphertext $\mathsf{c} = (\mathsf{c}[0], \mathsf{c}[1]) \in R_q^2$ of $m$, the goal is to homomorphically evaluate $\tau_j$ on the plaintext, i.e. to find $\mathsf{c}'$ such that $\langle \mathsf{c}', s\rangle = \tau_j(\langle \mathsf{c}, s\rangle)$. To achieve this, we first compute $\tau_j(\mathsf{c}) = (\tau_j(\mathsf{c}[0]), \tau_j(\mathsf{c}[1]))$ the ciphertext obtained by taking $\tau_j$ to the entries of $\mathsf{c}$. Then $\tau_j(\mathsf{c})$ is a valid encryption of $\tau_j(m)$ corresponding the secret key $\tau_j(\mathsf{sk})$. The key-switching procedure is then applied to $\tau_j(\mathsf{c})$, which has for consequence to generate a new ciphertext encrypting the same message under the original secret key $s$ instead of $\tau_j(s)$. This key switching algorithm presented below is adapted from [CDKS19].

## E.3 Key Switching algorithm (adapted from [CDKS19])

We now provide the key switching method *KeySwitch* and then discuss its correctness. It takes as input $\mathsf{c} = (\mathsf{c}[0], \mathsf{c}[1]) \in R_q^2, \mathbf{bk} = [\mathbf{h}_0 | \mathbf{h}_1] \in (R_q^l)^2$, and outputs $\mathsf{c}' = (\mathsf{c}'[0], \mathsf{c}'[1]) \in R_q^2$ as follows:

1 $\mathsf{c}'[0] \leftarrow \tau_j(\mathsf{c}[0])$
2 $\mathsf{c}'[0] \leftarrow \mathsf{c}'[0] + \big\langle \mathbf{g}^{-1}(\tau_j(\mathsf{c}[1])), \mathbf{h}_0 \big\rangle$
3 $\mathsf{c}'[1] \leftarrow \big\langle \mathbf{g}^{-1}(\tau_j(\mathsf{c}[1])), \mathbf{h}_1 \big\rangle$

***Correctness*** From the definition and with $s = (1, \mathsf{sk})$, the output ciphertext $\mathsf{c}' = (\mathsf{c}'[0], \mathsf{c}'[1]) \leftarrow KeySwitch(\mathsf{c}, \mathbf{bk})$ holds

$$\begin{aligned}\mathsf{c}'[0] + \mathsf{c}'[1]\,\mathsf{sk} &= \tau_j(\mathsf{c}[0]) + \left\langle \mathbf{g}^{-1}(\tau_j(\mathsf{c}[1])), \mathbf{h}_0 \right\rangle + \left\langle \mathbf{g}^{-1}(\tau_j(\mathsf{c}[1])), \mathbf{h}_1 \right\rangle \\ &\approx \tau_j(\mathsf{c}[0]) + \left\langle \mathbf{g}^{-1}(\tau_j(\mathsf{c}[1])), \tau_j(s).\mathbf{g} \right\rangle = \langle \tau_j(\mathsf{c}), \tau_j(\mathsf{sk}) \rangle = \tau_j(\langle \mathsf{c}, s \rangle)\end{aligned}$$

as desired.

## F Detailed Protocol $\Pi^{\mathcal{F}_{\mathsf{Lsss}}}$ when instantiated from BFV + CDKS*

In Fig. 11, we detail our MPC protocol $\Pi^{\mathcal{F}_{\mathsf{Lsss}}}$ when instantiated from $\mathsf{BFV} + \mathsf{CDKS}^*$, that we adapt to our distributed use-case. Notably, we use the alternative version of relinearization key generation presented in Appendix D.4.1, and follow Appendix D.4.3 to make it distributed. Note that this requires uniform random strings $\mathbf{a}$ and $\mathbf{d}_1$ to be in $R_q^l$, and to be common parameters in to allow some form of additivity (they were previously sampled locally in [FV12] and in [CDKS19] respectively). As a consequence, the function BFV.KeyGen detailed in section 3.1.2 that took some parameter $a \in R_q$ as input variable, is naturally extended to handle input $\mathbf{a} \in R_q^l$ and to return a public key $(\mathbf{b}, \mathbf{a}) \in R_q^{l \times 2}$).

The choice of parameters are discussed in section 6.2. Moreover, for security and correctness, we require Equation (10); and:

$$\frac{B_{\mathrm{C}}}{B_{sm}} = \mathrm{negl}(\lambda) \;\text{ and }\; \frac{2nNB}{B_{Enc}} = \mathrm{negl}(\lambda)\ . \tag{12}$$

// the former will be used in Lemma 22, the latter in Lemma 23

## G Practical Parameters Estimation

Recall that the common encryption key generated by the DKG comes as a $\mathsf{BFV} + \mathsf{CDKS}^*$ single-key, of the form $(\mathbf{b}, \mathbf{a}) \in R_q^{2l}$, thus of total bit-size $n.l.\log q$. Likewise, the common relinearization key is of the form $(\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2) \in R_q^{3l}$, thus of total bit-size $3.n.l.\log q$. In the table below we recall the parameters used in [CDKS19, 6.2, Table 2] (which are default settings in the library "SEAL"). Notice that in their table, $l$ is instead denoted "$\#p_i$". The authors indicate that these parameters were chosen following the "the HE security standard" [ACC+21], in order to achieve at least $128bits$ of security. Also, recall that [ACC+21] advises to set $\sigma = 3.2$.

| $n$ | $\log q$ | $l$ |
|---|---|---|
| $2^{13}$ | 218 | 4 |
| $2^{14}$ | 438 | 8 |
| $2^{15}$ | 881 | 16 |

Table 3: Parameters of [CDKS19, 6.2, Table 2], calibrated for at least 128 bits security according to [ACC+21].

## Protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ instantiated from BFV

**Participants:** $N$ players $P_1, \ldots, P_N$ and $|\mathscr{L}|$ input owners;
**Inputs** (for each input owner $\ell \in \mathscr{L}$): a plaintext $\Delta m_\ell$ with label $\overline{\Delta m_\ell}$.

**Setup.** Each player $P_i$:

- Sends $(\mathsf{Setup})$ to $\mathcal{F}_{\mathsf{LSSS}}$
- Obtains common uniform strings $(\mathbf{a}, \mathbf{d}_1) \leftarrow \overline{\mathcal{G}}_{\mathrm{URS}}$.

**Broadcast.**

- *Input and Randomness Distribution:* Upon $\mathsf{ready}$ from $\mathcal{F}_{\mathsf{LSSS}}$, each input owner $\ell \in \mathscr{L}$:
  (1) Samples $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(Enc)} \xleftarrow{\$} \mathscr{B}_{Enc,q}$ and $e_1^{(Enc)} \xleftarrow{\$} \Psi_q$ and sends $(\mathsf{input}, \{\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}}\}, \{\Delta m_\ell, u_\ell, e_{0,\ell}^{(Enc)}, e_{1,\ell}^{(Enc)}\})$ to $\mathcal{F}_{\mathsf{LSSS}}$. Then it goes offline.
- *Distributed Keys Generation:* Upon $\mathsf{ready}$ from $\mathcal{F}_{\mathsf{LSSS}}$, each player $P_i$:
  (1) Computes $(\mathsf{sk}_i, (\mathbf{b}_i, \mathbf{a})) \leftarrow \mathsf{BFV.KeyGen}(\mathbf{a})$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$. Sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSSS}}$ and $(\mathbf{b}_i, (\mathbf{d}_{0,i}, \mathbf{d}_{2,i}))$ over $\mathsf{BC}^{P_i}$.
- *Distributed Smudging Noise Generation:* Upon $\mathsf{ready}$ from $\mathcal{F}_{\mathsf{LSSS}}$, each player $P_i$:
  (1) Samples $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ and sends $(\mathsf{input}, \overline{e_{\mathrm{sm},i}}, e_{\mathrm{sm},i})$ to $\mathcal{F}_{\mathsf{LSSS}}$. //Once for each subsequent distributed decryption, if multiple circuits

**Local computation.** For all $\ell \in \mathscr{L}$, each player waits to receive $(\mathsf{stored}, \ell, *_\ell)$ from $\mathcal{F}_{\mathsf{LSSS}}$ for all fours variables of $\ell$'s "input and randomness distribution"; then sets $S_{\mathsf{c}} \subset \mathscr{L}$ the $\ell$'s for which no $*_\ell = \perp$. For all $P \in \mathscr{P}$, each player waits to receive $(\mathsf{stored}, P, *_P)$ from $\mathcal{F}_{\mathsf{LSSS}}$ for both instances in "distributed Keys Generation" and in "Distributed Smudging Noise Generation", and an output from all instances of $(\mathsf{BC}^P)_{P \in \mathscr{P}}$; then sets $S \subset \mathscr{P}$ the set of players for which no instance returned $\perp$.
Each player $P_i$:

- $\forall j \in S$, parses the outputs of $\mathsf{BC}^{P_j}$ as $(\mathbf{b}_j, (\mathbf{d}_{0,j}, \mathbf{d}_{2,j}))$ and computes $\mathbf{b} = \Sigma_{j \in S} \mathbf{b}_j$ and $\mathbf{rlk} = (\Sigma_{j \in S} \mathbf{d}_{0,j}, \mathbf{d}_1, \Sigma_{j \in S} \mathbf{d}_{2,j})$. Sets $a = \mathbf{a}[0]$ and $b = \mathbf{b}[0]$, and define the secret key as $\mathsf{sk} = \Sigma_{i \in S} \mathsf{sk}_i$ and the smudging noise $e_{\mathrm{sm}} = \Sigma_{i \in S} e_{\mathrm{sm},i}$ //accessible through $\mathcal{F}_{\mathsf{LSSS}}$, via the labels $\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}}$

**Asynchronous step.** Each player $P_i$:

(2) $\forall \ell \in S_{\mathsf{c}}$, given labels $(\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}})$, a key $\mathsf{ek} = (b, a)$, sends $\big(\mathsf{LCOpen}, \Lambda_{Enc}^{b,a}(\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}})\big)$ to $\mathcal{F}_{\mathsf{LSSS}}$, and obtains a ciphertect $\mathsf{c}_\ell$.

**Evaluation:** To evaluate a circuit C, each player $P_i$:

(3) Computes $\mathsf{c} \leftarrow \mathsf{BFV.Eval}(\mathrm{C}, \{\mathsf{c}_j\}_{j \in S_{\mathsf{c}}}, \mathbf{rlk}, \mathbf{b})$.

**Distributed Decryption:** Each player $P_i$:

(3) Given labels $(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})$, and a ciphertext $\mathsf{c}$, sends $\big(\mathsf{LCOpen}, \Lambda_{Dec+sm}^{\mathsf{c}}(\overline{\mathsf{sk}}, \overline{e_{sm}})\big)$ to $\mathcal{F}_{\mathsf{LSSS}}$.
- Upon receiving $(\Lambda_{Dec}^{\mathsf{c}},\ \mu)$ from $\mathcal{F}_{\mathsf{LSSS}}$, outputs $m := \Omega_{\mathsf{Dec}}(\mu)$.

Fig. 11: Protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$ instantiated from BFV

## H Further Details on the Proof of Theorem 3

### H.1 Pseudorandomness of BFV ciphertexts with uniformly generated public keys

First, we want to prove that considering a public key sampled uniformly at random, the ciphertext produced by $\mathsf{BFV.Enc}$ are pseudorandom under the RLWE assumption. The reason is that in the context of $\mathsf{Hybrid}_3$, i.e., in Lemma 23 the view of $\mathscr{E}$ is very similar to the one of the BFV scheme, except that the key is uniformly random. We formalize it by the game *Semantic* shown below:

**Setup** : The challenger generates samples $\mathbf{a}, \mathbf{b} \xleftarrow{\$} U(R_q^l)$ and sends $(\mathbf{a}, \mathbf{b})$ to $\mathcal{A}$.

**Query** : $\mathcal{A}$ chooses a $m \in R_k$ and sends it to the challenger.

**Challenge** The challenger picks a random $\beta \in \{0, 1\}$.

- If $\beta = 0$, it chooses $\mathsf{c}^* = (\mathsf{c}_0^*, \mathsf{c}_1^*) \xleftarrow{\$} R_q^2$ uniformly at random.
- If $\beta = 1$, it generates a valid ciphertext $\mathsf{c}^* = (\mathsf{c}_0^*, \mathsf{c}_1^*) \leftarrow \mathsf{BFV.Enc}(\mathsf{ek} = (\mathbf{b}[0], \mathbf{a}[0]), m)$.

**Guess** $\mathcal{A}$ gets $\mathsf{c}^* = (\mathsf{c}_0^*, \mathsf{c}_1^*)$ and outputs $\beta' \in \{0, 1\}$. It wins if $\beta' = \beta$.

***Lemma 19.*** *Let $\mathsf{pp} = (R_q, l, \mathscr{X}_q, R_k, \Psi_q)$ be parameters such that Assumption 13 holds and $\mathscr{B}_{Enc,q}$ that satisfies Equation 12 in §5. Then for any PPT adversary $\mathcal{A}$, the function $AdvCPA_{\mathcal{A}}^{Semantic}(\lambda) := \left|Pr[\beta = \beta'] - \frac{1}{2}\right|$, denoted as the advantage of $\mathcal{A}$, is negligible in $\lambda$.*

*Proof.* In case $\beta = 1$, the adversary is returned the pair $(\Delta m + u.b + e_0^{(Enc)}, a.u + e_1^{(Enc)}) \in R_q^2$, where the fixed $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(Enc)} \xleftarrow{\$} \mathscr{B}_{Enc,q}$ and $e_1^{(Enc)} \xleftarrow{\$} \Psi_q$ are secretly sampled. Subtracting the known $\Delta m$ from the left component, the pair constitutes 2 RLWE samples, namely: sample a fixed $u \xleftarrow{\$} \mathscr{X}_q$, then construct the first RLWE sample with $(b \leftarrow U(R_q), e_0^{(Enc)} \leftarrow \mathscr{B}_{Enc,q})$ and the second one with $(a \xleftarrow{\$} U(R_q), e_0^{(Enc)} \xleftarrow{\$} \Psi_q)$. Thus, by RLWE for $(\mathscr{X}_q, \Psi_q)$, and thus a fortiori for $(\mathscr{X}_q, \mathscr{B}_{Enc,q})$ (Equation 12), the two RLWE samples are indistinguishable from a sample in $U(R_q^2)$. □

### H.2 IND-CPA under Joint Keys

In [AJL+12, Lemma 3.4], it is proven that an adversary cannot distinguish the ciphertext of a chosen plaintext from a random string, even if the ciphertext if encrypted under a key of the form $\mathsf{ek} = (b + b', a)$, where $b'$ is adaptively generated by the semi-honest adversary after it saw $b$. Our goal is to adapt their result in the RLWE setting. Since we need only this result in the context of $\mathsf{Hybrid}_3$, i.e., in Lemma 23, we can consider that the honest key $\mathsf{ek} = (b, a)$ is generated uniformly at random, instead of generated by $\mathsf{BFV.KeyGen}$. We consider an experiment $JointKey(R_q, l, \mathscr{X}_q, R_k, \Psi_q, \mathscr{B}_{Enc,q})$ between an attacker $\mathcal{A}$ and a challenger defined as:

**Setup** The challenger generates samples $\mathbf{a}, \mathbf{b} \xleftarrow{\$} U(R_q^l)$ and sends $(\mathbf{a}, \mathbf{b})$ to $\mathcal{A}$.

**Query** $\mathcal{A}$ adaptively chooses: $t$ pairs $(\mathsf{sk}_i, \mathbf{e}_i^{(\mathbf{pk})})_{i \in \mathcal{I}}$, both terms being either $\perp$ or such that $\|\mathsf{sk}_i\| = 1$ and $\|\mathbf{e}_i^{(\mathbf{pk})}\| \leq l\,B$. Define $\mathsf{sk}' := \Sigma_{i \in \mathcal{I}} \mathsf{sk}_i$ where the $\perp$ values are set to 0, and likewise for $\mathbf{e}^{(\mathbf{pk})} := \sum_{i \in \mathcal{I}} \mathbf{e}_i^{(\mathbf{pk})}$. $\mathcal{A}$ outputs $\left\{\mathbf{b}' = -\mathbf{a}.\mathsf{sk}' + \mathbf{e}^{(\mathbf{pk})}, (\mathsf{sk}'_i)_{i \in \mathcal{I}}, (\mathbf{e}_i^{(\mathbf{pk})})_{i \in \mathcal{I}}\right\}$ to the challenger, along with some $m \in R_k$ of its choice.

**Challenge** The challenger sets $\mathbf{pk} = \mathbf{b} + \mathbf{b}'$ and picks a random $\beta \in \{0, 1\}$.

- If $\beta = 0$, it chooses $\mathsf{c}^* = (\mathsf{c}_0^*, \mathsf{c}_1^*) \xleftarrow{\$} R_q^2$ uniformly at random.
- If $\beta = 1$, it generates a valid ciphertext $\mathsf{c}^* = (\mathsf{c}_0^*, \mathsf{c}_1^*) \leftarrow \mathsf{BFV.Enc}(\mathsf{ek} = (\mathbf{pk}[0], \mathbf{a}[0]), m)$.

**Guess** $\mathcal{A}$ gets $\mathsf{c}^* = (\mathsf{c}_0^*, \mathsf{c}_1^*)$ and outputs $\beta' \in \{0, 1\}$. It wins if $\beta' = \beta$.

***Lemma 20.*** *Let $\mathsf{pp} = (R_q, l, \mathscr{X}_q, R_k, \Psi_q)$ be parameters such that Assumption 13 holds and $\mathscr{B}_{Enc,q}$ that satisfies Equation 12 in §5. Then for any PPT adversary $\mathcal{A}$, we have:*

$$Pr\big[JointKey_{\mathcal{A}}(R_q, l, \mathscr{X}_q, R_k, \Psi_q, \mathscr{B}_{Enc,q}) = 1\big] - 1/2 = negl(\lambda). \tag{13}$$

*Proof.* We construct an adversary $\mathcal{A}'$ playing the game of Lemma 19. $\mathcal{A}'$ uses as black box an adversary $\mathcal{A}$ for $JointKey(R_q, l, \mathscr{X}_q, R_k, \Psi_q, \mathscr{B}_{Enc,q})$, as follows. The challenger gives $\mathcal{A}'$ the value $(\mathbf{b}, \mathbf{a})$, and a ciphertext $(\mathsf{c}_0, \mathsf{c}_1)$ which is either chosen as $\mathsf{BFV.Enc}(\mathsf{ek} = (\mathbf{b}[0], \mathbf{a}[0]), 0)$ ($\beta = 1$) or is a sample in $U(R_q^2)$ ($\beta = 0$). Then $\mathcal{A}'$ gives $\mathbf{b}$ to $\mathcal{A}$ and gets back $\left(\mathbf{b}' = -\mathbf{a}.\mathsf{sk}' + \mathbf{e}^{(\mathbf{pk})}, sk', \mathbf{e}^{(\mathbf{pk})}, m\right)$ from $\mathcal{A}$, where $m$ is a challenge plaintext. Finally, $\mathcal{A}'$ sets $(\mathsf{c}_0^*, \mathsf{c}_1^*) = (\mathsf{c}_0 - \mathsf{c}_1.\mathsf{sk}', \mathsf{c}_1) \in R_q^2$, sends it to $\mathcal{A}$ and outputs the bit $\beta'$ obtained from $\mathcal{A}$.

It is easy to see that if $\beta = 0$, then $(\mathsf{c}_0^*, \mathsf{c}_1^*)$ is uniformly random. On the other hand, if $\beta = 1$, we can write $\mathsf{c}_0 = u.b + e_0^{(Enc)} \in R_q$ and $c_1^* = u.a + e_1^{(Enc)} \in R_q$ for some $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(Enc)} \xleftarrow{\$} \mathscr{B}_{Enc,q}$, $e_1^{(Enc)} \xleftarrow{\$} \Psi_q$ and $b = \mathbf{b}[0]$, $a = \mathbf{a}[0]$, and with $e^{(pk)} = \mathbf{e}^{(\mathbf{pk})}[0]$ :

$$\begin{aligned}
\mathsf{c}_0^* &= u.b + e_0^{(Enc)} - \mathsf{c}_1.\mathsf{sk}' = u.b + e_0^{(Enc)} - (u.a + e_1^{(Enc)}).sk' \\
&= u(b + b') + e_0^{(Enc)} - e_1^{(Enc)}.\mathsf{sk}' - u.e^{(pk)} \\
&\overset{s}{\equiv} u.(b + b') + e_0^{(Enc)}
\end{aligned}$$

The last equality states a statistical indistinguishability between the distributions of $e_0^{(Enc)} - e_1^{(Enc)}.\mathsf{sk}' - u.e^{(pk)}$ and of $e_0^{(Enc)}$, which we now prove. To start with, from equation (3), we have both $\|e_1^{(Enc)}.\mathsf{sk}'\| \leq nNB$ and $\|u.e^{(pk)}\| \leq nNB$. Thus, $\|e_1^{(Enc)}.\mathsf{sk}' - u.e^{(pk)}\| \leq 2nNB$. But on the other hand, $\|e_0^{(Enc)}\| \leq B_{Enc}$. We conclude since the parameters are chosen such that $\frac{2nNB}{B_{Enc}} = negl(\lambda)$ (cf Equation (12) in §D.7). This conclusion can be formalized as the "smudging Lemma 21" below, which implies that, in the sum $e_0^{(Enc)} - e_1^{(Enc)}.\mathsf{sk}' - u.e^{(pk)}$, we have that the distribution of $-e_1^{(Enc)}.\mathsf{sk}' - u.e^{(pk)}$ is "smudged-out" by the one of $e_0^{(Enc)}$. Therefore, $\mathcal{A}'$ acts indistinguishably from the challenger of the Game of Lemma 19, thus has the same advantage as $\mathcal{A}$. □

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 [DDE+23, 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 21 (Smudging lemma [AJL+12]).*** *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.*

### H.3 Full Details of the Proof of Theorem 3

**H.3.1 Description of the Simulator $\mathcal{S}$ of $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$** $\mathcal{S}$ Initiates in its head: sets of $N$ players and $\mathscr{L}$ of Owners, and may initially receive corruption requests from $\mathscr{E}$ for arbitrarily many owners and up to $t$ players, indexed by $\mathcal{I} \subset \mathscr{P}$. It simulates functionalities $\mathsf{BC}, \mathcal{F}_{\mathsf{LSSS}}$ following a correct behavior, apart from the value returned by $\mathcal{F}_{\mathsf{LSSS}}$ in the Output computation. For instance, receiving $\mathsf{activate}$ from $\mathscr{E}$ intended to some functionality, $\mathcal{S}$ internally sends it to the functionality then simulates the steps taken by the functionality accordingly. Upon every output from a simulated functionality to a simulated corrupt player, or, upon every message from a simulated functionality to the simulated $\mathcal{A}$, then $\mathcal{S}$ immediately forwards it to $\mathscr{E}$, as would have done the actual dummy $\mathcal{A}$.

- **Setup.**
  (0) Simulates the setup of $\mathcal{F}_{\mathsf{LSSS}}$, i.e., queries $\mathsf{ReqInput}$ for corrupt players then, upon receiving $\mathsf{Setup}$ or $\mathsf{activate}$ on behalf of all corrupt players, sends delayed output $\mathsf{ready}$ to players and owners.
  (0) Retrieves $(\mathbf{a}, \mathbf{d}_1)$ from $\overline{\mathcal{G}}_{\mathrm{URS}}$ and sends it to all on behalf of $\overline{\mathcal{G}}_{\mathrm{URS}}$
- **Input and randomnesses distribution:** Simulates of a correct behavior of $\mathcal{F}_{\mathsf{LSSS}}$. Moreover:
  (1) $\forall$ simulated honest $\ell \in \mathscr{L}$: sets $\widetilde{m_\ell} := 0$ and samples $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(Enc)} \xleftarrow{\$} \mathscr{B}_{Enc,q}$ and $e_1^{(Enc)} \xleftarrow{\$} \Psi_q$. Then sends $(\mathsf{input}, \{\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}}\}, \{\Delta m_\ell, u_\ell, e_{0,\ell}^{(Enc)}, e_{1,\ell}^{(Enc)}\})$ to $\mathcal{F}_{\mathsf{LSSS}}$.

(1) $\forall$ simulated corrupt $\ell \in \mathscr{L}$, upon $\Big(\mathsf{input}$ OR $\mathsf{activate}, \{\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}}\}, \{\Delta m_\ell, u_\ell, e_{0,\ell}^{(Enc)}, e_{1,\ell}^{(Enc)}\}\Big)$ from $\mathscr{E}$, sets $\widetilde{m_\ell} := 0$ if $m_\ell = \perp$ or $\widetilde{m_\ell} := m_\ell$ otherwise, and sends $(\mathsf{input}, \ell, m_\ell)$ to $\mathcal{F}_{\mathrm{C}}$.

- **Distributed Keys Generation:** Simulates of a correct behavior of $\mathcal{F}_{\mathsf{LSSS}}$. For every simulated honest $P_i \in \mathcal{H}$:
  (1) Samples $\mathsf{sk}_i \xleftarrow{\$} \mathscr{X}_q$ (never used) and $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$, and sends them to $\mathcal{F}_{\mathsf{LSSS}}$.
  (1) Samples $\mathbf{b}_i \xleftarrow{\$} U(R_q^l)$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \xleftarrow{\$} U(R_q^l \times R_q^l)$, sends them over $\mathsf{BC}^{P_i}$.
  As in the protocol, $\mathcal{S}$ sets $S_{\mathsf{c}} \subset \mathscr{L}$ the owners, resp. $S \subset \mathscr{P}$ the players, for which no instance returned $\perp$.
- **Threshold encryption (Shrink) and evaluation** Simulates correct behaviors. For instance in (2), denoting $\mathbf{b} := \sum_{j\in S} \mathbf{b}_j$ and $\mathbf{rlk} := (\sum_{j\in S} \mathbf{d}_{0,j}, \mathbf{d}_1, \sum_{j\in S} \mathbf{d}_{2,j})$, the simulated $\mathcal{F}_{\mathsf{LSSS}}$, for $\forall \ell \in S_{\mathsf{c}}$, delay-outputs: $\big(\Lambda_{Enc}^{a,b}, \widetilde{\mathsf{c}_\ell} := (\Delta\widetilde{m_\ell} + u_\ell\, b + e_{0,\ell}^{(Enc)}, u_\ell\, a + e_{1,\ell}^{(Enc)})\big)$.
- **Output computation:** Upon receiving $(\mathsf{evaluation}, y)$ from $\mathcal{F}_{\mathrm{C}}$, where by definition $y = \mathrm{C}(\{m_\ell\}_{\ell\in S_{\mathsf{c}}})$, then $\mathcal{S}$ simulates the following incorrect behavior:
  - $\mathcal{F}_{\mathsf{LSSS}}$ delay-outputs $\Big(\Lambda_{dec}^{\mathsf{c}},\ \mu^{\mathcal{S}} := \Delta y + \Sigma_{j\in S} e_{\mathrm{sm},j}\Big)$.

***Proof of indistinguishability with a real execution*** We go through a series of hybrid games, starting from the real execution $REAL_{\Pi_{\mathrm{C}}}$. For completeness they are spelled out in full details in §H.3. The view of $\mathscr{E}$ consists of its interactions with $\mathcal{A}/\mathcal{S}$, and of the outputs of the actual honest players. We deal with the latter once and for all in Lemma 22.

$\mathsf{Hybrid}_1$ ***[Simulated Decryption].*** $\mathcal{F}_{\mathsf{LSSS}}$ is modified in Output computation: there it, incorrectly, outputs $\mu^{\mathcal{S}} := \Delta.y + \Sigma_{j\in S} e_{\mathrm{sm},j}$, where $y := \mathrm{C}((m_\ell)_{\ell\in S_{\mathsf{c}}})$ is the evaluation in clear of the circuit on the actual inputs.

***Lemma 22.*** *The outputs of the actual honest players are the same in $REAL_{\Pi_{\mathrm{C}}}$ and $IDEAL_{\mathcal{F}_{\mathrm{C}},\mathcal{S},\mathscr{E}}$. Also, the views of $\mathscr{E}$ in $REAL_{\Pi_{\mathrm{C}}}$ and $\mathsf{Hybrid}_1$ are computationally indistinguishable.*

*Proof.* It is convenient to prove the two claims at once. The view of $\mathscr{E}$ is identical in $REAL_{\Pi_{\mathrm{C}}}$ and $\mathsf{Hybrid}_1$ until (3) included. There, for all $\ell \in S_{\mathsf{c}}$, consecutively to an instance of Share&Shrink, $\mathcal{F}_{\mathsf{LSSS}}$ delay-outputs a fresh $\mathsf{BFV}$ encryption $\mathsf{c}_\ell$ of $m_\ell$ under $\mathsf{ek} = (\mathbf{b}, \mathbf{a})$, following the terminology of §D.5. Thus, the evaluated $\mathsf{c} := \mathsf{BFV.Eval}(\mathrm{C}, \{\mathsf{c}_j\}_{j\in S_{\mathsf{c}}}, \mathbf{rlk}, \mathbf{b})$ is the same in both views. In the Output computation of $REAL_{\Pi_{\mathrm{C}}}$, the output of $\mathcal{F}_{\mathsf{LSSS}}$ is:

$$\mu = \mathsf{c}[0] + \mathsf{c}[1].\Sigma_{j\in S}\mathsf{sk}_j + \Sigma_{j\in S} e_{\mathrm{sm},j}\,, \tag{14}$$

with $e_{\mathrm{sm},j} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ for all $j \in S$. First, by Def-Prop 18, we have, for some noise $e^{(Dec)}$, with $\|e^{(Dec)}\| \leq B_{\mathrm{C}}$

$$\mathsf{c}[0] + \mathsf{c}[1].\Sigma_{j\in S}\mathsf{sk}_j = \Delta y + e^{(Dec)}\,. \tag{15}$$

Since $\|e_{\mathrm{sm},j}\| \leq B_{sm}$ for all $j \in S$, it follows from the choice of parameters (10) and the final remark in Def-Prop 18, that the output of honest players in $REAL_{\Pi_{\mathrm{C}}}$ is $m := \Omega_{\mathsf{Dec}}(\mu) = y$, which proves our first claim. Second, since we specified $\|e^{(Dec)}\|/N.B_{sm} = negl(\lambda)$ (equation (12)), it follows that the distribution of $\mu$, given by (14) is computationally indistinguishable from the one of $\Delta y + \Sigma_{j\in S} e_{\mathrm{sm},j}$, see the "smudging" Lemma 21 in §H for a further formalization of this fact. But the latter is by definition $\mu^{\mathcal{S}}$, which is exactly the output of $\mathcal{F}_{\mathsf{LSSS}}$ in $\mathsf{Hybrid}_1$. □

$\mathsf{Hybrid}_2$ ***[Random Public Keys].*** This is the same as $\mathsf{Hybrid}_1$ except that the additive contributions $(\mathbf{b}_i, (\mathbf{d}_{0,i}, \mathbf{d}_{2,i}))_{i\in\mathcal{H}}$ of honest players to the public and relinearization keys, are replaced by a sample in $U(R_q^{l\times 3})$. Indistinguishability from $\mathsf{Hybrid}_1$ follows from Corollary 14.

$\mathsf{Hybrid}_3$ ***[Bogus Honest Inputs]*** This is the same as $\mathsf{Hybrid}_2$ except that the input and randomness distribution on behalf of honest owners are computed with $\widetilde{m_\ell} := 0$, instead of with their actual inputs $m_\ell$. Importantly, the behavior of $\mathcal{F}_{\mathsf{LSSS}}$ is unchanged, i.e., correct until (3) included, then outputs $\mu^{\mathcal{S}} := \Delta y + \Sigma_{j \in S} e_{\mathrm{sm},j}$, where $y := \mathrm{C}((m_\ell)_{\ell \in S_c})$ is still the evaluation of the circuit on the *actual* inputs.

We now have that $\mathsf{Hybrid}_3$ and $IDEAL_{\mathcal{F}_{\mathrm{C}},\mathcal{S},\mathscr{E}}$ produce identical views to $\mathscr{E}$. Indeed, the behaviours of $\overline{\mathcal{G}}_{\mathrm{URS}}$, of the simulated ideal functionalities $(\mathcal{F}_{\mathsf{LSSS}}, \mathsf{BC})$, and of the honest parties in $\mathsf{Hybrid}_3$, are identical to the simulation done by $\mathcal{S}$.

***Lemma 23.*** $\mathsf{Hybrid}_2$ *and* $\mathsf{Hybrid}_3$ *are computationally indistinguishable.*

*Proof.* Since $\mathsf{Hybrid}_2$, the secret keys of the honest players ($P_i \in \mathcal{H}$) are no longer used in any computation. Furthermore, since honest players sample their contributions $\mathbf{b}_i$ to the $\mathsf{BFV}$ public key independently (uniformly at random), we can assume without loss of generality that corrupt contributions are generated after having seen the honest ones. We can thus apply Lemma 20 "IND-CPA under Joint Keys", which adapts the one of [AJL+12, Lemma 3.4] in the RLWE setting. It considers a uniform value $\mathbf{b}$ in $R_q$, then the adversary can add to it the sum $(\mathbf{b}', \mathbf{a})$ of $t$ $\mathsf{BFV}$ public keys which it semi-maliciously produces (with the same $\mathbf{a}$). The lemma states that the ciphertext of a chosen message under the sum of keys $(\mathbf{b} + \mathbf{b}', \mathbf{a})$, is still indistinguishable from a uniformly random value. The reduction, from multi-message, to this latter single-message statement, is straightforward. □

**H.3.2 Full details the series of hybrid games** In section §H.3.1, we detailed a simulator $\mathcal{S}$ such that no PPT environment $\mathscr{E}$, which can choose the honest inputs, observe the honest outputs, and fully controls $t$ out of the $N = 2t + 1$ players (via an adversary $\mathcal{A}$), can distinguish between: $(i)$ the real protocol $REAL_{\Pi^{\mathcal{F}_{\mathsf{LSSS}}}}$, where $\mathscr{E}$ interacts with the adversary $\mathcal{A}$, and honest players follow the actual protocol $\Pi^{\mathcal{F}_{\mathsf{LSSS}}}$, and $(ii)$ the ideal protocol $IDEAL_{\mathcal{F}_{\mathrm{C}},\mathcal{S},\mathscr{E}}$, where $\mathscr{E}$ interacts with $\mathcal{S}$, while honest players are connected only to $\mathcal{F}_{\mathrm{C}}$

We now spell out in full details the series of hybrid games used in §H.3.1 to prove the indistinguishability of the real and ideal worlds. The output of each game is the output of the environment. Changes with respect to the previous Hybrid are highlighted in blue. We abuse notations in that when we denote that "players perform some action", we mean implicitely that dishonest players perform it in accordance with a semi-Malicious behavior (§2.2), i.e., send part of, or none, of the messages instructed, and, when sending some, arbitrarily select their random parameters, as long as they are within the essential bounds of the prescribed distributions, i.e., $B$, $B_{sm}$, $B_{Enc}$.

$\mathsf{Hybrid}_1$*:*

- **Setup:** Each player $P_i \in \mathscr{P}$ does the following:
  (0) Send $(\mathsf{Setup})$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  (0) Retrieves $(\mathbf{a}, \mathbf{d}_1)$ from $\overline{\mathcal{G}}_{\mathrm{URS}}$.
- **Input distribution:** Upon $\mathsf{ready}$ from $\mathcal{F}_{\mathsf{LSSS}}$, each input owner $\ell \in \mathscr{L}$:
  (1) Samples $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(Enc)} \xleftarrow{\$} \mathscr{B}_{Enc,q}$ and $e_1^{(Enc)} \xleftarrow{\$} \Psi_q$ and sends $(\mathsf{input}, \{\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}}\}, \{\Delta m_\ell, u_\ell, e_{0,\ell}^{(Enc)}, e_{1,\ell}^{(Enc)}\})$ to $\mathcal{F}_{\mathsf{LSSS}}$. Then it goes offline.
- **Key distribution:** Upon $\mathsf{ready}$ from $\mathcal{F}_{\mathsf{LSSS}}$, each player $P_i$:
  (1) Computes $(\mathsf{sk}_i, (\mathbf{b}_i, \mathbf{a})) \leftarrow \mathsf{BFV.KeyGen}(\mathbf{a})$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$, and sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  (1) Sample $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ and sends $(\mathsf{input}, \overline{e_{\mathrm{sm},i}}, e_{\mathrm{sm},i})$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  (1) Sends $(\mathbf{b}_i, (\mathbf{d}_{0,i}, \mathbf{d}_{2,i}))$ over $\mathsf{BC}^{P_i}$.

As in the protocol, we denote $S_{\mathsf{c}} \subset \mathscr{L}$ the owners, resp. $S \subset \mathscr{P}$ the players, for which no instance returned $\perp$.

- **Threshold encryption (Shrink) and evaluation:** Each player $P_i$:
  (2) $\forall j \in S$, parses the outputs of $\mathsf{BC}^{P_j}$ as $(\mathbf{b}_j, (\mathbf{d}_{0,j}, \mathbf{d}_{2,j}))$.
  (2) Computes $\mathbf{b} = \Sigma_{j\in S}\mathbf{b}_j$ and $\mathbf{rlk} = (\Sigma_{j\in S}\mathbf{d}_{0,j}, \mathbf{d}_1, \Sigma_{j\in S}\mathbf{d}_{2,j})$. Sets $a = \mathbf{a}[0]$ and $b = \mathbf{b}[0]$.
  (2) $\forall \ell \in S_{\mathsf{c}}$, given labels $(\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}})$, a key $\mathsf{ek} = (b, a)$, sends $\left(\mathsf{LCOpen}, \Lambda_{Enc}^{b,a}(\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}})\right)$ to $\mathcal{F}_{\mathsf{LSSS}}$, and obtains a ciphertect $\mathsf{c}_\ell$.
  (3) Computes $\mathsf{c} \leftarrow \mathsf{BFV.Eval}(\mathrm{C}, \{\mathsf{c}_j\}_{j\in S_{\mathsf{c}}}, \mathbf{rlk}, \mathbf{b})$.
  (3) Given labels $(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})$, and a ciphertext $\mathsf{c}$, sends $\left(\mathsf{LCOpen}, \Lambda_{Dec+sm}^{\mathsf{c}}(\overline{\mathsf{sk}}, \overline{e_{sm}})\right)$ to $\mathcal{F}_{\mathsf{LSSS}}$.
- **Output computation:** Denote $y := \mathrm{C}(\{m_\ell\}_{\ell\in S_{\mathsf{c}}})$
  - $\mathcal{F}_{\mathsf{LSSS}}$ delay-outputs $\left(\Lambda_{dec}^{\mathsf{c}},\ \mu^{\mathcal{S}} := \Delta.y + \Sigma_{j\in S}e_{\mathrm{sm},j}\right)$ to all players. Upon receiving it from $\mathcal{F}_{\mathsf{LSSS}}$, players output $m \leftarrow \Omega_{\mathsf{Dec}}(\mu^{\mathcal{S}})$.

$\mathsf{Hybrid}_2$*:*

- **Setup:** Each player $P_i$:
  (0) Send $(\mathsf{Setup})$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  (0) Retrieves $(\mathbf{a}, \mathbf{d}_1)$ from $\overline{\mathcal{G}}_{\mathrm{URS}}$.
- **Input distribution:** Upon ready from $\mathcal{F}_{\mathsf{LSSS}}$, each input owner $\ell \in \mathscr{L}$:
  (1) Samples $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(Enc)} \xleftarrow{\$} \mathscr{B}_{Enc,q}$ and $e_1^{(Enc)} \xleftarrow{\$} \Psi_q$ and sends $(\mathsf{input}, \{\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}}\}, \{\Delta m_\ell, u_\ell, e_{0,\ell}^{(Enc)}, e_{1,\ell}^{(Enc)}\})$ to $\mathcal{F}_{\mathsf{LSSS}}$. Then it goes offline.
- **Key distribution:** Upon ready from $\mathcal{F}_{\mathsf{LSSS}}$, each player $P_i$:
  (1) *If $P_i$ corrupt*, then unchanged instructions: Computes $(\mathsf{sk}_i, (\mathbf{b}_i, \mathbf{a})) \leftarrow \mathsf{BFV.KeyGen.}(\mathbf{a})$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$, and sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  (1) *If $P_i$ honest:* Samples $\mathsf{sk}_i \xleftarrow{\$} \mathscr{X}_q$ and sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSSS}}$. Computes $\mathbf{b}_i \leftarrow U(R_q^l)$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow U(R_q^{2\times l})$.
  (1) Sample $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ and sends $(\mathsf{input}, \overline{e_{\mathrm{sm},i}}, e_{\mathrm{sm},i})$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  (1) Sends $(\mathbf{b}_i, (\mathbf{d}_{0,i}, \mathbf{d}_{2,i}))$ over $\mathsf{BC}^{P_i}$.

As in the protocol, we denote $S_{\mathsf{c}} \subset \mathscr{L}$ the owners, resp. $S \subset \mathscr{P}$ the players, for which no instance returned $\perp$.

- **Threshold encryption (Shrink) and evaluation:** Each player $P_i$:
  (2) $\forall j \in S$, parses the outputs of $\mathsf{BC}^{P_j}$ as $(\mathbf{b}_j, (\mathbf{d}_{0,j}, \mathbf{d}_{2,j}))$.
  (2) Computes $\mathbf{b} = (\Sigma_{j\in S}\mathbf{b}_j)$ and $\mathbf{rlk} = (\Sigma_{j\in S}\mathbf{d}_{0,j}, \mathbf{d}_1, \Sigma_{j\in S}\mathbf{d}_{2,j})$. Sets $a = \mathbf{a}[0]$ and $b = \mathbf{b}[0]$.
  (2) $\forall \ell \in S_{\mathsf{c}}$, given labels $(\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}})$, a key $\mathsf{ek} = (b, a)$, sends $\left(\mathsf{LCOpen}, \Lambda_{Enc}^{b,a}(\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}})\right)$ to $\mathcal{F}_{\mathsf{LSSS}}$, and obtains a ciphertect $\mathsf{c}_\ell$.
  (3) Computes $\mathsf{c} \leftarrow \mathsf{BFV.Eval}(\mathrm{C}, \{\mathsf{c}_j\}_{j\in S_{\mathsf{c}}}, \mathbf{rlk}, \mathbf{b})$.
  (3) Given labels $(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})$, and a ciphertext $\mathsf{c}$, sends $\left(\mathsf{LCOpen}, \Lambda_{Dec+sm}^{\mathsf{c}}(\overline{\mathsf{sk}}, \overline{e_{sm}})\right)$ to $\mathcal{F}_{\mathsf{LSSS}}$.
- **Output computation:** Each player $P_i$:
  - $\mathcal{F}_{\mathsf{LSSS}}$ delay-outputs $\left(\Lambda_{dec}^{\mathsf{c}},\ \mu^{\mathcal{S}} := \Delta.y + \Sigma_{j\in S}e_{\mathrm{sm},j}\right)$ to all players. Upon receiving it from $\mathcal{F}_{\mathsf{LSSS}}$, players output $m \leftarrow \Omega_{\mathsf{Dec}}(\mu^{\mathcal{S}})$.

$\mathsf{Hybrid}_3$*:*

- **Setup:** Each player $P_i$ does the following:
  (0) Send $(\mathsf{Setup})$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  (0) Retrieves $(\mathbf{a}, \mathbf{d}_1)$ from $\overline{\mathcal{G}}_{\mathrm{URS}}$.
- **Input distribution:** Upon $\mathsf{ready}$ from $\mathcal{F}_{\mathsf{LSSS}}$, every input owner $\ell \in \mathscr{L}$:
  (1) *If $\ell$ corrupt*: instructions unchanged.
  (1) *If $\ell$ honest*: Sets $\widetilde{m_\ell} := 0$ and samples $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(Enc)} \xleftarrow{\$} \mathscr{B}_{Enc,q}$ and $e_1^{(Enc)} \xleftarrow{\$} \Psi_q$ and sends $(\mathsf{input}, \{\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}}\}, \{\Delta m_\ell, u_\ell, e_{0,\ell}^{(Enc)}, e_{1,\ell}^{(Enc)}\})$ to $\mathcal{F}_{\mathsf{LSSS}}$.
- **Key distribution:** Upon $\mathsf{ready}$ from $\mathcal{F}_{\mathsf{LSSS}}$, each player $P_i$:
  (1) *If $P_i$ corrupt*, then unchanged instructions: Computes $(\mathsf{sk}_i, (\mathbf{b}_i, \mathbf{a})) \leftarrow \mathsf{BFV.KeyGen}.(\mathbf{a})$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{CDKS}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$, and sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  (1) *If $P_i$ honest:* Samples $\mathsf{sk}_i \leftarrow \mathscr{X}_q$ and sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSSS}}$. Computes $\mathbf{b}_i \leftarrow U(R_q^l)$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow U(R_q^{2\times l})$.
  (1) Sample $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ and sends $(\mathsf{input}, \overline{e_{\mathrm{sm},i}}, e_{\mathrm{sm},i})$ to $\mathcal{F}_{\mathsf{LSSS}}$.
  (1) Sends $(\mathbf{b}_i, (\mathbf{d}_{0,i}, \mathbf{d}_{2,i}))$ over $\mathsf{BC}^{P_i}$.

As in the protocol, we denote $S_{\mathsf{c}} \subset \mathscr{L}$ the owners, resp. $S \subset \mathscr{P}$ the players, for which no instance returned $\perp$.

- **Threshold encryption (Shrink) and evaluation:** Each player $P_i$:
  (2) $\forall j \in S$, parses the outputs of $\mathsf{BC}^{P_j}$ as $(\mathbf{b}_j, (\mathbf{d}_{0,j}, \mathbf{d}_{2,j}))$.
  (2) Computes $\mathbf{b} = \Sigma_{j\in S}\mathbf{b}_j$ and $\mathbf{rlk} = (\Sigma_{j\in S}\mathbf{d}_{0,j}, \mathbf{d}_1, \Sigma_{j\in S}\mathbf{d}_{2,j})$. Sets $a = \mathbf{a}[0]$ and $b = \mathbf{b}[0]$.
  (2) $\forall \ell \in S_{\mathsf{c}}$, given labels $(\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}})$, a key $\mathsf{ek} = (b, a)$, sends $\left(\mathsf{LCOpen}, \Lambda_{Enc}^{b,a}(\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(Enc)}}, \overline{e_{1,\ell}^{(Enc)}})\right)$ to $\mathcal{F}_{\mathsf{LSSS}}$, and obtains a ciphertect $\mathsf{c}_\ell$.
  (3) Computes $\mathsf{c} \leftarrow \mathsf{BFV.Eval}(\mathrm{C}, \{\mathsf{c}_j\}_{j\in S_{\mathsf{c}}}, \mathbf{rlk}, \mathbf{b})$.
  (3) Given labels $(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})$, and a ciphertext $\mathsf{c}$, sends $\left(\mathsf{LCOpen}, \Lambda_{Dec+sm}^{\mathsf{c}}(\overline{\mathsf{sk}}, \overline{e_{sm}})\right)$ to $\mathcal{F}_{\mathsf{LSSS}}$.
- **Output computation:** Each player $P_i$:
  - $\mathcal{F}_{\mathsf{LSSS}}$ delay-outputs $\left(\Lambda_{dec}^{\widetilde{\mathsf{c}}},\ \mu^{\mathcal{S}} := \Delta.y + \Sigma_{j\in S}e_{\mathrm{sm},j}\right)$ to all players. Upon receiving it from $\mathcal{F}_{\mathsf{LSSS}}$, players output $m \leftarrow \Omega_{\mathsf{Dec}}(\mu^{\mathcal{S}})$.

## References for the Appendices.


[ACGJ20] P. Ananth, A. R. Choudhuri, A. Goel, and A. Jain. "Towards Efficiency-Preserving Round Compression in MPC". In: *ASIACRYPT*. 2020.

[AM69] M. F. Atiyah and I. G. MacDonald. *Introduction To Commutative Algebra*. 1969.

[ANRX21] I. Abraham, K. Nayak, L. Ren, and Z. Xiang. "Good-case Latency of Byzantine Broadcast: a Complete Categorization". In: *PODC*. 2021.

[AZ21] P. Ananth and A. J. and Zhengzhong Jin and Giulio Malavolta. "Unbounded Multi-Party Computation from Learning with Errors". In: *EUROCRYPT*. 2021.

[BCG21] E. Boyle, R. Cohen, and A. Goel. "Breaking the $O(\sqrt{n})$-Bit Barrier: Byzantine Agreement with Polylog Bits Per Party". In: *PODC*. 2021.

[BCN18] C. Boschini, J. Camenisch, and G. Neven. "Relaxed lattice-based signatures with short zero-knowledge proofs". In: *International Conference on Information Security*. Springer. 2018, pp. 3–22.

[BH08] Z. Beerliová-Trubíniová and M. Hirt. "Perfectly-Secure MPC with Linear Communication Complexity". In: *Theory of Cryptography*. 2008.

[BHZ21] C. Badertscher, J. Hesse, and V. Zikas. "On the (Ir)Replaceability of Global Setups, or How (Not) to Use a Global Ledger". In: *TCC*. 2021.

[BMMR21] S. Badrinarayanan, P. Miao, P. Mukherjee, and D. Ravi. *On the Round Complexity of Fully Secure Solitary MPC with Honest Majority*. Cryptology ePrint Archive, Report 2021/241. `https://ia.cr/2021/241`. 2021.

[Bor96] M. Borcherding. "Levels of authentication in distributed agreement". In: *WDAG*. 1996.

[BPR11] A. Banerjee, C. Peikert, and A. Rosen. "Pseudorandom Functions and Lattices". In: *EUROCRYPT*. 2011.

[BS20] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. Version 0.5 Jan 2020. 2020.

[Can01] R. Canetti. "Universally composable security: A new paradigm for cryptographic protocols". In: *FOCS*. We refer to eprint 2000/067 version 02/20/2020. 2001.

[Can04] R. Canetti. "Universally Composable Signature, Certification, and Authentication". In: *CSFW*. IEEE Computer Society, 2004, p. 219.

[Can95] R. Canetti. "Studies in Secure Multiparty Computation and Applications". PhD thesis. 1995.

[CD20] I. Cascudo and B. David. "ALBATROSS: Publicly AttestabLe BATched Randomness Based On Secret Sharing". In: *ASIACRYPT*. 2020.

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

[CDPW07] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. "Universally Composable Security with Global Setup". In: *TCC*. 2007.

[CGGM00] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. "Resettable Zero-Knowledge (Extended Abstract)". In: *STOC*. 2000.

[CGHZ16] S. Coretti, J. A. Garay, M. Hirt, and V. Zikas. "Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions". In: *ASIACRYPT*. 2016.

[CKKS17] J. H. Cheon, A. Kim, M. Kim, and Y. Song. "Homomorphic encryption for arithmetic of approximate numbers". In: *ASIACRYPT*. 2017.

[CKR+20] H. Chen, M. Kim, I. P. Razenshteyn, D. Rotaru, Y. Song, and S. Wagh. "Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning". In: *ASIACRYPT*. 2020.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. "Universally Composable Two-Party and Multi-Party Secure Computation". In: *STOC*. 2002.

[Coh16] R. Cohen. "Asynchronous Secure Multiparty Computation in Constant Time". In: *PKC*. 2016.

[DDOPS01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. "Robust non-interactive zero knowledge". In: *CRYPTO 2001*. 2001.

[DGKN09] I. Damgård, M. Geisler, M. Kroigaard, and J. Nielsen. "Asynchronous multiparty computation: Theory and Implementation". In: *PKC*. 2009.

[DGKS21] D. Dachman-Soled, H. Gong, M. Kulkarni, and A. Shahverdi. "Towards a Ring Analogue of the Leftover Hash Lemma". In: *Journal of Mathematical Cryptology* (2021).

[DLS88] C. Dwork, N. Lynch, and L. Stockmeyer. "Consensus in the Presence of Partial Synchrony". In: *J. ACM* (1988).

[FN09] M. Fitzi and J. B. Nielsen. "On the Number of Synchronous Rounds Sufficient for Authenticated Byzantine Agreement". In: *DISC*. 2009.

[GHS12] C. Gentry, S. Halevi, and N. P. Smart. "Fully Homomorphic Encryption with Polylog Overhead". In: *EUROCRYPT*. 2012.

[GJPR21] A. Goel, A. Jain, M. Prabhakaran, and R. Raghunath. "On Communication Models and Best-Achievable Security in Two-Round MPC". In: *TCC*. 2021.

[GKLP16] J. A. Garay, A. Kiayias, N. Leonardos, and G. Panagiotakos. "Bootstrapping the Blockchain, with Applications to Consensus and Fast PKI Setup". In: *PKC*. 2016.

[GKOPZ20] J. Garay, A. Kiayias, R. M. Ostrovsky, G. Panagiotakos, and V. Zikas. "Resource-Restricted Cryptography: Revisiting MPC Bounds in the Proof-of-Work Era". In: *EUROCRYPT*. 2020.

[GOS06] J. Groth, R. Ostrovsky, and A. Sahai. "Perfect Non-interactive Zero Knowledge for NP". In: *EUROCRYPT*. 2006.

[JVC18] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. "GAZELLE: A low latency framework for secure neural network inference". In: *USENIX Security Symposium*. 2018.

[KMQR21] A. Kiayias, C. Moore, S. Quader, and A. Russell. *Efficient Random Beacons with Adaptive Security for Ungrindable Blockchains*. IACR ePrint 2021/1698. `https://ia.cr/2021/1698`. 2021.

[Lam06] L. Lamport. "Lower Bounds for Asynchronous Consensus". In: *Distrib. Comput.* (2006).

[LJLA17] J. Liu, M. Juuti, Y. Lu, and N. Asokan. "Oblivious neural network predictions via minionn transformations". In: *CCS*. 2017.

[LLM+20] C.-D. Liu-Zhang, J. Loss, U. Maurer, T. Moran, and D. Tschudi. "MPC with Synchronous Security and Asynchronous Responsiveness". In: *ASIACRYPT*. 2020.

[LM21] B. Li and D. Micciancio. "On the Security of Homomorphic Encryption on Approximate Numbers". In: *EUROCRYPT*. 2021.

[LPR13b] V. Lyubashevsky, C. Peikert, and O. Regev. "A toolkit for ring-LWE cryptography". In: *EUROCRYPT*. 2013.

[LSP82] L. Lamport, R. E. Shostak, and M. C. Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* (1982).

[MBH23] C. Mouchet, E. Bertrand, and J. Hubaux. "An Efficient Threshold Access-Structure for RLWE-Based Multiparty Homomorphic Encryption". In: *J. Cryptol.* (2023).