Robust Multiparty Computation from Threshold Encryption Based on RLWE

Antoine Urban and Matthieu Rambaud

Télécom Paris and Institut polytechnique de Paris
{antoine.urban,matthieu.rambaud}@telecom-paris.fr

Abstract. We consider protocols for secure multi-party computation (MPC) built from FHE under honest majority, i.e., for $n = 2t + 1$ players of which t are corrupt, that is robust. Surprisingly there exists no robust threshold FHE scheme based on BFV to design an MPC protocol. Precisely, all existing methods for generating a common relinearization key can abort as soon as one player deviates. We solve this issue, with a new relinearization key (adapted from [CDKS19, CCS'19]) which we show how to securely generate in parallel of the threshold encryption key, in the same broadcast. We thus obtain the first robust threshold BFV scheme, moreover using only one broadcast for the generation of keys instead of two previously.
Of independent interest, as an optional alternative, we propose the first threshold FHE decryption enabling simultaneously: (i) robustness over asynchronous channels 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

The generation and use of vast volumes of data to fuel innovative scientific breakthroughs pose a number of challenges in terms of data collection and efficiency. One potential solution lies in the delegation of data processing to public cloud service providers equipped with substantial computing resources. Nonetheless, concerns surrounding the privacy and security of outsourced data and analysis persist. In recent years, there have been notable advancements in cryptographic methods designed to enhance secure computation. Of these techniques, Multiparty Computation (MPC) and Fully Homomorphic Encryption (FHE) have received growing interest due to significant technical breakthroughs.

Threshold* FHE *(*ThFHE*). Fully Homomorphic Encryption (FHE) allows for the execution of arbitrary computations on encrypted data without the need for decryption. Over the years, several generations of FHE schemes have been proposed, with the latest based on the ring-learning-with-errors (RLWE) assumption gaining traction through implementation, and standardization [ACC+21]. Extending these constructions to multiple participants brings up the question of which key to encrypt under. Encrypting inputs under individual keys prevents

homomorphic evaluation, while a single key for all players creates a single point of failure for privacy if compromised. To remove this single point of failure and accommodate a broader range of use cases involving multiple players, threshold FHE (ThFHE) schemes have been developed [AJL+12], in which a secret key is split into a number of shares, so that only a threshold of players collaborating together can decrypt an encrypted secret.

ThFHE-based MPC. In scenarios involving multiple users, ThFHE-based techniques present a promising set of solutions for secure multiparty computation (MPC), where a set of n players collaborate to compute any function on their inputs, while preserving the confidentiality of the latter, due to their minimal communication overhead [AJL+12]. Instantiating an MPC protocol from a ThFHE scheme is not straightforward, and involves multiple steps:

Distributed Key Generation (DKG): a protocol in which the players collaboratively generate a common threshold encryption key ek for a FHE scheme, and where each player also receives a share of the secret key sk.
Input Distribution: players subsequently encrypt their respective inputs using the common threshold encryption key and broadcast the ciphertexts;
Evaluation: players (locally) perform homomorphic computations on the ciphertexts to evaluate the desired function;
Threshold Decryption: players finally jointly execute a threshold decryption protocol using their secret key shares to uncover the computation's output.

Robustness. In the realm of multiparty protocols, an often neglected yet crucial attribute is robustness, specifically referring to the need for a protocol to produce a correct output in a *constant number of rounds* whenever it is executed, even in the presence of malicious behavior. In the context of ThFHE-based MPC, the robust generation of threshold keys is proving challenging, and is our main goal.

1.1 Results

***Main result: the* first robust threshold BFV scheme and robust MPC.** In this work, we construct trBFV, the first robust threshold FHE scheme based on the BFV [Bra12; FV12] cryptosystem, and propose an MPC protocol as informally stated in Theorem 1 below.

Theorem 1 ((Informal) Robust MPC). *Consider $n = 2t + 1$ players, of which t are maliciously corrupt. There exists a robust protocol in 2 broadcasts + 1 asynchronous P2P rounds that UC implements secure evaluation of any arithmetic circuit.*

This result follows from our main contribution detailed in Section 1.1.1.

1.1.1 Robust relinearization key generation. To evaluate a circuit, players first perform a distributed key generation (DKG) protocol to establish a common threshold encryption key, which basically consists of each player P_i sampling a key pair $(\mathsf{sk}_i, \mathsf{ek}_i)$, using a (n, t)-linear secret sharing scheme ((n, t)-LSS, see

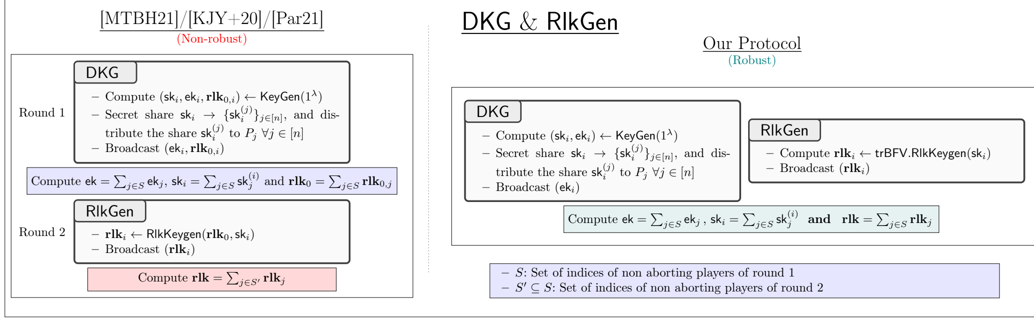


Fig. 1: We present in the left hand side the overall construction of previous DKG&RlkGen protocols [KJY+20; Par21; MTBH21]. First, each player P_i runs KeyGen to produce keys $(\mathsf{sk}_i, \mathsf{ek}_i, \mathbf{rlk}_{0,i})$. The secret key sk_i is secret-shared in n shares $\{\mathsf{sk}_i^{(j)}\}_{j=1}^n$, and each $\mathsf{sk}_i^{(j)}$ is distributed to P_j. The last two elements are broadcast and contributions are added together over the set S of indices of non-aborting players to form the common threshold encryption and intermediate relinearization keys ek and $\mathbf{rlk}_0$, as well as a key share sk_i. Then, players run RlkKeygen with their key sk_i and $\mathbf{rlk}_0$ to produce a contribution $\mathbf{rlk}_i$ that is broadcast. Once added together over the set S' of indices of non-aborting players of this second round, players can compute the relinearization key $\mathbf{rlk} = \sum_{j\in S'} \mathbf{rlk}_j$. On the right hand side, we present a sketch of our protocol. More specifically, to have robustness, players run in parallel KeyGen and our new relinearization key generation algorithm trBFV.RlkKeygen. This is possible because the latter requires only one round.

Definition 2) to divide sk_i into n shares such that only authorized subsets of $t+1$ of them can be used to reconstruct the original key, broadcasting its contribution ek_i, and distributing the shares of sk_i to the players. Then, players can set a threshold encryption key $\mathsf{ek} = \sum_{i\in S} \mathsf{ek}_i$ and secret key shares as the sum over the set S of the indices of players that have correctly broadcast a contribution.

To perform homomorphic computation, BFV (along with other RLWE-based FHE schemes such as CKKS [CKKS17]) requires the generation of an additional common "relinearization key" $\mathbf{rlk}$. For a secret key sk, it is described for BFV [FV12] as being of the following form:

$$\mathbf{rlk} = (\mathsf{sk}^2\mathbf{w} - \mathsf{sk}\cdot\mathbf{r} + \mathbf{e}^{(\mathbf{rlk})}, \mathbf{r}) \tag{1}$$

where $\mathbf{r}$ is an uniform random string, $\mathbf{e}^{(\mathbf{rlk})}$ some noise, and w a decomposition basis of dimension some l, i.e $\mathbf{w} = (w^0, w^1, \ldots, w^{l-1})^T$. Generating this relinearization key in a distributed way proves to be more complex than in the case of the threshold encryption key. Indeed, the presence of the term $\mathsf{sk}^2\mathbf{w}$ introduces a non-linearity. To overcome the challenge posed by the squaring of sk, various RlkGen protocols [KJY+20; Par21; MTBH21] for generating $\mathbf{rlk}$, have been proposed. We briefly discuss these RlkGen protocols to emphasize the novelty of our work. Overall, they have the following informal structure:

- In round 1: each player P_i generates a contribution $\mathbf{rlk}_{0,i}$ using its key sk_i.
- In round 2: each player P_i sums together the contributions $\mathbf{rlk}_0 = \sum_{i\in S} \mathbf{rlk}_{0,i}$, where S denotes the set of indices of non-aborting players in the first round.

Then, each P_i uses an algorithm $\mathsf{RlkKeygen}$ to compute a final contribution $\mathbf{rlk}_i \leftarrow \mathsf{RlkKeygen}(\mathbf{rlk}_0, \mathsf{sk}_i)$ and broadcasts $\mathbf{rlk}_i$.

Finally, the relinearization key is defined as $\mathbf{rlk} = \sum_{i \in S'} \mathbf{rlk}_i$, where S' is the set of indices of non-aborting players in this second round.

However, this generic protocol, illustrated in Fig. 1, has a major drawback, in that it is not robust; if some players take part in some of the rounds, but not all, then no **rlk** is generated. Specifically, [KJY+20; Par21; MTBH21] required S and S' to be equal for RlkGen to output. Otherwise, if the generation were done with non-equal sets S and S', then the resulting **rlk** would be incompatible with the ek produced in the first round as $\mathsf{ek} = \sum_{i \in S} \mathsf{ek}_i$, making the overall key generation non-robust.

We overcome this issue by introducing a new $\mathsf{trBFV.RlkKeygen}$ algorithm for generating an alternative relinearization key adapted from the multikey FHE scheme of [CDKS19], that departs from all previous approaches because it only applies a *linear map* to the secret key sk, *not a squaring*. This allows, as shown in Fig. 1, to design a RlkGen protocol to generate **rlk** in only one round that operates *in parallel* of the DKG, and to obtain a robust overall key generation.

1.1.2 Alternative threshold decryption enabling q power of a small prime.

Most previous threshold FHE schemes used the following *mainstream approach* for threshold decryption. To decrypt a ciphertext c, each player P_i did the following:

- First, it used its secret key share sk_i to compute its "decryption share" c_i of c, and added some locally generated *"smudging noise"* $e_{\mathrm{sm},i}$ to prevent leakage of any secret information, before sending the noisy decryption share $\tilde{\mathsf{c}}_i = \mathsf{c}_i + e_{\mathrm{sm},i}$.
- Second, each player used $t+1$ of the received noisy decryption shares to reconstruct the output.

As a result, when the secret sharing was instantiated with Shamir [Sha79], the smudging noises were multiplied by Lagrange coefficients during reconstruction. As explained in [BGG+18], this leads to use of a large $n!^2$ scaling factor, in order to clear-out the denominators of the Lagrange coefficients. Overall, this imposed the bit-size of the ciphertext modulus q to be $O(n \log n)$, which resulted in a $n\times$ blowup of the ciphertext length. [BGG+18] also required the modulus q to be a prime in order for the multiplied noise to be uniformly distributed modulo q.

A way around both these limitations is proposed in [BGG+18] with another threshold decryption protocol based on a $\{0,1\}$-LSSD scheme[1], which, instead of the Lagrange coefficients used in Shamir, employs binary coefficients to recover the output from the noisy decryption shares. This allows to remove the extra n in the modulus bit-size, i.e. $\log q = O(\log n)$. However, this also leads to a significant space overhead, as the size of each secret key share is at least $O(n^{4.2})$. Cheon et al. [CCK23] introduced a new scheme denoted TreeSSS, but with each share still of size $O(n^{2+o(1)})$.

[1] D stands for "derived" scheme [JRS17], which is stated equivalently as "with strong reconstruction" in [BS23].

Table 1: Threshold FHE schemes for n players, using modulus q. The last column indicates the size of the shares owned by a player, and the modulus-to-noise ratio refers to the ratio between the modulus and decryption noise of a ciphertext.

	LSS Scheme	Simulation Security	Modulus-to noise ratio	Modulus Size $O(\log q)$	Share Size
[BGG+18]	$\{0,1\}$-LSSD	✓	Superpoly	$\mathbf{O(\log n)}$	$O(n^{4.2})$
	Shamir			$O(n \cdot \log n)$	$\mathbf{O(1)}$
[CCK23]	Shamir	✓	Superpoly	$\mathbf{O(\log n)}$	$O(n^{2+o(1)})$
[BS23]	$\{0,1\}$-LSSD	✗	Poly	$\mathbf{O(\log n)}$	$O(n^{4.2})$
Our Scheme	Shamir	✓	Superpoly	$O(n \cdot \log n)$	$\mathbf{O(1)}$
	$\{0,1\}$-LSSD	✓	Superpoly	$\mathbf{O(\log n)}$	$O(n^{4.2})$
	Shamir with pre-shared noise	✓	Superpoly	$\mathbf{O(\log n)}$	$\mathbf{O(1)}$

Although this mainstream approach could be used in our protocol, we now propose an alternative optional approach for threshold decryption. It enables simultaneously (i) a $n!^3\times$ smaller total smudging noise, and (ii) 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* via some distributed protocol. To decrypt c, players use their key shares to perform all-at-once the decryption of c, added with one secret-shared smudging noise that can be obtained in amortized constant overhead. This first ingredient, that allows to remove the $n!^2$ scaling factor, was introduced by [GLS15][2], but was never later used to our knowledge. Second, in order to enable q of small size 2^e, we detail a Shamir sharing over $\mathbb{Z}/2^e\mathbb{Z}$, i.e., embed polynomials into Galois rings extensions [Feh98; ACD+19]. 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 $n!$. In short, our scheme instantiated with Shamir and pre-shared noise achieves a modulus size in $O(\log n)$ while maintaining shares of size $O(1)$, unlike related works as shown in Table 1.

1.1.3 Other threshold schemes with smaller noise but incompatible with MPC.

Some recent works [CSS+22; BS23] address an orthogonal size dependency, by replacing the statistical distance used to analyze the noise during the threshold decryption by the Rényi divergence. In more details, the threshold decryption of a ciphertext c allows recovering the message, but also reveals a small *decryption noise* term that depends on the given ciphertext and the secret key. It is precisely to prevent this leakage that some smudging noise is added to the decryption shares. As shown in Table 1, previous works [BGG+18; CCK23] required that the ratio between the smudging noise and the size of the decryption noise to be superpolynomial in the security parameter. This in turn requires the RLWE problem to be secure with a superpolynomial modulus-to-

[2] With whom we do not compare ourselves since they do not generate a **rlk** key.

noise ratio, which requires larger RLWE parameters. Recently, [CSS+22; BS23] proposed threshold FHE schemes with a polynomial modulus-to-noise ratio, via the Renyi divergence. However, this does not come without its own drawbacks. Importantly, they make clear that their scheme is not usable in MPC, i.e., do not offer composability guarantees. On the contrary, our approach produces a threshold decryption functionality in the simulation paradigm, making it much more versatile for use as a black box in complex protocols.

2 Model

2.1 Notations. All logarithms are in base two. 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 $[n]$. For two vectors $\mathbf{u},\mathbf{v}$ (in bold) we denote $\langle\mathbf{u},\mathbf{v}\rangle$ the dot product and, for a third vector $\mathbf{w}$, we denote $\mathbf{u}\langle\cdot\rangle(\mathbf{v},\mathbf{w}) := \big(\langle\mathbf{u},\mathbf{v}\rangle, \langle\mathbf{u},\mathbf{w}\rangle\big)$. We denote by λ the security parameter throughout the paper.

We consider a positive integer d, denoted the *lattice dimension*; a monic polynomial f of degree d; $k < q$ positive integers denoted plaintext and ciphertext moduli; and $R := \mathbb{Z}[X]/f(X)$. We denote $R_k = R/(k.R)$ and $R_q = R/(q.R)$ the residue rings of R modulo k and q. We denote $\lceil . \rceil, \lfloor . \rfloor, \lfloor . \rceil$ 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. All linear forms are succinctly specified as *linear combinations*, e.g., let $(\overline{x_i})_i$ denote *labels* of some variables $(x_i)_i$, then, $\sum_i l_i\overline{x_i}$ denotes $\big\{(x_i)_i \to \sum_i l_i x_i\big\}$.

2.2 Players 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 consider the Universal Composability (UC) model [Can01] with static corruptions. We consider a PPT machine, denoted as the Environment Env. It fully controls an entity denoted the "adversary" $\mathcal{A}$. At the beginning of the execution, $\mathcal{A}$ may corrupt up to t players of its choice. They behave as arbitrarily instructed by $\mathcal{A}$. 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 Env of every message received by corrupt players and from (simulated) functionalities. For simplicity, we present our protocol in the *semi-malicious* corruption model of [AJL+12], widely adopted since [GLS15].

2.3 Formalizing Eventual Delivery in UC. We now explain the high level idea of the mechanism, denoted fetch-*and*-delay, used to formalize eventual delivery following [KMTZ11; CGHZ16]. Every ideal functionality $\mathcal{F}$, when it needs to eventually deliver (ssid, v) to some entity P, engages in the following interaction. It notifies $\mathcal{A}$ of the output id (ssid), initializes a counter $D_{\mathsf{ssid}} \leftarrow 1$, which captures the delivery delay. Upon receiving (delay) from $\mathcal{A}$, it sets $D_{\mathsf{ssid}} \leftarrow D_{\mathsf{ssid}} + 1$. Upon receiving (fetch) from P, it sets $D_{\mathsf{ssid}} \leftarrow D_{\mathsf{ssid}} - 1$, as well as for all

other counters related to pending outputs for P. In addition, we specify that it leaks (fetch) to $\mathcal{A}$. It is left implicit that entities fetch as much as they can all. Since $\mathcal{A}$ is PPT, at some point it gets exhausted of pressing the button delay. So, after sufficiently many fetches, the counter drops down to 0. Then $\mathcal{F}$ can deliver (ssid, v) to P.

2.4 BC: Broadcast with eventual termination. We formalize in Fig. 5 in Appendix A.1 the ideal functionality of broadcast. It is parametrized by a sender $\mathcal{S}$ and by a set of receivers. It has the following properties: (Termination) all honest receivers eventually output, and (Consistency) any two honest receivers output the same value. Finally, (Validity) if the sender $\mathcal{S}$ is honest and input value x, all honest receivers output the same value x.

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

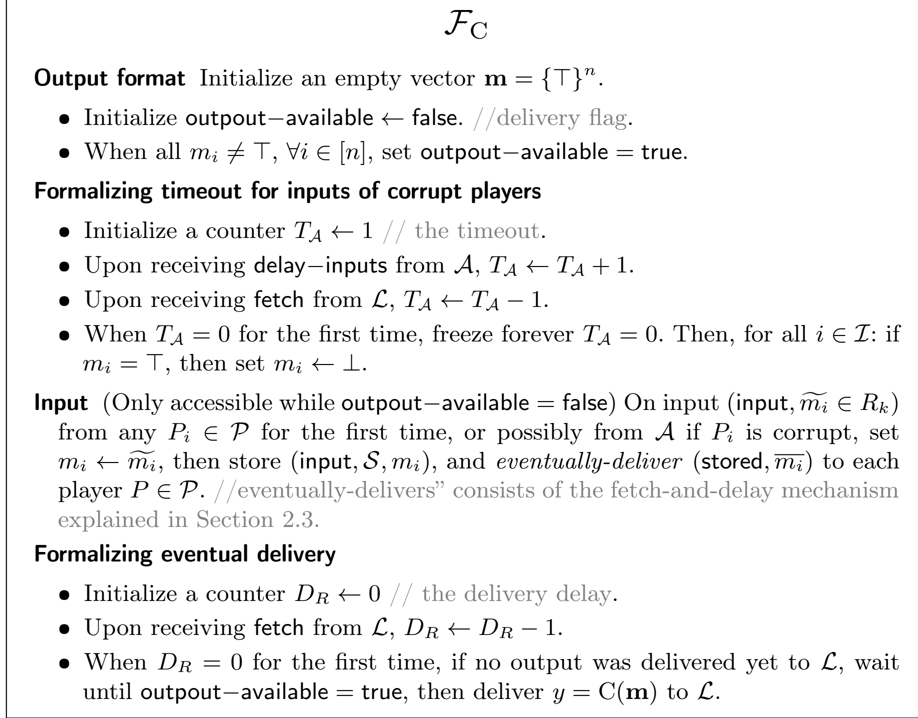
2.6 Bulletin Board PKI: bPKI. We present in Fig. 7 of Appendix A.3 the ideal functionality of a bulletin board of public keys, denoted as bPKI. Upon receiving a key pk_i from any player $P_i \in \mathcal{P}$, it stores (P_i, pk_i) and leaks this information to the adversary $\mathcal{A}$. Then, it *waits until it received a public key from every honest player* in $\mathcal{P}$, and sets a timeout. After it elapsed, sets to $\perp$ the keys of the players which did not give a key, and eventually delivers $\mathsf{pk} \leftarrow (\mathsf{pk}_i)_{i\in[n]}$.

2.7 Global Uniform Random String $\overline{\mathcal{G}}_{\mathbf{URS}}$. It samples uniformly at random a sequence of bits of length κ, denoted URS, then outputs it to all players.

2.8 Ideal Functionality of MPC $\mathcal{F}_{\mathbf{C}}$. The ideal functionality of MPC that we aim to UC implement, is formalized as $\mathcal{F}_{\mathrm{C}}$ in Fig. 2. It returns to an output learner $\mathcal{L}$ the evaluation of an arithmetic circuit $\mathrm{C} : (R_k \cup \{\perp\})^n \rightarrow R_k$ over inputs in R_k. For simplicity: C has n input gates, one single output gate, and $\mathcal{F}_{\mathrm{C}}$ expects one single input from each player, and delivers the output to $\mathcal{L}$.

The functionality works as follows. Upon receiving an input m_i from any player P_i, it stores $(P_i, \overline{m_i})$[3] and leaks this information to $\mathcal{A}$. Before $\mathcal{F}_{\mathrm{C}}$ delivers the output, it needs to wait for the inputs to be submitted. However, the adversary $\mathcal{A}$ can choose to never instruct corrupt players to send their inputs. To remedy this, the functionality i) waits until it receives an input from every honest player, ii) sets a timeout $T_{\mathcal{A}}$, then iii), after it elapsed, sets to $\perp$ the inputs of the (corrupt) players which did not give an input. Once the timeout expires, the output evaluation is delivered following a finite delay chosen by $\mathcal{A}$.

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



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

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

- Initialize outpout−available ← false. //delivery flag.
- When all $m_i \neq \top$, $\forall i \in [n]$, set outpout−available = true.

Formalizing timeout for inputs of corrupt players

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

Input (Only accessible while outpout−available = false) On input (input, $\widetilde{m_i} \in R_k$) from any $P_i \in \mathcal{P}$ for the first time, or possibly from $\mathcal{A}$ if P_i is corrupt, set $m_i \leftarrow \widetilde{m_i}$, then store (input, $\mathcal{S}, m_i$), and *eventually-deliver* (stored, $\overline{m_i}$) to each player $P \in \mathcal{P}$. //eventually-delivers" consists of the fetch-and-delay mechanism explained in Section 2.3.

Formalizing eventual delivery

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

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

3 Cryptographic Ingredients

We now detail the main ingredients needed for the remainder of the paper.

Ring Learning with Errors. Let Ψ_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 $s \leftarrow \mathscr{X}_q$, then any polynomially long sequence of samples in R_q^2 of the form $(a_i, b_i = s \cdot a_i + e_i)_i$, 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.

Gadget Decomposition. For later use in section 4, let us define the widely used, e.g., [GSW13; CDKS19], *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 $\Sigma_{i=0}^{l-1} u_i.g_i = x \pmod q$.

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 $B_1/B_2 = \mathrm{negl}(\lambda)$.

3.1 BFV [FV12]

We now describe the BFV cryptosystem, departing from [FV12; Bra12], by specifying that the key generation algorithm takes a public uniform random string (URS) denoted a as input, whereas in [FV12] a is instead sampled locally. The reason is that, for our DKG to operate (see Section 3.3), some form of additivity will be required between the keys. Let Ψ_q, $\mathscr{B}_{\mathsf{Enc},q}$ and $\mathscr{X}_q$ be distributions over R_q.

- $\mathsf{BFV.KeyGen}(\mathsf{pp} = (a \in R_q))$: Sample $e^{(\mathsf{ek})} \xleftarrow{\$} \Psi_q$ and $\mathsf{sk} \xleftarrow{\$} \mathscr{X}_q$, and output $\mathsf{ek} \leftarrow (-a \cdot \mathsf{sk} + e^{(\mathsf{ek})}, a) = (b, a)$ and sk.
- $\mathsf{BFV.Enc}(\mathsf{ek} = (b, a), m \in R_k)$: Sample the encryption randomnesses $e_0^{(\mathsf{Enc})} \xleftarrow{\$} \mathscr{B}_{\mathsf{Enc},q}$, $e_1^{(\mathsf{Enc})} \xleftarrow{\$} \Psi_q$, and $u \xleftarrow{\$} \mathscr{X}_q$.
Output $\mathsf{c} \leftarrow \left(\Delta m + u \cdot b + e_0^{(\mathsf{Enc})}, u \cdot a + e_1^{(\mathsf{Enc})}\right) \in R_q^2$.
- $\mathsf{BFV.Dec}(\mathsf{sk}, \mathsf{c})$: Given a ciphertext $\mathsf{c} = (\mathsf{c}[0], \mathsf{c}[1]) \in R_q^2$, compute $\mu \leftarrow \mathsf{c}[0] + \mathsf{c}[1] \cdot \mathsf{sk}$. Output $m \leftarrow [\left\lfloor \frac{k}{q}(\mu) \right\rceil]_k := \Omega_{\mathsf{Dec}}(\mu) \in R_k$, where Ω_{Dec} denotes a non-linear decoding function.

We defer to Section 4 the algorithm used for evaluation, since they depend on the chosen *relinearization key*, of which the choice is our main contribution.

Remark 1. In the following sections, we will only use the BFV example, but an analogous construction using CKKS can be obtained with an obvious adaptation.

3.2 Linear Secret Sharing, abstracted-out as ideal $\mathcal{F}_{\mathsf{LSS}}$ functionality

The main ingredient in building a robust threshold FHE scheme is using a (n, t)-linear secret sharing scheme ((n, t)-LSS formally defined in Definition 2), that enables to divide a secret s into n shares, with the property that only authorized subsets of $t + 1$ of them can be used to reconstruct the original secret. Interestingly, thanks to the linear property of the sharing, if secrets $m_1, \ldots, m_n$ have been shared, on input some linear form Λ, one can compute $\Lambda(\{m_i\}_i)$ on the shared inputs. In this section, we abstract this through a functionality $\mathcal{F}_{\mathsf{LSS}}$. A detailed description is available in Appendix B.

3.2.1 Functionality $\mathcal{F}_{\mathsf{LSS}}$. We specify, in Fig. 3, an ideal functionality for LSS, denoted $\mathcal{F}_{\mathsf{LSS}}$. It is parametrized by *i*) a set $\mathcal{P}$ of n players, *ii*) a list $\mathscr{S}$ of entities of the (possibly malicious) senders, where each $\mathcal{S} \in \mathscr{S}$ has a list of inputs: $(x_{\mathcal{S},\alpha})_{\alpha \in X_{\mathcal{S}}}$, identified by input labels $(\overline{x_{\mathcal{S},\alpha}})_{\alpha \in X_{\mathcal{S}}}$. We denote $X_{\mathcal{S}}$ the list of indices α of inputs of sender $\mathcal{S}$. Finally *iii*), we consider an output learner $\mathcal{L}$.

Setup. Before any sender starts interacting with $\mathcal{F}_{\mathsf{LSS}}$, it needs to wait until (Setup, P) is stored $\forall P \in \mathcal{P}$. However, the adversary $\mathcal{A}$ can choose to never instruct corrupt players to setup. To remedy this, we follow the *fetch-and-delay* mechanism explained in Section 2.3 and introduce a timeout $T_{\mathcal{A}}$.

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

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

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

Setup

- On input (Setup) from any $P \in \mathcal{P}$ for the first time, or possibly from $\mathcal{A}$ if P is corrupt: stores (Setup, P), and *eventually-delivers* (Setup, P) to each player $P \in \mathcal{P}$. //eventually-delivers" consists of the same fetch-and-delay mechanism as explained in Section 2.3.
- Initialize a counter $T_\mathcal{A} \leftarrow 1$ //Timing for setup
  - Upon receiving delay−Setup from $\mathcal{A}$, $T_\mathcal{A} \leftarrow T_\mathcal{A} + 1$.
  - Upon receiving fetch from any $P \in \mathcal{P}$, $T_\mathcal{A} \leftarrow T_\mathcal{A} - 1$.
  - When $T_\mathcal{A} = 0$ for the first time, freeze forever $T_\mathcal{A} = 0$. Then, send ready to every $\mathcal{S} \in \mathscr{S}$.

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

$\mathcal{A}$ delaying eventual delivery

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

Bookkeeping requests from honest players

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

LCOpen

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

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

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

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

Fig. 3: Sharing with Linear Combination functionality for a single linear form Λ

Opening. Let HOpeners be a set of players, initially empty. Any player P_i can call LCOpen for some linear form Λ, and is then included in HOpeners. When $|\mathsf{HOpeners}| \geq t+1$, and if $\mathcal{F}_{\mathsf{LSS}}$ has stored all the inputs appearing with nonzero coefficient in Λ, then $\mathcal{F}_{\mathsf{LSS}}$ eventually delivers its evaluation. We denote this mechanism a *collective opening*. Now consider the scenario where one isolated honest player would start the LCOpen protocol, i.e. revealing its share of the evaluation. Since it is hard to prevent t corrupt players from also publicly disclosing consistent shares, this results in the evaluation being publicly opened. We qualify such event as an *early opening*. In practice, we give to $\mathcal{A}$ the power to send an (open−order) to $\mathcal{F}_{\mathsf{LSS}}$, which triggers an immediate delivery of the evaluation to all players, as soon as one honest player requests (LCOpen).

3.2.2 Implementation of $\mathcal{F}_{\mathsf{LSS}}$. Our main goal is to build a protocol Π_{LSS} that implements $\mathcal{F}_{\mathsf{LSS}}$, i.e. that enables, after an unique round of broadcast, players to have a common view on a set of shared secrets. Subsequently, they can perform the 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. Π_{LSS} is detailed in Fig. 13 in Appendix B.4. Overall it can be outlined as follows.

To send a secret s to $\mathcal{F}_{\mathsf{LSS}}$, i.e., to share it, the first step is to generate a (n,t)-linear secret sharing of s. Let $[s^{(i)} : i \in [n]]$ be 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 verifiable secret sharing (PVSS)[4]. To open a linear map Λ over a set of shared secrets $(s_j)_j$: every player P_i decrypts its encrypted shares $(s_j^{(i)})_j$, then evaluates Λ on them. By linearity of the LSS scheme, the result is a partial opening share $z^{(i)}$ of $\Lambda((s_j)_j)$. Then it sends $z^{(i)}$ to all, via asynchronous channels. Finally, from any $t+1$ partial opening shares, the desired linear combination $\Lambda((s_j)_j)$ is efficiently reconstructible.

The main technical challenge we face is that we consider efficient homomorphic encryption schemes in which the ciphertext space is a polynomial ring R_q. In turn, this requires an efficient linear secret sharing scheme over polynomial rings. For this, different options exist, that are further detailed in Appendix B.2:

1. First, one can consider a class of Linear Secret Sharing Schemes, denoted as $\{0,1\}-$LSSD, in which the reconstruction coefficients are always binary. We refer to [JRS17] for an example of construction of such a scheme, which leads to a significant space overhead, as each share is now of size $O(n^{4.3})$.

To remove this overhead, one can naturally think of using Shamir. However, this scheme is instantiated over a field $\mathbb{F}$, as it is based on polynomial interpolation and involves the computation of Lagrange coefficients, that requires inverting elements of the form $\alpha_i - \alpha_j$, where α_i and α_j are public-points. Working over a field guarantees that all non-zero elements are units, hence that these coefficients exist. Our goal in (2) and (3) below is to sketch a variant of this classical case that works over polynomial rings. More details can be found in Appendix B.2.2.

[4]The terminology *verifiable*, is because when compiling to fully malicious security, it should be appended NIZKs of knowledge of plaintexts and of a degree t polynomial. State of the art implementations of PVSS can be found in [GV22].

2. Second, we recall the claim known since [Feh98], that it is possible to construct a Shamir scheme over polynomial rings as long as $\alpha_i - \alpha_j$ is invertible (see Definition 7), which exists when the prime factor q is of size at least $n+1$. We refer to [BS23, §2.2][KJY+20, IV. A] for examples of constructions.
3. Finally, we discuss 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]). We detail in Appendix.B.2.2 $R_{p^e}-\mathsf{Shamir}$, a Shamir scheme variant over Galois extensions of polynomial rings, following [Feh98; ACD+19].

We prove in Proposition 13 that Π_{LSS} (Fig. 13) does UC-implement $\mathcal{F}_{\mathsf{LSS}}$.

3.3 Distributed Key Generation

Our DKG follows the classical pattern of previous DKGs in one broadcast [FS01; BDO23]. Provided with a fixed public URS denoted a as input, each player P_i generates a key pair $(\mathsf{ek}_i, \mathsf{sk}_i)$, sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSS}}$ and broadcasts ek_i. In the second step, players set a threshold key pair without any interaction, as follows. Denote S the set of indices of non-aborting players, i.e., the ones that have broadcast a contribution ek_i and sent an input to $\mathcal{F}_{\mathsf{LSS}}$, then:

$$(\mathsf{ek} = (b, a) = (-\Sigma_{i \in S}\, a.\, \mathsf{sk}_i + e_i^{(\mathsf{ek})}, a)) \; . \tag{2}$$

Thus, the corresponding secret key is defined as $\mathsf{sk} = \Sigma_{i \in S} \mathsf{sk}_i$. Concretely, each player has a share of sk, consisting in the sum over S of its shares of the sk_i. In our formalism, each contribution is accessible via $\mathcal{F}_{\mathsf{LSS}}$. Note that since the adversary sees first the contributions ek_i of honest players, before it decides of the contributions of corrupt players, the obtained key pair $(\mathsf{sk}, \mathsf{ek})$ can be seen as generated by using what [BDO23] formalize as the BiasKeyGen subroutine. In Section 6.2, we will prove that our protocol, as a whole, UC implements MPC. Hence, it can be replaced by the ideal functionality of MPC. Thus the DKG has completely disappeared, and its bias is a no-issue.

4 trBFV, with robust relinearization key generation

In this section, we introduce a new relinearization key and describe its distributed generation in one single robust round. Along the line, we call trBFV the new robust threshold variant of BFV (presented in Section 3.1) obtained with this new ingredient.

In more details, let us recall that in order to perform homomorphic operations, an extra relinearization key denoted **rlk** is needed. The homomorphic multiplication of two BFV ciphertexts $\mathsf{c}_1, \mathsf{c}_2 \in R_q^2$ involves two steps:

(a) The first, denoted *"tensoring"* produces a *degree two* ciphertext consisting of three elements:

$$\hat{\mathsf{c}} = \left\lfloor \frac{k}{q} \mathsf{c}_1 \otimes \mathsf{c}_2 \right\rceil = (\hat{\mathsf{c}}[0], \hat{\mathsf{c}}[1], \hat{\mathsf{c}}[2]) \in R_q^3. \tag{3}$$ [5]

[5] Where $\mathsf{c}_1 \otimes \mathsf{c}_2 = (\mathsf{c}_1[0] \cdot \mathsf{c}_2[0], \mathsf{c}_1[0] \cdot \mathsf{c}_2[1] + \mathsf{c}_1[1] \cdot \mathsf{c}_2[0], \mathsf{c}_1[1] \cdot \mathsf{c}_2[1])$

(b) To reduce the degree back to one, a second step, denoted *relinearization*, must be carried out using **rlk** to turn $\hat{\mathsf{c}}$ into a "regular" BFV ciphertext $\mathsf{c}' = (\mathsf{c}'[0], \mathsf{c}'[1])$ which can be decrypted as the product of the plaintexts.

As previously discussed in Section 1.1.1, existing methods for distributed generation of a relinearization key were not robust. In what follows, we detail in Section 4.1 our new relinearization key and, in Section 4.1.1, its robust, distributed generation. Then, we justify in Section 4.2 simulatability of our relinearization key generation, before detailing in Section 4.3 how to perform homomorphic operations with this new key.

4.1 New Relinearization Key Generation.

Our relinearization algorithm heavily leverages the gadget toolkit introduced in Section 3. Notably, recall that $\mathbf{g}^{-1} : R_q \to R^l$ is a gadget decomposition corresponding to a gadget vector $\mathbf{g} \in R_q^l$. It also makes use of two uniform random strings, that come in the form of two vectors $(\mathbf{a}, \mathbf{d}_1) \in R_q^{2\times l}$, of which $a = \mathbf{a}[0]$ is, as described in Section 3.1, used to generate encryption keys. As a result, all algorithms that took $a \in R_q$ as input variable, are naturally extended to handle input $\mathbf{a} \in R_q^l$ (and notably KeyGen). We can now define the algorithm trBFV.RlkKeygen to generate a relinearization key.

- $(\mathbf{d}_0, \mathbf{d}_2) \in R_q^{2\times l} \leftarrow \mathsf{trBFV.RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk})$:
  - Sample $r \leftarrow \mathscr{X}_q$.
  - Sample $\mathbf{e}_0^{(\mathbf{rlk})} \leftarrow \Psi_q^l$, and set $\mathbf{d}_0 = -\mathsf{sk} \cdot \mathbf{d}_1 + \mathbf{e}_0^{(\mathbf{rlk})} + r \cdot \mathbf{g}$
  - Sample $\mathbf{e}_2^{(\mathbf{rlk})} \leftarrow \Psi_q^l$ and set $\mathbf{d}_2 = r \cdot \mathbf{a} + \mathbf{e}_2^{(\mathbf{rlk})} + \mathsf{sk} \cdot \mathbf{g}$

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

Interestingly, the overall algorithm to generate **rlk** is linear over the secret key sk, unlike previous ones [KJY+20; Par21; MTBH21].

4.1.1 Distributed Relinearization Key Generation.

To distributively generate a common **rlk**, one can leverage the additional linearity. In short, to build a RlkGen protocol, we let each player P_i compute an additive contribution to the relinearization key $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{trBFV.RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$ and broadcast it. From the set S of indices of players who have correctly broadcast their additive contributions to the relinearization key and to the threshold encryption key ek as in Equation (2), one can then compute:

$$\mathbf{rlk} := (\Sigma_{i\in S}\mathbf{d}_{0,i}, \mathbf{d}_1, \Sigma_{i\in S}\mathbf{d}_{2,i}) \tag{4}$$

4.2 Construction Details & Security

We now justify simulatability of our new **rlk** introduced in Section 4.1, firstly by giving the reasoning behind its construction, before proving security in Section 4.2.1.

Intuition. The intuitive rationale is that our $\mathbf{rlk}$ is none other than a particular case of an existing relinearization key! Indeed, the recent work of [CDKS19] proposed a n-out-of-n *multi*-key FHE scheme from BFV, and, therefore an algorithm to generate relinearization keys, which operate on multi-key ciphertexts. In this setting, a multi-key ciphertext associated to n players is of the form $\mathbf{c} = (\mathsf{c}_1, \mathsf{c}_2, \ldots, \mathsf{c}_n)$, is decryptable by the concatenated secret key $\mathbf{s} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, and relinearizable using the concatenated n relinearization keys $\{\mathbf{rlk}_i\}_{i\in[n]}$. Interestingly, we observe that if we only consider the particular single-key case (i.e. $n = 1$), we obtain exactly our $\mathbf{rlk}$ presented in Section 4.1! In the remainder of this section, we discuss this particularization in greater detail.

Detailed Construction Explanation: From multikey to single-key relinearization key. Let us slightly abuse future notation and denote $\mathsf{RlkKeygen}$ the linear map used in [CDKS19] to produce a relinearization key. In our context, the secret key sk comes as a sum $\sum_i \mathsf{sk}_i$ of (secret shared) contributions sk_i from non-aborting players, so we roughly need a robust protocol which generates $\mathbf{rlk} := \mathsf{RlkKeygen}(\mathsf{sk})$. This hints towards the blueprint of our robust distributed solution to generate $\mathbf{rlk}$: *in parallel* of linearly secret-sharing its contribution sk_i to the secret key, each player P_i broadcasts the corresponding contribution: $\mathbf{rlk}_i = (\mathbf{d}_{0,i}, \mathbf{d}_{1,i}, \mathbf{d}_{2,i}) = \mathsf{RlkKeygen}(\mathsf{sk}_i)$ to the relinearization key. Then, after it computed the threshold encryption key ek as in Equation (2), each player sets $\mathbf{rlk} = \sum_{i\in S} \mathbf{rlk}_i$, where S is the *same* set, i.e., of indices of players not aborting in the first round, as the one used to set ek. However, one hurdle remains in that in the linear map $\mathsf{RlkKeygen}$ defined in [CDKS19], the coefficient of sk_i, denoted $\mathbf{d}_{1,i}$, actually *depends* on the player making the contribution, since $\mathbf{d}_{1,i}$ is sampled by P_i. Hence, this prevents additivity between contributions from different players (in the setting of [CDKS19], no additivity was needed).

To solve this, we specify instead that $\mathbf{d}_1$ is in common, and given by an uniform random string (URS). The reason why fixing a common $\mathbf{d}_1$ does actually not degrade the security of the distributed protocol, compared to [CDKS19] is that $\mathbf{d}_1$, by definition, appears in clear in the public relinearization key. More particularly, in the proof of Corollary 2, we will show a reduction from the pseudorandomness of our threshold $\mathbf{rlk}$, into the pseudorandomness of a single-key $\mathbf{rlk}$, with loss only *linear* in n. To give an intuition, a toy model of our reduction is just the well-known reduction from the security of our DKG, into the security of RLWE. In this toy model, what the adversary sees are n samples $(a,\ a \cdot \mathsf{sk}_i + e_i^{(\mathsf{ek})})_{i\in[n]}$, all with the same public uniform randomness a but with *different independently sampled* secrets sk_i. So the idea is that the reduction to RLWE, upon receiving *one* RLWE challenge sample: $(a,\ a \cdot \mathsf{sk}_n + e_n^{(\mathsf{ek})})$, simply generates itself $n-1$ other challenges: $(a,\ a \cdot \mathsf{sk}_i + e_i^{(\mathsf{ek})})_{i\in[n-1]}$ with the same a, and handles them to our adversary.

4.2.1 Security.

In Corollary 2, we prove that, despite our specification of a common $\mathbf{d}_1$, the concatenation of all the honestly generated contributions $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{trBFV.RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$ to the common relinearization key, as

well as the contributions $\mathbf{ek}_i = (\mathbf{b}_i, \mathbf{a})$ to the threshold encryption key, is indistinguishable from a large uniform random string, under the *same* circular security assumption as implicitly made in [CDKS19] and detailed in Appendix C.2.

Consider a public sampling of a uniform string $(\mathbf{a}, \mathbf{d}_1) \in U(R_q^{l\times 2})$, and a polynomial number M of independent machines. Each of them generates a key pair $(\mathsf{sk}_m, \mathsf{ek}_m)$ by using KeyGen, all using the common public $\mathbf{a}$. Each machine m generates $(\mathbf{d}_{0,m}, \mathbf{d}_{2,m}) \leftarrow \mathsf{trBFV.RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_m)$. 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 2 (Security with Common Public Randomness). *Consider:*

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

Then the maximum distinguishing advantage $\mathsf{Adv}^{\lambda}_{\mathcal{D}_0^M}$ 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 $M\mathsf{Adv}^{\lambda}_{\mathcal{D}_0}$.

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 $\mathsf{Adv}_{\mathcal{D}_0}$, as a straightforward reduction shows (cf Appendix C.2). □

4.3 Homomorphic Evaluation of a circuit

We can now augment the definition presented in Section 3.1 with homomorphic operations. Consider two BFV ciphertexts $\mathsf{c}_1, \mathsf{c}_2 \in R_q^2$ and keys $\mathsf{ek} = (\mathbf{b}, \mathbf{a})$ and $\mathbf{rlk}$, then we have:

- (**Addition**) $\mathsf{trBFV.Add}(\mathsf{c}_1, \mathsf{c}_2)$: Return $\mathsf{c} = \mathsf{c}_1 + \mathsf{c}_2 \in R_q^2$.
- (**Multiplication**) $\mathsf{trBFV.Mult}(\mathsf{c}_1, \mathsf{c}_2, \mathbf{rlk}, \mathbf{b})$: Compute $\hat{\mathsf{c}} = \left\lfloor \frac{k}{q} \mathsf{c}_1 \otimes \mathsf{c}_2 \right\rceil \in R_q^3$ and return $\mathsf{c}' \leftarrow \mathsf{Relin}(\hat{\mathsf{c}}, \mathbf{rlk}, \mathbf{b})$ (cf Algorithm 1).
- $\mathsf{trBFV.Eval}(\mathrm{C}, (\mathsf{c}_i \in R_q^2)_{i\in[n]}), \mathbf{rlk}, \mathbf{b})$, for a circuit C with n input gates, return the evaluation obtained by applying $\mathsf{trBFV.Add}$ and $\mathsf{trBFV.Mult}$ gate by gate, with inputs the $(\mathsf{c}_i)_{i\in[n]}$.

New Relinearization. We now present our new relinearization algorithm.

Algorithm 1 Relin

Input: $\hat{\mathsf{c}} = (\hat{\mathsf{c}}[0], \hat{\mathsf{c}}[1], \hat{\mathsf{c}}[2]) \in R_q^3, \mathbf{rlk} = [\mathbf{d}_0|\mathbf{d}_1|\mathbf{d}_2] \in (R_q^l)^3, \mathbf{b} \in R_q^l$
Output $\mathsf{c}' = (\mathsf{c}'_0, \mathsf{c}'_1) \in R_q^2$
1: $\mathsf{c}'_0 \leftarrow \hat{\mathsf{c}}[0]$
2: $\mathsf{c}'_1 \leftarrow \hat{\mathsf{c}}[1]$
3: $\mathsf{c}'_2 \leftarrow \langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathbf{b} \rangle$
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: $\mathsf{c}'_1 \leftarrow \mathsf{c}'_1 + \langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathbf{d}_2 \rangle$

Correctness. Correctness follows from the proof of [CDKS19] adapted to our single-key context. In a nutshell, we have:

$$\mathbf{g}^{-1}(\mathsf{c}'_2) \langle\cdot\rangle (\mathbf{d}_0, \mathbf{d}_1) \langle\cdot\rangle (1, \mathsf{sk}) \approx r \cdot \mathsf{c}'_2 \text{ and } \left\langle \mathbf{g}^{-1}(\hat{\mathsf{c}}[2]), \mathbf{d}_2 \right\rangle \cdot \mathsf{sk} \approx -r \cdot \mathsf{c}'_2 + \hat{\mathsf{c}}[2] \cdot \mathsf{sk}^2$$

and thus,

$$\mathsf{c}'_0 + \mathsf{c}'_1 \mathsf{sk} \approx \hat{\mathsf{c}}[0] + \hat{\mathsf{c}}[1]\mathsf{sk} + \hat{\mathsf{c}}[2]\mathsf{sk}^2$$

We refer to Appendix C for further details about trBFV and for a complete noise analysis.

5 Threshold decryption

Recall from Section 3.1 that the decryption can be seen as a two steps process: (i) first the interactive opening of a linear map defined for BFV as,

$$\Lambda^{\mathsf{c}}_{\mathsf{Dec}} : \mathsf{sk} \to \mathsf{c}[0] + \mathsf{c}[1] \cdot \mathsf{sk}, \tag{5}$$

applied to the (secret shared) secret key sk, with public coefficients equal to the ciphertext c, (ii) followed by the local computation of a non-linear decoding function Ω_{Dec}. However, a direct adaptation from this decryption to the threshold setting is not trivial, when Ω_{Dec} is nontrivial, as the case in fully homomorphic encryption. Indeed, the output μ of (i) allows recovering the plaintext, but also reveals a small *decryption noise* term that depends on the given ciphertext and the secret key, as defined below:

Definition 3 (Decryption noise). *Let $\mathsf{c} \in \mathscr{C}$, $m \in \mathscr{M}$ and $\mathsf{sk} \in \mathscr{X}_q$. We define the "decryption noise" as $e^{(\mathsf{Dec})}(\mathsf{c}, \mathsf{sk}, m) := \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk}) - \Delta \cdot m$.*

Asharov et al. [AJL+12] demonstrated that the noisy output μ of (i) reveals too much information about the secret key. Thus, to prevent any information leakage about the secret key, [AJL+12] introduced the technique of adding additional noise to μ before it can be reconstructed. This "smudging" noise e_{sm} is, roughly, sampled uniformly in some large enough interval $[-B_{sm}, B_{sm}]$. Now consider an arithmetic circuit C, and denote B_{C} the upper-bound on the decryption noise of a ciphertext after evaluation of circuit C. The choice of B_{sm} is crucial to both the security and correctness of our MPC protocol. This translates into the following two requirements:

(a) First, the output of (i) $\mu = \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk})$ must be statistically close enough to the (scaled) plaintext circuit evaluation $\Delta.y$. Then, there should exist some level of noise B_{sm}, so that adding an uniform noise $e_{\mathrm{sm}} \in [-B_{sm}, B_{sm}]$ to both μ and y, makes them indistinguishable, while leaving correct the result: $y = \Omega_{\mathsf{Dec}}(\mu + e_{\mathrm{sm}})$. As stated in Lemma 25, the indistinguishability requirement imposes a level of noise high enough so that $B_{\mathrm{C}}/B_{sm} \leqslant \mathrm{negl}(\lambda)$ (Equation (8)).

(b) Second, the correctness requirement imposes that B_{C} added with this smudging noise stays small, i.e. we want that:

$$B_{\mathrm{C}} + n \cdot B_{sm} \leq \Delta/2, \tag{6}$$

as further explained in Appendix C.5.

In Sections 5.0.1 and 5.0.2, we give two methods for opening μ added with such noise, that are illustrated in Fig. 14.

5.0.1 Mainstream Threshold Decryption Method. The first protocol 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 c, which it sends. The reason for multiplying by $n!^2$ is to clear-out the denominators of the Lagrange coefficients applied at reconstruction (see [BGG+18]). 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 $B_{\mathrm{C}} + n.n!^3 \cdot B_{sm} < \Delta/2$ (for correctness of Lagrange reconstruction-then-rounding, or $q/4$ instead of $\Delta/2$ in their instantiation with GSW). In a nutshell, this method introduces an overhead of $n.n!^3$ on the ciphertext modulus q. This results in a $n\times$ blowup of the ciphertext length.

5.0.2 Improved Threshold Decryption Method. To keep the ciphertext size small, the second method, follows a forgotten approach, which we credit to [GLS15]. Players do not anymore blur their opening share of μ. Instead, they now reconstruct all at once the sum of μ and a common shared noise e_{sm}, i.e., they open the linear map defined as:

$$\Lambda^{\mathsf{c}}_{\mathsf{Dec}+sm} : (\mathsf{sk}, e_{\mathrm{sm}}) \to \mathsf{c}[0] + \mathsf{c}[1] \cdot \mathsf{sk} + e_{\mathrm{sm}} \tag{7}$$

The distributed generation of the noise is simply done 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 $B_{\mathrm{C}}+n.B_{sm} < \Delta/2$ (Eq. 6). Hence, the ciphertext expansion factor Δ has dependency in n which is only linear, instead of $n.n!^3$ in the previous method. Since the noise can be used only for one threshold decryption, players must precompute as much noises as many circuits to be subsequently evaluated. We will formalize this simple Distributed Noise Generation protocol 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 PVSS 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: $e_{\mathrm{sm}} = \sum_{i \in S} e_{\mathrm{sm},i}$.

6 MPC protocol

In Fig. 4, we formalize our trBFV scheme as an end-to-end MPC protocol in 2 broadcasts + one asynchronous P2P round, called $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$, assuming a bulletin board PKI and an URS. For simplicity, we describe and prove it in the $\mathcal{F}_{\mathsf{LSS}}$-hybrid model.

6.1 Protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ instantiated from trBFV

We instantiate protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ from trBFV. Notably, we use the alternative version of relinearization key generation presented in Section 4. Moreover, for security and correctness, we require Equation (6); and:

$$\frac{B_{\mathrm{C}}}{B_{sm}} = \mathrm{negl}(\lambda) \text{ and } \frac{2dnB}{B_{\mathsf{Enc}}} = \mathrm{negl}(\lambda) \ . \tag{8}$$

where B_{C} is a bound on the noise of a ciphertext after evaluation of circuit C (Eq.30) and B, B_{Enc} bounds on the encryption randomnesses (see Appendix C.3). Note that following Section 5, two possibilities exist for threshold decryption: (i) either by using the mainstream threshold decryption method (Section 5.0.1), that does not require pre-shared noises, (ii) or by using the second improved method (Section 5.0.2), in which players distributively generate a pre-shared noise in parallel with the DKG. We present the latter in Fig. 4.

6.2 Proofs of Theorem 1

By Proposition 13, Π_{LSS} UC implements $\mathcal{F}_{\mathsf{LSS}}$. Thus, the following Theorem 4 implies Theorem 1.

Theorem 4. *$\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ implemented from* trBFV *UC implements the ideal functionality $\mathcal{F}_{\mathrm{C}}$ for any semi-malicious adversary, in the $(\mathcal{F}_{\mathsf{LSS}}, \mathsf{BC})$-hybrid model with external resource $\overline{\mathcal{G}}_{URS}$.*

6.2.1 Description of the Simulator Sim of $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$. To prove Theorem 4, we describe a simulator Sim of $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$, that initiates in its head, a set of n players and may initially receive corruption requests from Env for up to t players, indexed by $\mathcal{I} \subset [n]$. It simulates functionalities $\mathsf{BC}, \mathcal{F}_{\mathsf{LSS}}$ following a correct behavior, apart from the value returned by $\mathcal{F}_{\mathsf{LSS}}$ during the threshold decryption. Upon every output from a simulated functionality to a simulated corrupt player, or, upon every message from a simulated functionality to the simulated $\mathcal{A}$, Sim instantly forwards it to Env, as would have done the actual $\mathcal{A}$.

Intuition. We now convey the main ideas of Sim by describing it via a sequence of incremental changes, starting from a real execution. In the last hybrid, the view of Env is generated solely by interaction with $\mathcal{F}_{\mathrm{C}}$, hence what we are describing is a simulator. The full details about Sim and the proofs are in Appendix D.

First, in Hybrid_1, we simulate decryption by modifying the behavior of $\mathcal{F}_{\mathsf{LSS}}$ in the threshold decryption. There it, incorrectly, outputs $\mu^{\mathsf{Sim}} := \Delta y + \Sigma_{j \in S} e_{\mathrm{sm},j}$,

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

Participants: n players $P_1, \ldots, P_n$, each with input m_i.
Bulletin-board PKI setup and CRS setup. Each P_i:

- Sends (Setup) to $\mathcal{F}_{\mathsf{LSS}}$ and obtains common URSs $(\mathbf{a}, \mathbf{d}_1) \leftarrow \overline{\mathcal{G}}_{\mathrm{URS}}$.

(1) **Interactive setup in one step of all-to-all broadcasts.**
Upon ready from $\mathcal{F}_{\mathsf{LSS}}$, each P_i:

- *Distributed Keys Generation - broadcasts:*
  - Computes $(\mathsf{sk}_i, (\mathbf{b}_i, \mathbf{a})) \leftarrow \mathsf{BFV.KeyGen}(\mathbf{a})$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{trBFV.RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$.
  - Sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSS}}$ and broadcasts $(\mathbf{b}_i, (\mathbf{d}_{0,i}, \mathbf{d}_{2,i}))$.
- *Distributed Smudging Noises Generation (in parallel of* DKG*):*
  - Samples $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ and sends $(\mathsf{input}, \overline{e_{\mathrm{sm},i}}, e_{\mathrm{sm},i})$ to $\mathcal{F}_{\mathsf{LSS}}$.

(1) **Formation of threshold keys (local):** Each player:

- *Reception of broadcasts:* Initializes an empty list $S \leftarrow \{\}$, and $\forall j \in [n]$, check if the data received from the broadcast of P_j parses as: $(\mathbf{b}_j, (\mathbf{d}_{0,j}, \mathbf{d}_{2,j}))$ then add j to S. Also, if the distributed smudging noise generation was activated, further check if $\mathcal{F}_{\mathsf{LSS}}$ did notify $(\mathsf{stored}, \overline{e_{\mathrm{sm},j}})$ (else, do not add j to S).
- *Adding the contributions of non-aborting players*: Computes

$$\mathbf{b} = \Sigma_{j \in S} \mathbf{b}_j \ , \tag{9}$$

sets the threshold keys: $\mathsf{ek} = (b = \mathbf{b}[0], a = \mathbf{a}[0])$ and $\mathsf{sk} = \Sigma_{i \in S} \mathsf{sk}_i$, and the noise $e_{\mathrm{sm}} = \Sigma_{i \in S} e_{\mathrm{sm},i}$ //accessible through $\mathcal{F}_{\mathsf{LSS}}$, via the labels $\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}}$, and:

$$\mathbf{rlk} = (\Sigma_{j \in S} \mathbf{d}_{0,j}, \mathbf{d}_1, \Sigma_{j \in S} \mathbf{d}_{2,j}) \tag{10}$$

(2) **Broadcast of encrypted inputs:** Each P_i:

- Samples $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(\mathsf{Enc})} \xleftarrow{\$} \mathscr{B}_{\mathsf{Enc},q}$ and $e_1^{(\mathsf{Enc})} \xleftarrow{\$} \Psi_q$. Computes $\mathsf{c}_i = (\Delta m_i + u \cdot b + e_0^{(\mathsf{Enc})}, u \cdot a + e_1^{(\mathsf{Enc})})$ then broadcasts it.

(2) **Evaluation (local):** Each P_i sets $S_{\mathsf{c}} \subset [n]$ the subset of indices of players from which it received a ciphertext c_j. Then it computes $\mathsf{c} \leftarrow \mathsf{trBFV.Eval}(\mathrm{C}, \{\mathsf{c}_j\}_{j \in S_{\mathsf{c}}}, \mathbf{rlk}, \mathbf{b})^a$.

(3) **Threshold Decryption:** Each P_i:

- Given labels $(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})$ and c, sends $\big(\mathsf{LCOpen}, \Lambda^{\mathsf{c}}_{\mathsf{Dec}+sm}(\overline{\mathsf{sk}}, \overline{e_{sm}})\big)$ to $\mathcal{F}_{\mathsf{LSS}}$;
- Upon receiving $(\Lambda^{\mathsf{c}}_{\mathsf{Dec}+sm}, \mu)$ from $\mathcal{F}_{\mathsf{LSS}}$, outputs $y := \Omega_{\mathsf{Dec}}(\mu)$.

[a] Without loss of generality, C sets to $\bot$ the non received inputs in $[n] \setminus S_{\mathsf{c}}$.

Fig. 4: MPC Protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$

where $y := \mathrm{C}((m_\ell)_{\ell \in S_\mathsf{c}})$ is the evaluation of the circuit on the actual inputs. Indistinguishability follows from the "smudging Lemma" (see Lemma 25).

Then, in 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 keys, are replaced by a sample in $U(R_q^{l \times 3})$. Indistinguishability from Hybrid_1 follows from Corollary 2.

Finally, in Hybrid_3, we replace the actual inputs m_ℓ of simulated honest players by $\widetilde{m_\ell} := 0$. Importantly, the behavior of $\mathcal{F}_{\mathsf{LSS}}$ is unchanged, i.e., correct until (3) included, then outputs $\mu^{\mathsf{Sim}} := \Delta y + \Sigma_{j \in S} e_{\mathrm{sm},j}$, where $y := \mathrm{C}((m_\ell)_{\ell \in S_\mathsf{c}})$ is still the evaluation of the circuit on the *actual* inputs. Thanks to the modifications so far, we can apply Lemma 24 "IND-CPA under Joint Keys", which adapts the one of [AJL+12, Lemma 3.4] in the RLWE setting, and argue that ciphertexts of chosen plaintexts are indistinguishable from random strings.

References

[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.

[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.

[BDO23] L. Braun, I. Damgård, and C. Orlandi. "Secure multiparty computation from threshold encryption based on class groups". In: *CRYPTO.* Springer. 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.

[Bra12] Z. Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *Advances in Cryptology – CRYPTO.* Ed. by R. Safavi-Naini and R. Canetti. 2012.

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

[BS23] K. Boudgoust and P. Scholl. "Simple Threshold (Fully Homomorphic) Encryption from LWE with Polynomial Modulus". In: *Advances in Cryptology - ASIACRYPT 2023.* 2023.

[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.

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

[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.

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

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

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

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

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

[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.

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

[FLL21] M. Fitzi, C.-D. Liu-Zhang, and J. Loss. "A New Way to Achieve Round-Efficient Byzantine Agreement". In: *PODC*. 2021.

[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).

[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.

[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.

[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).

[KMTZ11] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. “Universally Composable Synchronous Computation”. In: *TCC*. 2011.

[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.

[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).

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

[Sha79] A. Shamir. “How to share a secret”. In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782.

Supplementary Material

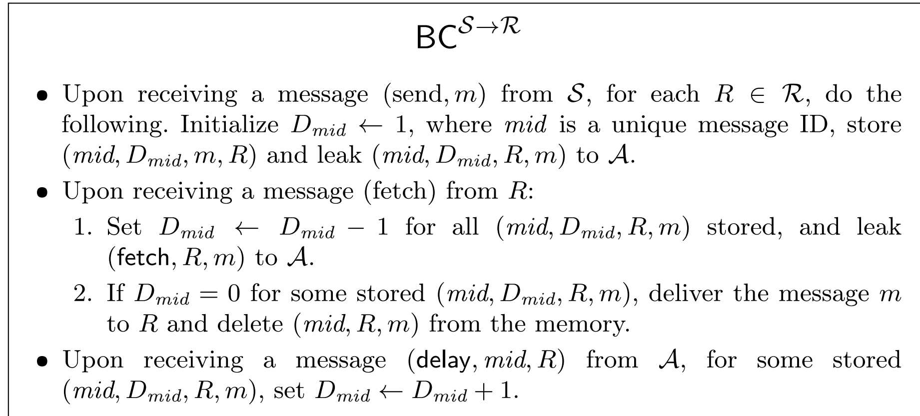
A Model: Further Formalism and Discussion

A.1 More on Broadcast BC

Definition 1. *A broadcast protocol [FLL21, Definition 1] involves a sender $\mathcal{S}$ and a set of receivers $\mathcal{R}$. It requires the following properties:*

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

We dub it as BC and formalize it in Fig. 5. Overall, it simply proceeds as follows. On receiving a message s from the sender $\mathcal{S}$, it sends s to each receiver $R \in \mathcal{R}$ by using the fetch-and-delay mechanism introduced in Section 2.3.



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

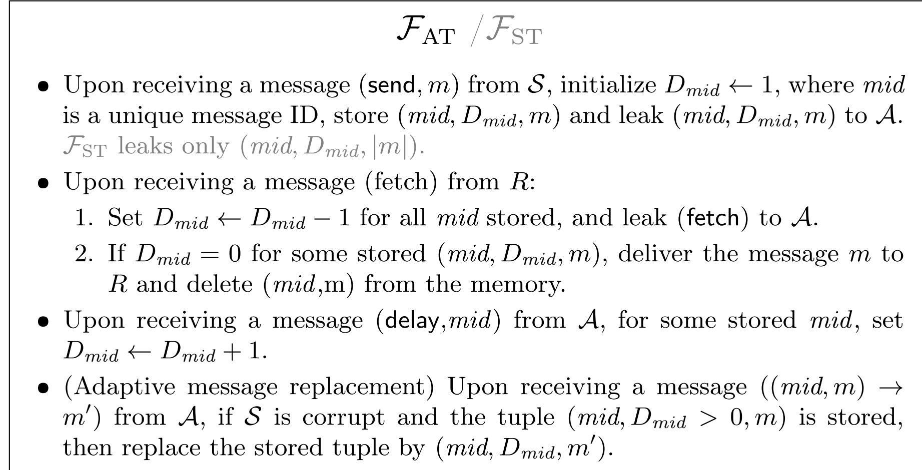
- Upon receiving a message (send, m) from $\mathcal{S}$, for each $R \in \mathcal{R}$, do the following. Initialize $D_{mid} \leftarrow 1$, where mid is a unique message ID, store (mid, D_{mid}, m, R) and leak (mid, D_{mid}, R, m) to $\mathcal{A}$.
- Upon receiving a message (fetch) from R:
  1. Set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

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

A.2 $\mathcal{F}_{\mathbf{AT}}$

In Fig. 6, we present our Authenticated Message Transmitting functionality $\mathcal{F}_{\text{AT}}$. Our baseline for $\mathcal{F}_{\text{ST}}$ is the functionality denoted $\mathcal{F}_{\text{ed-smt}}$ [KMTZ11]. For $\mathcal{F}_{\text{AT}}$, we made the addition to leak the contents of the messages to $\mathcal{A}$. We also incorporated two other additions, borrowed from the $\mathcal{F}_{\text{NET}}$ in [LLM+20]. The first consists in attaching a unique identifier to each message and counter, in order to give to $\mathcal{A}$ a control on the delay of each message individually. Notice that [CGHZ16] model this individual control by, instead, given the power to $\mathcal{A}$

to re-order messages not delivered yet. The second addition consists in forcing explicitely $\mathcal{A}$ to press (delay) to augment the delay by +1, instead of the (equivalent) formalization in which $\mathcal{A}$ enters the additional delay in unary notation.



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

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

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

A.3 Bulletin board PKI: bPKI

In Fig. 7, we present our bulletin board PKI functionality bPKI.

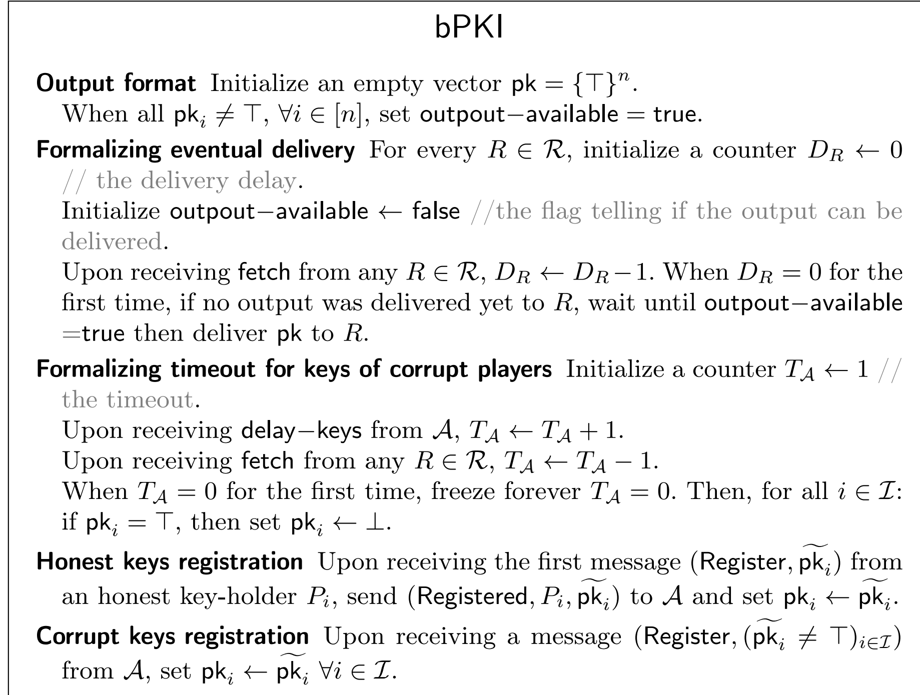
A.4 $\overline{\mathcal{G}}_{\text{URS}}$

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

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

In this section, we detail the instantiation of $\mathcal{F}_{\mathsf{LSS}}$ and the different challenges we face.

1. We first define in Appendix B.1, what is a (n, t)-Linear Secret Sharing scheme (LSS) and how it can be used to design a Publicly Verifiable Secret Sharing scheme (PVSS).
2. We then detail in Appendix B.2, how to implement a LSS scheme over polynomial rings.
3. We later prove in Appendix B.3, that a PVSS scheme is IND-CPA.
4. Finally in Appendix B.4, we detail the implementation of $\mathcal{F}_{\mathsf{LSS}}$ and its security.



bPKI

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

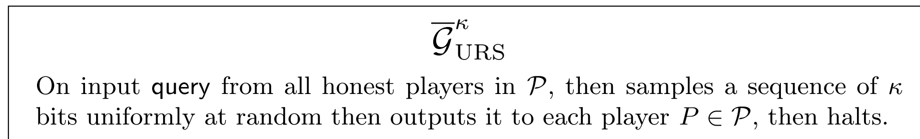
Formalizing eventual delivery For every $R \in \mathcal{R}$, initialize a counter $D_R \leftarrow 0$ // the delivery delay.
Initialize $\mathsf{outpout-available} \leftarrow \mathsf{false}$ //the flag telling if the output can be delivered.
Upon receiving fetch from any $R \in \mathcal{R}$, $D_R \leftarrow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

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

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

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

Fig. 7: The bulletin board of public keys functionality bPKI, parametrized by a set of n key-holders, of which the corrupt ones are indexed by $\mathcal{I} \subset [n]$, and by a set of receivers $\mathcal{R}$. It does not perform any check on the keys received.



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

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

Fig. 8: Uniform Random String.

B.1 Linear Secret Sharing

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

Definition 2. *((n,t)-LSS) Let $\mathbb{R}$ be a ring. A (n,t)-Linear Secret Sharing Scheme is defined by the following two algorithms:*

- $\mathsf{LSS.Share}(s \in \mathbb{R}, n, t)^6 \to (\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)})$*: For a given secret $s \in \mathbb{R}$, the sharing algorithm generates a vector $(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)})$ of shares, where $\mathbf{s}^{(i)}$ is the share of player P_i.*
- $\mathsf{LSS.Reco}(\{\mathbf{s}^{(i)}\}_{i\in\mathcal{U}}, \mathcal{U}) \to s$*: For any set $\mathcal{U}$ of size $t+1$ and shares $\{\mathbf{s}^{(i)}\}_{i\in\mathcal{U}}$, the reconstruction algorithm outputs a secret $s \in \mathbb{R}$.*

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

Furthermore it must satisfy the following properties.

1. ***Correctness**: For any set $\mathcal{U}$ of size $t+1$, the value s can be efficiently reconstructed from the set of shares $\{\mathbf{s}^{(i)}\}_{i\in\mathcal{U}}$, i.e. for any projection $S_{\mathcal{U}}$ of $S \leftarrow$* $\mathsf{LSS.Share}(s,n,t)$*, it holds that* $\mathsf{LSS.Reco}(S_{\mathcal{U}}, \mathcal{U}) = s$ *with probability* 1.
2. ***Privacy**: For all $\mathcal{V}$ such that $|\mathcal{V}| \le t$, and secrets $s_L, s_R \in \mathbb{R}$, then the shares of s_L and s_R output by* $\mathsf{LSS.Share}$ *follow the same distribution. More formally, we have:*

$$\left\{ \{\mathbf{s}_R^{(i)}\}_{i\in\mathcal{V}} \approx \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{V}} \middle| \begin{array}{l} \wedge \{\mathbf{s}_R^{(i)}\}_{i\in[n]} \leftarrow \mathsf{Share}(s_R, n, t) \\ \wedge \{\mathbf{s}_L^{(i)}\}_{i\in[n]} \leftarrow \mathsf{Share}(s_L, n, t) \end{array} \right\} \tag{11}$$

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

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

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

Moreover, for our UC proofs to go through, we require in addition the following two properties (4) and (5), which enable simulation of shares. They are satisfied by all linear secret sharings used in practice, e.g., Shamir[Sha79] and the

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

$\{0,1\}$-LSSD of [JRS17] (renamed $\{0,1\}$-LSS in [BS23]). Of possible independent interest, we provide a definition (Definition 5) of a subclass of $(n,t)-$LSS, which we call $(n,t)-$LSSD, and which encompasses both Shamir sharing, $\{0,1\}$-LSSD and the recent TreeSS scheme of Cheon et al. [CCK23]. Then in Proposition 5 we prove that a $(n,t)-$LSSD scheme satisfies both properties (4) and (5).

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

$$\Big| Pr[\mathcal{A}(\{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{V}}, \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] - Pr[\mathcal{A}(\{\mathbf{s}_R^{(i)}\}_{i\in\mathcal{V}}, \{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] \Big| \leq negl(\lambda) \tag{12}$$

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

$$\Big| Pr[\mathcal{A}(\{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{V}}, \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] - Pr[\mathcal{A}(\{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{V}}, \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] \Big| \leq negl(\lambda) \tag{13}$$

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

Example: Shamir Secret Sharing. For the purpose of this section, we consider a finite field $\mathbb{F}$ and assume that each player $P_i \in \mathcal{P}$ is associated with a non-zero element $\alpha_i \in \mathbb{F}$ such that if $i \neq j$ then $\alpha_i \neq \alpha_j$. We recall the secret-sharing scheme of Shamir [Sha79] that implements a (n,t)-LSS scheme based on polynomial interpolation in a finite field. As a reminder, let us first define what are Lagrange coefficients.

Definition 3. *Given $\mathcal{U} \subseteq [n]$ with $|\mathcal{U}| = t+1$, we denote as Lagrange coefficients the values $\{\lambda_i^{\mathcal{U}}\}_{i\in\mathcal{U}}$ computed as*

$$\lambda_i^{\mathcal{U}} = \prod_{j\in\mathcal{U}, j\neq i} \frac{\alpha_0 - \alpha_j}{\alpha_i - \alpha_j} \tag{14}$$

Intuitively, Shamir uses polynomial evaluation to share some secret and interpolation to reconstruct it from shares. In more details, to share a value $s \in \mathbb{F}$ using Shamir, the dealer samples at random a polynomial $f(Y) \in F_{\leq t}[Y]$ of degree at most t, such that $f(0) = s$. The shares corresponding to each player P_i are then define as the evaluation of f in α_i, i.e. $s^{(i)} = f(\alpha_i)$. The reconstruction of the secret is done by doing a Lagrange interpolation at α_0 from any set of

$t+1$ shares.

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

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

$$s = \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} s^{(i)} \tag{15}$$

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

B.1.1 Publicly Verifiable Secret Sharing (PVSS)

Let $\mathsf{PKE} = (\mathsf{EKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be any public key encryption scheme satisfying IND-CPA (see [GV22]). We introduce the following definition:

Definition 4. *(Publicly Verifiable Secret Sharing (PVSS)) Let us consider the following randomized function* PVSS*, parametrized by n strings $(\mathsf{pk}_i^{\mathsf{PKE}})_{i \in [n]}$. On input $s \in R_q$ compute $(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)}) \leftarrow \mathsf{LSS.Share}(s)$, and output $\left[\mathsf{PKE.Enc}(\mathsf{pk}_i^{\mathsf{PKE}}, \mathbf{s}^{(i)})\right]_{i \in [n]}$ along with a NIZK proof π of correct sharing for the following relation.*

$$\mathscr{R}_{\mathsf{Share}} = \left\{ \begin{array}{l} x = (\{\mathsf{pk}_i^{\mathsf{PKE}}\}_{j \in [n]}, \mathsf{enc{-}shares}) \\ w = (s, \mathbf{r}, \{\rho_i\}_{i \in [n]}) \end{array} \middle| \begin{array}{l} \wedge \{\mathbf{s}^{(i)}\}_{i \in [n]} \leftarrow \mathsf{LSS.Share}(s; \mathbf{r}) \\ \wedge \mathsf{enc{-}shares} \leftarrow [\mathsf{Enc}(\mathsf{pk}_i^{\mathsf{PKE}}, \mathbf{s}^{(i)}; \rho_i)]_{i \in [n]} \end{array} \right\} \tag{16}$$

PVSS *is* IND-CPA *for any $\mathcal{A}$ being given at most t secret keys $\left(\mathsf{dk}_i^{\mathsf{PKE}}\right)_{i \in \mathcal{I} \subset [n], |\mathcal{I}| \le t}$, as will be shown in Appendix B.3.*

Remark 2. By convention, encryption under an incorrectly formatted public key $\mathsf{pk}^{\mathsf{PKE}}$, e.g., $\perp$, returns the plaintext itself.

Remark 3. We describe the MPC protocol in Section 6 in the semi-malicious model, for which the NIZK proof can be dropped. This leads to the manipulation of new structures, denoted as Public Secret Sharing, namely a PVSS without a proof of correctness.

B.2 How to implement a linear Secret Sharing scheme over a Polynomial Ring

Our goal in this section is to propose instantiations of (n,t)-LSS schemes as defined in Definition 2 over polynomial rings. There are two main difficulties in constructing such schemes. First, following Definition 2 one need to be able to instantiate two efficient functions ShSim and ShInfer from these schemes, which will turn out to be very important for our UC proofs. Second, these schemes must be defined over polynomial rings, which turned out to be not obvious, for example in the case of Shamir. To address these challenges, we follow the roadmap below:

1. We first introduce in Appendix B.2.1, a subclass of $(n,t)-\mathsf{LSS}$ that we denote as $(n,t)-\mathsf{LSSD}$ in Definition 5. The latter will be used as a helper to describe a common instantiation strategy for ShSim and $\mathsf{ShInfer}$ that encompass classic sharing schemes, eg. the classical $\{0,1\}-\mathsf{LSSD}$ scheme of [JRS17].
2. Then, we present in Appendix B.2.2 our new Shamir scheme over polynomial rings, denoted as $R_p-\mathsf{Shamir}$, and show that it also is a $(n,t)-\mathsf{LSSD}$ scheme.

In a nutshell, we show the following:

$$R_p-\mathsf{Shamir} \cup (n,t)-\{0,1\}-\mathsf{LSSD} \underset{\text{Prop. 6\&9}}{\subset} (n,t)-\mathsf{LSSD} \underset{\text{Definition 5}}{\subset} (n,t)-\mathsf{LSS} \tag{17}$$

B.2.1 $(n,t)-\mathsf{LSSD}$ Following [JRS17], we now define $(n,t)-\mathsf{LSSD}$ in Definition 5, presented as a subclass of $(n,t)-\mathsf{LSS}$ (cf Definition 2). This definition will help us in Proposition 5 to describe efficient simulation and inference strategies for instantiating $(n,t)-\mathsf{LSS}$ schemes.

Definition 5. *($(n,t)-\mathsf{LSSD}$ adapted from [JRS17]) A (n,t)-LSSD scheme is defined by the following two algorithm:*

– $\mathsf{LSSD.Share}(s \in R_q, n, t) \to (\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)})$*: There exists a share matrix $\mathbf{M} \in R_q^{p\times l}$ with positive integers $p = \sum_{i=1}^{n} p_i, l$ and associate a partition T_i of $[p]$ of size $|T_i| = p_i$ to each player P_i, $\forall i \in [n]$. For a given secret $s \in R_q$ the sharing algorithm samples random values $r_2, \ldots, r_l \leftarrow R_q$ and generates a vector $(\mathsf{sh}_1, \ldots, \mathsf{sh}_p)^T = \mathbf{M} \cdot (s, r_2, \ldots, r_l)^T$. The share for P_i is a set of entries $\mathbf{s}^{(i)} = \{\mathsf{sh}_j\}_{j\in T}$.*

– $\mathsf{LSSD.Reco}(\{\mathbf{s}^{(i)}\}_{i\in\mathcal{U}}, \mathbf{M}) \to s$*: For any set $\mathcal{U} \subseteq [n]$ such that $|\mathcal{U}| > t$, one can efficiently find the coefficient $\{c_j^{\mathcal{U}}\}_{j\in\cup_{\mathcal{U}}T}$ such that*

$$\sum_{j\in\cup_{\mathcal{U}}T} c_j^{\mathcal{U}} \cdot \mathbf{M}[j] = (1, 0, \ldots, 0). \tag{18}$$

Given such coefficients, the secret can be recovered simply by computing

$$s = \sum_{j\in\cup_{\mathcal{U}}T} c_j^{\mathcal{U}} \cdot \mathsf{sh}_j. \tag{19}$$

The coefficients $\{c_j^{\mathcal{U}}\}$ are called recovery coefficients.

Our goal is then to show that a $(n,t)-\mathsf{LSSD}$ scheme verifies the properties (4) and (5) of simulatability and inference of a $(n,t)-\mathsf{LSS}$ scheme. In order to achieve this, we first adapt the following definition from [BGG+18].

Definition 6. *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of players. We define the following:*

- *A set of players $S \subseteq \mathcal{P}$ is a* maximal invalid player set *if $|S| \leq t$ but for every $P_i \in \mathcal{P} \setminus S$, we have $|S \cup \{P_i\}| > t$.*

- *A set of players $S \subseteq \mathcal{P}$ is a* minimal valid player set *if $|S| > t$ and for every $S' \subsetneq S$, we have $|S'| \leq t$.*

Let LSS *be a $(n,t)-$*LSSD *scheme with share matrix $\mathbf{M} \in R_q^{p \times l}$. For a set of indices $T \subseteq [p]$, we say that T is a* valid share set *if $(1, 0, \dots, 0) \in \mathsf{span}(\{\mathbf{M}[j]\}_{j \in T})$, and an* invalid share set *otherwise. We also define the following:*

- *A set of indices $T \subseteq [p]$ is a* maximum invalid share set *if T is an invalid share set, but for any $i \in [p] \setminus T$, the set $T \cup \{i\}$ is a valid share set.*
- *A set of indices $T \subseteq [p]$ is a* minimal valid share set *if T is a valid share set, but for any $T' \subsetneq T$, T' is an invalid share set.*

Proposition 5. *From any $(n,t)-$*LSSD *scheme, there exists an efficient instantiation of* ShSim *and* ShInfer *following Definition 2.*

Proof. We leverage Definition 6 to outline simulation and inference strategies common to all $(n,t)-$LSSD schemes as defined in Definition 5, so that we can use functions ShSim and ShInfer as wrappers independent of the instantiation.

Simulation Strategy : We now detail the overall strategy to implement ShSim, i.e. the computation of shares $\{\mathbf{s}^{(i)}\}_{i \in [n] \setminus \mathcal{V}}$ from $\{\mathbf{s}^{(i)}\}_{i \in \mathcal{V}}$ and some secret s.

1. Compute a maximal invalid share set $\{\mathsf{sh}_j\}_{j \in T}$ where $T^* = \bigcup_{i \in \mathcal{V}} T_i$.
2. To simulate $\mathbf{s}^{(i)} = \{\mathsf{sh}_j\}_{j \in T}$, compute for all $j \in T_i$:
   - If $j \in T_i \bigcap T^*$, then set $\widetilde{\mathsf{sh}}_j = \mathsf{sh}_j$.
   - If $j \notin T_i \bigcap T^*$, then compute a minimal valid share set $T \subseteq T^* \cup \{j\}$. Such set T exists since T^* is a maximal invalid share set, and we have $\sum_{j \in T} c_j \cdot \mathsf{sh}_j = s$. Therefore, as long as $j \in T$, we have:

     $$\widetilde{\mathsf{sh}}_j = (c_j)^{-1} s - \sum_{j \in T \setminus \{j\}} (c_j)^{-1} c_j \cdot \mathsf{sh}_j \tag{20}$$

     Finally, set $\mathbf{s}^{(i)} = \{\widetilde{\mathsf{sh}}_j\}_{j \in T}$.

Inference Strategy : To implement ShInfer, i.e the computation of shares $\{\mathbf{s}^{(i)}\}_{i \in \mathcal{V}, |\mathcal{V}| \leq t}$ from shares $\{\mathbf{s}^{(i)}\}_{i \in \mathcal{U} = [n] \setminus \mathcal{V}}$, one can follow this simple strategy:

1. First, reconstruct $s \leftarrow \mathsf{LSS.Reco}(\{\mathbf{s}^{(i)}\}_{i \in \mathcal{U} = [n] \setminus \mathcal{V}}, \mathcal{U})$.
2. Then, without lose of generality, choose t shares $\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(t)}$ among $\{\mathbf{s}^{(i)}\}_{i \in \mathcal{U} = [n] \setminus \mathcal{V}}$, and follow the steps described above for the "Simulation Strategy" with inputs s and these shares.
3. Output the simulated $\{\mathbf{s}^{(i)}\}_{i \in \mathcal{V}}$.

□

*Example of $\{0,1\}-$*LSSD *[JRS17]*

Property 6. *The $\{0,1\}-$*LSSD *scheme described in [JRS17] is a $(n,t)-$*LSSD *scheme.*

Proof. This property directly follows from [JRS17, Theorem 3]. Importantly, one can efficiently instantiate ShSim and ShInfer from it following the strategy described in the proof of Proposition 5. □

Remark 4. Finally, let us note that the recent TreeSS scheme presented in [CCK23] also is a $(n,t)-$LSSD scheme following their Proposition F.1. Therefore, the same reasoning is trivially valid.

B.2.2 Shamir Secret-Sharing in R_q The usual Shamir secret-sharing scheme described in Appendix B.1 is instantiated over a field $\mathbb{F}$. Indeed, Shamir is based on polynomial interpolation and involves the computation of Lagrange coefficients, that requires inverting elements of the form $\alpha_i - \alpha_j$, where α_i and α_j are public-points. Working over a field guarantees that all non-zero elements are units, hence that these coefficients exist. Our goal is to propose a variant of this classical case that works over polynomial rings.

Reminders. It is a known result since [Feh98] that using a ring is possible, as long as the set of Shamir public-points forms an *exceptional sequence* [ACD+19; CDN15] as defined in Definition 7 below.

Definition 7. *From [ACD+19] For a ring $\mathbb{R}$, the sequence $\alpha_1, \ldots, \alpha_n$ of elements of $\mathbb{R}$ is an* exceptional sequence *if $\alpha_i - \alpha_j$ is a unit in $\mathbb{R}$ for all $i \neq j$.*

Provided with an exceptional sequence over $\mathbb{R}$, we then have the following Theorem 7.

Theorem 7. *From [ACD+19] Let $\mathbb{R}$ be a commutative ring and $\alpha_1, \ldots, \alpha_n$ be an exceptional sequence in $\mathbb{R}$. Then, a Shamir secret-sharing scheme instantiated in $\mathbb{R}$ with Shamir public-points, $\alpha_1, \ldots, \alpha_n$, is correct and secure.*

What remains to be seen is how to build an exceptional sequence from R_q.

Construction of an Exceptional Sequence. To build an exceptional sequence for R_q, we distinguish two cases.

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, itself possibly small $p \leqslant n$ ([CH18; GIKV23]), then the case of composite q follows from the CRT. The construction is conceptually as follows. Consider an irreducible polynomial $\overline{Q(T)} \in \mathbb{F}_p[T]$ of degree $d := \lceil \log_p(n+1) \rceil$, then an arbitrary lift Q in $\mathbb{Z}/q\mathbb{Z}$. Finally, embed R_q in the R_q-algebra $S := R_q[T]/Q(T)$, 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 $\mathbb{Z}/q\mathbb{Z}$".

In [ACD+19], they observe that B contains a p^d-sized exceptional sequence, i.e. at least $n+1 = 2^{\log(n+1)}$ elements, denoted $(\alpha_0 := 0, \alpha_1, \ldots, \alpha_n)$, such that

all pair-wise differences $\alpha_i - \alpha_j$ for $i \neq j$ are invertible. From them, we deduce a linear secret-sharing over B, that we denote $\mathsf{LSS}[B]$. By tensorisation of $\mathsf{LSS}[B]$, 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 $\mathsf{LSS}[S]$ over $S := R_q \otimes_{\mathbb{Z}/q\mathbb{Z}} B$. Thus, we can apply to S and to these evaluation points $(\alpha_i)_{i=0,\dots,n}$ the same previous construction as for Shamir defined in Appendix B.1. R_q being a sub-ring of S, we have that $\mathsf{LSS}[S]$ particularizes to a R_q-linear sharing over R_q as desired.

Property 8. *($R_p-\mathsf{Shamir}$) Let e be an integer and p a prime. There exists a Shamir secret-sharing scheme instantiated in R_p that is correct and secure.*

Property 9. *$R_p-\mathsf{Shamir}$ is a $(n,t)-\mathsf{LSSD}$ scheme.*

Proof. This directly follows from Definition 5, when considering the Vandermonde matrix as sharing matrix. □

To summarize, we consider an exceptional sequence $\alpha_1, \dots, \alpha_n$, where each α_i will be treated as a Shamir public-point. To share a secret s, sample a polynomial h at random in $S[X]_t^{(s)}$, then output $\{h(\alpha_i)\}_{i\in[n]}$. Each share, which is in S, is therefore encoded as d elements of R_q. Then for reconstruction use the Lagrange polynomials $\Pi_{j\neq i}(X - \alpha_j)/(\alpha_i - \alpha_j)$.
Note 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.

Uniformity of any t shares of any given secret.

Property 10. *Let p be a prime and e and integer. For every $s \in R_p$, for any subset of t indices $\mathcal{I} \subset [n]$, the distribution of shares $(s^{(i)})_{i\in\mathcal{I}}$ output by $R_p-\mathsf{Shamir.Share}(s)$ is $U(R_p^t)$.*

Proof. For any commutative ring $\mathbb{R}$ with unit 1, we denote as $\mathbb{R}[Y]_t$ the polynomials of degree $\leq t$. Let us first introduce, for a set $\mathcal{U} \subset (\alpha_i)_{i\in[0,\dots,n]}$, the following map:

– $\mathsf{Eval}_{\mathcal{U}} : h \in \mathbb{R}[Y]_t \to [h(\alpha_i), \alpha_i \in \mathcal{U}]$: the map returning the evaluations at points of $\mathcal{U}$.

By [ACD+19, Thm 3], for every $(t+1)$-sized $\mathcal{U}$, we have that $\mathsf{Eval}_{\mathcal{U}}$ is an *isomorphism.* Then, we have the randomized function $\mathsf{LSS}[\mathbb{R}].\mathsf{Share} : \mathbb{R} \longrightarrow \mathbb{R}^n$, defined as: on input a secret $s \in \mathbb{R}$, sample $h \leftarrow U\big(\mathbb{R}[Y]_t^{(s)}\big)$ then return $\mathsf{Eval}_{(\alpha)}(h)$ denoted shares of s.

By surjectivity (isomorphism) of $\mathsf{Eval}_{\{0\}\cup\mathcal{I}} : \mathbb{R}[Y]_t \to \mathbb{R}^{t+1}$ for any t-sized subset $\mathcal{I}$ of indices of $(\alpha_i)_{i\in[0,\dots,n]}$, we have surjectivity (isomorphism) of $\mathsf{Eval}_{\mathcal{I}} : \mathbb{R}[Y]_t^{(s)} \to \mathbb{R}^t$ for any fixed $s \in \mathbb{R}$. Furthermore, the map $\mathsf{Eval}_{\mathcal{I}}$ being also linear, we have that it maps the uniform distribution onto the uniform distribution.

When LSS is instantiated with $R_p-\mathsf{Shamir}$, we have the desired result. □

B.3 Proof of IND-CPA of Publicly Verifiable Secret Sharing

Proposition 11 states that any PPT adversary $\mathcal{A}$ corrupting at most t players, has negligible advantage in distinguishing between the encrypted $(n,t)-\mathsf{LSS}$ sharings of any two chosen secrets $(s_L, s_R) \in R_q^2$. Recall that in this section, we consider any public key encryption scheme $\mathsf{PKE} = (\mathsf{EKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfying IND-CPA (see [GV22]).

Proposition 11 (*IND-CPA *of encrypted sharing). *For any integers $0 \leq t \leq n$, we consider the following game between an adversary $\mathcal{A}_{\mathsf{PVSS}}$ and an oracle $\mathscr{O}$. $\mathscr{O}$ is parametrized by a secret $b \in \{L, R\}$ (left or right oracle).*

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

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

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

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

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

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

Fig. 9: IND-CPA of encrypted sharing

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

In order to prove Proposition 11, our goal is then to bound the advantage by any adversary $\mathcal{A}_{\mathsf{PVSS}}$ in the game presented in Fig. 10, by the maximum advantage of an adversary $\mathcal{A}_{\mathsf{PKE}}$ in the (n-t)-keys variant indistinguishability game for PKE presented in Fig. 12.

Proof. We now consider the game of IND-CPA *of encrypted sharing with plaintext adversary shares*. We denote again its oracles as $\mathscr{O}^L$ and $\mathscr{O}^R$, although now they return in the clear the t corrupt shares. We first define two apparent modifications of $\mathscr{O}^L$ and $\mathscr{O}^R$, denoted as cotL and $\widetilde{\mathscr{O}}^R$, which only differ from the previous, in that they *first* sample the corrupt shares $(\mathbf{s}^{(i)})_{i\in\mathcal{I}} \stackrel{\$}{\leftarrow} R_q^t$[7] uniformly at random, *then* simulate shares for $[n] \setminus \mathcal{I}$ using the latter and s_L or s_R.

Actually, by the secrecy of the secret sharing scheme, they produce exactly the same distribution as $\mathscr{O}^L$ and $\mathscr{O}^R$. We describe them below in Fig. 10, then formalize the previous claim in Equation (21).

[7] Here we do as if all shares were in R_q

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

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

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

- $\widetilde{\mathscr{O}}^{\boldsymbol{L}}$ **:** samples $(\mathbf{s}^{(i)})_{i\in\mathcal{I}} \xleftarrow{\$} R_q^t$ and simulates shares for $[n]\setminus\mathcal{I}$. Precisely, it interpolates $(\{\mathbf{s}^{(i)}\}_{i\in[n]\setminus\mathcal{I}}) \leftarrow \mathsf{ShSim}(\{\mathbf{s}^{(i)}\}_{i\in\mathcal{I}}, s_L)$ and returns $\big((\mathbf{s}^{(i)})_{i\in\mathcal{I}}, (\mathsf{Enc}_{\mathsf{pk}_i}(\mathbf{s}^{(i)}))_{i\in[n]\setminus\mathcal{I}}\big)$;
- $\widetilde{\mathscr{O}}^{\boldsymbol{R}}$**:** samples $(\mathbf{s}^{(i)})_{i\in\mathcal{I}} \xleftarrow{\$} R_q^t$ and simulates shares for $[n]\setminus\mathcal{I}$. Precisely, it interpolates $(\{\mathbf{s}^{(i)}\}_{i\in[n]\setminus\mathcal{I}}) \leftarrow \mathsf{ShSim}(\{\mathbf{s}^{(i)}\}_{i\in\mathcal{I}}, s_R)$ and returns $\big((\mathbf{s}^{(i)})_{i\in\mathcal{I}}, (\mathsf{Enc}_{\mathsf{pk}_i}(\mathbf{s}^{(i)}))_{i\in[n]\setminus\mathcal{I}}\big)$;

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

Fig. 10: IND-CPA of encrypted sharing with plaintext adversary shares.

For any possibly unlimited adversary $\mathcal{A}_{\mathsf{PVSS}}$,

(21)
$$|\Pr(1 \leftarrow \mathcal{A}^{\widetilde{\mathscr{O}}}_{\mathsf{PVSS}}) - \Pr(1 \leftarrow \mathcal{A}^{\mathscr{O}}_{\mathsf{PVSS}})| = 0 \text{ and } |\Pr((1 \leftarrow \mathcal{A}^{\widetilde{\mathscr{O}}}_{\mathsf{PVSS}}) - \Pr(1 \leftarrow \mathcal{A}^{\mathscr{O}}_{\mathsf{PVSS}})| = 0$$

To conclude the proof, we introduce an intermediary oracle, defined as $\widetilde{\mathscr{O}}^Z$ in Fig. 11. $\widetilde{\mathscr{O}}^Z$ is the common modification of $\widetilde{\mathscr{O}}^L$ and $\widetilde{\mathscr{O}}^R$, which sets to 0 the $n-t$ honest plaintext shares. In particular, it completely ignores the request (s_L, s_R) given to it.

$\widetilde{\mathscr{O}}^{\boldsymbol{Z}}$ **:** samples $(\mathbf{s}^{(i)})_{i\in\mathcal{I}} \xleftarrow{\$} R_q^t$; sets $\mathbf{s}^{(i)} := 0\ \forall i \in [n]\backslash\mathcal{I}$; returns $\big((\mathbf{s}^{(i)})_{i\in\mathcal{I}}, (\mathsf{Enc}_{\mathsf{pk}_i}(\mathbf{s}^{(i)}))_{i\in[n]\setminus\mathcal{I}}\big)$;

Fig. 11: Intermediate oracle $\widetilde{\mathscr{O}}^Z$ for the game presented in Fig. 10.

From Property 10 of uniform independence of the t plaintext shares $\big(\mathbf{s}^{(i)}\big)_{i\in\mathcal{I}}$, we conclude that the distinguishing advantage between both $\widetilde{\mathscr{O}}^L$ and $\widetilde{\mathscr{O}}^R$, with $\widetilde{\mathscr{O}}^Z$, is negligible.

Claim 12. *The maximum distinguishing advantage with $\widetilde{\mathscr{O}}^Z$ is less than the one for $(n-t)$-keys* IND-CPA *for* PKE*.*

We recall the game defining it, from which the *Claim* should be clear enough. It is between a challenger $\mathcal{A}_{\mathsf{PKE}}$, and an oracle $\mathscr{O}_{\mathsf{PKE}}$ parametrized by a secret $b \in \{E, 0\}$.

$\mathsf{Game}^{\mathcal{A}}_{(n-t)-\mathsf{IND\text{-}CPA}}$

Setup. For each $i \in [n-t]$, $\mathscr{O}_{\mathsf{PKE}}$ generates $(\mathsf{dk}_i, \mathsf{pk}_i) \leftarrow \mathsf{EKeyGen}(1^\lambda)$ and shows pk_i to $\mathcal{A}_{\mathsf{PKE}}$.

Query. $\mathcal{A}_{\mathsf{PKE}}$ gives to $\mathscr{O}_{\mathsf{PKE}}$ $(n-t)$ chosen plaintexts $(s_h)_{h\in[n-t]}$, then $\mathscr{O}_{\mathsf{PKE}}$ replies depending on $b \in \{E, 0\}$.
$\mathscr{O}^{\boldsymbol{E}}_{\mathsf{PKE}}$ returns $(\mathsf{Enc}_{\mathsf{pk}_h}(s_h))_{h\in[n-t]}$;
$\mathscr{O}^{\mathbf{0}}_{\mathsf{PKE}}$ returns $(\mathsf{Enc}_{\mathsf{pk}_h}(0))_{h\in[n-t]}$.

Guess. $\mathcal{A}_{\mathsf{PKE}}$ gets some $(\mathsf{c}_h)_{h\in[n-t]}$ and outputs $b' \in \{E, 0\}$. It wins if $b' = b$.

Fig. 12: $(n-t)$-keys IND-CPA

Recall that the distinguishing advantage in this $\mathsf{Game}^{\mathcal{A}}_{(n-t)-\mathsf{IND\text{-}CPA}}$ game, is upper-bounded by $n-t$ times the advantage for one-message indistinguishability, see e.g. [BS20, Thm 5.1].

We now fully formalize the proof of the *Claim*, as the following straightforward reduction from the game $(\widetilde{\mathscr{O}}^L/\widetilde{\mathscr{O}}^Z)$ (and likewise $(\widetilde{\mathscr{O}}^Z/\widetilde{\mathscr{O}}^R)$) into the $n-t$-keys IND-CPA game $(\mathscr{O}^E_{\mathsf{PKE}}/\mathscr{O}^0_{\mathsf{PKE}})$. The reduction works as follows.

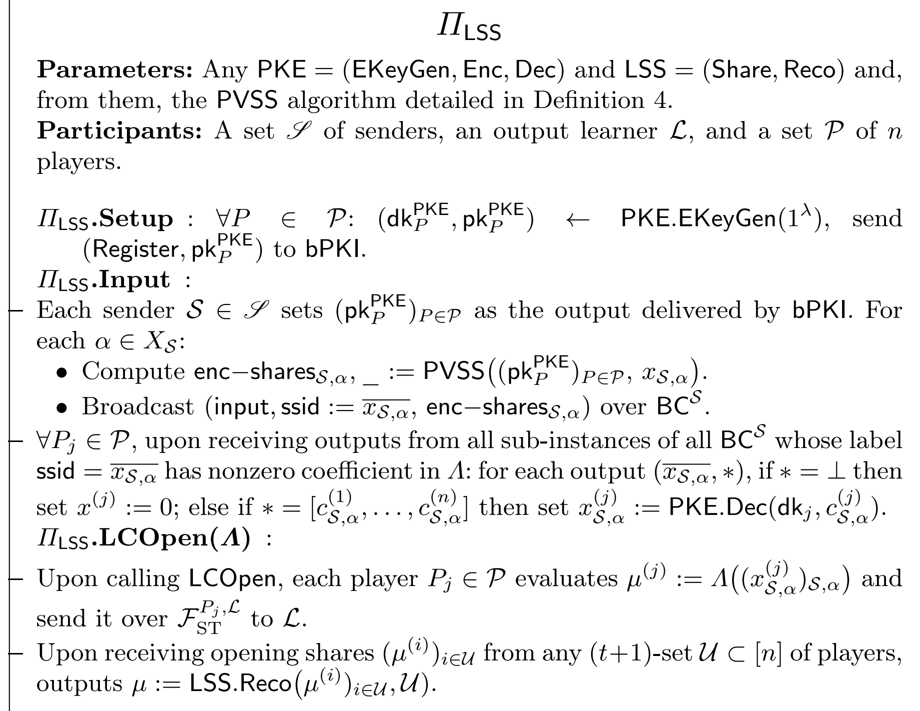
1. Upon receiving a set of keys $(\mathsf{pk}^{\mathsf{PKE}}_h)_{h\in\mathcal{H}}$ from $\mathscr{O}_{\mathsf{PKE}}$, then $\mathcal{A}_{\mathsf{PKE}}$ samples itself t key pairs $(\mathsf{dk}^{\mathsf{PKE}}_i, \mathsf{pk}^{\mathsf{PKE}}_i)_{i\in\mathcal{I}}$, initiates $\mathcal{A}_{\mathsf{PVSS}}$, reorganizes the indices so that the indices chosen by $\mathcal{A}_{\mathsf{PVSS}}$ correspond to $\mathcal{I}$, gives to $\mathcal{A}_{\mathsf{PVSS}}$ the total $n = |\mathcal{H}| + |\mathcal{I}|$ public keys and furthermore gives to $\mathcal{A}_{\mathsf{PKE}}$ the t secret keys $(\mathsf{dk}^{\mathsf{PKE}}_i)_{i\in\mathcal{I}}$.
2. Upon receiving one challenge (s_L, s_R) from $\mathcal{A}_{\mathsf{PVSS}}$, $\mathcal{A}_{\mathsf{PKE}}$ samples $(\mathbf{s}^{(i)})_{i\in\mathcal{I}} \xleftarrow{\$} R_q^t$ and interpolates $(\{\mathbf{s}^{(i)}\}_{i\in[n]\setminus\mathcal{I}}) \leftarrow \mathsf{ShSim}(\{\mathbf{s}^{(i)}\}_{i\in\mathcal{I}}, s_L)$ which it gives to its oracle $\mathscr{O}_{\mathsf{PKE}}$ as a request.
3. Upon receiving the response ciphertexts $(\mathsf{c}_h)_{h\in\mathcal{H}}$ from $\mathscr{O}_{\mathsf{PKE}}$, it then computes the n-sized vector V consisting of:
   – The entries in $\mathcal{I}$ equal to the plaintexts $(\mathbf{s}^{(i)})_{i\in\mathcal{I}}$ that $\mathcal{A}_{\mathsf{PKE}}$ generates itself.
   – The remaining entries are set to the $\{\mathsf{c}_h\}_{h\in\mathcal{H}}$ received from $\mathscr{O}_{\mathsf{PKE}}$.
   And sends it to $\mathcal{A}_{\mathsf{PVSS}}$ as response to its challenge.
4. Upon answer a bit b from $\mathcal{A}_{\mathsf{PVSS}}$, then $\mathcal{A}_{\mathsf{PKE}}$ outputs the same bit b to $\mathscr{O}_{\mathsf{PKE}}$.

The *Claim* now follows from the fact that in the case $\mathscr{O}^E_{\mathsf{PKE}}$, then $\mathcal{A}_{\mathsf{PVSS}}$ is facing the same behavior as $\widetilde{\mathscr{O}}^L$, while in the case $\mathscr{O}^0_{\mathsf{PKE}}$, then $\mathcal{A}_{\mathsf{PVSS}}$ is facing the same behavior as $\widetilde{\mathscr{O}}^Z$. Thus the distinguishing advantage of $\mathcal{A}_{\mathsf{PKE}}$ is the same as the one of $\mathcal{A}_{\mathsf{PVSS}}$, which concludes the proof.

□

B.4 Implementation of $\mathcal{F}_{\mathsf{LSS}}$.

We now detail in Fig. 13 protocol Π_{LSS} that instantiates $\mathcal{F}_{\mathsf{LSS}}$ in the $(\mathsf{BC}, \mathcal{F}_{\mathrm{ST}}, \mathsf{bPKI})$-hybrid model. Recall from Section 3.2.1 that we consider a set $\mathcal{P}$ of n players, a set $\mathscr{S}$ of senders and an output learner $\mathcal{L}$.



$\mathit{\Pi}_{\mathsf{LSS}}$

Parameters: Any $\mathsf{PKE} = (\mathsf{EKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and $\mathsf{LSS} = (\mathsf{Share}, \mathsf{Reco})$ and, from them, the PVSS algorithm detailed in Definition 4.
Participants: A set $\mathscr{S}$ of senders, an output learner $\mathcal{L}$, and a set $\mathcal{P}$ of n players.

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

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

Fig. 13: Protocol for secret-sharing then delayed linear combination

Proposition 13. *Protocol $\mathit{\Pi}_{\mathsf{LSS}}$ UC implements $\mathcal{F}_{\mathsf{LSS}}$*

Proof. For simplicity, we construct a simulator for an honest $\mathcal{L}$[8] and 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 Open.

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

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

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

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

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

witness tape of $\mathcal{S}$.

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

Claim 14. *$REAL_{\mathcal{A}} \equiv \mathsf{Hybrid}^{\mathsf{0Share}}$.*
Proof: Since $\mathsf{ShSim}^{\mathcal{I}}$ simulates perfectly, they are identical to the ones of the Real execution.

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

Claim 15. $\mathsf{Hybrid}^{\mathsf{0Share}} \equiv \mathsf{Hybrid}^{\mathcal{F}}$.
Proof: By correctness of Π_{LSS}, $y = \widetilde{y}$ so the view of Env is unchanged.

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

Claim 16. $\mathsf{Hybrid}^{\mathcal{F}} \equiv \mathsf{Hybrid}^{\mathsf{0Share}}$.
Proof: Since the private decryption keys dk_h of all honest players $h \in \mathcal{H}$ are not used anymore, we have that the IND-CPA *property of* PVSS *stated in Proposition 11 applies. Thus the view of* Env *is indistinguishable from the one in the previous game.*

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

Claim 17. $\mathsf{Hybrid}^{\mathsf{0Share}} \equiv \mathsf{Hybrid}^{\mathsf{ShInfer}}$.
Proof: The inferred shares are identical to the ones in the previous game, by the property of $\mathsf{ShInfer}^{\mathcal{H}}$.

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

□

C Complements on trBFV

The goal of this section is to provide details regarding what has been introduced in Section 4. First, in Appendix C.1, we recall some generalities that we will use throughout this section. Then in Appendix C.2, we detail the circular security assumption made in [CDKS19], which we use as a basis to demonstrate the security of our relinearization key generation protocol. Then in Appendix C.3, we define the *decryption noise* and prove, as a warmup, correctness of the decryption of a fresh encryption. In Appendix C.4, we detail the homomorphic properties that can be added to the standalone scheme. Finally in Appendix C.5, we perform a complete noise analysis after homomorphic evaluation of a circuit.

C.1 Generalities

Notation. We adopt the notations introduced in Section 2.1. For any element $\widetilde{r} = \Sigma_{i=0}^{n-1} \widetilde{r}_i X^i \in R$, with $R := \mathbb{Z}[X]/f(X)$, 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 $\|r\| := \|\widetilde{r}\|$.

Recall that we denote $\Delta = \lfloor q/k \rfloor$, the integer division of q by k. We denote vectors of some length l (see Section 3) in bold, e.g. $\mathbf{a}$. For such vector $\mathbf{r} = (r_1, \ldots, r_l) \in R_q^l$, we define $\|\mathbf{r}\| := \max_i |\widetilde{r}_i|$. For two polynomials p and h in R_q whose polynomial modulus is a degree-d power of 2 cyclotomic, we have

$$\|ph\| \leq d\|p\|\|h\|. \tag{22}$$

C.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 $(d, q, \mathscr{X}_q, \Psi_q)$ since it uses the same encryption algorithm as 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 $R_q^{4\times l}$. 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 now detail 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 18. Consider an oracle $\mathscr{O}_{\mathcal{D}}$ which samples $\mathbf{a} \xleftarrow{\$} U(R_q^l)$ then KeyGenerates one BFV key pair $(\mathsf{sk}, \mathsf{ek})$, then samples $\mathbf{d}_1 \xleftarrow{\$} U(R_q^l)$, then, using $\mathsf{RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk})$, 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 18. *Define the distribution:*
$$\mathcal{D}_0 := \Big\{(\mathbf{b}, \mathbf{a}, \mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2) : (\mathbf{a}, \mathbf{d}_1) \leftarrow U(R_q^l)^2,\ \mathsf{sk} \leftarrow \mathscr{X}_q, (\mathbf{e}^{(\mathsf{ek})}, \mathbf{e}_0^{(\mathbf{rlk})}, \mathbf{e}_2^{(\mathbf{rlk})}) \leftarrow (\Psi_q^l)^3,$$
$$r \leftarrow \mathscr{X}_q,\ \mathbf{b} := -\mathbf{a}\,\mathsf{sk} + \mathbf{e}^{(\mathsf{ek})},\ \mathbf{d}_0 := -\mathsf{sk}\,\mathbf{d}_1 + \mathbf{e}_0^{(\mathbf{rlk})} + r\,\mathbf{g},\ \mathbf{d}_2 := r\,\mathbf{a} + \mathbf{e}_2^{(\mathbf{rlk})} + \mathsf{sk}\,\mathbf{g}\Big\}$$
Then the maximum distinguishing advantage $\mathsf{Adv}_{\mathcal{D}}^{\lambda}$ *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 (Appendix 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.

C.2.1 How Assumption 18 appears in [CDKS19] Assumption 18 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 $\mathsf{UniEnc}_{\mathbf{a}}$, 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 Appendix B.1 that for any (chosen plaintext) μ, 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 $\mathsf{UniEnc}_{\mathbf{a}}(\mathsf{sk}, \mu)$, then the *single output* $(\mathbf{b}, \mathsf{UniEnc}_{\mathbf{a}}(\mathsf{sk}, \mu))$ is indistinguishable from a single sample in $U(R_q^{l\times 5})$. Next, they assume that (Security) also holds when the chosen μ is replaced by the secret key sk itself, which is exactly what we spelled-out in Assumption 18. Concretely, in their $\mathsf{UniEnc}_{\mathbf{a}}$, 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.

C.3 Warmup: Correctness & Decryption Noise of a Fresh Encryption

We first introduce some definitions:

Definition 19 (Decryption noise). *Let* $\mathsf{c} \in R_q^2$, $m \in R_k$ *and* $\mathsf{sk} \in \mathscr{X}_q$. *We define the "decryption noise" as* $e^{(\mathsf{Dec})}(\mathsf{c}, \mathsf{sk}, m) := \Lambda_{\mathsf{Dec}}^{\mathsf{c}}(\mathsf{sk}) - \Delta.m$.

Proposition 20 (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* $|e^{(\mathsf{Dec})}(\mathsf{c}, \mathsf{sk}, m)| < \frac{\Delta}{2}$, *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: $\mathsf{c} = (\Delta m + u \cdot b + e_0^{(\mathsf{Enc})}, u \cdot a + e_1^{(\mathsf{Enc})})$, where $\|u\| \leq 1$, $\left\|e_0^{(\mathsf{Enc})}\right\| \leq B_{\mathsf{Enc}}$ and $\left\|e_1^{(\mathsf{Enc})}\right\| \leq B$.

Let us denote $e^{(fresh)} := e^{(\mathsf{Dec})}(\mathsf{c}, \mathsf{sk}, m) := \mathsf{c}[0] + \mathsf{c}[1] \cdot \mathsf{sk} - \Delta m$ its decryption noise (Proposition 20). Recall that by definition we have that $\mathsf{c}[0] + \mathsf{c}[1] \cdot \mathsf{sk} = \Delta m + e^{(fresh)}$. With $e^{(\mathsf{ek})} = \mathbf{e}^{(\mathbf{pk})}[0]$, we have

$$\mathsf{c}[0] + \mathsf{c}[1] \cdot \mathsf{sk} = \Delta m + u \cdot e^{(\mathsf{ek})} + e_0^{(\mathsf{Enc})} + \mathsf{sk} \cdot e_1^{(\mathsf{Enc})}$$

$$\left\| e^{(fresh)} \right\| \leq B_{\mathsf{Enc}} + d \| e^{(\mathsf{ek})} \| + d \cdot B \cdot \| \mathsf{sk} \| := B_{fresh} \tag{23}$$

C.4 Correctness & Decryption Noise of Homomorphic Operations

In C.4.1 and C.4.2, we upper-bound the additional noise introduced respectively by $\mathsf{trBFV.Add}$ and $\mathsf{trBFV.Mult}$ as the sum of the bounds given by Equations (26) and (28).These bounds are obtained by particularizing the analysis of [CDKS19] in the single key setting, and turning their variances into essential upper-bounds.

C.4.1 Noise Analysis of Addition Let us consider two ciphertexts c_1 and c_2 such that $\mathsf{c}_1[0] + \mathsf{c}_1[1] \cdot \mathsf{sk} = \Delta m_1 + e_1^{(\mathsf{Dec})}$ and $\mathsf{c}_2[0] + \mathsf{c}_2[1] \cdot \mathsf{sk} = \Delta m_2 + e_2^{(\mathsf{Dec})}$. Let $\mathsf{c}_{\mathsf{Add}} = \mathsf{trBFV.Add}(\mathsf{c}_1, \mathsf{c}_2)$ be the homomorphic sum of c_1 and c_2, and let us define the "decryption noise of an addition" as $e^{(\mathsf{Add})} := e^{(\mathsf{Dec})}(\mathsf{c}_{\mathsf{Add}}, \mathsf{sk}, m_1 + m_2)$.

Thus we have $\mathsf{c}_{\mathsf{Add}}[0] + \mathsf{c}_{\mathsf{Add}}[1] \cdot \mathsf{sk} = \Delta [m_1 + m_2]_k + e^{(\mathsf{Add})}$, with $m_1 + m_2 = [m_1 + m_2]_k + k \cdot r$ for $\|r\| \leq 1$ and

$$\| e^{(\mathsf{Add})} \| = \| e_1^{(\mathsf{Dec})} + e_2^{(\mathsf{Dec})} + r_k(q)\, r \| \leq \| e_1^{(\mathsf{Dec})} \| + \| e_2^{(\mathsf{Dec})} \| + r_k(q) \tag{24}$$

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

C.4.2 Noise Analysis of Multiplication & Relinearization Let us consider two ciphertexts c_1 and c_2 such that $\mathsf{c}_1[0] + \mathsf{c}_1[0] \cdot \mathsf{sk} = \Delta m_1 + e_1^{(\mathsf{Dec})}$ and $\mathsf{c}_2[0] + \mathsf{c}_2[1] \cdot \mathsf{sk} = \Delta m_2 + e_2^{(\mathsf{Dec})}$. Recall from Section 4.3 that the multiplication of two ciphertexts involves two steps that introduce noise: a *tensoring* operation followed by a *relinearization*.

1. 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 sk and plaintext $m_1\, m_2$", and denote it $e^{(tens)}$, as:

$$\hat{\mathsf{c}}[0] + \hat{\mathsf{c}}[1] \cdot \mathsf{sk} + \hat{\mathsf{c}}[2] \cdot \mathsf{sk}^2 = \Delta [m_1\, m_2]_k + e^{(tens)} \tag{25}$$

Using [FV12, Lemma 2], we conclude that

$$\| e^{(tens)} \| \leq d \cdot k \left(\| e_1^{(\mathsf{Dec})} \| + \| e_2^{(\mathsf{Dec})} \| \right) (d \cdot \| \mathsf{sk} \| + 1) + 2k^2 \cdot d^2 \left(\| \mathsf{sk} \| + 1 \right)^2. \tag{26}$$

This shows that the noise is roughly multiplied by the factor $2 \cdot k \cdot d^2 \cdot n$.

2. Relinearization. Second, a relinearization is performed using a key, denoted **rlk**, generated distributively by the new $\mathsf{RlkKeygen}$ algorithm detailed in Section 4. Recall that $\mathbf{rlk} = (\sum_{i \in n} \mathbf{d}_{0,i}, \mathbf{d}_1, \sum_{i \in n} \mathbf{d}_{2,i})$ where $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{trBFV.RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$.

Consider a degree two ciphertext $\hat{\mathsf{c}}$ with decryption noise $e^{(tens)}$ with respect to plaintext m ($m_1 \cdot m_2$ in our context) and secret key sk. Let us now recall that algorithm Relin presented in Algorithm 1, 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 $\mathsf{c}' = (\mathsf{c}'[0], \mathsf{c}'[1]) \in R_q^2$.

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

$$\hat{\mathsf{c}}[0] + \hat{\mathsf{c}}[1]\,\mathsf{sk} + \hat{\mathsf{c}}[2]\,\mathsf{sk}^2 = \mathsf{c}'[0] + \mathsf{c}'[1]\,\mathsf{sk} + e^{(relin)} \tag{27}$$

Unrolling the Relin algorithm presented in Algorithm 1, we obtain:

$$\|e^{(relin)}\| \le \|err_1\| + \|err_2\| \le d \cdot l \cdot n \cdot B_g \cdot B + 2d^2 \cdot l^2 \cdot n^2 \cdot B_g \cdot B \tag{28}$$

From Equations (26) and (28) we deduce:

Proposition 21 (Decryption noise of a product). *Consider two ciphertexts c_1 and c_2 of m_1 and m_2 respectively under a key $\big(\boldsymbol{b} = -\mathbf{a}\cdot\mathsf{sk}+\boldsymbol{b}\,, \mathbf{a}\big) \in R_q^{2\times l}$, with decryption noises (Definition 19) denoted $e_i^{(\mathsf{Dec})} := \mathsf{c}_i[0] + \mathsf{c}_i[1] \cdot \mathsf{sk} - \Delta m_i$, $i \in \{1,2\}$. Consider any $\mathbf{rlk} = (\sum_{i \in n} \mathbf{d}_{0,i}, \mathbf{d}_1, \sum_{i \in n} \mathbf{d}_{2,i})$ where $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \mathsf{trBFV.RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$, denote $\mathsf{c}' := \mathsf{trBFV.Mult}(\mathsf{c}_1, \mathsf{c}_2, \mathbf{rlk}, \boldsymbol{b})$, then $e^{(\mathsf{Dec})}(\mathsf{c}', \mathsf{sk}, m_1 m_2)$ is dominated by $k \cdot d^2 \cdot n (\|e_1^{(\mathsf{Dec})}\| + \|e_2^{(\mathsf{Dec})}\|) + 2d^2 \cdot l^2 \cdot n^2 \cdot B_g \cdot B$.*

C.5 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 an evaluated ciphertext, does return the correctly evaluated plaintext.

Definition 22 (Decryption noise of a circuit: B_{C}). *For any arithmetic circuit C of depth L, with input gates indexed by n, we consider the largest norm of the decryption noise $e^{(\mathsf{Dec})}(\mathsf{c}, \mathsf{sk}, y)$ of a ciphertext c, over the previous choices, and over the choices: of elements $(m_i \in R_k)_{i \in [n]}$, and of arbitrary fresh* BFV *Encryptions of them $(\mathsf{c}_i)_{i \in [n]}$; denoting $\mathsf{c} := \mathsf{trBFV.Eval}(\mathrm{C}, (\mathsf{c}_i)_i, \mathbf{rlk}, \boldsymbol{b})$ and $y := \mathrm{C}((m_i)_i)$. From Definition 19 and Fig. 14, it follows that, for any y and c as above,* if *the second threshold decryption method is used with a level of noise B_{sm} such that:*

$$B_{\mathrm{C}} + n.B_{sm} < \frac{\Delta}{2} \tag{29}$$

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

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 n. Using Proposition 21, we estimate an upper bound on the decryption noise of the evaluated ciphertext as:

$$C_1^L.B_{fresh} + C_2 \Sigma_{i=0}^{L-1} C_1^i \leq C_1^L.B_{fresh} + L.C_2.C_1^{L-1} \tag{30}$$

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

C.6 More on Threshold Decryption

We recap in Fig. 14 our protocol for threshold decrypting some ciphertext c, and highlight the two main approaches introduced in Section 5.

Threshold Decryption protocol

Participants: n players P_i, $i = 1, \ldots, n$;
Inputs: a public ciphertext c; shared decryption key sk in $\mathcal{F}_{\mathsf{LSS}}$ with label $\overline{\mathsf{sk}}$ // concretely, in the form of a PVSS of sk.
Outputs: decryption $\mathsf{Dec}(\mathsf{sk}, \mathsf{c})$
Additional input for 2nd method: secret shared smudging noise $e_{sm} \in [-n.B_{sm}, n.B_{sm}]$ in $\mathcal{F}_{\mathsf{LSS}}$ with label $\overline{e_{\mathrm{sm}}}$ (usable only for one decryption).

(1st method - mainstream): Each player P_i, given a ciphertext c and a secret share sk_i of the secret key:

- Generates its decryption share of the decryption (i), i.e. $\mu_i = \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk}_i)$ (see Equation (5)). Samples a noise $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$. Then, multicasts over P2P channels its "noisy decryption share" $\mu_i = \mu_i + n!^2 e_{\mathrm{sm},i}$. // The $n!^2$ factor is not needed if $\{0,1\}$-LSSD is used [JRS17; CCK23];
- Each player P_i waits until it receives noisy decryption shares from a subset $\mathcal{U} \subset [n]$ of indices of $t+1$ players: $(\mu_j)_{j\in\mathcal{U}}$. Denote $(\lambda_j^{\mathcal{U}})_{j\in\mathcal{U}}$ the Lagrange reconstruction coefficients corresponding to the set $\mathcal{U}$ (see Definition 3). It sets $\mu = \sum_{j\in\mathcal{U}} \lambda_j^{\mathcal{U}} \mu_j$ the smudged decryption and outputs $\Omega_{\mathsf{Dec}}(\mu)$.

(2nd method, with smaller noise): Each player P_i:

- Given labels $(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})$, and a ciphertext c, sends $\big(\mathsf{LCOpen}, \Lambda^{\mathsf{c}}_{\mathsf{Dec}+sm}(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})\big)$ (see Equation (7)) to $\mathcal{F}_{\mathsf{LSS}}$, obtains $\widetilde{\mu}$, and outputs $m = \Omega_{\mathsf{Dec}}(\mu)$.

Fig. 14: Threshold Decryption Protocol, when the (n,t)-LSS scheme is instantiated with Shamir.

D Further Details on the Proof of Theorem 1

D.1 Pseudorandomness of BFV ciphertexts with uniformly generated encryption keys

First, we want to prove that considering ann encryption key sampled uniformly at random, the ciphertexts produced by $\mathsf{BFV.Enc}$ are pseudorandom under the

RLWE assumption. The reason is that in the context of Hybrid_3, i.e., in Lemma 27 the view of Env is very similar to the one of the BFV scheme, except that the key is uniformly random. We formalize it by the game $\mathsf{Game}_{\mathsf{Semantic}}$ shown below:

$\mathsf{Game}_{\mathsf{Semantic}}$

Setup. The challenger generates samples $a, b \xleftarrow{\$} U(R_q)$ and sends (a, 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 $\mathsf{c}^* = (\mathsf{c}_0^*, \mathsf{c}_1^*) \leftarrow \mathsf{BFV.Enc}(\mathsf{ek} = (b, a), m)$.

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

Fig. 15: Pseudorandomness of BFV ciphertexts with uniformly generated encryption keys

Lemma 23. *Let* $\mathsf{pp} = (R_q, l, \mathscr{X}_q, R_k, \Psi_q, \mathscr{B}_{\mathsf{Enc},q})$ *be parameters suited for our MPC protocol presented in Section 6.1, i.e. such that Corollary 2 holds and that satisfies Equation* (8). *Then for any* PPT *adversary* $\mathcal{A}$*, the function* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Semantic}}(1^\lambda) := \left|Pr[\beta = \beta'] - \frac{1}{2}\right|$*, denoted as the advantage of* $\mathcal{A}$*, is negligible in* λ*.*

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

Thus, by RLWE for $(\mathscr{X}_q, \Psi_q)$, and thus a fortiori for $(\mathscr{X}_q, \mathscr{B}_{\mathsf{Enc},q})$ (Equation 8), the two RLWE samples are indistinguishable from a sample in $U(R_q^2)$. □

D.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 $\mathsf{ek}' = (\mathbf{b}+\mathbf{b}', \mathbf{a})$, where $\mathbf{b}'$ is adaptively generated by the semi-honest adversary after it saw $\mathbf{b}$. Our goal is to adapt their result in the RLWE setting. We consider an experiment $\mathsf{JointKey}(R_q, l, \mathscr{X}_q, R_k, \Psi_q, \mathscr{B}_{\mathsf{Enc},q})$ between an attacker $\mathcal{A}$ and a challenger defined as:

$\mathsf{JointKey}(R_q, l, \mathscr{X}_q, R_k, \Psi_q, \mathscr{B}_{\mathsf{Enc},q})$

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 $(\mathsf{sk}_i, \mathbf{e}_i^{(\mathsf{ek})})_{i\in\mathcal{I}}$ for some set $\mathcal{I}$ of indices. Both terms being either $\perp$ or such that $\|\mathsf{sk}_i\| = 1$ and $\|\mathbf{e}_i^{(\mathsf{ek})}\| \leq l \cdot B$. Define $\mathsf{sk}' := \sum_{i\in\mathcal{I}} \mathsf{sk}_i$ where the $\perp$ values are set to 0, and likewise for $\mathbf{e}^{(\mathsf{ek})} := \sum_{i\in\mathcal{I}} \mathbf{e}_i^{(\mathsf{ek})}$.
$\mathcal{A}$ outputs $\{\mathbf{b}' = -\mathbf{a}.\mathsf{sk}' + \mathbf{e}^{(\mathsf{ek})}, (\mathsf{sk}'_i)_{i\in\mathcal{I}}, (\mathbf{e}_i^{(\mathsf{ek})})_{i\in\mathcal{I}}\}$ 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 $\beta' = \beta$.

Fig. 16: IND-CPA under Joint Keys Game

The aim of this result is to be used in the broader context of MPC. That is the reason why, we consider that the honest key $\mathsf{ek} = (\mathbf{b}, \mathbf{a})$ is generated uniformly at random, instead of generated by BFV.KeyGen. Moreover, this specific use-case has an impact on the choice of parameters, which will be discussed in Assumption 2 and Equation (8).

Lemma 24. *Let* $\mathsf{pp} = (R_q, l, \mathscr{X}_q, R_k, \Psi_q)$ *be parameters such that Corollary 2 holds and* $\mathscr{B}_{\mathsf{Enc},q}$ *that satisfies Equation* (8). *Then for any* PPT *adversary* $\mathcal{A}$, *we have:*

$$Pr\left[\mathsf{JointKey}_{\mathcal{A}}(R_q, l, \mathscr{X}_q, R_k, \Psi_q, \mathscr{B}_{\mathsf{Enc},q}) = 1\right] - 1/2 = negl(\lambda).$$

Proof. Our goal is to show a reduction, from this IND-CPA under Joint Keys Game, into the $\mathsf{Game}_{\mathsf{Semantic}}$ game of security of BFV presented in Appendix D.1. We therefore construct an adversary $\mathcal{A}'$ playing the former game. $\mathcal{A}'$ uses as black box an adversary $\mathcal{A}$ for $\mathsf{JointKey}(R_q, l, \mathscr{X}_q, R_k, \Psi_q, \mathscr{B}_{\mathsf{Enc},q})$, as follows.

1. 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 $U(R_q^2)$ $(\beta = 0)$.
2. Then $\mathcal{A}'$ gives $\mathbf{b}$ to $\mathcal{A}$ and gets back $(\mathbf{b}' = -\mathbf{a} \cdot \mathsf{sk}' + \mathbf{e}^{(\mathsf{ek})}, \mathsf{sk}', \mathbf{e}^{(\mathsf{ek})}, m)$ from $\mathcal{A}$, where m is a challenge plaintext.
3. 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 β' 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 $\mathsf{c}_0 = u \cdot b + e_0^{(\mathsf{Enc})} \in R_q$ and $c_1^* = u \cdot a + e_1^{(\mathsf{Enc})} \in R_q$ for some $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(\mathsf{Enc})} \xleftarrow{\$} \mathscr{B}_{\mathsf{Enc},q}$, $e_1^{(\mathsf{Enc})} \xleftarrow{\$} \Psi_q$, and $b = \mathbf{b}[0]$, $a = \mathbf{a}[0]$, and with

$e^{(\mathsf{ek})} = \mathbf{e}^{(\mathsf{ek})}[0]$:

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

The last equality states a statistical indistinguishability between the distributions of $e_0^{(\mathsf{Enc})} - e_1^{(\mathsf{Enc})} \cdot \mathsf{sk}' - u \cdot e^{(\mathsf{ek})}$ and of $e_0^{(\mathsf{Enc})}$, which we now prove.

To start with, from equation (22), we have both $\|e_1^{(\mathsf{Enc})} \cdot \mathsf{sk}'\| \leq dnB$ and $\|u \cdot e^{(\mathsf{ek})}\| \leq dnB$. Thus, $\|e_1^{(\mathsf{Enc})} \cdot \mathsf{sk}' - u \cdot e^{(\mathsf{ek})}\| \leq 2dnB$. But on the other hand, $\|e_0^{(\mathsf{Enc})}\| \leq B_{\mathsf{Enc}}$. We conclude since the parameters are chosen such that $2dnB/B_{\mathsf{Enc}} = negl(\lambda)$ (cf Equation (8)). This conclusion can be formalized as the "smudging Lemma 25" below, which implies that, in the sum $e_0^{(\mathsf{Enc})} - e_1^{(\mathsf{Enc})} \cdot \mathsf{sk}' - u \cdot e^{(\mathsf{ek})}$, we have that the distribution of $-e_1^{(\mathsf{Enc})} \cdot \mathsf{sk}' - u \cdot e^{(\mathsf{ek})}$ is "smudged-out" by the one of $e_0^{(\mathsf{Enc})}$. Therefore, $\mathcal{A}'$ acts indistinguishably from the challenger of $\mathsf{Game}_{\mathsf{Semantic}}$ of Lemma 23, 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 25 (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.*

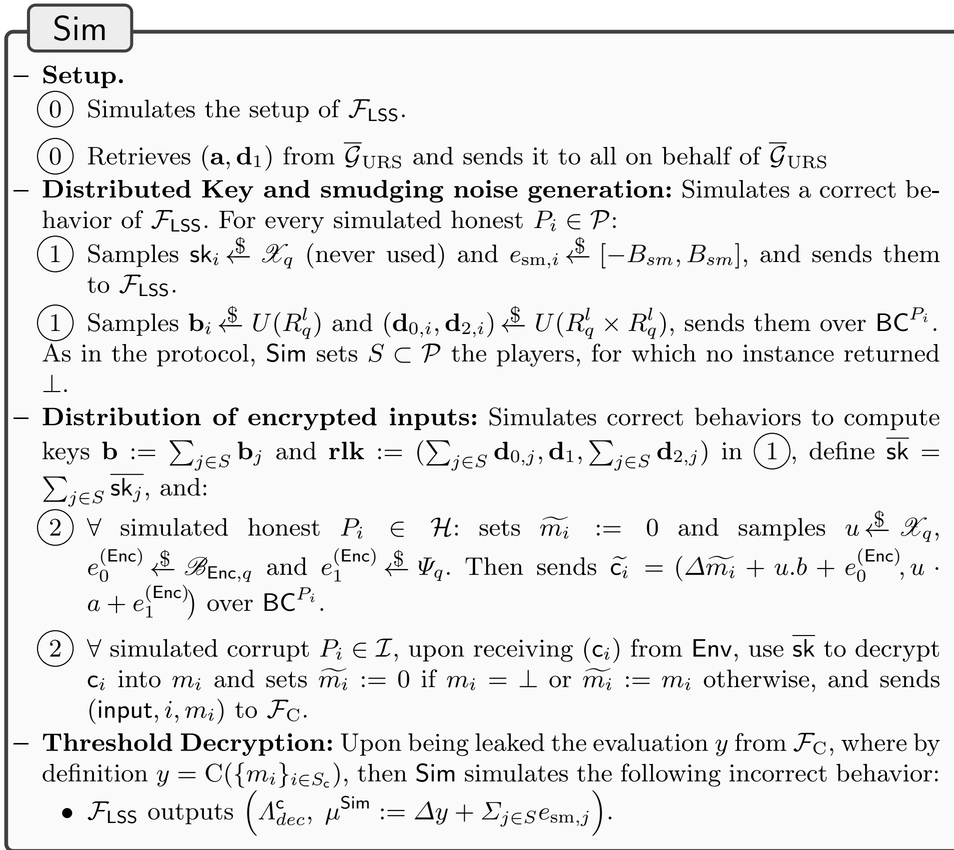
D.3 Description of the simulator

We describe in Fig. 17 our simulator Sim introduced in Section 6.2.1.

D.3.1 Hybrids, and proofs of indistinguishabilities We go through a series of hybrid games, starting from the real execution $REAL_{\Pi}$. The view of Env consists of its interactions with $\mathcal{A}$/Sim, and of the outputs of the actual honest players. We deal with the latter once and for all in Lemma 26.

Hybrid_1 ***[Simulated Decryption]****.* $\mathcal{F}_{\mathsf{LSS}}$ is modified in the threshold decryption: there it, incorrectly, outputs $\mu^{\mathsf{Sim}} := \Delta.y + \Sigma_{j \in S} e_{\mathrm{sm},j}$, where $y := \mathrm{C}((m_i)_{i \in S})$ is the evaluation in clear of the circuit on the actual inputs.

Lemma 26. *The outputs of the actual honest players are the same in $REAL_{\Pi}$ and $IDEAL_{\mathcal{F},\mathsf{Sim},\mathsf{Env}}$. Also, the views of Env in $REAL_{\Pi}$ and Hybrid_1 are computationally indistinguishable.*



Sim

- **Setup.**
  (0) Simulates the setup of $\mathcal{F}_{\mathsf{LSS}}$.

  (0) Retrieves $(\mathbf{a}, \mathbf{d}_1)$ from $\overline{\mathcal{G}}_{\mathrm{URS}}$ and sends it to all on behalf of $\overline{\mathcal{G}}_{\mathrm{URS}}$
- **Distributed Key and smudging noise generation:** Simulates a correct behavior of $\mathcal{F}_{\mathsf{LSS}}$. For every simulated honest $P_i \in \mathcal{P}$:
  (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 $\mathcal{F}_{\mathsf{LSS}}$.

  (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 BC^{P_i}.
  As in the protocol, Sim sets $S \subset \mathcal{P}$ the players, for which no instance returned $\perp$.
- **Distribution of encrypted inputs:** Simulates correct behaviors to compute keys $\mathbf{b} := \sum_{j \in S} \mathbf{b}_j$ and $\mathbf{rlk} := (\sum_{j \in S} \mathbf{d}_{0,j}, \mathbf{d}_1, \sum_{j \in S} \mathbf{d}_{2,j})$ in (1), define $\overline{\mathsf{sk}} = \sum_{j \in S} \overline{\mathsf{sk}}_j$, and:
  (2) $\forall$ simulated honest $P_i \in \mathcal{H}$: sets $\widetilde{m_i} := 0$ and samples $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(\mathsf{Enc})} \xleftarrow{\$} \mathscr{B}_{\mathsf{Enc},q}$ and $e_1^{(\mathsf{Enc})} \xleftarrow{\$} \Psi_q$. Then sends $\widetilde{\mathsf{c}_i} = (\Delta \widetilde{m_i} + u.b + e_0^{(\mathsf{Enc})}, u \cdot a + e_1^{(\mathsf{Enc})})$ over BC^{P_i}.

  (2) $\forall$ simulated corrupt $P_i \in \mathcal{I}$, upon receiving (c_i) from Env, use $\overline{\mathsf{sk}}$ to decrypt c_i into m_i and sets $\widetilde{m_i} := 0$ if $m_i = \perp$ or $\widetilde{m_i} := m_i$ otherwise, and sends (input, i, m_i) to $\mathcal{F}_{\mathrm{C}}$.
- **Threshold Decryption:** Upon being leaked the evaluation y from $\mathcal{F}_{\mathrm{C}}$, where by definition $y = \mathrm{C}(\{m_i\}_{i \in S_{\mathsf{c}}})$, then Sim simulates the following incorrect behavior:
  - $\mathcal{F}_{\mathsf{LSS}}$ outputs $\left(\Lambda_{dec}^{\mathsf{c}},\ \mu^{\mathsf{Sim}} := \Delta y + \Sigma_{j \in S} e_{\mathrm{sm},j}\right)$.

Fig. 17: Description of the simulator

Proof. It is convenient to prove the two claims at once. The view of Env is identical in $REAL_{\Pi}$ and Hybrid_1 until (3) included. There, for all $i \in S_{\mathsf{c}}$, a fresh BFV encryption c_i of m_i under $\mathsf{ek} = (b, a)$ is broadcast, following the terminology of Appendix C.3. Thus, the evaluated $\mathsf{c} := \mathsf{trBFV.Eval}(\mathrm{C}, \{\mathsf{c}_j\}_{j \in S}, \mathbf{rlk}, \mathbf{b})$ is the same in both views. In the threshold decryption of $REAL_{\Pi}$, the output of $\mathcal{F}_{\mathsf{LSS}}$ is:

$$\mu = \mathsf{c}[0] + \mathsf{c}[1].\Sigma_{j \in S} \mathsf{sk}_j + \Sigma_{j \in S} e_{\mathrm{sm},j} \ , \tag{31}$$

with $e_{\mathrm{sm},j} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ for all $j \in S$. First, by Definition 22, we have, for some noise $e^{(\mathsf{Dec})}$, with $\|e^{(\mathsf{Dec})}\| \leq B_{\mathrm{C}}$

$$\mathsf{c}[0] + \mathsf{c}[1].\Sigma_{j \in S} \mathsf{sk}_j = \Delta y + e^{(\mathsf{Dec})} \ . \tag{32}$$

Since $\|e_{\mathrm{sm},j}\| \leq B_{sm}$ for all $j \in S$, it follows from the choice of parameters (6) and the final remark in Definition 22, that the output of honest players in $REAL_{\Pi}$ is $y := \Omega_{\mathsf{Dec}}(\mu)$, which proves our first claim. Second, since we specified $\|e^{(\mathsf{Dec})}\|/n.B_{sm} = negl(\lambda)$ (equation (8)), it follows that the distribution of μ, given by Equation (31) is computationally indistinguishable from the one of $\Delta y + \Sigma_{j \in S} e_{\mathrm{sm},j}$ (see Lemma 25 for a further formalization of this fact). But the latter is by definition μ^{Sim}, which is exactly the output of $\mathcal{F}_{\mathsf{LSS}}$ in Hybrid_1. □

Hybrid_2 ***[Random Keys].*** This is the same as 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 encryption and relinearization keys, are replaced by a sample in $U(R_q^{l \times 3})$. Indistinguishability from Hybrid_1 follows from Corollary 2.

Hybrid_3 ***[Bogus Honest Inputs]*** This is the same as Hybrid_2 except that the input and randomness distribution on behalf of honest players are computed with $\widetilde{m_i} := 0$, instead of with their actual inputs m_i. Importantly, the behavior of $\mathcal{F}_{\mathsf{LSS}}$ is unchanged, i.e., correct until (3) included, then outputs $\mu^{\mathsf{Sim}} := \Delta y + \Sigma_{j \in S} e_{\mathrm{sm},j}$, where $y := \mathrm{C}((m_i)_{i \in S})$ is still the evaluation of the circuit on the *actual* inputs.

We now have that Hybrid_3 and $IDEAL_{\mathcal{F},\mathsf{Sim},\mathsf{Env}}$ produce identical views to Env. Indeed, the behaviours of $\overline{\mathcal{G}}_{\mathrm{URS}}$, of the simulated ideal functionalities $(\mathcal{F}_{\mathsf{LSS}}, \mathsf{BC})$, and of the honest players in Hybrid_3, are identical to the simulation done by Sim.

Lemma 27. Hybrid_2 *and* Hybrid_3 *are computationally indistinguishable.*

Proof. Since 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 BFV encryption 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 24 "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 BFV encryption 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. $\square$