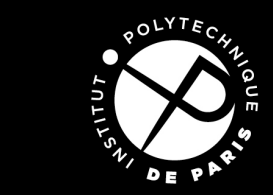
Efficient delegated secure multiparty computation

Antoine Urban



INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAT050

Thèse de doctorat

Efficient Secure Delegated Computation

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 Ecole doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 18/12/2024, par

ANTOINE URBAN

Composition du Jury :

Renaud SIRDEY Directeur de recherche, CEA, Université Paris-Saclay	Président/ Rapporteur
Chen-Da Liu ZHANG Directeur de recherche, HSLU	Rapporteur
Melek ONEN Professeur, Eurecom	Examinatrice
Phillipe GABORIT Professeur, Université de Limoges	Examinateur
Duong Hieu PHAN Professeur, Telecom Paris	Directeur de thèse
Matthieu RAMBAUD Maitre de conférence, Telecom Paris	Co-encadrant de thèse

Remerciments

Tout d'abord, je tiens à exprimer ma profonde gratitude à Matthieu Rambaud, dont l'expertise scientifique et l'enthousiasme ont constitué une véritable ligne directrice pour ma thèse. Je suis également profondément reconnaissant envers Hieu Phan pour sa disponibilité, sa bienveillance, ainsi que ses précieux conseils et commentaires, qui m'ont aidé à affiner mes recherches et mon approche de la résolution de problèmes.

Je souhaite également adresser mes sincères remerciements aux rapporteurs, Renaud Sirdey et Chen-Da Liu Zhang, pour le temps précieux qu'ils ont consacré à lire et à évaluer avec attention le manuscrit de ma thèse. Leurs commentaires constructifs et leurs remarques éclairées ont grandement contribué à améliorer la qualité de mon travail, et je leur en suis profondément reconnaissant.

Je tiens à remercier chaleureusement tous les membres du jury présents lors de la soutenance de mon doctorat – Melek Onen, Philippe Gaborit, Renaud Sirdey et Chen-Da Liu Zhang. Je leur exprime ma profonde gratitude pour leur présence, leur engagement, et l'intérêt qu'ils ont témoigné envers mes travaux de recherche.

Je remercie également de tous coeurs les doctorants et post-doctorants de l'équipe C2 du laboratoire LTCI, ainsi que l'ensemble des membres de Télécom Paris. Je suis ravi de voir que l'équipe s'est beaucoup agrandie dernièrement, et si je n'ai pas eu la chance de travailler avec tout le monde, j'ai sincèrement apprécié votre compagnie. Votre présence, votre soutien et votre amitié ont rendu mon doctorat d'autant plus enrichissant.

Enfin, je tiens à exprimer ma vive gratitude aux membres de ma famille et mes amis, à commencer par mes parents, pour leur soutien et leurs encouragements indéfectibles. A mon frère et ma soeur, merci de m'avoir encouragé et motivé tout au long de ce parcours. Je tiens également à remercier tous mes amis qui ont suivi de près ou de loin tout ce cheminement, avec une mention spéciale pour Gauthier et Franc cois pour m'avoir supporté pendant 3 ans.

Résumé

Avec l'essor des services de stockage et de calcul dans le cloud, il est désormais possible de déléguer la gestion des données à des infrastructures dématérialisées pour se concentrer uniquement sur leur analyse. Cette architecture permet également de combiner facilement des données issues de sources diverses pour en extraire des informations utiles. Cependant, la confidentialité des données externalisées demeure un défi crucial, freinant encore de nombreux cas d'usage lorsqu'il s'agit de données sensibles.

Le calcul sécurisé multipartite (MPC) offre une solution à ce problème. Il permet à un ensemble de n participants, chacun possédant ses propres inputs, de calculer une fonction f sur l'ensemble des données tout en préservant leur confidentialité et en garantissant l'exactitude des résultats. Dans cette thèse, nous adoptons le paradigme du cloud computing pour explorer la délégation sécurisée de protocoles MPC. Plus précisément, nous étudions un cadre où un ensemble d'input-owners délègue un calcul à des serveurs non fiables, responsables de l'essentiel des tâches computationnelles, tout en respectant des garanties strictes de sécurité et de confidentialité.

Pour répondre à ces besoins, nous nous appuyons sur des solutions basées sur le chiffrement complètement homomorphe (FHE). Cet outil puissant permet d'effectuer des calculs directement sur des données chiffrées, offrant ainsi des garanties de confidentialité. Trois exigences principales guident notre approche:

i) Limiter le protocole à un nombre fixe de rounds de communication. Plus précisément, quel que soit la fonction f à évaluer, nous exigeons que le protocole consiste en un nombre constant de rounds, incluant un ou plusieurs broadcasts initiaux suivi d'échanges peer-to-peer;

ii) Assurer la robustesse en garantissant que les participants honnêtes obtiennent le résultat correct même en cas de comportements malveillants d'une minorité. Plus précisément, nous exigeons que pour un certain seuil de corruption t, tant que $n - t$ participants prennent correctement part au calcul sécurisé, alors les participants honnêtes ont la garantie de recevoir le résultat, même si un sous-ensemble d'au plus t participants se comporte de manière malveillante (c'est-à-dire envoie des messages malformés ou abandonne). En pratique, cette propriété empêche tout participants d'obtenir un avantage déloyal en retenant le résultat final ou en sabotant le protocole;

iii) Permettre une délégation simple des calculs sans prétraitement complexe ni charge computationnelle excessive pour les propriétaires des données. Plus précisément, nous exigeons que la partie de l'évaluation coûteuse en termes de calcul soit déléguée par les "input-owners" vers un ensemble de participants non fiables, tout en préservant la confidentialité des inputs.

Dans la première partie de cette thèse, nous établissons un cadre théorique pour la construction de protocoles de calcul délégué. Ce framework se veut aussi général que possible afin de s'adapter à une grande diversité de contextes. Pour cela, nous proposons de dissocier les inputs des fonctions à évaluer. Ainsi, la délégation s'en voit faciliter puisqu'aucun calcul préalable spécifique à une fonction ne doit être effectué par les inputs-owners. De plus, les inputs distribués peuvent être réutilisées pour évaluer différentes fonctions sans nécessiter de nouvelle intervention de leurs propriétaires. Dans la deuxième partie de cette thèse, nous proposons deux protocoles répondant aux exigences pratiques que nous avons mis en avant. Ce faisant, nous présentons deux contributions principales:

1. Tout d'abord, nous introduisons le premier protocole de MPC robuste utilisant un schéma efficace de FHE basé sur l'hypothèse RLWE. Traditionnellement, la construction d'un protocole de MPC utilisant un schéma de FHE repose sur la génération de clés communes pour tous les participants, permettant de réaliser des calculs sur leurs différentes inputs. Cependant, la génération robuste de ces clés s'avère difficile dans le cas des schémas basés sur RLWE, et aucun travail antérieur n'a réussi à résoudre ce problème. Dans cette thèse, nous introduisons une nouvelle variante linéaire du cryptosystème BFV et montrons comment exploiter cette linéarité pour obtenir une génération robuste des clés;

2. Ensuite, nous introduisons une approche générique pour construire des protocoles de MPC permettant d'obtenir les meilleurs propriétés pratiques: permettre de facilement déléguer le calcule en utilisant un nombre optimal d'un unique broadcast initial, tout en permettant une évaluation efficace, y compris dans des scénarios à grande échelle impliquant un grand nombre de participants. Pour souligner les applications pratiques de ce nouveau protocole, appelé Share&Shrink, nous l'avons implémenté et testé par rapport aux méthodes de l'état de l'art.

Ces différentes avancées offrent des solutions concrètes aux défis posés par le MPC, en combinant simplicité, efficacité et robustesse. Elles ouvrent des perspectives prometteuses pour des applications pratiques dans divers contextes, où la sécurité des données et l'efficacité computationnelle sont cruciales, y compris dans des scénarios impliquant un très grand nombre de participants.

Abstract

With the rise of storage and computing services in the cloud, it has become possible to delegate data management to remote infrastructures, allowing users to focus solely on data analysis. This architecture also facilitates the seamless combination of data from various sources to extract valuable insights. However, ensuring the confidentiality of outsourced data remains a critical challenge, hindering many potential use cases involving sensitive information.

Secure multiparty computation (MPC) provides a solution to this problem. It enables a group of n participants, each holding private inputs, to compute a function f over their collective data while preserving the privacy of the inputs and ensuring the accuracy of the results. In this thesis, we adopt the cloud computing paradigm to explore the secure delegation of MPC protocols. Specifically, we examine a framework where a group of input-owners delegates computations to untrusted servers, which handle the bulk of the computational workload while adhering to strict security and confidentiality guarantees.

To meet these needs, we rely on solutions based on fully homomorphic encryption (FHE). This powerful tool enables computations directly on encrypted data, ensuring data confidentiality. Three key requirements guide our approach:

i) Limiting the protocol to a fixed number of communication rounds. Specifically, regardless of the function f to be evaluated, we require that the protocol consist of a constant number of rounds, including one or more initial broadcasts followed by peer-to-peer exchanges.

ii) Ensuring robustness by guaranteeing that honest parties obtain the correct result even in the presence of malicious behavior by a minority. Specifically, we require that for a certain corruption threshold t, as long as $n - t$ parties correctly take part in the secure computation, the honest parties are guaranteed to receive the result, even if a subset of at most t participants behaves maliciously (e.g., by sending malformed messages or aborting). In practice, this property prevents any participant from gaining an unfair advantage by withholding the final result or sabotaging the protocol.

iii) Enabling simple delegation of computations without complex preprocessing or excessive computational burden on the input-owners. Specifically, we require that the computationally expensive part of the evaluation be delegated by the input-owners to a set of untrusted parties, while preserving the confidentiality of the inputs.

In the first part of this thesis, we establish a theoretical framework for constructing delegated MPC protocols. This framework is designed to be as general as possible to adapt to a wide range of contexts. To achieve this, we propose decoupling the inputs from the functions to be evaluated. This separation simplifies delegation since no prior computation specific to a function needs to be performed by the input-owners. Moreover, the distributed inputs can be reused to evaluate different functions without requiring further intervention from their owners.

In the second part of this thesis, we propose two protocols that meet the practical requirements we have outlined. In doing so, we present two main contributions:

1. We introduce the first robust MPC protocol using an efficient FHE scheme based on the RLWE assumption. Traditionally, constructing an MPC protocol from an FHE scheme relies on the generation of common keys for all participants, enabling computations on their respective inputs. However, robust key generation has proven challenging for RLWE-based schemes, and no prior work has successfully addressed this issue. In this thesis, we introduce a new linear variant of the BFV cryptosystem and demonstrate how to leverage this linearity to achieve robust key generation.

2. Next, we present a generic approach for constructing MPC protocols that achieve the best practical properties: enabling easy delegation of computation using an optimal single initial broadcast, while allowing efficient evaluation, even in large-scale scenarios involving many participants. To highlight the practical applications of this new protocol, called Share&Shrink, we implemented it and tested it against state-of-the-art methods.

These various advancements provide concrete solutions to the challenges of MPC, combining simplicity, efficiency, and robustness. They open up promising prospects for practical applications in diverse contexts where data security and computational efficiency are critical, including scenarios involving a very large number of participants.

Contents

List of Figures

Introduction and Technical Overview

In this dissertation, we consider the problem of secure multiparty computation (or MPC for short), in which a number of participants want to compute a function f on inputs while guaranteeing interesting security properties such as the privacy of the inputs and the correctness of the computation. In particular, we will focus on the setting where a number of participants, known as *input-owners*, provide inputs, and where n additional parties, known as the *computation parties*, are responsible for offloading as many computing tasks as possible from the input-owners. Importantly, both input-owners and computation parties should trust each other as little as possible.

This setting is especially relevant with the rise in popularity of cloud storage and analytics. These advancements allow users to delegate data management concerns (scalability, availability, infrastructure) and focus solely on their analysis. While combining data from multiple sources offers valuable insights (e.g., in medical research, advertising), ensuring user privacy from cloud providers during this analysis is a significant challenge, which can have a direct impact on the choice of whether or not to carry out a certain processing operation on private data. Therefore, it is not surprising that the construction of practical secure MPC protocols suited for delegation has received a lot of interest.

Numerous techniques [BFLS91; KMR11; PRV12; CF13; BFR13; GKR15; PHGR16; SVV16; WJB+17; CCKP19; FNP20; WHZ24] have been proposed to achieve verifiable delegation of computation. These techniques allow one (or more) resource-constrained parties to delegate a computation to a powerful, untrusted server. They ensure that the returned result is computed correctly, even if the powerful server tries to cheat, and that the work required to verify the correctness of the result is significantly less than the work needed to compute the function itself. Motivated by practical scenarios, a number of recent works [JNO14; BIK+17; BBG+20; RSY21] focus on large-scale secure computation, where a function needs to be evaluated on thousands or even millions of private inputs. To this end, for instance, Alon et al. [ANOS24] recently proposed the GMPC model, specially designed to capture the scenarios in which a very large company (such as Apple, Meta, or Google) is interested in executing a secure computation over its users' data[1]. There, it is reasonable to assume that users (e.g. phones) can communicate only via the server in a star-network, since it's not practical to have, say, 100M users communicating with each other arbitrarily. However, this efficient server-aided model relies on the continued functionality and security of the server. Any server malfunction or compromise could disrupt the entire computation.

[1] Variations of this setting have been used in large-scale real-world applications, such as for health statistics through the COVID-19 Exposure Notification system developed jointly by Apple and Google [21], or for secure advertising measurement by Meta [23b; MMT+24], that is used for queries with up to 1 billion records.

In this dissertation, we focus on the secure delegation of a computation that must be practical in the most generic setting possible, and, in particular, even when a supposedly reliable helper server is not available. We therefore introduce the following set of practical requirements:

Constant-round MPC. We require that a group of mutually distrustful parties is able to securely compute a function of their inputs in a constant number of communication rounds. Following Yao's seminal work [Yao82], garbled circuits have become a core concept for 2-party secure computation. They enable one party to "encrypt" a function (circuit) while another party privately evaluates it. For more than two parties, Beaver, Micali, and Rogaway [BMR90] proposed the BMR protocol, which allows for a dishonest majority of parties to securely evaluate a function of their inputs while maintaining a constant-round communication complexity. Overall, the latter consists of two steps: first, an offline phase that involves the secure garbling of a circuit, independent of the actual inputs. Then, an input-dependent online phase, in which parties simply send some keys before locally evaluating the circuit. Subsequently, an extensive number of constant-round protocols have been proposed [LSS16; WRK17; ACGJ18; LPSY19; ABT19; ACGJ19; HSS20; ACGJ20] mainly following variations of this BMR protocol. However, garbled-circuit-based protocols seem to inherently have a scaling issue with a large number of participants. Indeed, the circuit size typically grows linearly with the number of parties, leading to increased data processing and communication bandwidth requirements.

Another classic approach for building constant-round MPC protocols [AJL+12; LTV12; GLS15; MW16; BJMS20; Par21; DMR+21; BMMR23] is to use a homomorphic encryption scheme. The latter is a versatile tool for locally evaluating a function on encrypted inputs and is the main focus of this dissertation.

Guaranteed Output Delivery (GOD) under honest majority. We require that for some corruption threshold t, as long as $n - t$ parties correctly participate in the secure computation, then honest parties have the guarantee to receive the output, even if a subset of at most t parties behave maliciously (i.e. send malformed messages or abort). This property, also denoted as *robustness* when a protocol needs to produce a correct output in a *constant number of rounds*, is particularly important and has been studied in many contexts, such as for threshold signature schemes [GJKR96; Sho00; SS01; Bol02; RRJ+22], distributed key generation protocols [KMM+23; Kat24], decentralized random beacons protocols [CD17; KRDO17; GLOW21], or MPC [IKLP06; GLS15; BJMS20]. In practice, this property prevents any party from gaining an unfair advantage by withholding the final result or sabotaging the process [CL17]. For instance, consider several companies collaborating on a confidential market research project on some private data (e.g., customer demographics). It is paramount that all participating companies receive the final market analysis report, even if some of them experience temporary internet outages or if a competitor attempts to manipulate the outcome. Without this guarantee, companies are unlikely to participate in the protocol in the first place for fear of giving away a competitive edge.

In this dissertation, we will consider a set of n parties, of which a minority of $t < n/2$ of them can be corrupted by an adversary (i.e. we set $n = 2t + 1$, also denoted as *honest majority*), and will guarantee a robust execution, i.e. honest parties output the computation result in a constant number of rounds whatever the behavior of the adversary.

Delegability. We require that most of the computationally expensive evaluation must be out-

sourced by the so-called *input-owners* to a set of n untrusted computation parties while safeguarding the confidentiality of the inputs. Note that contrary to some MPC protocols [GGP10; BGV11] that require input-owners to perform a costly pre-processing phase before delegating a computation, we require that no function-specific data be sent or known by the input-owners before the computation begins.

RLWE. Finally, let us recall that in this dissertation, we focus on secure computation from fully homomorphic encryption (FHE, formally defined in Chapter 1). In recent years, several generations of FHE schemes have been proposed, with the latest based on the ring-learning-with-errors (RLWE) assumption [LPR13a] gaining traction through implementation [23c], and standardization [ACC+21]. For efficiency, we therefore require our protocols to be as scheme-independent as possible, and in particular, to be built from the latest efficient FHE schemes based on the RLWE assumption.

Finding a way to securely and practically delegate a computation with these properties is not straightforward and poses several challenges that we address in this dissertation.

Challenges and Contributions

We now turn to the main challenges we have faced, and the new solutions we have introduced to achieve an efficient and practical secure delegation of any computation.

Challenge 1: Building a Robust threshold-FHE scheme based on RLWE

Our first goal is to design a robust MPC protocol based on a fully homomorphic encryption (FHE) scheme. To this end, the common start of FHE-based MPC protocols [CDN01; DPSZ12; AJL+12; CLO+13; BGG+18; MTBH21; BDO23; IKC+24] is a *distributed key generation* protocol (DKG) executed by all n parties. During the latter, each party P_i generates a contribution sk_i and takes part in the multiparty protocol to reach the following starting point:

$$\mathsf{sk}_S = \mathsf{Sk}_S(\{\mathsf{sk}_i\}_{i\in S}), \mathsf{ek}_S = \mathsf{Ek}_S(\mathsf{sk}_S), \tag{1}$$

i.e. a shared secret/encryption key pair, where Sk_S is an algorithm for generating a secret key from a set of contributions coming from a subset $S \subseteq [n]$ of indices of parties who did not abort during the DKG, and Ek_S is an algorithm for generating the corresponding encryption key. S is a critical subset, which we will discuss later. Importantly, the secret key sk_S is never known to any single party.

To run their homomorphic evaluation algorithm, some FHE schemes require the generation of some additional common key, denoted as *relinearization key*[2]. In a similar way to the above-mentioned DKG protocol, one therefore needs another relinearization key generation protocol (RlkGen) to generate this key:

$$\mathbf{rlk}_S = \mathsf{RLK}_S(\mathsf{sk}_S), \tag{2}$$

where RLK_S is an algorithm to generate the relinearization key that corresponds to the same secret key sk_S presented in Equation (1) generated by the DKG.

[2]Or sometimes as *evaluation key*.

However, although conceptually similar to the well-known DKG protocols, the design of these RlkGen protocols is proving challenging, and no solution, to be used for building an MPC protocol with the properties we want, currently exists. Overall, two problems emerged:

1. First, some protocols [GLS15; BHP17; BGG+18; CSW23] completely bypassed the generation of a relinearization key and have chosen not to use the latest RLWE-based schemes, but instead other LWE-based [Reg05] schemes such as GSW [GSW13]. The former does not require any such relinearization key. However, this comes at the cost of a less efficient homomorphic evaluation.
2. Second, a large number of recent works [KJY+20; MTBH21; Par21] proposed RlkGen protocols for generating a relinearization key. Unfortunately, none of them are robust; as soon as one party in S deviates, no relinearization key is generated, which prevents the design of a robust MPC protocol.

In this dissertation, we focus on the latter issue. The robust generation of a common relinearization key is therefore our first goal.

Contribution 1: The first robust Key Generation for RLWE-based threshold-FHE scheme.

Our first contribution, presented in Chapter 3, is to design a protocol for reaching a distributed state in which the parties simultaneously generate, the secret key sk_S, and the keys ek_S and $\mathbf{rlk}_S$ corresponding to sk_S introduced in Equations (1) and (2), all in one round. As a result, we have that either a party aborts or is in S. Once this state is reached - and this is the main novelty- it no longer matters whether a party in S aborts or not as in previous works [KJY+20; MTBH21; Par21]. Indeed, $t+1$ cooperating parties[3] are always able to decrypt a ciphertext c with the shared key sk_S and eventually output the result.

In other words, this contribution enables the design of the first robust MPC protocol under honest majority from a RLWE-based FHE scheme.

Technical Summary:

We now provide technical explanations about our contribution that can be found in greater detail in Chapter 3. We take the BFV [FV12] cryptosystem as an example of a RLWE-based FHE scheme, in which the relinearization key is described, for a secret key sk, as being of the following form:

$$\mathbf{rlk} = (\mathsf{sk}^2\mathbf{w} - \mathsf{sk}\cdot\mathbf{r} + \mathbf{e}^{(\mathbf{rlk})}, \mathbf{r}) \tag{3}$$

where $\mathbf{r}$ is a uniform random string, $\mathbf{e}^{(\mathbf{rlk})}$ some noise, and w a decomposition basis of dimension some l, i.e $\mathbf{w} = (w^0, w^1, \ldots, w^{l-1})^T$. The presence of the term $\mathsf{sk}^2\mathbf{w}$ in Equation (3) introduces a challenging non-linearity. To overcome the latter, various RlkGen protocols [KJY+20; Par21; MTBH21] for generating $\mathbf{rlk}$, have been proposed. Overall, they have the following informal structure:

[3] Which is always possible since we assume an honest majority, i.e. $n = 2t+1$.

- In round 1: each party P_i generates a contribution $\mathbf{rlk}_{0,i}$ using its key sk_i, and sends it. It also shares its secret key via *Publicly Verifiable Secret Sharing* (PVSS) [Sta96; Sch99]. The latter primitive, which is described in detail later in Chapter 2[4], is commonly used in many distributed key generation protocols [CD17; CD20; Gro21; GV22; Kat24; CD24] to generate a shared secret in one round, so that each party possesses a share of this secret.
- In round 2: each party 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 parties in the first round. Then, each P_i uses an algorithm RelinKeyGen to compute a final contribution $\mathbf{rlk}_i \leftarrow \mathsf{RelinKeyGen}(\mathbf{rlk}_0, \mathsf{sk}_i)$ and sends $\mathbf{rlk}_i$. Also, note that the shared secret key roughly consists in the sum $\mathsf{sk}_S = \sum_{i \in S} \mathsf{sk}_i$ of the keys sk_i over the set S.

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

However, in this generic protocol, illustrated in Figure 1, if some parties take part in some of the rounds, but not all, then no $\mathbf{rlk}$ 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 $\mathbf{rlk}_{S'}$ would be incompatible with the ek_S produced in the first round, making the overall key generation non-robust.

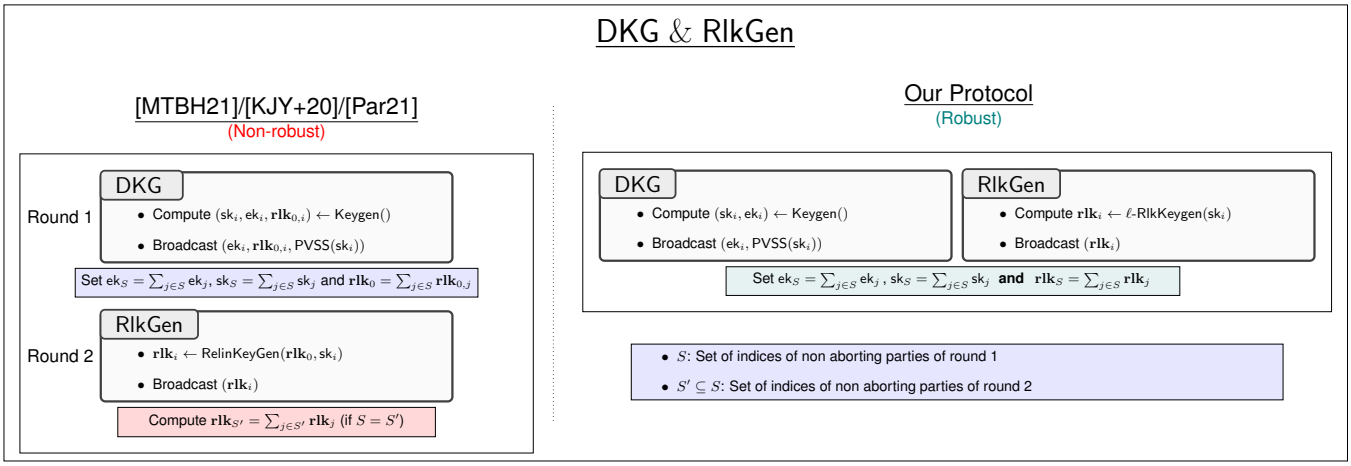


Figure 1: We present on the left-hand side the overall construction of previous protocols [KJY+20; Par21; MBH23]. First, each party P_i runs Keygen to produce keys $(\mathsf{sk}_i, \mathsf{ek}_i, \mathbf{rlk}_{0,i})$. Contributions are added together over the set S of indices of non-aborting parties to form the common secret, threshold encryption, and intermediate relinearization keys sk_S, ek_S and $\mathbf{rlk}_0$. Then, parties run RelinKeyGen 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 parties of this second round, parties can compute the relinearization key $\mathbf{rlk} = \sum_{j \in S'} \mathbf{rlk}_j$ if $S = S'$. On the right-hand side, we present a sketch of our protocol. More specifically, to have robustness, parties run in parallel Keygen and our relinearization key generation algorithm ℓ-RlkKeygen.

[4]It roughly consists in parties publishing encrypted shares of secrets along with proofs of share validity, so that all parties are able to non-interactively verify whether a given sharing is correct, and to obtain a share, so that a set of shares can be used to reconstruct the shared secret.

We overcome these issues by detailing a ℓ-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_S, *not a squaring*. This results in a linear variant of BFV, denoted as ℓ-BFV, which, as shown in Figure 1, enables the design of a RlkGen protocol that generates $\mathbf{rlk}_S$ in only one round, running *in parallel* with the DKG. This process yields a robust overall key generation, as summarized in Table 1.

	# Rounds for DKG & RlkGen	Robustness
[KJY+20]	2	✗
[Par21]	2	✗
[MTBH21]	2	✗
Chapter 3	1	✓

Table 1: DKG and RlkGen properties for various Threshold-FHE schemes.

What we ultimately end up with is a robust threshold-FHE scheme based on RLWE, which we call trBFV. It enables the construction of a robust MPC protocol, as described later in Figure 2a. Once a common threshold encryption key has been generated, encrypted inputs can be distributed, and parties are able to (locally) perform homomorphic computations on the ciphertexts to evaluate the desired function. Finally, parties jointly execute a threshold decryption protocol to uncover the computation's output.

Challenge 2: MPC when only one initial broadcast is available

Overall, MPC protocols can be characterized in terms of (i) the security guarantees they achieve, (ii) the setup they require, (iii) the corruption threshold t they support, and (iv) the kinds of communication required in each round (broadcast, peer-to-peer channels,...). It is known that secure computation is possible in two broadcast rounds [GIKR02; AJL+12; GLS15; DHRW16; MW16; BLPV18]. However, in practice, implementing a broadcast channel requires either multiple rounds of peer-to-peer communication [DR85; FH06; KK06; GP21; WMR+23] or special channels (such as blockchains), and is therefore costly and difficult in itself. This difficulty was for instance pointed out in the comments[5] of NIST's recent call for standardization of multiparty threshold-cryptography[23a]. For this reason, a number of recent works [CGZ20; RU21; DMR+21; GJPR21; BMMR23; DRSY23] have undertaken to fully characterize MPC protocols that are *broadcast-optimal*, i.e. that require only one broadcast in one of the rounds. In this dissertation, we restrict the study to protocols with GOD under honest majority, provided with one initial broadcast followed by any number of asynchronous peer-to-peer communication rounds.

[5] e.g. From J. Katz, C. Komlo, X. Meng, and N. Smart *"We have observed that protocol implementers are often unsure about how to implement a "broadcast channel" in practice, and there is no general guidance available about how protocols that rely on a broadcast channel should be implemented in a point-to-point network. We therefore encourage NIST to require submissions to either explicitly state that their security analysis assumes a broadcast channel (and then suggest how such a channel should be implemented), or otherwise provide a protocol specification and proof of security in the point-to-point communication model (i.e., without the assumption of a broadcast channel)."*

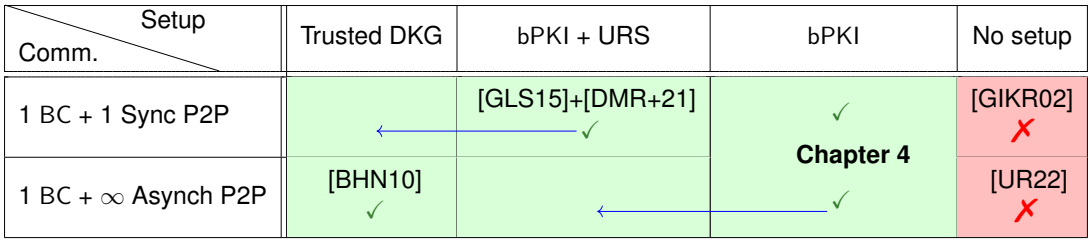


Table 2: Feasibility and impossibility of MPC with GOD under honest majority with different setups and communication patterns. URS stands for a public uniform random string, and bPKI for a bulletin-board PKI.

With these requirements, various setups can be considered:

- *Trusted* DKG *setup*, i.e. when a common threshold encryption key and secret key shares have been distributed to the parties by a trusted dealer. Under this setup, Beerliová, Hirt and Nielsen [BHN10] have shown the feasibility of MPC with GOD under honest majority in one broadcast.
- *Bulletin Board PKI* (bPKI) + URS *setup*, i.e. when a bare bulletin board of public keys (as detailed in Section 1.9) is available to parties, and a single uniform random string is given. Under this setup, Damgård et a. [DMR+21] have shown feasibility of MPC in one broadcast.

Thus, what remains to be seen is what is achievable when a bulletin board PKI is available but no URS.

Contribution 2: Feasibility of MPC with the sole setup of a bulletin board PKI

In Chapter 4, we complete the theoretical picture of honest majority MPC with GOD from one initial round of broadcast (BC) followed by a number of peer-to-peer (P2P) communications, and show its feasibility in the sole setup of a bulletin board PKI. The overall (in)feasibility picture is summarized in Table 2.

Challenge 3: Delegated MPC in one Broadcast with an Efficient Evaluation

Our last goal is to address the shortcomings of the previous threshold-FHE based MPC protocols [AJL+12; BGG+18; KJY+20; MTBH21; Par21; BDO23], as well as our protocol introduced in Chapter 3, namely the need for two initial broadcast rounds before the evaluation: a first one for the generation of the common keys[6] and the second one for the distribution of the encrypted inputs. However, as discussed previously, the use of broadcast is costly in practice. Therefore, significant effort is currently put to remove or minimize the use of broadcast in MPC protocols [FN09; BHN10; GGOR13; PR18; CGZ20; GJPR21; DMR+21; DRSY23], under different setup settings.

[6] A common threshold encryption key and a common relinearization key

Overall, previous attempts [GLS15; BJMS20] to reduce the number of broadcast rounds in FHE-based MPC protocols are based on multikey-homomorphic encryption schemes [LTV12]. The latter enables a homomorphic evaluation directly over ciphertexts encrypted under different keys. Thus, by using such schemes, no DKG is needed before the distribution of the inputs, effectively reducing the number of broadcasts to just one: parties start with the local generation of keys, directly followed by the (unique) broadcast of inputs encrypted under the different keys. However, the use of a multikey scheme is not without its own issues and leads to two main practical problems:

Problem 1: Delegability.

In [GLS15], Gordon et al. proposed the first robust threshold multikey-FHE scheme requiring one single broadcast. Unfortunately, their scheme is not compatible with efficient RLWE-based FHE schemes and most importantly, the MPC protocol they built from it cannot enable delegation, i.e. input-owners cannot easily outsource the costly homomorphic evaluation to a set of n untrusted computation parties.

Problem 2: Efficient Evaluation.

To remedy the former issue, Badrinarayanan et al. [BJMS20] proposed another approach for building an MPC protocol with GOD in one broadcast that leverages a more generic usage of a multikey-HE scheme, enabling the use of efficient RLWE-based schemes. However, their protocol still suffers from a major drawback: the sizes of the (multikey) ciphertexts are at least linear in the number $|\mathcal{Q}|$ of *input-owners* who distributed an input. As a consequence, the multikey evaluation is at least $|\mathcal{Q}|\times$ less efficient than when using a threshold-FHE scheme[7], which prevents use-cases involving a large number of inputs.

Our third goal is therefore to propose a new FHE-based delegated MPC protocol that offers the best of both threshold-FHE based [KJY+20; MTBH21; Par21; MBH23] and multikey-FHE based [GLS15; BJMS20] protocols, i.e. an efficient delegable homomorphic evaluation with only one initial broadcast.

Contribution 3: Share&Shrink: A Generic Protocol for Efficient Delegated MPC with GOD from one broadcast.

In Chapter 4, we propose a new generic protocol, denoted Share&Shrink, that enables building a *broadcast-optimal* MPC protocol, while performing the homomorphic evaluation on ciphertexts whose size does not depend on the number of inputs, leading to an efficient computation. Importantly, this generic protocol can be built from various schemes, including our linear RLWE-based ℓ-BFV scheme introduced in Chapter 3, allowing for greater modularity according to the use-cases considered. We summarize in Table 3 the properties of our protocol as well as the previous works.

Technical Summary:

We now provide technical explanations about our contribution that can be found in greater detail in Chapter 4. Overall, Share&Shrink departs from previous threshold-FHE based protocols [KJY+20; MTBH21; Par21] and performs *in parallel*: a DKG protocol, and a distribution of

[7] In which, we recall, the evaluation is independent of the number of distributed inputs.

Protocol	1 BC + asynch P2P	GOD for $t < n/2$	Delegability	Size of ciphertexts	RLWE Compatible
[KJY+20][MTBH21] [Par21][MBH23]	✗	✗	✓	$\lvert\mathscr{C}\rvert$	✓
Chapter 3 (trBFV)	✗	✓	✓	$\lvert\mathscr{C}\rvert$	✓
[GLS15]	✓	✓	✗	$\lvert\mathscr{C}\rvert$	✗
[BJMS20]	✓	✓	✗	$\lvert\mathcal{Q}\rvert \cdot \lvert\mathscr{C}\rvert$	✓
Chapter 4	✓	✓	✓	$\lvert\mathscr{C}\rvert$	✓

Table 3: MPC for $n = 2t + 1$ parties and $|\mathcal{Q}|$ input-owners. Here, $|\mathscr{C}|$ represents the size of a regular FHE ciphertext.
The last column refers to the ability for the protocol to be built from efficient RLWE-based FHE schemes. The "Size" is the one of the ciphertexts which undergo homomorphic evaluation. The "Delegability" refers to the ability for computation to be outsourced to a set of untrusted computation parties. We did not mark as GOD the protocols that must be completely restarted when one party aborts in the middle.

ciphertexts of inputs under a common threshold encryption key. This allows obtaining in only one broadcast, ciphertexts of sizes independent of the number $|\mathcal{Q}|$ of input-owners, and thus an efficient evaluation.

The main intuition is that some encryption schemes, and notably ℓ-BFV, the linear variant of BFV we introduced in Chapter 3, can be expressed as a set of linear maps. Importantly, this includes the key generation, encryption, and decryption algorithms, as later defined in Section 1.10.4. In particular, encryption consists of the evaluation of a linear map $\Lambda_{\mathsf{Enc}}^{\mathsf{ek}}$, parametrized by a threshold encryption key ek, over the secret input (and some encryption randomnesses). Provided with linearly secret-shared inputs, encryption then consists in opening $\Lambda_{\mathsf{Enc}}^{\mathsf{ek}}$ evaluated over the shared inputs, which can be done in one step of *peer-to-peer asynchronous messages* thanks to the properties of the sharing[8].

Overall, our Share&Shrink protocol is outlined in Figure 2b, and can be synthesized as follows:

1. **Share.** Parties run a DKG protocol in one round of broadcast. The pattern is the same as in [FS01]. Namely, each party P_i generates an additive contribution sk_i to the secret key, and ek_i to the threshold encryption key. It broadcasts ek_i and shares sk_i (as previously, in the form of a PVSS [Sta96; Sch99]).

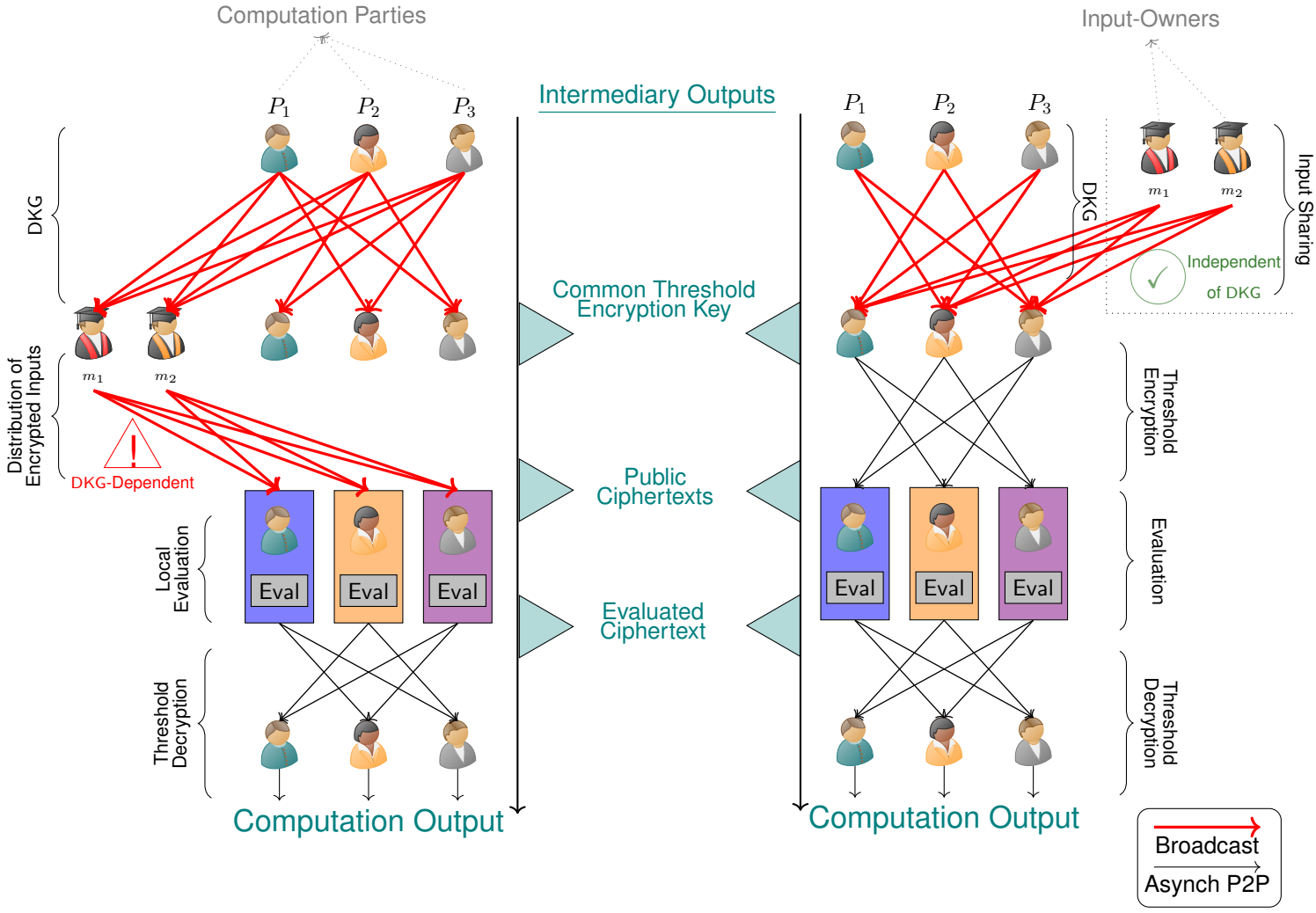
   In parallel, input-owners also share their inputs and encryption randomnesses via PVSSs[9].
2. **Shrink.** Each party locally sets the common *threshold encryption key* ek as, roughly, the sum of the ek_i's for which the sk_i's were correctly shared. Parties then perform the threshold encryption of the shared inputs under ek in one step of peer-to-peer messages using the previously introduced linear map $\Lambda_{\mathsf{Enc}}^{\mathsf{ek}}$, thereby "shrinking" them down to the sizes of ciphertexts encrypted under this single ek.

In the end, parties obtain a common view on inputs encrypted under a common threshold encryption key ek. They can thus proceed with the efficient evaluation and later the threshold

[8] This has been used extensively in many protocols, see e.g. [CDD+98; BBPT14; CDGK22; AAPP23]
[9] Which, we recall, is described in Chapter 2

decryption, which requires no further broadcast.

Importantly, as highlighted in Figure 2, the major difference with previous approaches based on threshold-FHE schemes [KJY+20; Par21; MBH23], is that the input distribution is not dependent on the DKG. In other words, inputs are distributed without any knowledge of the encryption key under which they will be encrypted, which means they can be shared in parallel with the DKG. This reduces the number of initial broadcasts to just one, and, as a bonus, enables input-owners to remain *lightweight*, in the sense that they do not need to participate, monitor, or receive the outputs of a DKG protocol, nor implement the full FHE libraries. Say differently, they can simply share their inputs and leave the protocol.



(a) Overview of Threshold-FHE based MPC.

(b) Overview of Share&Shrink

Figure 2: Comparison of the threshold-FHE-based approach in 2 broadcasts, and our broadcast-optimal Share&Shrink protocol for delegating to some computation parties the secure evaluation of some circuit on inputs provided by a set of input-owners.

Finally, let us compare our approach to previous broadcast-optimal protocols in terms of practical computation complexity. To this end, we consider the protocol of [BJMS20] in-

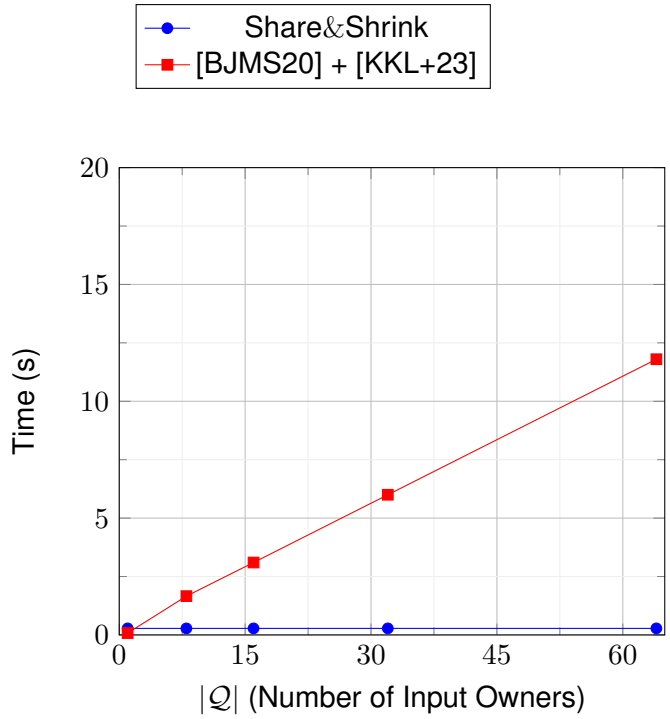


Figure 3: Comparison of median times for a multiplication between two ciphertexts, when using [BJMS20] instantiated from the multikey scheme of [KKL+23] and our Share&Shrink method instantiated leveraging the ℓ-BFV scheme introduced in Chapter 3. 100 runs for various number of input-owners were executed.

stantiated with the efficient multikey scheme of [KKL+23] based on BFV [FV12], and our Share&Shrink protocol instantiated leveraging the ℓ-BFV scheme introduced in Chapter 3. In Figure 3, we compare the running time of a multiplication between two ciphertexts for different numbers of input-owners ranging from 1 to 64. Overall, this verifies that the running time of a multiplication grows almost linearly with the number of input-owners when using a multikey scheme, while Share&Shrink brings down this duration to a small constant, independent of the number of input-owners as expected.

Details about this experiment, including implementation and parameter selection, are further discussed in Chapter 4.

Organization

This dissertation is organized as follows:

- In Chapter 1, we present the main concepts and definitions relevant to describe FHE-based MPC protocols.
- In Chapter 2, we formalize a general framework for linear secret sharing, that is used throughout this dissertation for constructing threshold schemes.
- In Chapters 3 and 4, we present our main aforementioned contributions.
- In Chapter 5, we conclude this dissertation and discuss future research directions.

Chapter 1

Definition and Constructions

Contents

The goal of this first chapter is to introduce the reader to some of the key theoretical existing concepts in the area of secure multiparty computation. First in Sections 1.1 and 1.2 we introduce some basic terms and notations to present our main topic of study, Multiparty Computation, in Sections 1.3 to 1.6. From Sections 1.7 to 1.9, we detail our formalism, before defining in Section 1.10 primitives that will be used throughout this work. Finally, in Sections 1.11 and 1.12, we give an overview of the design of an MPC protocol which will be refined and detailed in the following sections.

1.1 Terminology

For greater clarity, we now introduce the main terms that will be used to describe our multiparty cryptographic constructions.

System. It refers to a set of participants and rules according to which a cryptographic goal is achieved.

Party. It refers to each participant in a system, modeled as probabilistic interactive Turing machines [GMR85] that run in polynomial time (in some security parameter λ).

Algorithm. It refers to any non-interactive cryptographic function, i.e. a set of instructions executed locally without any communication between parties.

Protocol. It refers to any interactive cryptographic function, i.e. a set of algorithm executions composed of communication between parties.

Let us give a concrete example of these notions. We will later refer to a system whose goal is to securely compute a function applied on a number of inputs held by a set of n parties. It outputs the result through a "threshold decryption" protocol, that consists, as per the definition, of a set of local instructions, referred to as a "decryption" algorithm, executed by the different parties and followed by some interactions between them.

1.2 Notation

We now introduce some notations that will be used throughout this work. We denote $x \xleftarrow{\$} \mathscr{D}$ the sampling of x according to distribution $\mathscr{D}$, and $\overrightarrow{\mathscr{D}}$ a vector of distributions. Cardinality of a set X is denoted as $|X|$. For a finite set E, we denote $U(E)$ the uniform distribution on E. The set of positive integers $[1, \ldots, n]$ is denoted $[n]$. We denote by λ the security parameter throughout the dissertation. $\{0,1\}^*$ denotes bitstrings of arbitrary lengths. All logarithms are in base two.

We consider a positive integer d, 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 respectively. 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 $\mathbf{u}^{\langle \cdot \rangle}(\mathbf{v}, \mathbf{w}) := \big(\langle \mathbf{u}, \mathbf{v} \rangle, \langle \mathbf{u}, \mathbf{w} \rangle\big)$. Finally, $\otimes : R_q^2 \times R_q^2 \to R_q^3$ denotes the tensor product.

All linear forms are are succinctly specified as *formal linear combinations*, e.g., let $(\overline{x_i})_i$ denote *labels* of some variables $(x_i)_i$, then, $\sum_i l_i \overline{x_i}$ denotes $\big\{(x_i)_i \to \sum_i l_i x_i\big\}$. In a general sense, a linear map between abelian groups $g : (E, +) \to (F, *)$ is such that $g(e_1 + e_2) = g(e_1) * g(e_2)$, with some further compatibility to multiplication by scalar constants (either polynomials, or integers, possibly modulo some prime p or prime power p^e, depending on the cases).

1.3 Multiparty Computation (MPC)

In secure multiparty computation (MPC), we consider a setting where n parties $\mathscr{P} = \{P_1, \ldots, P_n\}$, each P_i having an input m_i, want to securely compute a given function $f(m_1, \ldots, m_n)$, while preserving the confidentiality of their inputs, i.e. nothing should be learned about the inputs $m_1, \ldots, m_n$, except possibly from what is leaked about the output $y = f(m_1, \ldots, m_n)$.

In its simplest conception, we can imagine that there is a *trusted third party* that receives inputs from the various parties who wish to participate, computes the result y in a truthful way, and sends it to the participants without revealing any input m_i. The aim of MPC is to create a protocol that allows this kind of scenario to be instantiated in the real world, where a trusted third party is not available.

1.4 Arithmetic circuit

The first natural question to ask is how to calculate the desired function f. Following Yao's seminal work [Yao82], most approaches for constructing MPC protocols involves initially expressing f as an arithmetic circuit C, followed by a strategy to compute C in a secure way. Informally, an arithmetic circuit takes as inputs some values and is allowed to either add or multiply two values it has already computed.

In more detail, an arithmetic circuit over a field $\mathbb{F}$ is a directed acyclic graph whose vertices are called gates and whose edges are called wires. The gates represent either the addition or the multiplication operation over the field, that can be applied to the values. Let us consider inputs $M = \{m_1, \ldots, m_n\}$. The computation flow is as follows: every input gate, i.e. of in-degree 0 is labeled by either a variable from M or a field element from $\mathbb{F}$. Then, the outputs of these gates are successively processed through the renaming connected gates until to reach a gate of out-degree 0, i.e. the output gate, where the result of the computation is retrieved. In practice, two important metrics come into play: the size and the depth. The size of a circuit is the total number of gates in it, and the depth of a circuit is the length of the longest path between two input and output gates.

1.5 Adversary Model and System Goals

In cryptography, we often consider a system to be "secure" by showing that it is difficult to break with reasonable probability. Hence, we generally consider a probabilistic polynomial time (PPT) referred to as an *"adversary"*, denoted $\mathcal{A}$, whose goal is to break a property supposedly maintained by a system. In the case of MPC, we are trying to ensure that nothing is learned about the parties' inputs, except from the output y.

In practice, if the incentive is high enough, we can well imagine the possibility of several parties colluding, i.e. adopting a behavior that takes into account the pooling of their information, to increase their chances of breaking the properties of a system. That is why, the adversary is generally given the capability to corrupt simultaneously a number $0 < t < n$ of parties. Then, our goal is to build a protocol that is secure as long as the adversary does not corrupt more than t parties. The set of indices in $[n]$ corresponding to *corrupt* parties is denoted by $\mathcal{I}$, while the set of indices of the remaining *honest* parties, is denoted by $\mathcal{H} = [n] \setminus \mathcal{I}$.

Multiple threshold adversarial structures exist, to account for the many ways to model risks. In this dissertation, we gonna assume an *honest majority*, i.e. $t < n/2$. In this case, the set of honest parties always accounts for a majority of the participants, no matter how the adversary selects corrupt parties.

To be complete, let us note that classically, systems may also be considered in the dishonest majority case, i.e. $t < n$, where the adversary is given more power and may corrupt up to $t = n - 1$ parties, or the two-thirds honest majority, where $t < n/3$. The choice of these bounds is important, as it will determine which properties are achievable for the system, and the means of achieving them.

Type of Corruption. So far, we have seen that to be considered secure, a system must be able to withstand an adversary who has the ability to corrupt a certain number of its participants. We now turn to the details of what the latter are capable of doing to achieve their goal.

We previously said that these corrupt parties, coordinated by the adversary, can share information received during the protocol execution, with the aim of breaking the system, but we haven't yet touched on their capabilities. We therefore introduce two different types of corruption:

Passive/semi-honest. An adversary is said to be passive if the corrupt parties follow the rules specified by the protocol faithfully. The adversary is able to see all the internal state of the corrupt parties, including messages received and sent by these, but cannot change their behavior.

Active/malicious. An adversary is said to be active if it fully controls the corrupt parties, which includes making them do whatever arbitrary action it desires during the execution of the protocol. The need to protect a system against this more realistic scenario stems from the fact that there is no way for honest parties to verify that a given message follows the protocol since, by principle, it should not reveal any information.

As one would expect, protocols secured against malicious adversaries are usually more difficult to design and are likely to be less efficient compared to their passively secure equivalent.

Output Guarantees. Apart from privacy, an MPC protocol is expected to return an output, which an adversary might want to prevent. We therefore introduce three notions, from the weakest to the strongest, that categorize the guarantees achieved by different protocols:

Security with abort. A protocol is secure with abort if the adversary may obtain the output while honest parties do not (this abort may be *unanimous*, i.e. if one honest party aborts, then all honest parties also abort, or *selective*).

Fairness. A protocol is fair if the adversary cannot learn the output when the honest parties do not also learn it, i.e. either all parties abort, or none of them do.

Guaranteed Output Delivery (GOD). A protocol guarantees output delivery if, no matter what the adversary does, the honest parties will learn the output.

In this dissertation, we focus on protocols that achieve the strongest notion of *Robustness*, i.e. that guarantee output delivery in a constant number of rounds.

Efficiency. When designing a cryptographic system, we are generally interested in how "efficient" it is. To this end, there exists a number of metrics among which the most common are the following:

Communication complexity. MPC involves interactions during which data are transmitted. A crucial metric in practice is therefore the number of bits transmitted by all parties in a system throughout an execution.

Computation complexity. The amount of computation performed locally by each party is also a crucial metric, since this can be a limiting factor in practice.

1.6 Delegated MPC

So far, we have only considered a set $\mathscr{P}$ of parties, each of whom has inputs, performs a computation, and then learns the result. This model is restrictive in practice, for a number of reasons. Firstly, there are situations in which the output does not have to be public, but a single designated party has to learn it. More importantly, computation requires significant resources, both in terms of computational power and bandwidth. Thus, there are scenarios in which the owners of the inputs do not have the necessary resources or will to participate in such protocols. Finally, some scenarios involve a large number of parties, some of whom may not be able to remain online for the entire duration of the computation. We therefore present an amended model, that we call *delegated* MPC, and introduce, in addition to the parties in $\mathscr{P}$, the following new kind of machines:

Input-Owner: Owner of an input that does not directly take part in the computation.

Output-Learner: Designated receiver of the computation output.

We denote $\mathcal{Q}$ the set of input-owners and $\mathcal{L}$ the output learner[1].

Practical Example. In this model, the circuit evaluation can be outsourced to a cloud-like service, by providing it with the inputs and necessary cryptographic materials. The input-owners can arbitrarily go offline during the evaluation and reconnect for the final output step if they are meant to receive it.

Consider for instance the case of multiple private sensing devices that wish to delegate the computation of some analytic computation to a set of external companies that operate a network of computing parties. The latter can generate all the cryptographic material required for the computation, receive encrypted data, and collectively compute the desired function before sending the result to an output learner, say a client.

In this setting, the overhead for each input-owner is independent of the total number of participants, i.e. of the number of parties, other input-owners, and output learners. This enables to scale MPC to a large number of participants and even enables it for resource-constrained parties. Finally, note that the generation of the cryptographic material by the computing parties does not necessarily need to be repeated for each owner, which makes the protocol more efficient at scale.

[1] This can trivially be extended to multiple learners.

1.7 Simulation-based Security

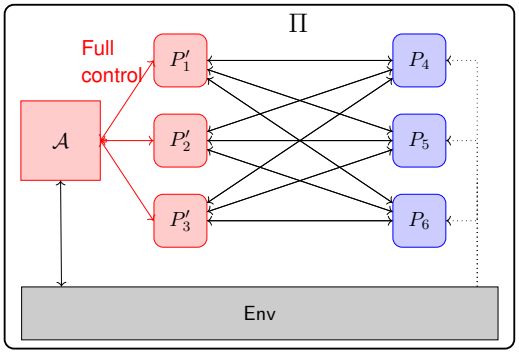
Motivation. The main idea behind MPC is above all to build "secure" systems. In our case, let us recall from Section 1.3 that the goal is to prevent an adversary, who can corrupt a subset of parties, from learning something about honest parties' inputs, except from the output y. The idea of defining a system's security in terms of its ability to resist an adversary is not unique to MPC, but is used in all areas of cryptography. For example, when considering an encryption scheme, security is classically formalized as a security game (see for instance Definition 5), in which an adversary can choose a number of pairs of plaintext messages, and its objective is to determine whether the ciphertexts it receives represent encryptions of the first or the second plaintext in the pairs.

The case of multiparty computation is, however, more complicated in that $i)$ it is a distributed protocol during which the parties communicate with each other, $ii)$ the adversary (see Section 1.5) is possibly given additional powers to change the behavior of the corrupt parties and, $iii)$ it receives an output y. Therefore, to prove security, a more complex approach has to be adopted based on the *real vs ideal world* paradigm [Lin17].

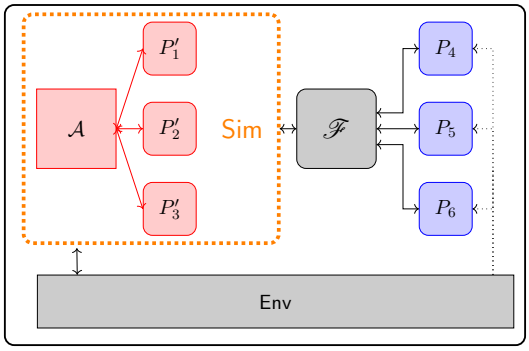
High-level intuition. Remember that in Section 1.3, we said that in its simplest conception, we could imagine that in an ideal world, the parties would send their inputs to a trusted third party and only receive the result. This intuition forms the basis of what we are going to describe as the *ideal world*. In the *real world*, the adversary corrupts a subset of parties and carries out the protocol by interacting with the other parties. The aim seems to be to show that an adversary can't distinguish between the two. However, in this description, the adversary sees interactions between parties in the real world, whereas, in the ideal world, only the inputs are sent to the trusted party and the output is received from it. One ingredient is missing, that we will now formalize.

Description. We consider in Figure 1.1 a machine, called the *environment*, that fully controls an adversary $\mathcal{A}$ and may send inputs to honest parties and which is forwarded their outputs in real-time by honest parties. Its task is to distinguish the real world from the ideal world. In the real world detailed in Figure 1.1a, the adversary $\mathcal{A}$ corrupts a subset of parties, and executes the protocol Π by interacting with the honest parties, to eventually learn the output. In the ideal world detailed in Figure 1.1b, we model the trusted third party as an interactive agent, called *functionality* and denoted $\mathscr{F}$, that is able to receive some inputs, perform some computation, and send an output. Honest parties interact with $\mathscr{F}$, while the adversary interacts through a proxy, called the *simulator*, that connects to $\mathscr{F}$ on behalf of the corrupt parties, and that sends messages to corrupt parties. That way, as the adversary still interacts in the ideal world with what it thinks are the honest parties, there is no obvious way to distinguish between the two.

Proof Idea. We illustrate the high-level proof idea in Figure 1.2. The simulator Sim initiates in its head a set of n parties and may initially receive up to t corruption requests from the environment. Then, Sim creates a simulated honest party in mind for each real, or *dummy*, honest party and lets the simulated honest parties play the protocol Π with the external corrupt parties as in the real world protocol. The simulator can send inputs to and receive outputs from the functionality $\mathscr{F}$ that represents the intended computation to be performed securely. This



(a) Real execution of protocol Π with dummy adversary $\mathcal{A}$. The Environment Env interacts with the system by: its full control on $\mathcal{A}$, its power to give an input to honest participants, and its power to learn the outputs.



(b) Ideal execution: dummy honest participants perform the dummy protocol with the functionality $\mathcal{F}$; the simulator Sim interacts on the right with $\mathcal{F}$ with the same interface as $\mathcal{A}$ and corrupt entities in the dummy protocol, and on the left, interacts with Env with the same interface as the dummy adversary in the real protocol.

Figure 1.1: Real vs Ideal world paradigm

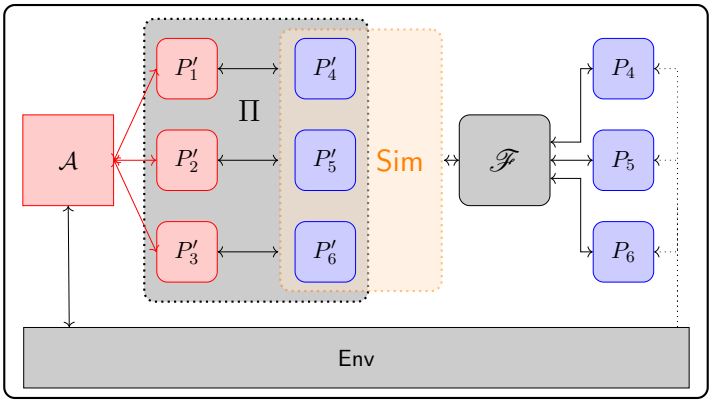


Figure 1.2: Simulation-based proof idea. The simulator Sim simulates honest parties for each real honest party, and executes the protocol Π with the external corrupt parties, while interacting with $\mathcal{F}$. The goal is to prove that the view of Env in the simulation is indistinguishable from the real protocol execution.

serves as the primary tool for Sim to establish an indistinguishable scenario for the environment compared to the real world.

To summarize we introduce the following components in addition to the parties:

Functionality. A functionality is a machine that receives inputs from parties, performs some computation, and sends an output back to the parties.

Environment. The environment is a machine that fully controls an adversary and which may send inputs to honest parties and which is forwarded their outputs in real-time.

Simulator. A simulator acts as an interface between an ideal functionality and the adversary in the ideal world. It can send inputs to and receive outputs from the functionality while being connected to the adversary as in the real world.

We now say that a protocol is secure if no environment can distinguish between the real

and ideal worlds.

1.7.1 Formalizing Eventual Delivery in UC

We now explain the high-level idea of the mechanism used to formalize eventual delivery following [KMTZ11; CGHZ16]. Every ideal functionality $\mathscr{F}$, 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) and initializes a counter $D_{\text{ssid}} \leftarrow 1$, which captures the delivery delay. Upon receiving (delay) from $\mathcal{A}$, it sets $D_{\text{ssid}} \leftarrow D_{\text{ssid}} + 1$. Upon receiving (fetch) from P, it sets $D_{\text{ssid}} \leftarrow D_{\text{ssid}} - 1$, as well as for all other counters related to pending outputs for P. In addition, we specify that it leaks (fetch) to $\mathcal{A}$[2]. It is left implicit that entities fetch as much as they can all. Since $\mathcal{A}$ is PPT, at some point, it gets exhausted from pressing the button delay. So, after sufficiently many fetches more, the counter drops down to 0. Then $\mathscr{F}$ can deliver (ssid, v) to P.

Remark. *The session identifier of the MPC protocol,* sid*, is left implicit in all calls to functionalities. Some calls to functionalities are parametrized by sub-session identifiers* ssid*. For instance, in* $\mathcal{F}_{\text{LSS}}$ *and* BC *described Sections 1.8.2 and 2.2,* ssid*s are encoded by labels of variables.*

1.7.2 MPC Functionality

The ideal functionality of MPC that we aim to UC implement, is formalized as $\mathscr{F}_{\mathsf{C}}$ in Fig. 1.3. It returns to an output learner $\mathcal{L}$ the evaluation of a public arithmetic circuit $\mathsf{C} : (\mathcal{M} \cup \{\perp\})^n \to \mathcal{M}$ over inputs in some space $\mathcal{M}$. For simplicity: C has n input gates, one single output gate, $\mathscr{F}_{\mathsf{C}}$ expects exactly one single input from each party, and delivers the output to an output learner $\mathcal{L}$. For more clarity in the presentation, C is hardcoded in $\mathscr{F}_{\mathsf{C}}$. However, our MPC protocols actually allow parties to *adaptively* choose C based on the list of non-$\perp$ inputs received.

The functionality works as follows. Upon receiving an input from any party P_i, it stores $(P_i, \overline{m_i})$[3] and leaks this information to $\mathcal{A}$. Before $\mathscr{F}_{\mathsf{C}}$ delivers the output, it needs to wait for the inputs to be submitted. However, the adversary $\mathcal{A}$ can choose to never instruct corrupt parties to send their inputs. To remedy this, we follow the *fetch-and-delay* mechanism explained in Section 1.7.1 and introduce a timeout $T_{\mathcal{A}}$. In brief, the functionality:

1. waits until it receives an input from every honest party,
2. sets a timeout $T_{\mathcal{A}}$,
3. then, after it elapsed, sets to $\perp$ the inputs of the parties (necessarily corrupt) which did not give an input.

Once the timeout expires, the output-available flag is permanently set to true, and no more input can be submitted. Finally, the output evaluation is eventually delivered following a finite delay chosen by $\mathcal{A}$.

[2]This precision is not present in previous works [KMTZ11; CGHZ16; LLM+20], since that way, the adversary knows at any moment in time when an output will be delivered.

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

$\mathscr{F}_{\mathsf{C}}$

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

- Initialize output-available $\leftarrow$ false. //the flag telling if the output can be delivered.
- When all $m_i \neq \top$, $\forall i \in [n]$, set output-available $=$ true. //it is now frozen

Formalizing timeout for inputs of corrupt parties

- 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 output-available $=$ false) On input $(\mathsf{input}, \widetilde{m_i} \in \mathcal{M})$ from any $P_i \in \mathscr{P}$ for the first time, or possibly from $\mathcal{A}$ if P_i is corrupt, set $m_i \leftarrow \widetilde{m_i}$, then store $(\mathsf{input}, \mathcal{S}, m_i)$, and *eventually-deliver* $(\mathsf{stored}, \overline{m_i})$ to each party $P \in \mathscr{P}$. //eventually-delivers" consists of the same fetch-and-delay mechanism as explained in Section 1.7.1.

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 output-available $=$ true, then deliver $y = \mathsf{C}(\mathbf{m})$ to $\mathcal{L}$.

Figure 1.3: Functionality of secure circuit evaluation. Each input m_i is identified by a public label $\overline{m_i}$.

Straightforward Generalizations of $\mathscr{F}_{\mathsf{C}}$. In our MPC protocols presented in Chapters 3 and 4, messages of parties do not depend on the actual circuit to be computed. Thus, our protocol actually achieves the "reusability" property ([BJMS20]), i.e., it implements a more general functionality to which honest parties can possibly provide circuits to be computed *after* the distribution of the inputs by the input-owners, possibly with some $\perp$ or badly formed messages for some of them. We do formalize and implement such a functionality in Chapter 2 for the specific case of linear combinations, as this will help the future description of our MPC protocols.

As such, our definition of $\mathscr{F}_{\mathsf{C}}$ implies that only $\mathcal{L}$ outputs. However, it straightforwardly generalizes to capture protocols delivering outputs to a set of output learners, be they included, overlapping, or disjunct, from the set $\mathscr{P}$ of parties.

1.8 Communication Model

In this dissertation, we consider two ways of sending a message from some sender to some receiver(s).

- *Asynchronous peer-to-peer channel*: Asynchronous communication models real-world networks (like the Internet), where limited guarantees exist as to the reception of sent messages. We describe in Section 1.8.1 a functionality for asynchronous authenticated public/secure message transmitting with eventual delivery, where the adversary is given the power to schedule the delivery of messages within some finite (unknown to the honest sender and/or receiver) delay D. Importantly, two parties may receive two different messages from the same corrupt sender.

- *Broadcast channel*: We describe in Section 1.8.2 a functionality for broadcast, where each broadcast message recipient has the guarantee that all other recipients received the same message within some known bound Δ.

In practice, implementing a broadcast channel requires either multiple rounds of peer-to-peer communication [DR85; FH06; KK06; GP21; WMR+23] or special channels (such as blockchains), and is therefore costly in itself. This difficulty was pointed out in the comments of NIST's recent call for standardization of multiparty threshold-cryptography[23a]. That is why, in this dissertation, we are going to limit its use as much as possible to the initial distribution of inputs, in order to ensure *input provision*, i.e. to guarantee that the inputs of all the honest parties are considered for the circuit evaluation.

1.8.1 Authenticated message transmitting

We formalize in Figure 1.4 the ideal functionality of asynchronous *public authenticated* message transmitting with eventual delivery, denoted as $\mathcal{F}_{\mathsf{AT}}$. 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*. We also define a stronger variant, denoted $\mathcal{F}_{\mathsf{ST}}$ for *secure transmitting*, which leaks only the length of the messages. Only messages intended for the output-learner $\mathcal{L}$ will go through this stronger variant.

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

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

Figure 1.4: 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 $\mathcal{F}_{\mathsf{ST}}$ is described inline.

Our baseline for $\mathcal{F}_{\mathsf{ST}}$ is the functionality denoted $\mathcal{F}_{\text{ed-smt}}$ [KMTZ11]. For $\mathcal{F}_{\mathsf{AT}}$, we made the addition to leak the contents of the messages to $\mathcal{A}$. We also incorporated two other changes, borrowed from the $\mathcal{F}_{\mathsf{NET}}$ in [LLM+20]. The first consists in attaching a unique identifier to each message and counter, in order to give $\mathcal{A}$ control on the delay of each message individually. Notice that [CGHZ16] model this individual control by, instead, giving the power to $\mathcal{A}$ to reorder messages not delivered yet. The second consists in forcing explicitly $\mathcal{A}$ 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.

1.8.2 Broadcast

Definition 1. *A broadcast protocol [FLL21, Def 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 inputs value m, all honest receivers output the same value m.*

We dub it as BC *and formalize it in Figure 1.5.*

Overall, it simply proceeds as follows. On receiving a message m from the sender $\mathcal{S}$, it sends m to each receiver $R \in \mathcal{R}$ by using the same procedure as $\mathcal{F}_{\mathsf{AT}}$.

Remark. *There exists another functionality for broadcast in [CP23; AAPP22], which is denoted as $\mathcal{F}_{\textit{ACast}}$. The difference is that for every given $R \in \mathcal{R}$, $\mathcal{F}_{\textit{ACast}}$ may* never *deliver s to R if the*

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

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

Figure 1.5: Ideal functionality of reliable broadcast. It is parametrized by a sender $\mathcal{S}$ and a set of receivers $\mathcal{R}$.

adversary does not allow to, nonwithstanding other honest parties could have been delivered s. The reason is that they use the classical UC framework of delayed output *of Canetti [Can01], in which the delivery of every single output from a functionality needs to be allowed by the adversary.*

1.9 Other UC functionalities

We now define other functionalities which will be used to build our protocols.

Public Key Infrastructure. A public key infrastructure (PKI) is responsible for facilitating the authentication and distribution of public keys by essentially maintaining a database of identity/public key pairs.

We present in Figure 1.6 the ideal functionality of a bulletin board of public keys, denoted as bPKI. Upon receiving a key pk_i from any party $P_i \in \mathscr{P}$, 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 party $P_i \in \mathscr{P}$,*
- sets a timeout,
- then after it elapsed, sets to $\perp$ the keys of the (necessarily corrupt) parties which did not give a key.

Then it sets as $\mathsf{pk} \leftarrow (\mathsf{pk}_i)_{i\in[n]}$[4] the vector of all keys, and eventually delivers it to all the system.

Remark. *The* bPKI *functionality, for our usage limited to publication of public keys, could be traded by the assumption denoted {Bare/Untrusted/Bulletin board} public key setup (PKI)" [ACGJ18; BCG21; GJPR21]. Importantly,* bPKI *does not perform any check on the written strings, it displays them to all parties.*

[4] With possibly some $\mathsf{pk}_i = \perp$.

bPKI

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

Formalizing eventual delivery For every $R \in \mathcal{R}$, initialize a counter $D_R \leftarrow 0$ // the delivery delay.
Initialize $\mathsf{output\text{-}available} \leftarrow \mathsf{false}$ //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 $\mathsf{output\text{-}available}$ =true then deliver pk to R.

Formalizing timeout for keys of corrupt parties Initialize a counter $T_{\mathcal{A}} \leftarrow 1$ // the timeout.
Upon receiving $\mathsf{delay\text{-}keys}$ 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 $\mathsf{pk}_i \leftarrow \widetilde{\mathsf{pk}_i}$ $\forall i \in \mathcal{I}$.

Figure 1.6: The bulletin board of public keys functionality bPKI, parametrized by a set of n key-holders, of which the corrupt ones are indexed by $\mathcal{I} \subset [n]$, and by a set of receivers $\mathcal{R}$. It does not perform any checks on the keys received. A published key that is not in the correct distribution is automatically considered as $\perp$ by honest parties.

Non-Interactive Zero Knowledge Proof.

Definition 2 (Zero-Knowledge Proof)**.** *A pair of probabilistic polynomial time interactive programs P, V is a zero-knowledge proof if the following properties are satisfied:*

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

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

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

We present in Figure 1.7 the ideal functionality of *non-interactive zero-knowledge arguments of knowledge*, denoted as $\mathscr{F}_{\text{NIZK}}$ and mainly borrowed from [GOS06]. It is parametrized by an NP relation $\mathscr{R}$. Upon request of a prover P exhibiting some public input x and knowledge of some secret witness w, it verifies if $(x, w) \in \mathscr{R}$ then deletes w from its memory. If the verification passes, then $\mathscr{F}_{\text{NIZK}}$ eventually delivers a string π to P. We denote Π the space of such strings π. During the delay of output, $\mathcal{A}$ has the power to set the value of π. If it does not use this power, then $\mathscr{F}_{\text{NIZK}}$ sets π to a default value π_0. Upon subsequent input of the same string π and x from any verifier, $\mathscr{F}_{\text{NIZK}}$ then confirms to the verifier that P knows some witness for x.

Remark. *The main difference with [GOS06], in which $\mathcal{A}$ could delay forever the delivery of π, is the introduction of a time-out, based on the* fetch*-and-*delay *mechanism. Sticking to this model would have prevented us from specifying a protocol with guaranteed output delivery (GOD) in case of honest majority.*

UC implementations of $\mathscr{F}_{\text{NIZK}}$ exist, which do not require honest majority [DDOPS01], but at the cost of requiring a uniform random string (URS). Nonwithstanding that [DDOPS01] allows the same URS to be reused in concurrent executions, the bottom-line is that the URS needs to be part of a local setup in their implementation. Without an honest majority assumption, then [CDPW07] prove that UC NIZK is non-implementable in the global common random string model, i.e., which we formalized as $\overline{\mathcal{G}}_{\text{URS}}$ in the particular case where the string is uniform.

Remark. *The need for a URS can be escaped under honest majority, provided access to* bPKI*, thanks to the technique denoted multi-string CRS [GO14; BJMS20].*

Uniform Random String. We present in Figure 1.8 the ideal functionality of *uniform random string*, denoted $\overline{\mathcal{G}}_{\text{URS}}$ and mainly presented as a particular case of $\mathscr{F}_{crs}$ in [CLOS02]. It samples uniformly at random a sequence of bits of pre-defined length κ, denoted URS, then outputs it to all parties.

Remark. *We strictly upgrade the security of our model in that we allow the string produced by $\overline{\mathcal{G}}_{URS}$ to be directly observed by the Environment. In particular, our simulator will not have the choice but to use the URS provided by $\overline{\mathcal{G}}_{URS}$.*

$\mathscr{F}_{\text{NIZK}}$

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

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

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

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

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

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

Figure 1.7: Non-interactive zero-knowledge functionality

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

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

Figure 1.8: Uniform Random String Functionality.

1.9.1 Model Summary

In this dissertation, we aim at building some protocol $\Pi_{\mathscr{F}}$ that securely instantiates a functionality $\mathscr{F}$ with the help of some other functionality, say $\mathscr{F}'$, that is, in the $\mathscr{F}'$-*hybrid model*. This functionality $\mathscr{F}'$ serves as an intermediary trusted entity that parties can use to assist them in securely computing $\mathscr{F}$. Indeed, $\Pi_{\mathscr{F}}^{\mathscr{F}'}$ could possibly be expressed in a much simpler way using $\mathscr{F}'$.

By proving the UC security of $\Pi_{\mathscr{F}'}^{\mathscr{G}}$ in a typically simpler $\mathscr{G}$-hybrid model, we simplify the proof of $\Pi_{\mathscr{F}}^{\mathscr{F}'}$, by not considering a big "monolithic" protocol, but instead a simpler variant in this $\mathscr{F}'$-hybrid model. The main result of the UC framework is that the composed protocol still instantiates $\mathscr{F}$ in the $\mathscr{G}$-hybrid model.

In what follows, we will rely heavily on this principle and prove, notably in Chapter 2, an intermediate functionality that will serve to simplify the proofs of MPC protocols in Chapters 3 and 4. We can summarize the system in which we will be working as follows:

Participants: We consider a set $\mathscr{P}$ of n parties, of which up to t can be corrupt by a PPT adversary $\mathcal{A}$. We also consider a set $\mathcal{Q}$ of (possibly corrupt) input-owners and an output learner $\mathcal{L}$.

Functionalities: To help us to build MPC protocols, we described in Sections 1.8 and 1.9 functionalities $\mathcal{F}_{\mathsf{AT}}, \mathsf{BC}, \mathsf{bPKI}, \mathscr{F}_{\mathsf{NIZK}}$ and $\overline{\mathcal{G}}_{\mathsf{URS}}$.

Semi-Malicious Corruptions: In our protocols and proofs, we will consider what we define as *Semi-Malicious Corruptions*, following [AJL+12; BHP17; GLS15; BJMS20]. Semi-maliciously corrupt parties continuously forward to $\mathcal{A}$ their outputs received from ideal functionalities, and act arbitrarily as instructed by $\mathcal{A}$. For instance, they can possibly not send some messages although the protocol instructs them to. However, when a corrupt entity M inputs a message m to $\mathcal{F}_{\mathsf{AT}}$ or BC, then the sending of m must be compatible with the requirements of the protocol, with respect to: (i) all outputs of instances of $\overline{\mathcal{G}}_{\mathsf{URS}}$, bPKI, BC required for sending m (ii) an internal witness tape that M must have, of the form (x, r) with x an input and r of the same length as all random coins that an honest party would have been meant to have tossed upon sending m. M can however use conflicting (x, r) when sending different messages m, m'. Finally, we also require that the semi-malicious adversary $\mathcal{A}$ can only send an output v to BC^M for some corrupt M only if: either v could have been input to BC^M by M itself according to the above rule, or if $v = \perp$. Notice that we *do not* impose any condition for the sending of some m on bPKI.

Robustness : Briefly, we say that an MPC protocol is *robust* if, in every execution in which the adversary provides the inputs required by BC, then all honest parties obtain an output in a constant number of rounds. We argue that our definition coincides with the classical definition of robustness, as soon as BC and $\mathcal{F}_{\mathsf{AT}}$ are implemented with protocols or resources that eventually deliver messages.

1.10 Cryptographic Definitions

We now provide the definitions of the main notions that are relevant to this work. Recall from Section 1.2 that we consider a security parameter λ and will mainly use a computational security argument to show that an adversary's advantage to successfully attack a scheme is negligible in λ.

For the specific case of homomorphic operations, we also consider another parameter κ to account for the ability to carry out successive operations and will require that the probability of incorrect decryption after some homomorphic operations be a negligible function in κ.

1.10.1 Linear Secret Sharing.

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

Definition 3 ((n,t)-LSS)**.** *Let $\mathcal{R}$ be a ring. A (n,t)-Linear Secret Sharing Scheme is defined by the following two algorithms:*

- $\mathsf{LSS.Share}(s \in \mathcal{R}, n, t)$[5] $\rightarrow (\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)})$*: For a given secret $s \in \mathcal{R}$, the sharing algorithm generates a vector $(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)})$ of shares, where $\mathbf{s}^{(i)}$ is the share of party P_i.*

- $\mathsf{LSS.Reco}(\{\mathbf{s}^{(i)}\}_{i\in\mathcal{U}}, \mathcal{U}) \rightarrow s$*: For any set $\mathcal{U}$ of size $t+1$ and shares $\{\mathbf{s}^{(i)}\}_{i\in\mathcal{U}}$, the reconstruction algorithm outputs a secret $s \in \mathcal{R}$.*

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

Furthermore, it must satisfy the following properties.

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

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

$$\left\{ \{\mathbf{s}_R^{(i)}\}_{i\in\mathcal{V}} \approx \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{V}} \;\middle|\; \begin{array}{l} \wedge \{\mathbf{s}_R^{(i)}\}_{i\in[n]} \leftarrow \mathsf{Share}(s_R, n, t) \\ \wedge \{\mathbf{s}_L^{(i)}\}_{i\in[n]} \leftarrow \mathsf{Share}(s_L, n, t) \end{array} \right\} \tag{1.1}$$

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

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

[5] We leave implicit the randomness used for the sharing. It will be made explicit in Definition 8

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 $\{x_i\}_{i\in S} \in \mathcal{R}$. Then, we have that $\{\Lambda(\{\mathbf{x}_i^{(1)}\}_{i\in S}), \ldots, \Lambda(\{\mathbf{x}_i^{(n)}\}_{i\in S})\} = [\Lambda(\{x_i\}_{i\in S})]$.

Moreover, for our UC proofs to go through, we require in addition the following two properties (4) and (5), which enable the simulation of shares. They are satisfied by all linear secret sharings used in practice, e.g., Shamir[Sha79] and the $\{0,1\}$-LSSD of [JRS17] (renamed $\{0,1\}$-LSS in [BS23]). Of possible independent interest, in Appendix A, we provide a definition (Definition 39) 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 41 we prove that a $(n,t)-$LSSD scheme satisfies both properties (4) and (5).

4. ***Simulatability**: Additionally, we require the existence of an efficient algorithm* 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 $s_L, s_R \in \mathcal{R}$, 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 $\{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{U}} \leftarrow \mathsf{ShSim}(\{\mathbf{s}_R^{(i)}\}_{i\in\mathcal{V}}, s_L)$,*

$$\left| Pr[\mathcal{A}(\{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{V}}, \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] - Pr[\mathcal{A}(\{\mathbf{s}_R^{(i)}\}_{i\in\mathcal{V}}, \{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] \right| \leq negl(\lambda) \tag{1.2}$$

5. ***Inference of Shares**: Finally, we require the existence of an efficient algorithm* 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 $s_L, s_R \in \mathcal{R}$, 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 $\{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{V}} \leftarrow \mathsf{ShInfer}(\{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}})$,*

$$\left| Pr[\mathcal{A}(\{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{V}}, \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] - Pr[\mathcal{A}(\{\widetilde{\mathbf{s}}_L^{(i)}\}_{i\in\mathcal{V}}, \{\mathbf{s}_L^{(i)}\}_{i\in\mathcal{U}}) = 1] \right| \leq negl(\lambda) \tag{1.3}$$

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

Example: Shamir Secret Sharing [Sha79]. For the purpose of this section, we consider a finite field $\mathbb{F}$ and assume that each party $P_i \in \mathscr{P}$ is associated with a non-zero element $\alpha_i \in \mathbb{F}$ such that if $i \neq j$ then $\alpha_i \neq \alpha_j$. We recall the secret-sharing scheme of Shamir [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 4. *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*

$$\lambda_i^{\mathcal{U}} = \prod_{j\in\mathcal{U}, j\neq i} \frac{\alpha_0 - \alpha_j}{\alpha_i - \alpha_j} \tag{1.4}$$

Intuitively, Shamir uses polynomial evaluation to share some secrets and interpolation to reconstruct them from shares. In more detail, to share a value $s \in \mathbb{F}$ using Shamir, the dealer samples at random a polynomial $f(Y) \in F_{\leq t}[Y]$ of degree at most t, such that $f(0) = s$. The shares corresponding to each party 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:

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 $i \in [n]$.

Shamir.Reco$(\{s^{(i)}\}_{i \in \mathcal{U}}, \mathcal{U})$: To reconstruct s from shares $\{s^{(i)}\}_{i \in \mathcal{U}}$ and for $|\mathcal{U}| > t$, compute

$$s = \sum_{i \in \mathcal{U}} \lambda_i^{\mathcal{U}} s^{(i)} \tag{1.5}$$

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.

Remark. *To relate to Definition 39 later presented in Appendix A, let us note that in the case of Shamir, we use the Vandermonde matrix as the sharing matrix* $\mathbf{M}$.

$$\begin{pmatrix} 1 & \alpha_1^1 & \alpha_1^2 & \ldots & \alpha_1^t \\ 1 & \alpha_2^1 & \alpha_2^2 & \ldots & \alpha_2^t \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha_n^1 & \alpha_n^2 & \ldots & \alpha_n^t \end{pmatrix} \tag{1.6}$$

The simulation and the inference of shares follow easily by Lagrange interpolation.

1.10.2 Public Key Encryption (PKE).

We continue with the most basic cryptographic primitive: encryption.

Definition 5 (PKE)**.** *A public-key encryption scheme consists of the following algorithms:*

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

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

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

Moreover, a PKE *scheme satisfies the following properties*

1. ***Security:*** *There exists different levels of security of a public encryption scheme:*

   IND-CPA *(Indistinguishability Under Chosen-Plaintext Attacks): Informally, it means that an adversary can not find which message is encrypted after receiving a ciphertext of one of the two messages of its choice;*

   IND-CCA *(Indistinguishability Under adaptive Chosen-Ciphertext Attacks): Informally, it is similar to the previous definition except that now, the adversary has additional access to a decryption oracle, with the only restriction being that it cannot request the decryption of the challenge ciphertext.*

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

$$\frac{\mathsf{Game}^{\mathcal{A}}_{\mathsf{IND\text{-}CPA}}(1^\lambda)}{}$$

$1:\ b \xleftarrow{\$} \{0,1\}$

$2:\ (\mathsf{dk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{Keygen}(1^\lambda)$

$3:\ (\mathsf{state}, m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk})$

$4:\ \mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$

$5:\ b' \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}, \mathsf{c}, \mathsf{state})$

$6:\ \mathbf{return}\ b = b'$

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

$$|2.\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Enc}} - 1| \leq \mathit{negl}(\lambda) \tag{1.7}$$

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

$$\mathsf{Dec}(\mathsf{dk}, \mathsf{Enc}(\mathsf{pk}, m)) = m\,. \tag{1.8}$$

A user with the pair $(\mathsf{dk}, \mathsf{pk})$ of decryption-public keys can publish pk to everyone likely to send him a message. However, dk needs to be stored privately.

Example: ElGamal. Let us describe the PKE of ElGamal [Elg85], which is composed of three algorithms $(\mathsf{PKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and defined on a cyclic group $(\mathbb{G}, \cdot)$ of prime order p in

which DDH is hard. Let g be any generator of $\mathbb{G}$.

ElGamal Encryption Scheme

$\mathsf{PKeyGen}(1^\lambda)$ Given a security parameter λ, it chooses $x \leftarrow_\$ \mathbb{Z}_p$ and outputs the public key $\mathsf{pk} = g^{-x}$ and the decryption key $\mathsf{dk} = x$.

$\mathsf{Enc}(\mathsf{pk}, M)$ To encrypt a message $M \in \mathbb{G}$ using public key pk, it chooses $r \leftarrow_\$ \mathbb{Z}_p$ and outputs the ciphertext $\mathsf{c} = (M \cdot \mathsf{pk}^r, g^r) \in \mathbb{G}^2$.

$\mathsf{Dec}(\mathsf{dk}, \mathsf{c})$ Given $\mathsf{c} = (c_1, c_2)$ and the decryption key sk, it computes $c_1 \cdot \mathsf{c}^{\mathsf{dk}}$.

The scheme is correct as $c_1 \cdot c_2^{\mathsf{dk}} = M \cdot \mathsf{pk}^r \cdot (g^r)^{\mathsf{dk}} = M \cdot g^{-\mathsf{dk}r + r\mathsf{dk}} = M$.

1.10.3 Homomorphic Encryption.

An encryption scheme is said *partially* homomorphic if one can define an additional algorithm Add (or Mult) taking as input two ciphertexts X of a value x and Y of a value y and producing a ciphertext Z of value $x + y$ (or $x \times x$).

An encryption scheme that has simultaneously the additive and multiplicative properties, is said to be *fully* homomorphic. In that case, we will define in the Definition 6 below a new Eval algorithm whose purpose is to evaluate an arithmetic circuit C. Note that the latter algorithm may require an additional relinearization (or evaluation) key. Because this homomorphic evaluation requires proper parametrization in practice, we introduce an abstract homomorphic parameter κ that will be used to define correctness.

Definition 6 (Fully Homomorphic Encryption Scheme (FHE))**.** *A fully homomorphic encryption scheme (*FHE*) consists in a message space $\mathscr{M}$; spaces of secret, encryption and relinearization keys $\mathscr{X}$, $\mathscr{E}k$, $\mathscr{R}lk$; randomness spaces $\mathscr{B}_{Key}$ and $\overrightarrow{\mathscr{B}_{Enc}}$ (for key generation, and encryption) and a ciphertext space $\mathscr{C}$; along with the following probabilistic polynomial-time algorithms:*

- ***Setup*** $pp \leftarrow \mathsf{Setup}(1^\lambda)$*: On input the security parameter λ, the setup algorithm outputs a set of public parameters* pp*.*
- ***Keys Generation*** $(\mathsf{sk}, \mathsf{ek}, \mathsf{rlk}) \leftarrow \mathsf{Keygen}(\mathsf{pp})$*: On input some public parameters* pp*, the key generation algorithm samples a secret key* $\mathsf{sk} \xleftarrow{\$} \mathscr{X}$*, and outputs the encryption key* $\mathsf{ek} \in \mathscr{E}k$ *and the relinearization key* $\mathsf{rlk} \in \mathscr{R}lk$[6]*.*
- ***Encryption*** $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{ek} \in \mathscr{E}k, m \in \mathscr{M})$*: On input public parameters* pp*, an encryption key* ek *and a plaintext m, the encryption algorithm outputs a ciphertext* c *of m under* ek*.*
- ***Evaluation*** $\mathsf{c}' \leftarrow \mathsf{Eval}(\mathsf{C}, \mathsf{rlk}, \mathsf{c}_1, \ldots, \mathsf{c}_n)$*: On input public parameters* pp*, an encryption key* ek*, a relinearization key* rlk*, a circuit* $\mathsf{C} : \mathscr{M}^n \to \mathscr{M}$*, and a set of ciphertexts* $\mathsf{c}_1, \ldots, \mathsf{c}_n$*, the evaluation algorithm outputs a ciphertext* c*.*

[6] Sometimes denoted as *evaluation key*

- ***Decryption*** $m' \leftarrow \mathsf{Dec}(\mathsf{sk} \in \mathscr{X}, \mathsf{c} \in \mathscr{C})$*: On input a ciphertext* c *and a secret key* sk*, the decryption algorithm outputs* m'*.*

  Moreover, a FHE *scheme satisfies the following properties:*

  1. ***Semantic Security:*** *Similar to Definition 5, we say that a* FHE *scheme* $\mathcal{E} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* IND-CPA *secure if, no* PPT *adversary* $\mathcal{A}$ *has an non-negligible advantage in the* $\mathsf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{Enc}}(1^\lambda)$ *game.*

  2. ***Correctness:*** *For any arithmetic circuit* $\mathsf{C} : \mathscr{M}^n \to \mathscr{M}$ *and input messages* $m_1, \ldots, m_n$*, there exists a public parameterization* pp *such that, for* $(\mathsf{sk}, \mathsf{ek}, \mathsf{rlk}) \leftarrow \mathsf{Keygen}(\mathsf{pp}, 1^\lambda)$*,*

     $$Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{C}, \mathsf{rlk}, \mathsf{c}_1, \ldots, \mathsf{c}_n)) \neq \mathsf{C}(m_1, \ldots, m_n)] \leq negl(\kappa). \tag{1.9}$$

1.10.4 Linear Homomorphic Encryption.

We observe that in a number of encryption schemes, key generation, encryption, and decryption are essentially *linear maps*. For simplicity's sake, we assume that all operations take place over $\mathbb{Z}/q\mathbb{Z}$-modules, where q is a known modulus[7]. A linear map between two $\mathbb{Z}/q\mathbb{Z}$-modules $g : (E, +) \to (F, *)$ satisfies $g(e_1 + e_2) = g(e_1) * g(e_2)$ for $e_1, e_2 \in E$, and, $g(a \cdot e) = a \cdot g(e)$ for all $a \in \mathbb{Z}/q\mathbb{Z}$. More precisely, in such schemes, key generation is a linear function in a secret key sk and some randomness, encryption is linear in a plaintext and some randomnesses, while decryption is, roughly, linear in sk. This linearity property, essential in multiparty settings, implies that there exists a public map which, given an offset (roughly: sk') on the secret key, maps encryptions under sk into ciphertexts that have the same distribution as fresh encryptions under $\mathsf{sk} + \mathsf{sk}'$. Moreover, this linearity supports partial homomorphic operations, such as addition and/or multiplication, in some form. The following Definition 7 provides a wrapper for all such schemes, formalized using generic linear maps ($\Lambda_{\mathsf{EKeyGen}}$, Λ_{Enc} and Λ_{Dec}).

Definition 7 (Linear Homomorphic Encryption (ℓ-HE))**.** *A linear homomorphic encryption scheme (*ℓ-HE*) consists in a message space* $\mathscr{M}$*; spaces of secret and encryption keys* $\mathscr{X}$*,* $\mathscr{E}k$*; randomness spaces:* $\mathscr{B}_{Key}$ *and* $\overrightarrow{\mathscr{B}_{\mathsf{Enc}}}$*, both (subsets of) vector spaces over* $\mathbb{Z}/s\mathbb{Z}$*-modules ; and a ciphertext space* $\mathscr{C}$*; along with the following PPT algorithms and properties:*

- $\mathsf{Setup}(1^\lambda)$*: On input the security parameter* λ*, the setup algorithm outputs a set of public parameters* pp *and a uniform random string* a*.*

- $\mathsf{Keygen}(\mathsf{pp}, a)$*: On input some public parameters* pp*, and a URS* a*, the key generation algorithm samples a secret key* $\mathsf{sk} \xleftarrow{\$} \mathscr{X}$ *and a key randomness* $\rho^{key} \xleftarrow{\$} \mathscr{B}_{Key}$*. When* $\mathscr{B}_{Key}$ *is not closed under addition, the randomness* ρ^{key} *is drawn small enough such that the sum of* n *such terms remains statistically within* $\mathscr{B}_{Key}$*. In our multiparty setting, this will ensure that a common key resulting from a sum of keys remains valid, as illustrated in Equations* (3.6) *and* (4.1)*. The algorithm outputs an encryption key* $\mathsf{ek} \leftarrow \Lambda^a_{\mathsf{EKeyGen}}(\mathsf{sk}, \rho^{key}) \in \mathscr{E}k$[8]*, where* $\Lambda^a_{\mathsf{EKeyGen}}$ *is a public* linear map *with coefficients determined by the URS* a*.*

[7] However, this formalization can be adapted to settings where q is unknown, such as in the CL [CL15] case, where secret-sharing operates over $\mathbb{Z}$, as explained in [BDO23].

[8] sometimes, along with a *relinearization key* $\mathsf{rlk} \in \mathscr{R}lk$, that must also be linear in sk

- $\mathsf{Enc}(\mathsf{pp}, \mathsf{ek} \in \mathscr{E}k, m \in \mathscr{M})$*: On input public parameters* pp*, an encryption key* ek *and a plaintext* m*, the encryption algorithm samples a vector of randomnesses* $\rho_{\mathsf{Enc}}^{\vec{}} \xleftarrow{\$} \overrightarrow{\mathscr{B}_{Enc}}$ *and outputs a ciphertext* $\mathsf{c} \leftarrow \Lambda_{\mathsf{Enc}}^{\mathsf{ek}}(m, \rho_{\mathsf{Enc}}^{\vec{}}) \in \mathscr{C}$*, where* $\Lambda_{\mathsf{Enc}}^{\mathsf{ek}}$ *is a public fixed* linear map*.*

- $\mathsf{Dec}(\mathsf{sk} \in \mathscr{X}, \mathsf{c} \in \mathscr{C})$*: On input a ciphertext* c *and a secret key* sk*, the decryption algorithm computes* $\mu \leftarrow \Lambda_{\mathsf{Dec}}^{\mathsf{c}}(\mathsf{sk})$*, where* $\Lambda_{\mathsf{Dec}}^{\mathsf{c}}$ *is a public linear map, and either outputs a plaintext* $m = \Omega_{\mathsf{Dec}}(\mu)$ *or the symbol* $\perp$*, where* Ω_{Dec} *denotes a non-linear decoding function.*

- IND-CPA *and correctness, as reminded in Definition 5;*

Examples. Definition 7 is verified by various classical schemes such as BGN [BGN05], exponentiated ElGamal [ISO19; TLC+23], GSW [GSW13] or CL [CL15], as further discussed in Section 4.3. Interestingly, this definition wraps schemes with different degrees of homomorphic capabilities, ranging from those with limited homomorphism to fully homomorphic encryption schemes.

In Section 3.3.2, we detail a linear scheme called ℓ-BFV, which is the first RLWE-based ℓ-HE scheme with fully homomorphic capabilities. For clarity, we refer to this subcategory of schemes as ℓ-FHE.

Why linearity matters? The most important takeaway is that a ℓ-HE scheme has linearity properties for both key generation and encryption/decryption.

The former can be dubbed as a being *key homomorphic*, namely, that provided with a common set of public parameters pp and with a uniform random string a, the addition of two encryption keys ek_1 and ek_2 leads to a valid encryption key ek′. Said differently, for $\mathsf{sk}_1, \mathsf{sk}_2 \xleftarrow{\$} \mathscr{X}$ and $\rho_{key,1}, \rho_{key,2} \xleftarrow{\$} \mathscr{B}_{Key}$, we have that $\mathsf{ek}' \leftarrow \Lambda_{\mathsf{EKeyGen}}^{a}(\mathsf{sk}_1, \rho_{key,1}) + \Lambda_{\mathsf{EKeyGen}}^{a}(\mathsf{sk}_2, \rho_{key,2}) = \Lambda_{\mathsf{EKeyGen}}^{a}(\mathsf{sk}_1 + \mathsf{sk}_2, \rho_{key,1} + \rho_{key,2}) \in \mathscr{E}k$. This property enables the efficient distributed generation of a common encryption key, which is the backbone of any ℓ-HE-based MPC protocol as explained in Section 1.11.

The latter will facilitate threshold decryption, mainly thanks to the fact that, provided with a ciphertext c, we have that $\Lambda_{\mathsf{Dec}}^{\mathsf{c}}(\mathsf{sk}_1) + \Lambda_{\mathsf{Dec}}^{\mathsf{c}}(\mathsf{sk}_2) = \Lambda_{\mathsf{Dec}}^{\mathsf{c}}(\mathsf{sk}_1 + \mathsf{sk}_2)$. This allows decryption to be carried out when the secret key sk is distributed among different parties.

1.10.5 Threshold Fully Homomorphic Encryption (ThFHE).

We work in a distributed context, so we need to adapt the standalone Definition 6 in the event of several parties wishing to perform a computation together. This leads to the definition of a new family of *threshold* (or sometimes multiparty) FHE schemes, or ThFHE in short, that makes it possible to design MPC protocols as described in Section 1.11. In this dissertation, we do not directly define this new family, but instead describe in Chapters 3 and 4 how to efficiently transform any FHE scheme that verifies the linearity properties outlined in Definition 7, into a ThFHE scheme using a formalism introduced in Chapter 2.

Generally speaking, the main idea behind threshold schemes is that only a set of more than t parties out of n collaborating together can decrypt an encrypted secret. To this end, the

secret key sk is now distributed such that each of the parties has a share of it, and the key generation and the decryption algorithms are now interactive protocols.

1.10.6 Ring Learning with Error.

The security of encryption schemes usually relies on some hardness assumptions, such as the learning with error (LWE) assumption introduced by Regev [Reg05]. In this dissertation, we will largely rely on a variant of this cryptographic assumption over polynomial rings, that is assumed to be computationally hard enough to be used for cryptography.

Notations Recall that 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 respectively. Let $\mathscr{X}_q$ and Ψ_q be distributions over R_q, where the coefficients of the latter are sampled from a bounded discrete Gaussian distribution of small variance σ^2 and small bound B.

The* decisional-*Ring Learning with Errors Problem (RLWE)**.** The 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 $(a_i, b_i = s\,a_i + e_i)_i$, 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.

1.10.7 Gadget Decomposition.

A gadget decomposition technique is commonly used for constructing efficient lattice-based FHE schemes [BIP+22; CGGI20; JLP23] for reducing the noise growth of homomorphic operations. Below, let us define the widely used, e.g., [GSW13; CDKS19; GMP19], *gadget toolkit*:

1. Gadget vector: $\mathbf{g} = (g_0, g_1, ..., g_{l-1}) \in R_q^l$; and integers l and a (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 $\sum i = 0^{l-1} u_i.g_i = x \pmod q$.

1.11 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 parties collaborates to compute any function on their inputs, while preserving the confidentiality of those inputs, due to their minimal communication overhead [AJL+12].

Overall, instantiating an MPC protocol from a ThFHE scheme is not straightforward, and involves multiple steps described in Figure 1.9 and recalled below:

Distributetd Key Generation (DKG)**:** a protocol in which the parties (1) collaboratively establish a common threshold encryption key ek for a FHE scheme, and each party P_i also obtains a share sk_i of the secret key sk. The threshold encryption key ek is then made public to potential input-owners;

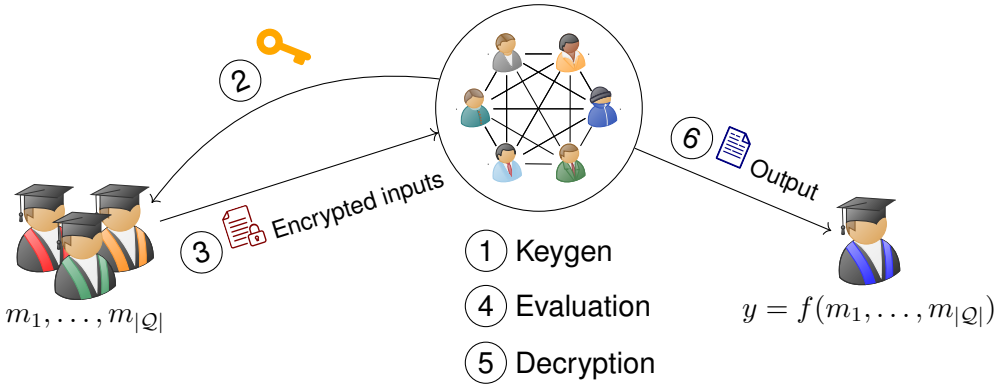


Figure 1.9: Parties generate keys and send the threshold encryption key to the input-owners, that encrypt their inputs and broadcast them back. Then, the computation is performed on these encrypted data and parties collectively perform a threshold decryption towards the output learner.

Input Distribution: input-owners encrypt their inputs using ek and (3) broadcast them;

Evaluation: parties (locally) perform (4) homomorphic computations on the ciphertexts to evaluate the desired function;

Threshold Decryption: parties then (5) jointly execute a threshold decryption protocol to uncover the computation's output, which is then sent to the output learner.

A lower-level description will be done in Chapters 3 and 4. For ease of comprehension, let us note that the protocol described above is made up of 3 rounds of interaction: one for generating the keys, one to distribute the encrypted inputs, and a last one for threshold decryption.

Delegated Computation. The main advantage of using a fully homomorphic encryption scheme to securely evaluate a circuit is that the computation can be performed non-interactively once the keys and inputs have been provided. This makes it easy to apply for delegated MPC, as defined in Section 1.6 and depicted in Figure 1.9, where the "heavy" part of the computation is distributed to a set of computation parties. This approach allows our MPC protocol to scale efficiently to a large number of input-owners, as the complexity of the evaluation remains independent of their number.

Another interesting use case made possible by this model is to leverage a single keyless semi-honest entity, such as a cloud server, to perform the computation. In this scenario, parties are only responsible for generating the keys and for decrypting the output, while the circuit is solely computed by the cloud server. This opens the door to other cases, in which parties would be resource-constrained.

1.12 Related Works

1.12.1 LSS-based MPC.

Linear Secret Sharing schemes (LSS, see Definition 3) have emerged as a popular foundation for constructing MPC protocols. Their appeal lies in their simplicity and the homomorphic properties that enable efficient computations on secret-shared data. Following the seminal work by Ben-Or, Goldwasser, and Wigderson [BGW88], many approaches [Bea91; DPSZ12] have been proposed to build more efficient and robust protocols, and have led to a number of practical implementations such as the Sharemind framework [BLW08] and the MP-SPDZ library [Kel20].

However, LSS-based MPC protocols still face practical limitations that can hinder their real-world usability. One major limitation is the high communication complexity, which grows with the number of participants and the size of the circuit. This issue is exacerbated in wide-area network settings, where latency can significantly impact performance and increase the overall computation time. Additionally, these protocols involve a number of communication rounds proportional to the depth of the circuit being computed. Finally, most protocols, including those following the SPDZ framework [DPSZ12], require a single-use, correlated random value shared among all parties, with a size proportional to both the circuit size and the number of participants. Distributing these randomnesses securely can be challenging and often requires the use of alternative primitives such as FHE [KPR18] or oblivious-transfer [KOS16], which can quickly become the dominant factor in the overall protocol execution time.

1.12.2 Multikey FHE.

Multikey FHE schemes, as introduced by López-Alt et al. [LTV12], enable the evaluation of homomorphic operations directly over ciphertexts encrypted under different secret keys. This paves the way to an ad-hoc form of MPC, where no DKG is run before the distribution of the inputs, as will be later shown in Figure 4.1b. Thus, these schemes allow for a dynamic set of parties to enter the protocol i.e. they enable to include on-the-fly new parties during the circuit computation.

The primary application of multikey schemes is to design MPC protocols with minimal rounds. For instance, granted that parties have access to a common uniform random string, multikey schemes enable the construction of two-round actively secure MPC protocols, as demonstrated in [MW16]. However, this low number of rounds comes at the cost of a much greater computation complexity than their single-key FHE counterparts. In turn, even recent schemes [CDKS19; CCS19; KMS24; KKL+23; KÖA23] have their ciphertext size and homomorphic operation complexity that increase at least linearly with n.

We observe that while at first sight the use of a multikey FHE scheme seems particularly suited to delegated MPC, it actually poses major challenges. First, in its classical conception, input-owners encrypt their inputs under their own keys, before distributing them. This effectively prevents separate roles for input-owners and computing parties, which is essential for easy delegation of the computation. Maybe even more importantly, computation under a multikey scheme scales poorly with the number of input-owners involved. Their use is therefore still restricted to limited use-cases. However, let us point out that we propose in Chapter 4 a new

approach that can be considered as a hybrid between multikey and standard threshold constructions, where no DKG needs to be run before distributing encrypted inputs while obtaining ciphertext sizes independent of the number of parties n.

Chapter 2

$\mathcal{F}_{\mathsf{LSS}}$: Linear Secret Sharing Functionality

Contents

In this chapter, we formalize a Linear Secret Sharing functionality $\mathcal{F}_{\mathsf{LSS}}$ that will prove a very useful tool to later design MPC protocols. More specifically, we are interested in how a set of parties can efficiently delegate a computation on their inputs. The computation performed is ideally as general as possible, and the system is adaptable to as many settings as possible. In MPC protocols, it is generally agreed that the parties settled on a circuit to be computed before any operation is undertaken or any input is distributed. However, this assumption turns out to be a strong one when many parties are considered, as some may be added during the course of the protocol, and not all parties may agree on the calculations to be made later.

In short, we are aiming for a functionality with a *reusability property*, where the inputs and the functions to be computed are dissociated. That way, no function-specific precomputation needs to be carried out by input-owners and distributed inputs can be re-used multiple times for different functions. Therefore, even if input-owners leave the system, the functionality has enough information to pursue and eventually output.

For later use in Chapters 3 and 4, we focus on the special case of linear combinations. Intuitively, this follows on from the remarks made earlier in Section 1.10.4, where we noted that for some encryption schemes, the key generation, encryption, and decryption algorithms are essentially linear maps. The ability to share inputs and compute linear combinations on them will come in handy when thresholdizing these ℓ-HE schemes, as this will help us to reduce the number of interactions and improve the efficiency of distributed protocols.

We describe in Figure 2.1 the overall idea of the functionality, that we call $\mathcal{F}_{\mathsf{LSS}}$, that will be used to compute linear combinations of shared data. In a nutshell, a set of senders are able to send their inputs $m_1, \ldots, m_n$ to the functionality without knowing which linear combination will be computed on them. Later, on input a linear map Λ, the linear combination $\Lambda(\{m_i\}_i)$ is computed and output. Importantly, distributed inputs can be reused for several different computations (say of another linear combination $\Lambda'(\{m_i\}_i)$) without requiring the senders to send anything again.

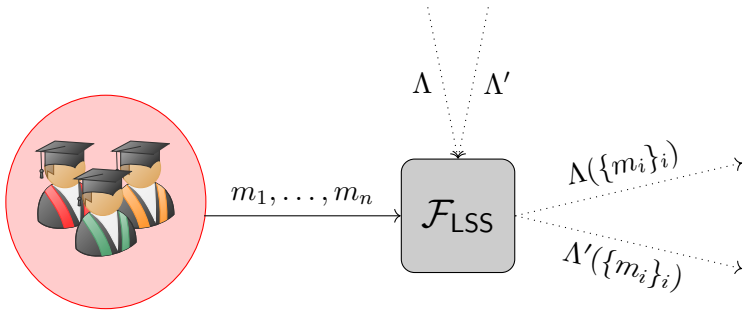


Figure 2.1: Overall design requirements of $\mathcal{F}_{\mathsf{LSS}}$. Senders distributed their inputs to the functionality, that on input a linear map Λ, computes and outputs the linear combination.

Let us give a dummy example to better understand how it is intended to work. Let us imagine that an unknown number of senders wants to compute an average on a set of personal data, e.g. their salary, without a priori knowing this number. The functionality $\mathcal{F}_{\mathsf{LSS}}$ we aim at building accepts a certain number of values, and is later parameterized by a linear map Λ, here to compute an average. It outputs the average over these data. Importantly, $\mathcal{F}_{\mathsf{LSS}}$ can

later consider another linear map Λ' and compute the linear combination without the senders having to take any other action.

This chapter is organized as follows: we first detail the technical challenges in Section 2.1, before formalizing our new Linear Secret Sharing functionality $\mathcal{F}_{\mathsf{LSS}}$ in Section 2.2. Then, we define the useful cryptographic primitive of Publicly Verifiable Secret Sharing (PVSS) in Section 2.3, before discussing in Section 2.4 how to implement a LSS scheme over a polynomial Ring for later use in Chapters 3 and 4. We prove in Section 2.5 that our construction of PVSS is IND-CPA, before finally describing in Section 2.6 the implementation of $\mathcal{F}_{\mathsf{LSS}}$ and its security.

2.1 Technical Challenges

We detail in Sections 2.1.1 and 2.1.2 the technical challenges for $\mathcal{F}_{\mathsf{LSS}}$ to be instantiated, for later use in the rest of this dissertation.

2.1.1 Efficient Linear Secret Sharing over R_q.

We will consider in Chapters 3 and 4 efficient homomorphic encryption schemes in which the ciphertext space is a polynomial ring R_q, as defined in Section 1.10.6. We thus need to have instantiations of a (n, t)-LSS scheme (see Definition 3) defined over polynomial rings. There are two main difficulties in constructing such schemes.

1. First, following Definition 3 in Section 1.10.1, one needs to be able to instantiate two efficient algorithms ShSim and ShInfer from a (n, t)-LSS scheme, which will turn out to be very important in Section 2.1.2. To this end, we introduce a new subclass of $(n, t)-$LSS denoted as $(n, t)-$LSSD, for which we detail a generic instantiation strategy for ShSim and ShInfer.

2. Second, as previously stated, the considered schemes must be defined over polynomial rings, which turned out not to be obvious, for example in the case of Shamir. Thus, we present a new variant of Shamir defined over a polynomial ring, denoted as R_{p^e}-Shamir. Importantly, it is defined over R_q, including the useful case where q is a power $q = p^e$ of a prime, itself possibly small $p \leqslant n$. The latter case proves useful in practice [CH18; GIKV23] to speed-up homomorphic operations for the BGV[BGV12] and BFV [FV12] FHE schemes.

In short, we explain how to implement a Linear Secret Sharing scheme over a polynomial ring, which will help us throughout the rest of this dissertation.

2.1.2 Simulatability of PVSS, without straight-line extraction.

We now look back to Definition 3 and the introduction of the notion of "Inference of shares", which was not present in the formalism used so far in [BGG+18; CCK23]. This is not just an addition of formalism, but, an important ingredient in the design of effective cryptographic schemes and MPC protocols. Indeed, this allows us to describe simulators for PVSS/MPC that do not perform straight-line extractions, but instead use rewinding, allowing the use of more efficient schemes.

In more detail, in our UC proof in Section 2.6.2, we introduce a hybrid in which ShInfer is used to infer corrupt shares instead of being extracted from the broadcast PVSSs (formally defined in Definition 8 later in Section 2.3). As a by-product, it positively answers the question raised by Shrestha-Bhat-Kate-Nayak in the first version of [SBKN21, p5], whether PVSS would be simulatable. Moreover, under honest majority, we achieve it from any NIZK Argument of Knowledge with simulation-soundness after possibly rewinding, i.e., weak simulation extractability [FKMV12], in other words, not necessarily in straight-line. This includes Bulletproofs and Spartan [BBB+18; Set20], as very recently shown under DLOG by [DG23].

We were later informed of other works doing simulation proofs for PVSS [CD20; GHK+21; Div22], but all of them require NIZK with straight-line, a.k.a. online, extractability, which ruled-out Bulletproofs and Spartan.

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

We now specify, in Figure 2.2, an ideal functionality for linear secret sharing, denoted $\mathcal{F}_{\mathsf{LSS}}$. It is parametrized by $i)$ a set $\mathscr{P}$ of n parties, $ii)$ 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 $\forall P \in \mathscr{P}$. However, the adversary $\mathcal{A}$ can choose to never instruct corrupt parties to setup. To remedy this, we follow the *fetch-and-delay* mechanism explained in Section 1.7.1 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 parties. The former cannot be subsequently updated; once sent, the sender $\mathcal{S}$ is committed to the submitted values.

Opening. Let HOpeners be a set of parties, initially empty. Any party P_i can call LCOpen for some linear map Λ, and is then included in HOpeners.

Upon receiving LCOpen for some linear map Λ from $t+1$ honest parties, i.e. 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 party would start the LCOpen protocol, i.e. starting revealing its shares of the evaluation. Since it is hard to prevent t corrupt parties from also publicly disclosing compatible consistent shares, this results in the evaluation being publicly opened. We qualify such an 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 parties, as soon as one honest party requests (LCOpen).

2.3 Publicly Verifiable Secret Sharing (PVSS)

Let $\mathsf{PKE} = (\mathsf{PKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be any public key encryption scheme as defined in Definition 5 in Section 1.10.2, satisfying IND-CPA. We introduce the following definition:

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

Participants: A set $\mathscr{S}$ of senders, an output learner $\mathcal{L}$, and a set $\mathscr{P}$ of parties.
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 $P \in \mathscr{P}$ for the first time, or possibly from $\mathcal{A}$ if P is corrupt: stores (Setup, P), and *eventually-delivers* (Setup, P) to each party $P \in \mathscr{P}$. //eventually-delivers" consists of the same fetch-and-delay mechanism as explained in Section 1.7.1.
- Initialize a counter $T_{\mathcal{A}} \leftarrow 1$ //Formalizing timing for setup of corrupt parties
  - Upon receiving delay - Setup from $\mathcal{A}$, $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} + 1$.
  - Upon receiving fetch from any $P \in \mathscr{P}$, $T_{\mathcal{A}} \leftarrow T_{\mathcal{A}} - 1$.
  - When $T_{\mathcal{A}} = 0$ for the first time, freeze forever $T_{\mathcal{A}} = 0$. Then, send ready to every $\mathcal{S} \in \mathscr{S}$.

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

$\mathcal{A}$ delaying eventual delivery

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

Bookkeeping requests from honest parties

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

LCOpen

- [Early Opening] If |HOpeners| ≥ 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 $(\mathsf{open\text{-}order}, \Lambda)$ from $\mathcal{A}$, if no output was delivered yet to $\mathcal{L}$, then send $(\mathsf{ssid} = \Lambda, y := \Lambda((x_{\mathcal{S},\alpha})_{\mathcal{S},\alpha}))$ to $\mathcal{L}$.
- [Collective Opening] If |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 $(\mathsf{ssid} = \Lambda, y := \Lambda((x_{\mathcal{S},\alpha})_{\mathcal{S},\alpha})$ to $\mathcal{L}$.

[a] Once a sender $\mathcal{S}$ (or $\mathcal{A}$) send an input $x_{\mathcal{S},\alpha}$ with label $\overline{x_{\mathcal{S},\alpha}}$, the former cannot be subsequently updated.
[b] Appended with "$x_{\mathcal{S},\alpha} = \perp$" when this is the case.
[c] Recall that we consider in this description a unique Λ. If multiple are considered, then several sets $\mathsf{HOpeners}_\Lambda$ must also be considered.

Figure 2.2: Sharing with Linear Combination functionality for one single linear map Λ. sid omitted

Definition 8. *(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

$$\mathscr{R}_{\mathsf{Share}} = \left\{ \begin{array}{l} x = (\{\mathsf{pk}_i^{\mathsf{PKE}}\}_{j\in[n]}, \mathsf{enc\text{-}shares}) \\ w = (s, \mathbf{r}, \{\rho_i\}_{i\in[n]}) \end{array} \middle| \begin{array}{l} \wedge \{\mathbf{s}^{(i)}\}_{i\in[n]} \leftarrow \mathsf{LSS.Share}(s; \mathbf{r}) \\ \wedge \mathsf{enc\text{-}shares} \leftarrow [\mathsf{Enc}(\mathsf{pk}_i^{\mathsf{PKE}}, \mathbf{s}^{(i)}; \rho_i)]_{i\in[n]} \end{array} \right\} \tag{2.1}$$

PVSS *is* IND-CPA *secure for any $\mathcal{A}$ being given at most t decryption keys $\left(\mathsf{dk}_i^{\mathsf{PKE}}\right)_{i\in\mathcal{I}\subset[n],|\mathcal{I}|\leq t}$, as will be shown in Section 2.5.*

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

Remark. *We describe the MPC protocols in Chapters 3 and 4 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.*

State-of-the-art implementations of PVSS can be found in [GV22; KMM+23]. The former includes NIZKs of smallness of the secret, which will be needed, e.g., for sharing noises.

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

2.4 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 3 over polynomial rings. There are two main difficulties in constructing such schemes. First, following Definition 3 in Section 1.10.1 one needs to be able to instantiate two efficient algorithms ShSim and ShInfer, which will turn out to be very important for our UC proofs as explained Section 2.1.2. Second, these schemes must be defined over polynomial rings, which turned out not to be obvious, for example in the case of Shamir. To address these challenges, we follow the roadmap below:

1. We first introduce a subclass of $(n,t)-$LSS that we denote as $(n,t)-$LSSD in Definition 39 in Appendix A. The latter will be used as a helper to describe a common instantiation strategy for ShSim and ShInfer that encompasses classic sharing schemes.
2. We then recall in Section 2.4.1 the classical $\{0,1\}-$LSS scheme of [JRS17] and show in Property 11 that it is indeed a $(n,t)-$LSSD scheme. In particular, we show that it verifies both the simulatability and inference properties.
3. Finally, we present in Section 2.4.2 our new Shamir scheme over polynomial rings, denoted as R_{p^e}-Shamir, and show that it also is a $(n,t)-$LSSD scheme.

In a nutshell, we show in this section the following:

$$R_{p^e}\text{-Shamir} \cup (n,t)-\{0,1\}-\text{LSS} \underset{\text{Prop. } 11\&15}{\subset} (n,t)-\text{LSSD} \underset{\text{Definition 39}}{\subset} (n,t)-\text{LSS} \tag{2.2}$$

$(n,t)-$LSSD**.** As a preliminary, we define $(n,t)-$LSSD in Definition 39 in Appendix A. The purpose of this definition, which is adapted from [JRS17], is to serve as a helper to prove that Shamir, the $\{0,1\}-$LSS scheme of [JRS17] and the recent TreeSS scheme of Cheon et al. [CCK23] all verify properties (4) and (5) of Definition 3. We give details, in Proposition 41, about a common strategy to instantiate ShSim and ShInfer for all these schemes.

2.4.1 Constructing a $(n,t)-$LSSD scheme from a $(n,t)-\{0,1\}$-LSS scheme

We first consider a class of linear secret sharing schemes, denoted as $(n,t)-\{0,1\}$-LSS, in which the reconstruction coefficients are always binary (see Definition 9). More precisely, such a secret sharing scheme divides a secret s into a set of shares $\mathbf{s}^{(1)},\dots,\mathbf{s}^{(n)}$ such that each share consists of a set of elements $\mathbf{s}^{(i)} = \{\mathbf{s}_j^{(i)}\}_{j\in[l]}$ for a fixed bound l. For any set $S \subseteq [n]$ that satisfies the access structure, i.e such that $|S| > t$, there exists a subset $S' \subseteq \bigcup_{i\in S'} \mathbf{s}^{(i)}$ such that $\sum_{S'} \mathbf{s}_j^{(i)} = s$, i.e. with binary recovery coefficients.

Definition 9. *(Strong $\{0,1\}$-Reconstruction). We say that a* LSS *scheme has strong $\{0,1\}$-reconstruction if for any secret s and $(\mathbf{s}^{(1)},\dots,\mathbf{s}^{(n)}) =$ LSS.Share(s), for any valid set of shared elements $T \subseteq [n]\times[l]$, there exists a subset $T' \subseteq T$ such that $\sum_{i,j\in T'} \mathbf{s}_j^{(i)} = s$, where $\mathbf{s}^{(i)} = (\mathbf{s}_1^{(i)},\dots,\mathbf{s}_l^{(i)})$.*

We denote any LSS scheme with this reconstruction property as a $\{0,1\}$-LSS scheme. In other words, the reconstruction function takes a $0/1$ combination of its inputs.

In the following definition, we require this to hold not only for any valid set of shared elements, but also for any set of shares corresponding to a valid set of more than $t+1$ parties, i.e. when each party receives multiple shares. This property defines the notion of a $\{0,1\}$-LSSD scheme in [JRS17].

Definition 10. *($(n,t)-\{0,1\}$-LSS particularized from [JRS17]) Let $\mathscr{P} = \{P_1,\dots,P_n\}$ be a set of parties. We denote by $\{0,1\}$-LSS$_n$ the collection of all sets of subsets $\mathcal{U} \in \mathcal{P}(\mathscr{P})$[1] with a size of at least $t+1$, such that there exists an $l \in \mathbb{N}$ polynomial in n and some set of parties $\mathscr{P}' = \{P'_1,\dots,P'_{ln}\}$ such that we can associate the party $P_i \in \mathscr{P}$ with the parties $P'_{l(i-1)}, P'_{l(i-1)+1},\dots,P'_{li} \in \mathscr{P}'$ as follows.*

For every $\mathcal{U} \subseteq [n]$, $|\mathcal{U}| > t$ if and only if the set $\mathcal{U}'$ of parties of $\mathscr{P}'$ is associated with a party in $\mathcal{U}$, $|\mathcal{U}'| > t$. More precisely, for every $\mathcal{U} \subseteq [n]$,

$$\Big|\bigcup_{i\in\mathcal{U}}\{P_i\}\Big| > t \text{ if and only if } \Big|\bigcup_{i\in\mathcal{U}}\{P'_{l(i-1)}, P'_{l(i-1)+1},\dots,P'_{li}\}\Big| > t. \tag{2.3}$$

[1] Where $\mathcal{P}(\mathscr{P})$ represents the power set of $\mathscr{P}$.

Construction. We refer to [JRS17] for an example of construction of a $(n,t)-\{0,1\}-$LSS scheme. Overall, it uses a classic result of Valiant [Val84] to describe a randomized construction of a monotone Boolean formula for threshold functions with size $O(n^{5.3})$, before describing a "folklore" algorithm to go from special monotone formulas to $(n,t)-\{0,1\}-$LSS sharing matrices (following the formalism of Definition 39 in Appendix A).

Property 11. *The $(n,t)-\{0,1\}-$*LSS *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 41 in Appendix A. □

Remark. *Note that the simulatability of a $\{0,1\}-$*LSS *scheme had been used without being formalized in [BGG+18; CCK23]. Inference, on the other hand, is a new ingredient that makes it possible not to use straight-line extraction in our UC proofs*

Remark. *This construction leads to an average share size of $O(n^{4.3})$. Note that following Definition 3, such scheme has the property that the distribution of any set of t shares is $U(R_q^t)$.*

Remark. *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.*

2.4.2 Shamir Secret-Sharing in R_q

The usual Shamir secret-sharing scheme described in Section 1.10.1 is instantiated over a field $\mathbb{F}$. Indeed, Shamir is based on polynomial interpolation and involves the computation of Lagrange coefficients, which 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. Overall, we will proceed as follows:

1. First, 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 12), which exists when the prime factor q is of size at least $n+1$.
2. Then, we propose a new construction, denoted as R_{p^e}-Shamir that extends this result to the useful case where q is a power $q = p^e$ of a prime, itself possibly small $p \leqslant n$.

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 12 below.

Definition 12. *From [ACD+19] For a ring $\mathcal{R}$, the sequence $\alpha_1, \ldots, \alpha_n$ of elements of $\mathcal{R}$ is an* exceptional sequence *if $\alpha_i - \alpha_j$ is a unit in $\mathcal{R}$ for all $i \neq j$.*

Provided with an exceptional sequence over $\mathcal{R}$, we then have the following Theorem 13.

Theorem 13. *From [ACD+19] Let $\mathcal{R}$ be a commutative ring and $\alpha_1, \ldots, \alpha_n$ be an exceptional sequence in $\mathcal{R}$. Then, a Shamir secret-sharing scheme instantiated in $\mathcal{R}$ 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 (or Shamir public points) 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 $\{(i,j) \in [0,\ldots,n]^2, i \neq j\}$ 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 $(\alpha_0 := 0, \alpha_1, ..., \alpha_n)$, 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 $(\alpha_i)_{i=0,\ldots,n}$ the same previous construction as for Shamir defined in Section 1.10.1. 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 14. *(R_{p^e}-Shamir) Let e be an integer and p a prime. There exists a Shamir secret-sharing scheme instantiated in R_{p^e} that is correct and secure.*

Property 15. R_{p^e}-Shamir *is a* $(n,t)-$LSSD *scheme.*

Proof. This directly follows from Definition 39 presented in Appendix A, when considering the Vandermonde matrix presented in Equation (1.6) as sharing matrix. □

To summarize, we consider an exceptional sequence $\alpha_1, \ldots, \alpha_n$, 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 16. *Let p be a prime and e and integer. For every $s \in R_{p^e}$, for any subset of t indices $\mathcal{I} \subset [n]$, the distribution of shares $(s^{(i)})_{i\in\mathcal{I}}$ output by R_{p^e}-Shamir.Share(s) is $U(R_{p^e}^t)$.*

Proof. For any commutative ring $\mathcal{R}$ with unit 1, we denote as $\mathcal{R}[Y]_t$ 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:

- $\mathsf{Eval}_{\mathcal{U}} : h \in \mathcal{R}[Y]_t \to [h(\alpha_i), \alpha_i \in \mathcal{U}]$: 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 LSS$[\mathcal{R}]$.Share : $\mathcal{R} \longrightarrow \mathcal{R}^n$, defined as: on input a secret $s \in \mathcal{R}$, sample $h \leftarrow U\big(\mathcal{R}[Y]_t^{(s)}\big)$ then return $\mathsf{Eval}_{(\alpha_i)_{i\in[n]}}(h)$ denoted shares of s.

By surjectivity (isomorphism) of $\mathsf{Eval}_{\{0\}\cup\mathcal{I}} : \mathcal{R}[Y]_t \to \mathcal{R}^{t+1}$ for any t-sized subset $\mathcal{I}$ of indices of $(\alpha_i)_{i\in[0,\dots,n]}$, we have surjectivity (isomorphism) of $\mathsf{Eval}_{\mathcal{I}} : \mathcal{R}[Y]_t^{(s)} \to \mathcal{R}^t$ for any fixed $s \in \mathcal{R}$. 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^e}-Shamir, we have in conclusion the desired result. □

2.5 Proof of IND-CPA of Publicly Verifiable Secret Sharing

Proposition 17 states that any PPT adversary $\mathcal{A}$ corrupting at most t parties, has negligible advantage in distinguishing between the encrypted $(n,t)-$LSS sharings of any two chosen secrets $(s_L, s_R) \in R_q^2$. Recall that in this section, we consider any public key encryption scheme PKE $=$ (PKeyGen, Enc, Dec) satisfying IND-CPA as defined in Definition 5 in Section 1.10.2.

Proposition 17 (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).*

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

Setup. $\mathcal{A}_{\mathsf{PVSS}}$ *gives to $\mathcal{O}$: 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 $i \in [n]\backslash\mathcal{I}$, $\mathcal{O}$ generates $(\mathsf{dk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PKeyGen}(1^\lambda)$ 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 $(\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(n)}) \leftarrow \mathsf{Share}(s_L)$ and returns $(\mathsf{Enc}_{\mathsf{pk}_i}(\mathbf{s}^{(i)}))_{i\in[n]}$

$\mathscr{O}^{\boldsymbol{R}}$: computes $(\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(n)}) \leftarrow \mathsf{Share}(s_R)$ 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 $b' = b$.*

Figure 2.3: IND-CPA of encrypted sharing

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

Proof Strategy: In order to prove Proposition 17, we follow the following steps:

1. We first consider in Figure 2.4 the straightforward reduction into the slightly easier variant of the game, which we denote as IND-CPA *of encrypted sharing with plaintext adversary shares*. There, the adversary $\mathcal{A}_{\mathsf{PVSS}^*}$ is given shares of corrupt parties in the clear[2].
2. We then introduce in Figure 2.5 an intermediary oracle that sets the shares of honest parties to 0.
3. Finally, we describe in Figure 2.6 the $(n-t)$-keys IND-CPA game for a PKE scheme.

Our goal is then to bound the advantage by any adversary $\mathcal{A}_{\mathsf{PVSS}^*}$ in the game presented in Figure 2.4, by the maximum advantage of an adversary $\mathcal{A}_{\mathsf{PKE}}$ in the (n-t)-keys variant indistinguishability game for PKE presented in Figure 2.6.

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 $\widetilde{\mathscr{O}}^L$ and $\widetilde{\mathscr{O}}^R$, which only differ from the previous, in that they *first* sample the corrupt shares $(\mathbf{s}^{(i)})_{i\in\mathcal{I}} \xleftarrow{\$} R_q^t$[3] 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 produced exactly the same distribution as $\mathscr{O}^L$ and $\mathscr{O}^R$. We describe them below in Figure 2.4, then formalize the previous claim in Equation (2.4).

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

Setup. $\mathcal{A}_{\mathsf{PVSS}^*}$ gives to $\mathcal{O}$: 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 $i \in [n]\backslash\mathcal{I}$, $\mathcal{O}$ generates $(\mathsf{dk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PKeyGen}(1^\lambda)$ 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 $[n] \setminus \mathcal{I}$. Pre-

[2]Notice that this variant is strictly easier when $\mathcal{A}_{\mathsf{PVSS}^*}$ badly generated some of its keys, and thus is unable to decrypt the shares of corrupt parties. Formally, the reduction from $\mathsf{Game}^{\mathcal{A}_{\mathsf{PVSS}}}_{\mathsf{IND-PVSS}}$ to $\mathsf{Game}^{\mathcal{A}_{\mathsf{PVSS}^*}}_{\mathsf{IND-PVSS}^*}$ proceeds as follows. Consider a challenger $\mathcal{A}_{\mathsf{PVSS}}$ of Proposition 17. The reduction forwards the requests of $\mathcal{A}_{\mathsf{PVSS}}$ to its oracle. It receives the plaintext shares $(\mathbf{s}^{(i)})_{i\in\mathcal{I}}$ and the ciphertexts $(c_i)_{i\in[n]\setminus\mathcal{I}}$. It forwards to $\mathcal{A}_{\mathsf{PVSS}^*}$: $(\mathsf{Enc}_{\mathsf{pk}_i}(\mathbf{s}^{(i)})_{i\in\mathcal{I}}$ and the same ciphertexts $(c_i)_{i\in[n]\setminus\mathcal{I}}$.

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

cisely, 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 $b' = b$.

Figure 2.4: IND-CPA of encrypted sharing with plaintext adversary shares.

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

$$(2.4)\quad |\mathsf{Pr}(1 \leftarrow \mathcal{A}^{\widetilde{\mathcal{O}}^L}_{\mathsf{PVSS}^*}) - \mathsf{Pr}(1 \leftarrow \mathcal{A}^{\mathcal{O}^L}_{\mathsf{PVSS}^*})| = 0 \text{ and } |\mathsf{Pr}((1 \leftarrow \mathcal{A}^{\widetilde{\mathcal{O}}^R}_{\mathsf{PVSS}^*}) - \mathsf{Pr}(1 \leftarrow \mathcal{A}^{\mathcal{O}^R}_{\mathsf{PVSS}^*})| = 0$$

To conclude the proof, we introduce an intermediary oracle, defined as $\widetilde{\mathcal{O}}^Z$ in Figure 2.5. $\widetilde{\mathcal{O}}^Z$ is the common modification of $\widetilde{\mathcal{O}}^L$ and $\widetilde{\mathcal{O}}^R$, which sets to 0 the $n-t$ honest plaintext shares. In particular, it completely ignores the request (s_L, s_R) given to it.

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

Figure 2.5: Intermediate oracle $\widetilde{\mathcal{O}}^Z$ for the game presented in Figure 2.4.

From Property 16 of uniform independence of the t plaintext shares $(\mathbf{s}^{(i)})_{i\in\mathcal{I}}$, we conclude that the distinguishing advantage between both $\widetilde{\mathcal{O}}^L$ and $\widetilde{\mathcal{O}}^R$, with $\widetilde{\mathcal{O}}^Z$, is negligible.

Claim 18. *The maximum distinguishing advantage with $\widetilde{\mathcal{O}}^Z$ 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 $\mathcal{O}_{\mathsf{PKE}}$ parametrized by a secret $b \in \{E, 0\}$.

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

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

Query. $\mathcal{A}_{\mathsf{PKE}}$ gives to $\mathcal{O}_{\mathsf{PKE}}$ $(n-t)$ chosen plaintexts $(s_h)_{h\in[n-t]}$, then $\mathcal{O}_{\mathsf{PKE}}$ replies depending on $b \in \{E, 0\}$. In the former case, it behaves as $\mathscr{O}^E_{\mathsf{PKE}}$ (actual $n-t$-keys encryption), in the latter case as $\mathscr{O}^0_{\mathsf{PKE}}$ (encryptions of 0).

$\mathscr{O}^{\boldsymbol{E}}_{\mathbf{PKE}}$ returns $(\mathsf{Enc}_{\mathsf{pk}_h}(s_h))_{h\in[n-t]}$;
$\mathscr{O}^{\mathbf{0}}_{\mathbf{PKE}}$ 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 $b' = b$.

Figure 2.6: $(n-t)$-keys IND-CPA Game

Recall that the distinguishing advantage in this $\mathsf{Game}^{\mathcal{A}_{\mathsf{PKE}}}_{(n-t)-\mathsf{IND\text{-}CPA}}$ game, is upper-bounded by $n-t$ times the advantage for one-message indistinguishability, see e.g. [BS, 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 $(\mathcal{O}^E_{\mathsf{PKE}}/\mathcal{O}^0_{\mathsf{PKE}})$. The reduction works as follows.

1. Upon receiving a set of keys $(\mathsf{pk}^{\mathsf{PKE}}_h)_{h\in\mathcal{H}}$ from $\mathcal{O}_{\mathsf{PKE}}$, 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 $\mathcal{A}_{\mathsf{PVSS}^*}$ (we keep the same syntax, but actually s_R is never used here), $\mathcal{A}_{\mathsf{PKE}}$ samples $(\mathbf{s}^{(i)})_{i\in\mathcal{I}} \xleftarrow{\$} R^t_q$ 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 $\mathcal{O}_{\mathsf{PKE}}$ as a request.
3. Upon receiving the response ciphertexts $(\mathsf{c}_h)_{h\in\mathcal{H}}$ from $\mathcal{O}_{\mathsf{PKE}}$, 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 $\mathcal{O}_{\mathsf{PKE}}$.
     And sends it to $\mathcal{A}_{\mathsf{PVSS}^*}$ as response to its challenge.
4. Upon receiving as answer a bit b from $\mathcal{A}_{\mathsf{PVSS}^*}$, then $\mathcal{A}_{\mathsf{PKE}}$ outputs the same bit b to $\mathcal{O}_{\mathsf{PKE}}$.

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

Without computing the probabilities, the idea why $\mathcal{A}_{\mathsf{PKE}}$ has non-negligible advantage with this strategy is that:

- In the case where the ciphertexts $\{c_h\}_{h\in\mathcal{H}}$ are encryptions of the actual $n-t$ shares $\{\mathbf{s}^{(h)}\}_{h\in\mathcal{H}}$, then $\mathcal{A}_{\mathsf{PVSS}^*}$ receives from $\mathcal{A}_{\mathsf{PKE}}$ a correctly generated PVSS of s.

- In the case where the ciphertexts $\{c_h\}_{h\in\mathcal{H}}$ are encryptions of 0, then, by Property 16 of uniform independence of the t plaintext shares $(\mathbf{s}^{(i)})_{i\in\mathcal{I}}$, we have that what $\mathcal{A}_{\mathsf{PVSS}^*}$ receives from $\mathcal{A}_{\mathsf{PKE}}$ is undistinguishable from a sample in the distribution $\mathcal{V}$.

□

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

We now detail in Figure 2.7 a protocol Π_{LSS} that instantiates $\mathcal{F}_{\mathsf{LSS}}$ in the $(\mathsf{BC}, \mathcal{F}_{\mathsf{AT}}, \mathsf{bPKI})$-hybrid model. Recall from Section 2.2, that we consider a set $\mathscr{P}$ of n parties, a set $\mathscr{S}$ of senders, and an output-learner $\mathcal{L}$.

Overall, we make use of a linear secret sharing scheme detailed in Section 2.4 as well as a public key encryption scheme PKE. The combination of the two will be used to distribute the inputs in the form of publicly verifiable secret sharings (PVSS). The linearity of the LSS scheme will then enable the evaluation of linear maps on the shared inputs and open the outputs.

Π_{LSS}

Parameters: Any $\mathsf{PKE} = (\mathsf{PKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and $(n,t)-\mathsf{LSS} = (\mathsf{Share}, \mathsf{Reco})$ schemes and, from them, the PVSS algorithm detailed in Definition 8.

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

Π_{LSS}**.Input** :

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

Π_{LSS}**.LCOpen(Λ)** :

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

Figure 2.7: Protocol for secret-sharing then linear combination

In practice, the main feature of this protocol is that, after a unique round of broadcast, parties have a common view of the set of shared secrets. Subsequently, they can perform the threshold opening of the evaluation of *any* linear map over the shared secrets, using only one step of all-to-all asynchronous peer-to-peer messages.

How to compute a linear map Λ on shared secrets? We consider a system in which each party $P_i \in \mathscr{P}$ has published a public key $\mathsf{pk}_i^{\mathsf{PKE}}$ on bPKI. Now, let us consider a set of senders $\mathscr{S}$ that wishes to share some secret values and evaluate some linear map Λ on them.

To share a secret s, each sender $\mathcal{S}$ first generates a (n,t)-linear secret sharing of it. Let $\{s^{(i)}\}_{i\in[n]}$ the vector of shares obtained. It then encrypts each share $s^{(i)}$ under P_i's public key $\mathsf{pk}_i^{\mathsf{PKE}}$. Following Definition 8, the n-sized vector of ciphertexts obtained is called a publicly verifiable secret sharing (PVSS)[4]. This vector is then broadcast.

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

Remark. *In the malicious case, we let senders append NIZKs of knowledge of plaintexts and of a degree t polynomial as defined in Definition 8*

Security.

Proposition 19. *Protocol Π_{LSS} UC implements $\mathcal{F}_{\mathsf{LSS}}$*

The intuition is that, if PKE perfectly hides the plaintexts, then we are almost brought back to the basic protocol in [CDN15, p. III] for computing a linear combination with information-theoretical MPC. The "almost" being that Π_{LSS} furthermore prevents malicious senders from aborting after having sent shares to some parties but none to others, thanks to the broadcast. We provide a detailed proof in Sections 2.6.1 and 2.6.2.

2.6.1 Description of the Simulator Sim

We describe in Figure 2.8 the simulator for an honest $\mathcal{L}$[5] and a linear map Λ. Sim initiates in its head: a sets $\mathscr{P}$ of n parties and $\mathscr{S}$ of senders, and may initially receive corruption requests from Env for arbitrarily many senders and up to t parties, indexed by $\mathcal{I} \subset [n]$.

It simulates the functionalities $(\mathsf{bPKI}, \mathcal{F}_{\mathsf{AT}}, \mathsf{BC})$ following a correct behavior. For instance, upon receiving some instruction from Env intended to some functionality, Sim internally sends it to the functionality then simulates the steps taken by the functionality accordingly. Upon every output from a simulated functionality to a simulated corrupt party, or, upon every message from a simulated functionality to Sim acting as $\mathcal{A}$, then Sim immediately forwards it to Env, as would have done the actual dummy $\mathcal{A}$.

Following its specification, $\mathcal{F}_{\mathsf{LSS}}$ relays to Sim every message from the actual dummy honest parties (LCOpen). Sim has an exact control over the delays of *early* and of *collective opening*, which is enabled by the instructions open-order and delay which it sends to $\mathcal{F}_{\mathsf{LSS}}$.

Remark. *Note that for simplicity, we describe the simulator for the opening of only one evaluation of one linear map. The case of multiple openings is handled as in [CDN15, p127], when they simulate each new* LCOpen*.*

[4] Recall that in the current semi-malicious model, the appended NIZK proof is dropped

[5] The case where the output learner is corrupt is easy. Namely, the simulator plays Π_{LSS} honestly, then indistinguishability follows from the correctness of Π_{LSS}.

Sim

Share : Sim does the following:

- For any simulated honest sender $\mathcal{S} \in \mathscr{S}$, the simulator Sim sets $\widetilde{x_{\mathcal{S},\alpha}} := 0$ for all $\alpha \in X_{\mathcal{S}}$, and follows the protocol.
- For any simulated corrupt $\mathcal{S} \in \mathscr{S}$, the simulator Sim first extracts the secret $x_{\mathcal{S},\alpha}$ for all $\alpha \in X_{\mathcal{S}}$ by simply decrypting shares of simulated honest parties, and sends (input, $\overline{x_{\mathcal{S},\alpha}}$, $x_{\mathcal{S},\alpha}$) to $\mathcal{F}_{\mathsf{LSS}}$.

LCOpen : Upon being leaked by $\mathcal{F}_{\mathsf{LSS}}$ that a dummy honest party P_i sent $(\mathsf{LCOpen}, \Lambda)$, then do the following. If all $\overline{x_{\mathcal{S},\alpha}}$ appearing with nonzero coefficient in Λ are stored and if the simulated P_i was already assigned a simulated opening share of $y := \Lambda((x_{\mathcal{S},\alpha})_{\mathcal{S},\alpha})$, then, the simulated P_i sends this share to $\mathcal{L}$. Otherwise, do the following steps to assign simulated opening shares to P_i, then proceed as above.

1. *If* $\mathcal{F}_{\mathsf{LSS}}$ did not leak the stored output y yet, then, assign an arbitrary opening share to P_i.
2. *Else* this means that Sim was just leaked the evaluation y from $\mathcal{F}_{\mathsf{LSS}}$. Then, do the following assignments from now on. Simulated parties who followed the protocol so far have received a set $\{\mathsf{enc\text{-}shares}_{\mathcal{S},\alpha}\}_{\mathcal{S}\in\mathscr{S},\alpha\in X_{\mathcal{S}}}$ of PVSSs of some $x_{\mathcal{S},\alpha}$. The simulator Sim does the following:
   (a) First, it decrypts the honest shares $\big(x^{(i)}_{\mathcal{S},\alpha}\big)_{i\in\mathcal{H},}$ of $\{\mathsf{enc\text{-}shares}_{\mathcal{S},\alpha}\}_{\mathcal{S}\in\mathscr{S},\alpha\in X_{\mathcal{S}}}$, and computes the opening shares $\{\widetilde{\mu^{(i)}} = \Lambda\big((x^{(i)}_{\mathcal{S},\alpha})_{\alpha\in X_{\mathcal{S}},\mathcal{S}\in\mathscr{S}}\big)\}_{i\in\mathcal{H}}$.
   (b) Second, it does a corrupt share inference: from the $t+1$ opening shares $\{\widetilde{\mu^{(i)}}\}_{i\in\mathcal{H}}$ of $\widetilde{y}$ held by simulated honest parties, it applies ShInfer to deduce the t corrupt opening shares $\{\widetilde{\mu^{(i)}}\}_{i\in\mathcal{I}}$.
   (c) Finally, it deduces the (unique) $n-t$ opening shares $\{\mu^{(i)}\}_{i\in\mathcal{H}}$ of honest parties as $\mathsf{ShSim}(y, (\widetilde{\mu^{(i)}})_{i\in\mathcal{I}})$.

Figure 2.8: Description of the simulator for Π_{LSS}

2.6.2 Proof of indistinguishability with a Real execution

We go through a series of hybrid games, starting from the real execution $\mathrm{REAL}_{\mathcal{A}}$. The view of Env consists of its interactions with $\mathcal{A}/\mathsf{Sim}$, and of the outputs of the actual honest parties.

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

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

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

In this hybrid, we change the method of computation of the opening shares of honest parties. To do so, we first define quantities denoted *Inferred Corrupt Opening Shares* $(\mu^{(i)})_{i\in\mathcal{I}}$, nonwithstanding corrupt parties 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 $\left(x^{(i)}_{\mathcal{S},\alpha}\right)_{i\in\mathcal{I}}$ 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 $\left(x^{(i)}_{\mathcal{S},\alpha} := 0\right)_{i\in\mathcal{I}}$, otherwise (ii) this implies that $*$ is a correctly formed PVSS. Thus in this case, we define as $\left(x^{(i)}_{\mathcal{S},\alpha}\right)_{i\in\mathcal{I}}$ the plaintext shares read on the witness tape of $\mathcal{S}$.

For all $i \in \mathcal{I}$ we set $\mu^{(i)} := \Lambda\left((x^{(i)}_{\mathcal{S},\alpha})_{\alpha\in X_{\mathcal{S}},\mathcal{S}\in\mathscr{S}}\right)$. 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 parties as $\mathsf{ShSim}(\widetilde{y}, (\mu^{(i)})_{i\in\mathcal{I}})$.

Claim 20. *REAL$_{\mathcal{A}}$* $\equiv \mathsf{Hybrid}^{\mathsf{0Share}}$.
Proof: Since ShSim *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 21. $\mathsf{Hybrid}^{\mathsf{0Share}} \equiv \mathsf{Hybrid}^{\mathcal{F}_{\mathsf{LSS}}}$.
Proof: By correctness of Π_{LSS}, $y = \widetilde{y}$ *so the view of* Env *is unchanged.*

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

Claim 22. $\mathsf{Hybrid}^{\mathcal{F}_{\mathsf{LSS}}} \equiv \mathsf{Hybrid}^{\mathsf{0Share}}$.
Proof: Since the private decryption keys dk_h *of all honest parties* $h \in \mathcal{H}$ *are not used anymore, we have that the* IND-CPA *property of* PVSS *stated in Proposition 17 in Section 2.5 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 $\mathsf{enc\text{-}shares}_{\mathcal{S},\alpha}$ broadcast by corrupt senders $\mathcal{S}$. First, decrypt the honest shares of $\mathsf{enc\text{-}shares}_{\mathcal{S},\alpha}$ using, again, the honest secret keys $(\mathsf{dk}_h)_{h\in\mathcal{H}}$. From them, compute the opening shares $\{\widetilde{\mu^{(i)}}\}_{i\in\mathcal{I}}$ and use them to infer the corrupt shares using $\mathsf{ShInfer}^{\mathcal{H}}$.

Claim 23. $\mathsf{Hybrid}^{\mathsf{0Share}} \equiv \mathsf{Hybrid}^{\mathsf{ShInfer}}$.
Proof: The inferred shares are identical to the ones in the previous game, by the property of $\mathsf{ShInfer}^{\mathcal{H}}$.

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

2.7 Chapter Summary

In this chapter, we introduced a new Linear Secret Sharing functionality $\mathcal{F}_{\mathsf{LSS}}$ and showed how to implement it. Along the way, we detailed efficient instantiations of (n, t)-LSS schemes over polynomial rings and discussed simulatability of PVSS without straight-line extraction, allowing the use of more efficient schemes. In the following Chapters 3 and 4, we will show how to use this functionality as a foundation for building delegated MPC protocols.

Chapter 3

trBFV: a Robust RLWE scheme with application to MPC

Contents

In this chapter, we propose the first FHE-based MPC protocol based on RLWE, as introduced in Section 1.10.6, that is robust. More specifically, considering a set of n parties, of which up to t can be corrupt by an adversary $\mathcal{A}$, this protocol guarantees that honest parties output the computation result in a constant number of rounds. Adding some robustness in a protocol is typically achieved by employing a (n, t)-LSS scheme, as introduced in Definition 3 and detailed in Chapter 2, to secret-share parties secret keys, and distributing the shares. That way, even if t parties abort, the remaining $t + 1$ ones have enough information to pursue and eventually output. In what follows, we consider the BFV scheme [FV12] as an example of a RLWE-based FHE scheme, and introduce trBFV, our new robust threshold-FHE scheme built upon it. However, an analogous construction using other RLWE-based schemes such as CKKS [CKKS17] can be obtained with a straightforward adaptation.

Using secret sharing to make a RLWE-based MPC protocol more robust has been studied in many works [BGG+18; MTBH21; Par21; MBH23], following the blueprint of Asharov et al. [AJL+12]. However, none of them has succeeded in proposing a robust end-to-end protocol, due to the unique challenges that arise when combining a RLWE-based FHE scheme with MPC. Two steps are particularly challenging:

Distributed Key Generation (DKG) : DKG protocols enable a set of parties to establish, in a distributed way, a common threshold encryption key. This has been widely studied for discrete-log-based schemes [Ped91; CKLS02; KG21; Kat24], and more recently for RLWE-based schemes [MTBH21; KJY+20; Par21]. However, the latter present unique challenges, mainly due to the fact that their Eval algorithm (see Definition 6) requires some additional public key, denoted as *relinearization key* for BFV, that proves difficult to generate in a distributed and robust way. In this chapter, we propose the first robust DKG for RLWE-based FHE schemes.

Threshold Decryption $(\mathsf{ThresholdDec})$: ThresholdDec protocols enable a set of parties, provided with some common threshold encryption key ek and secret key shares $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ of the secret key sk to decrypt a ciphertext c. Existing protocols, which we review in Section 3.2.2, either require non-compact (i.e. of size at least linear in n) ciphertexts or non-compact key shares, which, in both cases, introduce a significant overhead in practice. In this chapter, we detail an efficient threshold decryption protocol with compact ciphertexts and key shares, leveraging the functionality $\mathcal{F}_{\mathsf{LSS}}$ introduced in Chapter 2.

This chapter is organized as follows: We first introduce the BFV FHE scheme and detail the main challenges that arise when building a threshold and robust version of it (Section 3.1). Then, we detail our contributions (Section 3.2) and compare them with previous related works. In Section 3.3, we present an alternative relinearization key, allowing us to describe a linear variant of BFV, denoted ℓ-BFV. In Section 3.4, we show how the latter helps us describe trBFV, our new robust threshold scheme. Then, in Section 3.5, we detail an improved threshold decryption protocol. Finally in Section 3.6, we conduct a rigorous noise analysis, before describing our MPC protocol in Section 3.7 and its security in Section 3.8.

3.1 Preliminaries

We introduce in Section 3.1.1 the BFV scheme, and the overall idea behind the creation of a threshold version from it. In Sections 3.1.2 and 3.1.3, we detail the main challenges that we

face in designing our robust MPC protocol from BFV.

Notations Recall that we denote $R_k = R/(k.R)$ and $R_q = R/(q.R)$ the residue rings of R modulo k and q. 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 $\|r\| := \max_i |\widetilde{r_i}|$. Let Ψ_q, $\mathscr{B}_{\mathsf{Enc},q}$ and $\mathscr{X}_q$ be distributions over R_q. We denote $\lceil . \rfloor, \lfloor . \rfloor, \lfloor . \rceil$ the rounding to the next, previous, and nearest integer respectively, and $[.]_k$ the reduction of an integer modulo k into R_k. When applied to polynomials or vectors, these operations are performed coefficient-wise. Let $\Delta = \lfloor q/k \rfloor$ be the integer division of q by k. We denote vectors of some length l (see Section 1.10.7) in bold, e.g. $\mathbf{r}$. 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

$$\|p \cdot h\| \leq d\|p\|\|h\|. \tag{3.1}$$

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

Unless otherwise stated, we consider the arithmetic in R_q and polynomial reductions are omitted in the notation.

3.1.1 The BFV FHE scheme [FV12].

We now describe in Figure 3.1 the Brakerski-Fan-Vercauteren [FV12] FHE scheme seen as a mere ℓ-HE scheme, following Definition 7. Departing from [FV12], we specify that the encryption key generation algorithm takes a fixed public uniform random string (URS) denoted a as input, while a is sampled locally in [FV12]. The reason is that, for our distributed key generation (DKG) to operate, some form of additivity will be required between the keys. Intuitively, this specification of a as a URS is harmless, since $t+1 = n-t$ honest encryption keys (ek_i) generated with this same a are $t+1$ instances of RLWE with the same public a, hence, are indistinguishable from $t+1$ uniform randomnesses.

BFV is based on two kinds of secrets, commonly sampled from small-normed yet different distributions: The key distribution is denoted $\mathscr{X}_q$[1]. The error distribution Ψ_q (or $\mathscr{B}_{\mathsf{Enc},q}$ for encryption) over R_q has coefficients distributed according to a centered discrete Gaussian with standard deviation σ (resp σ_{Enc}) and truncated support over $[-B, B]$ (resp B_{Enc}) where σ and B are cryptosystem parameters.

BFV is an approximate FHE scheme over the message space R_k, i.e. its ciphertexts contain an error that increases during the homomorphic operations, and its decryption algorithm outputs approximations of the messages. While noisy messages may suffice for certain applications, the majority of use cases often demand a fixed level, or even exact arithmetic. To achieve this, BFV uses plaintext encoding and decoding methods. Specifically, it consists in amplifying the plaintext by a factor Δ to switch the locations of the message and the noise in a

[1] And is typically chosen as $\mathcal{R}_3$, where coefficients are uniformly distributed in $\{-1, 0, 1\}$

BFV Scheme

Public Input: URS $a \in R_q$

BFV.Keygen(a) : Sample $e^{(\mathsf{ek})} \xleftarrow{\$} \Psi_q$ and $\mathsf{sk} \xleftarrow{\$} \mathscr{X}_q$, and define the linear map $\Lambda^a_{\mathsf{EKeyGen}}$: $(\mathsf{sk}, e^{(\mathsf{ek})}) \to (-a \cdot \mathsf{sk} + e^{(\mathsf{ek})}, a)$.

Output $\mathsf{ek} \leftarrow \Lambda^a_{\mathsf{EKeyGen}}(\mathsf{sk}, e^{(\mathsf{ek})}) = (-a \cdot \mathsf{sk} + e^{(\mathsf{ek})}, a) = (b, a)$ and sk.

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 $u \xleftarrow{\$} \mathscr{X}_q$. Define the linear map $\Lambda^{b,a}_{\mathsf{Enc}}$: $(\Delta \cdot m, u, e_0^{(\mathsf{Enc})}, e_1^{(\mathsf{Enc})}) \to \left(\Delta \cdot m + u \cdot b + e_0^{(\mathsf{Enc})},\ u \cdot a + e_1^{(\mathsf{Enc})}\right)$.

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

[In the formalism of Definition 7, the space of encryption randomness is thus $\overrightarrow{\mathscr{B}_{Enc}} = \mathscr{B}_{\mathsf{Enc},q} \times \Psi_q \times \mathscr{X}_q$.]

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

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

Figure 3.1: The BFV scheme

ciphertext, and employing rescaling-and-rounding techniques during decoding. Then, for encryption randomnesses $e_0^{(\mathsf{Enc})} \xleftarrow{\$} \mathscr{B}_{\mathsf{Enc},q}$, $e_1^{(\mathsf{Enc})} \xleftarrow{\$} \Psi_q$, $u \xleftarrow{\$} \mathscr{X}_q$ and an encryption key $\mathsf{ek} = (b, a)$, the encryption of a message $m \in R_k$ can be seen as:

$$\mathsf{c} = (\Delta \cdot m + u \cdot b + e_0^{(\mathsf{Enc})}, u \cdot a + e_1^{(\mathsf{Enc})}) \tag{3.2}$$

Following this, the decryption of a ciphertext $\mathsf{c} = (\mathsf{c}[0], \mathsf{c}[1])$ can be seen as a two-step process. The first step requires the secret key to compute a noisy plaintext in R_q as:

$$\mathsf{c}[0] + \mathsf{sk} \cdot \mathsf{c}[1] = \Delta \cdot m + e_{\mathsf{c}}, \tag{3.3}$$

where e_{c} is the ciphertext overall error, or ciphertext noise (to be later specified in Section 3.6).

In the second step, the message is decoded from the noisy term in R_q to a plaintext in R_k, by rescaling-and-rounding:

$$\left[\lfloor \frac{k}{q}(\Delta \cdot m + e_{\mathsf{c}}) \rceil\right]_k = [\lfloor m + r \cdot k + v \rceil]_k \tag{3.4}$$

where v has coefficients in $\mathbb{Q}$. Thus, the decryption outputs m when $\|v\| < \frac{1}{2}$.

Hence, the correctness of the scheme depends on the ciphertext noise e_{c}, that must be kept below $\frac{\Delta}{2}$ throughout the computation.

Remark. *Note that one of our goals is to minimize the increase in the ciphertext noise (as introduced in Equation* (3.3)*) for the correctness to hold, without having an explosion of parameter values. Some operations, and notably the relinearization (see Algorithm 1), require multiplying a ciphertext with large R_q elements. To reduce the noise blowup, we often rely on a prior decomposition of R_q elements into an auxiliary basis to reduce their norm. Specifically, a common choice is to consider a decomposition basis w of dimension $l = \lceil \log_w(q) \rceil$, i.e $\mathbf{w} = (w_0, w_1, \ldots, w_{l-1})^T$. Then, for any element $z \in R_q$, we have $z = \sum_{i=0}^{l-1} w_i \cdot z_i$, where z_i denotes the i-th coefficient of z in the decomposition basis.*

Remark. *For better efficiency, it is common to use* packing *techniques; namely to FFT-like transform the plaintext polynomials in order to enable coefficient-wise encrypted arithmetic. Intuitively, this enables encoding up to d messages into d independent ciphertext slots, where d is the polynomial degree. We refer to [SV12] for details.*

Security. For later use in the security proof of our MPC protocol in Section 3.8, we require that considering an encryption key sampled uniformly at random[2], the ciphertext produced by BFV.Enc is pseudorandom under the RLWE assumption.

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

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 $\beta' = \beta$.

Figure 3.2: Pseudorandomness of BFV ciphertexts with uniformly generated encryption keys

Lemma 24. *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 later presented in Section 3.7.1, i.e. such that Assumption 42 holds and that satisfies Equation* (3.23)*. 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 $(a \xleftarrow{\$} U(R_q), e_1^{(\mathsf{Enc})} \xleftarrow{\$} \Psi_q)$.

[2]Note that this slightly departs from the classical notion, to be used in our MPC proof where keys are replaced by uniformly random values in R_q.

Thus, by RLWE for $(\mathscr{X}_q, \Psi_q)$ introduced in Section 1.10.6, and thus a fortiori for $(\mathscr{X}_q, \mathscr{B}_{\mathsf{Enc},q})$ (Equation 3.23), the two RLWE samples are indistinguishable from samples in $U(R_q^2)$. □

The multiparty Case: Distributed key Generation. We are interested in a distributed setting, where several parties participate in the algorithms introduced in Figure 3.1, and in particular, in the key generation and decryption. Overall, the main idea is to let all n parties perform a distributed key generation (DKG) protocol to establish a common threshold encryption key, and to use the linearity of the decryption in a ℓ-HE scheme to design a distributed version of it.

To facilitate understanding, we will describe in the first place the easiest case where the keys are additively shared. We will later explain how to adapt this to the case where a (n, t)-LSS scheme (see Definition 3) is used instead. First, let us denote by sk_i the secret key locally sampled by party P_i. Then, using an additive structure, we define the common secret key sk as:

$$\mathsf{sk} = \sum_{i \in S} \mathsf{sk}_i, \tag{3.5}$$

where $S \subseteq [n]$ is defined as the set of indices of non-aborting parties in the DKG.

Π_{EKeyGen}

Private Input for P_i: sk_i (secret key share)
Public Input $a \in R_q$ (uniform random string)
Public Output: ek (common threshold encryption key)

Round (1) Each party P_i:

1. Samples $e_i^{(\mathsf{ek})} \xleftarrow{\$} \Psi_q$;
2. Outputs $\mathsf{ek}_i = (-a \cdot \mathsf{sk}_i + e_i^{(\mathsf{ek})})$.

Output : $\mathsf{ek} = (\sum_{i \in S} \mathsf{ek}_i, a)$, where $S \subseteq [n]$ denotes the indices of parties that then a contribution in the first round.

Figure 3.3: Protocol for common threshold encryption key generation

Now, we would like to distributively generate a common encryption key ek from sk. In order to emulate the BFV.Keygen algorithm, we design in Figure 3.3 a protocol Π_{EKeyGen}. This requires some added additivity, which we provide in the form of a common *uniform random string* a uniformly sampled in R_q and agreed upon by all the parties. Thus, at the end of the execution of Π_{EKeyGen}, parties have access to:

$$\mathsf{ek} = \sum_{i \in S} \Lambda^a_{\mathsf{EKeyGen}}(\mathsf{sk}_i, e_i^{(\mathsf{ek})})^3 = \Lambda^a_{\mathsf{EKeyGen}}(\sum_{i \in S}(\mathsf{sk}_i, e_i^{(\mathsf{ek})})). \tag{3.6}$$

[3]Note that the URS a is left outside of the sum.

i.e. a BFV encryption key for sk following Figure 3.1. Note that S denotes again the set of indices of non-aborting parties in the DKG.

Remark. *Due to its distributed generation, the encryption key noise has a larger magnitude that grows linearly in n. However, that does not incur serious practical obstacles, even for large n.*

3.1.2 Challenge 1: Robust Relinearization Key Generation.

So far, we have described how a set of parties is able to distributively generate a common threshold encryption key. However, as introduced in Definition 6, an additional key, denoted as *relinearization key* in the specific case of BFV, is needed to be able to perform homomorphic operations. For a secret key sk, it is described in [FV12] as being of the form

$$\mathbf{rlk} = (\mathsf{sk}^2\mathbf{w} - \mathsf{sk}\cdot\mathbf{r} + \mathbf{e}^{(\mathbf{rlk})}, \mathbf{r}) \tag{3.7}$$

where $w < q$ is a decomposition basis of dimension $l = \lceil \log_w(q) \rceil$, i.e. $\mathbf{w} = (w^0, w^1, \ldots, w^{l-1})^T$, $\mathbf{r} \in R_q^l$ a uniform random string and $\mathbf{e}^{(\mathbf{rlk})}$ some noise.

Generating this key in a distributed way proves to be more complex than in the case of the 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 Π_{RlkGen} protocols [KJY+20; Par21; MTBH21] have been proposed. Overall, they have the following informal structure:

- In round (1): each party P_i generates a contribution $\mathbf{rlk}_{0,i}$ using its key sk_i.
- In round (2): each party 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 parties in the first round. Then each P_i uses an algorithm RelinKeyGen to compute a final contribution $\mathbf{rlk}_i \leftarrow \mathsf{RelinKeyGen}(\mathbf{rlk}_0, \mathsf{sk}_i)$ and broadcasts $\mathbf{rlk}_i$.

At the end of this protocol, 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 parties in this second round.

Main issue: Robustness. We observe that this generic protocol, illustrated in Figure 3.4, has a major drawback, in that it is not robust; if some parties take part in some of the rounds, but not all of them, then no $\mathbf{rlk}$ is generated. Specifically, [KJY+20; Par21; MTBH21] required S and S' to be equal for Π_{RlkGen} to output. Otherwise, if the generation were done with non-equal sets S and S', then the resulting $\mathbf{rlk}$ would be incompatible with the ek produced in the first round as $\mathsf{ek} = \sum_{i\in S} \Lambda^a_{\mathsf{EKeyGen}}(\mathsf{sk}_i, e_i^{(\mathsf{ek})})$, making the distributed key generation non-robust. In Section 3.2.1, we overcome this issue by detailing an algorithm ℓ-RlkKeygen for generating an alternative relinearization key, that allows, as shown in Figure 3.4, to design a RlkGen protocol to generate $\mathbf{rlk}$ in only one round that operates in parallel with the DKG, and to obtain a robust overall key generation.

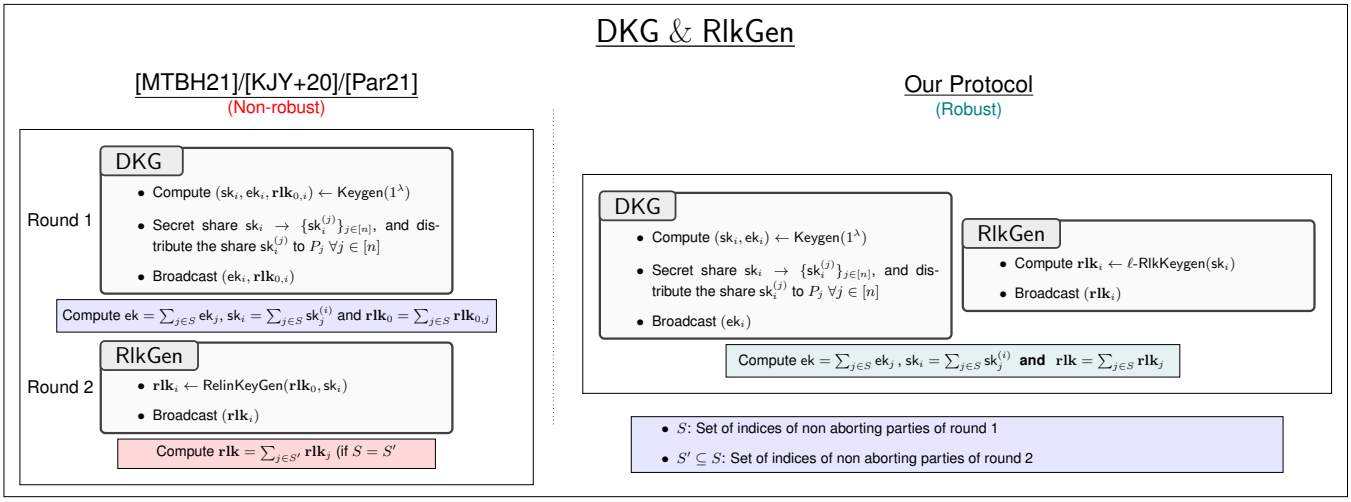


Figure 3.4: We present on the left-hand side the overall construction of previous protocols [KJY+20; Par21; MTBH21]. First, each party P_i runs Keygen to produce keys $(\mathsf{sk}_i, \mathsf{ek}_i, \mathbf{rlk}_{0,i})$. The secret key sk_i is secret-shared in n shares $\{\mathsf{sk}_i^{(j)}\}_{j=1}^n$, 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 parties to form the common threshold encryption and intermediate relinearization keys ek and $\mathbf{rlk}_0$, as well as a key share sk_i. Then, parties run $\mathsf{RelinKeyGen}$ 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 parties of this second round, parties can compute the relinearization key $\mathbf{rlk} = \sum_{j\in S'} \mathbf{rlk}_j$ if $S = S'$. On the right-hand side, we present a sketch of our protocol. More specifically, to have robustness, parties run in parallel Keygen and our relinearization key generation algorithm $\ell\text{-}\mathsf{RlkKeygen}$.

3.1.3 Challenge 2: Threshold Decryption.

Finally, the last challenge consists of figuring out how to distributively decrypt a ciphertext c encrypted under a common threshold encryption key ek.

A naive approach would be to let the set of parties follow a protocol that would emulate $\mathsf{BFV.Dec}$. Recall from Figure 3.1, that the latter algorithm consists of applying a linear map $\Lambda_{\mathsf{Dec}}^{\mathsf{c}}$ in the secret key, followed by the computation of a non-linear decoding function Ω_{Dec}. Now, let us consider a setting where a common secret key sk has been generated and each party P_i owns an additive share sk_i of it. Then, to decrypt c, each P_i would compute $\mu_{\mathrm{dec},i} \leftarrow \Lambda_{\mathsf{Dec}}^{\mathsf{c}}(\mathsf{sk}_i)$ and broadcast it. Then, after summing all contributions together, it would output $m \leftarrow \Omega_{\mathsf{Dec}}(\sum_{i\in S} \mu_{\mathrm{dec},i})$. Unfortunately, this simple approach does not work straight away, as each contribution $\mu_{\mathrm{dec},i}$ leaks some information about sk_i (see also, e.g., [BGG+18, §2.1]), which ruins the security of the scheme.

To circumvent this issue, Asharov et al. [AJL+12] introduced the technique of adding additional noise to the decryption output before it can be opened. This *smudging* noise e_{sm} is, roughly, sampled uniformly in some large enough interval $[-B_{sm}, B_{sm}]$[4]. All in all, the threshold decryption protocol for some ciphertext c followed by most previous approaches [BGG+18; MTBH21; KJY+20; Par21] can be summed up informally as:

[4]More details will be provided in Section 3.5

(1) Each party P_i used its secret key share sk_i to compute its *decryption share* of c: $\mu_{\mathrm{dec},i} \leftarrow \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk}_i)$. Then, it locally sampled some "smudging noise" $e_{\mathrm{sm},i}$ and broadcast a noisy $\mu_i = \mu_{\mathrm{dec},i} + e_{\mathrm{sm},i}$.

(2) Each party used the received noisy decryption shares to reconstruct the output as $m \leftarrow \Omega_{\mathsf{Dec}}(\sum_{i \in S} \mu_i)$.

Main issue: Robust and Efficient Threshold Decryption. A major difficulty arises when trying to make this threshold decryption protocol robust. To do this, the basic idea is to share the key sk not using an additive sharing, but a (n, t)-LSS scheme as defined in Definition 3 to add some redundancy. In doing so, even if the corrupt parties abort, the remaining $t + 1$ honest ones have enough material to reconstruct the output. Details about the transition to an (n, t)-LSS scheme will be done in Section 3.5. However, this naively leads to a significant drain on efficiency.

Following the discussion held in Section 2.4, we consider two alternatives for instantiating the (n, t)-LSS scheme, which introduce their own challenges when parties use their secret key shares during the threshold decryption.

- When instantiated with Shamir (see Section 1.10.1), the added smudging noises are multiplied by Lagrange coefficients during the reconstruction. As explained in [BGG+18], this led to the use of a large $n!^2$ scaling factor, in order to later clear-out the denominators of the Lagrange coefficients. Overall, this imposed the bit-size of the ciphertext modulus q to be $(n \log n)$, which resulted in a $n\times$ blowup of the ciphertext length, as observed in [GLS15].
- When using a $\{0, 1\}$-(n, t)-LSS scheme as defined in Definition 10 in Section 2.4.1, this allowed to remove the extra n in the modulus bit-size, i.e. $\log\, q = O(\log n)$, as instead of the Lagrange coefficients used in Shamir's scheme, it employs binary coefficients to recover the output from the noisy decryption shares. However, this also led to a significant space overhead, as the size of each secret key share is at least $O(n^{4.2})$.

In Section 3.2.2, we overcome these issues by leveraging the $\mathcal{F}_{\mathsf{LSS}}$ functionality to describe an alternative approach for threshold decryption which enables a $(n!)^3\times$ smaller total smudging noise, while keeping regular key share size.

3.2 Our Contributions

In this chapter, we present trBFV, the first robust threshold BFV scheme, and propose an MPC protocol as informally stated in Theorem 25 below.

Theorem 25. *(Informal) Consider $n = 2t + 1$ parties, 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 follows from our main contribution detailed in the following Section 3.2.1.

3.2.1 Main Contribution: A Robust Distributed Key Generation for RLWE-based FHE.

We propose an alternative relinearization key $\mathbf{rlk}$ for BFV, enabling the construction of the first robust key generation protocol for a RLWE-based FHE scheme. The key innovation lies in the newly proposed ℓ-RlkKeygen algorithm, which, unlike all previous approaches, applies a *linear map* to the secret key sk, rather than a squaring operation. This helps defining a linear variant of BFV, called ℓ-BFV, which underpins the RlkGen protocol described in Figure 3.4. Contrary to previous protocols, the generation of $\mathbf{rlk}$ is done *in parallel* with the generation of the common threshold encryption key, and is guaranteed to terminate under an honest majority. This is summarized in Table 3.1.

Related Works. Recent protocols [KJY+20; Par21; MTBH21] have been proposed for the distributed generation of $\mathbf{rlk}$, however, none of them are robust. The main challenge is that the generation of $\mathbf{rlk}$ requires the squaring of sk; let us analyze how these previous works address it.

Kim et al. [KJY+20] proposed a protocol based on additive secret sharing, where each party P_i first uses its key sk_i to encrypt sk_i and broadcasts this intermediate contribution dubbed as $\mathbf{rlk}_{0,i}$. Then, all received contributions are added together over S (the set of indices of non-aborting parties) to form an intermediate key $\mathbf{rlk}_0$, i.e. an encryption of $\mathsf{sk} = \sum_{i \in S} \mathsf{sk}_i$ under sk. Each party P_i runs an algorithm RelinKeyGen using once again its key sk_i and $\mathbf{rlk}_0$, that computes an encryption of $\mathsf{sk}_i \cdot \mathsf{sk}$ under sk by multiplying $\mathsf{sk}_i \cdot \mathbf{rlk}_0$. P_i samples a large smudging noise e_i and broadcasts $\mathbf{rlk}_i := \mathsf{sk}_i \cdot \mathbf{rlk}_0 + e_i$. Finally, the latter contributions are added together over the set S' of indices of non-aborting parties in this second round. Hence, if S and S' are not equal, the $\mathbf{rlk}$ produced becomes incompatible with the ek produced in the first round. Park [Par21] proposed a concurrent protocol to [KJY+20], where party P_i essentially uses for the second round, its secret key sk_i along with the common threshold encryption key ek generated over the set S in the first round (instead of $\mathbf{rlk}_0$), in order to generate its contribution $\mathbf{rlk}_i$ to $\mathbf{rlk}$. However, as before, if the same set of participants does not take part in both rounds, then the generated key is not correct, making it a non-robust protocol.

Finally, Mouchet et al. [MBH23] presented a framework to use the n-out-of-n relinearization key generation introduced in [MTBH21] in a $(t+1)$-out-of-n threshold manner. However, to make it efficient, the authors assumed that parties are able to determine (or optimistically guess) a set of at least $t+1$ online parties, in order to first convert Shamir shares into additive ones before performing the relinearization key generation. This caused the protocol to stale if one of the selected parties did not provide its share. Recently, [MCPT24] proposed a retry mechanism that addressed the latter challenge posed in [MBH23], at the cost, however, of a non-constant number of rounds, contrary to what we expect for our robustness requirement.

	# broadcast for DKG & RlkGen	Robustness
[KJY+20]	2	✗
[Par21]	2	✗
[MTBH21]	2	✗
trBFV	1	✓

Table 3.1: DKG and RlkGen properties for various Threshold-FHE schemes.

3.2.2 Leveraging $\mathcal{F}_{\mathsf{LSS}}$ for an alternative threshold decryption with smaller ciphertext.

For decryption, most previous works [BGG+18; KJY+20; Par21] followed the *mainstream approach* derived from [AJL+12] to decrypt a ciphertext c, each party 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 party used $t+1$ of the received noisy decryption shares to reconstruct the output.

As explained in Section 3.1.3, this leads to a significant overhead, either in the ciphertexts or in the size of the key shares.

In this chapter, we leverage the functionality $\mathcal{F}_{\mathsf{LSS}}$ introduced in Chapter 2 to easily describe an alternative threshold decryption protocol, that enables a $n!^3\times$ smaller total smudging noise, while keeping regular key shares size. We first observe, as made explicit in Figure 3.1, that the first step of the BFV decryption algorithm consists in the computation of a linear map $\Lambda^{\mathsf{c}}_{\mathsf{Dec}}$, to which must be added a smudging noise e_{sm}. The latter operation being linear, we can define $\Lambda^{\mathsf{c}}_{\mathsf{Dec}+sm}$, the linear map $(\mathsf{sk}, e_{\mathrm{sm}}) \to \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk}) + e_{\mathrm{sm}}$. Thus, provided with some *pre-generate common secret-shared smudging noises*[5], parties can leverage the $\mathcal{F}_{\mathsf{LSS}}$ functionality to perform all-at-once the opening of the evaluation of the linear map of decryption, added with one secret-shared smudging noise. This ingredient, which allows to remove the $n!^2$ scaling factor, has been introduced by [GLS15], but was never later used to our knowledge.

Moreover, as a bonus of using $\mathcal{F}_{\mathsf{LSS}}$, we can use a modulus q of small size 2^e, we use Shamir sharing over $\mathbb{Z}/2^e\mathbb{Z}$, i.e., embed polynomials into Galois rings extensions [Feh98; ACD+19], thereby allowing efficient implementations [CH18; GIKV23]. Notice that alone, this last ingredient would *not* have been applicable. Indeed, without the alternative threshold decryption, i.e., with the mainstream approach, then it would have been required that q has no factor in common with $n!$.

Related Works with smaller noise but incompatible with MPC. Some recent works [CSS+22; BS23] addressed an orthogonal size dependency, by replacing the statistical distance used to analyze the noise during the threshold decryption by the Rényi divergence. In more detail, the threshold decryption of a ciphertext c allows recovering the message, but also reveals a small

[5] One for every subsequent opening to be done

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

$^{(a)}$ Presented in Section 3.5.2.

Table 3.2: Threshold FHE schemes for n parties, using lattice of dimension d and modulus q. The last column indicates the size of the shares owned by a party, and the modulus-to-noise ratio refers to the ratio between the modulus and decryption noise of the ciphertext of the output.

decryption noise term (see Definition 29 in Section 3.6) 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 3.2, previous works [BGG+18; CCK23] required the ratio between this smudging noise, hence, the modulus, and the size of the decryption noise of the ciphertext to be superpolynomial in the security parameter. This, in turn, required the RLWE problem to be secure with a superpolynomial modulus-to-noise ratio, which required larger RLWE parameters. Recently, [CSS+22; BS23] proposed threshold FHE schemes with a polynomial modulus-to-noise ratio.

However, the latter schemes do not come without their own drawbacks. First, due to the Rényi divergence technique, both [CSS+22] and [BS23] require a predetermined bound l on the number of decryption queries, causing the modulus to scale with $\sqrt{l}$. Conversely, in our scheme, the modulus does not need to increase with the number of decryption queries. More importantly, [BS23] made 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 usable as a black box in complex protocols.

3.3 ℓ-BFV, with Linear Relinearization Key Generation

We now present ℓ-BFV, the single-key FHE scheme which will later be used in Section 3.4 as the baseline to build our robust threshold FHE scheme. ℓ-BFV is a particularization of the n-out-of-n multikey scheme of [CDKS19] to the specific single-key case (i.e. for $n = 1$), and can also be seen as a variant of BFV [FV12] with a linear relinearization key generation. In this section, we first describe our apparently new relinearization key generation in Section 3.3.1, before explicitly detailing ℓ-BFV in Section 3.3.2. Finally in Section 3.3.3, we explain how to perform homomorphic operations with this alternative key.

3.3.1 Relinearization Key Generation.

Let us recall that in order to perform homomorphic operations, an extra relinearization key denoted $\mathbf{rlk}$ is needed. The homomorphic multiplication of two BFV ciphertexts $\mathsf{c}_1, \mathsf{c}_2 \in R_q^2$

ℓ-RlkKeygen

Public Inputs: URS $\mathbf{a} \in R_q^l$ and $\mathbf{d}_1 \in R_q^l$.

ℓ-RlkKeygen$(\mathbf{a}, \mathbf{d}_1, \mathsf{sk})$: Given $\mathsf{sk} \in R_q$ and $(\mathbf{a}, \mathbf{d}_1) \in R_q^{2\times l}$:

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

Figure 3.5: ℓ-RlkKeygen Relinearization Key Generation

involves two steps:

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

$$\hat{\mathsf{c}} = \left\lfloor \frac{k}{q} \mathsf{c}_1 \otimes \mathsf{c}_2 \right\rceil = (\hat{\mathsf{c}}[0], \hat{\mathsf{c}}[1], \hat{\mathsf{c}}[2]) \in R_q^3.^{6} \tag{3.8}$$

(b) To reduce the degree back to one, a second step, denoted *relinearization*, must be carried out using $\mathbf{rlk}$ 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.

Remember from Section 3.1.2 that the relinearization key $\mathbf{rlk}$ used in previous works [KJY+20; Par21; MTBH21] was quadratic in the secret key. We now present an alternative key that is only linear in the secret key.

Relinearization Key Generation. Our relinearization algorithm heavily leverages the gadget toolkit introduced in Section 1.10.7. 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 3.1.1, used to generate encryption keys. We can now define in Figure 3.5 the algorithm ℓ-RlkKeygen to generate a relinearization key. Interestingly, the overall algorithm to generate $\mathbf{rlk}$ is linear over the secret key sk, unlike previous ones [KJY+20; Par21; MTBH21].

Construction Intuition. The intuitive rationale behind this algorithm 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 parties is of the form $\mathbf{c} = (\mathsf{c}_1, \mathsf{c}_2, \ldots, \mathsf{c}_n)$, is decryptable by the

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

ℓ-BFV Scheme

Public Inputs: URSs: $\mathbf{a} \in R_q^l, \mathbf{d}_1 \in R_q^l$

ℓ-BFV.Keygen($\mathbf{a} \in R_q^l$) : Sample $\mathbf{e}^{(\mathsf{pk})} \xleftarrow{\$} \Psi_q^l$ and $\mathsf{sk} \xleftarrow{\$} \mathscr{X}_q$. Define $\Lambda^{a}_{\mathsf{EKeyGen}} : (\mathsf{sk}, \mathbf{e}^{(\mathsf{pk})}) \to (-\mathbf{a} \cdot \mathsf{sk} + \mathbf{e}^{(\mathsf{pk})}, \mathbf{a}) = (\mathbf{b}, \mathbf{a})$.

Output $\mathbf{ek} \leftarrow \Lambda^{\mathbf{a}}_{\mathsf{EKeyGen}}(\mathsf{sk}, \mathbf{e}^{(\mathsf{pk})}) = (-\mathbf{a} \cdot \mathsf{sk} + \mathbf{e}^{(\mathsf{pk})}, \mathbf{a}) = (\mathbf{b}, \mathbf{a})$ and sk.

ℓ-BFV.RelinKeyGen($\mathbf{a} \in R_q^l, \mathbf{d}_1 \in R_q^l, \mathsf{sk}$) : Parse the URSs as $(\mathbf{a} \in R_q^l, \mathbf{d}_1 \in R_q^l)$ and sample $r \leftarrow \mathscr{X}_q$, $\mathbf{e}_0^{(\mathbf{rlk})}, \mathbf{e}_2^{(\mathbf{rlk})} \leftarrow \Psi_q^l$, and define $\Lambda^{\mathbf{a},\mathbf{d}_1,\mathbf{g}}_{\mathsf{RlkGen}} : (\mathsf{sk}, r, \mathbf{e}_0^{(\mathbf{rlk})}, \mathbf{e}_2^{(\mathbf{rlk})}) \to (-\mathsf{sk} \cdot \mathbf{d}_1 + \mathbf{e}_0^{(\mathbf{rlk})} + r \cdot \mathbf{g}, \mathbf{d}_1, r \cdot \mathbf{a} + \mathbf{e}_2^{(\mathbf{rlk})} + \mathsf{sk} \cdot \mathbf{g})$.

Output the relinearization key $\mathbf{rlk} \leftarrow \Lambda^{\mathbf{a},\mathbf{d}_1,\mathbf{g}}_{\mathsf{RlkGen}}(\mathsf{sk}, r, \mathbf{e}_0^{(\mathbf{rlk})}, \mathbf{e}_2^{(\mathbf{rlk})})$.

ℓ-BFV.Enc($\mathbf{ek} = (\mathbf{b}[0] = b, \mathbf{a}[0] = a), m \in R_k$) : Sample the encryption randomnesses $e_0^{(\mathsf{Enc})} \xleftarrow{\$} \mathscr{B}_{\mathsf{Enc},q}$, $e_1^{(\mathsf{Enc})} \xleftarrow{\$} \Psi_q$, and $u \xleftarrow{\$} \mathscr{X}_q$.

Define $\Lambda^{b,a}_{\mathsf{Enc}} : (\Delta m, u, e_0^{(\mathsf{Enc})}, e_1^{(\mathsf{Enc})}) \to (\Delta m + u \cdot b + e_0^{(\mathsf{Enc})}, u \cdot a + e_1^{(\mathsf{Enc})})$.

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

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

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

Figure 3.6: ℓ-BFV Scheme

concatenated secret keys $\mathbf{s} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, and is 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 Figure 3.5!

Further construction details will be presented in Section 3.4.3.

3.3.2 ℓ-BFV

We now describe ℓ-BFV in Figure 3.6, which features a linear relinearization key generation. To emphasize its linearity, we define it using *linear maps*. Specifically, we introduce $\Lambda_{\mathsf{EKeyGen}}$ and $\Lambda_{\mathsf{RlkGen}}$ to highlight that both encryption and relinearization key generation are linear in the secret key and some randomnesses, Λ_{Enc} to indicate that encryption is linear in a plaintext and some randomnesses, and Λ_{Dec} to show that decryption is, roughly, linear in the secret key.

3.3.3 Homomorphic Evaluation of a circuit

We can now augment, in Figure 3.7, the definition presented in Section 3.3.2 with homomorphic operations on ℓ-BFV ciphertexts using keys $\mathbf{ek} = (\mathbf{b}, \mathbf{a})$ and $\mathbf{rlk}$.

Relinearization. We now present our relinearization algorithm adapted from [CDKS19].

ℓ-BFV Scheme (Continuation of Figure 3.6)

ℓ-BFV.Add$(\mathsf{c}_1, \mathsf{c}_2)$: Given two ciphertexts c_1 and c_2, output $\mathsf{c} = \mathsf{c}_1 + c_2 \in R_q^2$.

ℓ-BFV.Mult$(\mathsf{c}_1, \mathsf{c}_2, \mathbf{rlk}, \mathbf{b})$: Given two ciphertexts c_1 and c_2, and keys $\mathbf{rlk}$ and $\mathbf{b}$, compute $\hat{\mathsf{c}} = \left\lfloor \frac{k}{q} \mathsf{c}_1 \otimes \mathsf{c}_2 \right\rceil \in R_q^3$.
Output $\mathsf{c}' \leftarrow \mathsf{Relin}(\hat{\mathsf{c}}, \mathbf{rlk}, \mathbf{b})$ (cf Algorithm 1).

ℓ-BFV.Eval$(\mathsf{C}, (\mathsf{c}_i \in R_q^2)_{i\in[n]}, \mathbf{rlk}, \mathbf{b})$, for an arithmetic circuit C with n input gates, output the evaluation c' obtained by applying ℓ-BFV.Add and ℓ-BFV.Mult gate by gate, with inputs the $(\mathsf{c}_i)_{i\in[n]}$.

Figure 3.7: Homomorphic Properties of ℓ-BFV

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:

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

and thus,

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

A full analysis is given in Section 3.6.3.

3.3.4 Bootstrapping

To evaluate deeper circuits, one can use a "bootstrapping" algorithm that homomorphically brings back the size of the decryption noise of a ℓ-BFV ciphertext (see Definition 29), roughly to that of a fresh ciphertext. As for the relinearization, we particularize the bootstrapping of a multikey construction, namely the one of [CDKS19, §5], to our single-key setting. The elementary homomorphic operations in bootstrapping require an auxiliary *bootstrapping key*, that we discuss below.

Bootstrapping Pipeline In short, [CDKS19] follow the algorithm improved by [CH18], which consists of four steps, denoted as (1) Modulus raise, (2) Linear Transformation, (3) Extraction and (4) the Inverse Linear Transformation.

ℓ-BkKG

Public Inputs: URS $\mathbf{h}_1 \in R_q^l$.

ℓ-BkKG$(\mathbf{h}_1, \mathsf{sk}, j)$: Given $\mathsf{sk} \in R_q$ and $\mathbf{h}_1 \in R_q^l$, sample $\mathbf{e}^{(\mathbf{bk})} \leftarrow \Psi_q^l$ and output $\mathbf{bk}(j) := \big(\mathbf{h}_1,\ \mathbf{h}_0(j) = -\mathsf{sk} \cdot \mathbf{h}_1 + \mathbf{e}^{(\mathbf{bk})} + \tau_j(\mathsf{sk}) \cdot \mathbf{g}\big)$.

Figure 3.8: ℓ-BkKG bootstrapping Key Generation

Technically, the linear transformation requires the homomorphic evaluation of the rotation of plaintext slots. In addition [CDKS19] also requires the homomorphic evaluation of "Galois elements slot-by-slot". In [GHS12, p23] it is explained how these two evaluations, i.e., operations on plaintexts, can be decomposed into additions, scalar multiplications and applications of automorphisms of $R_k = \mathbb{Z}_k[X]/(X^d + 1)$, denoted $\{\tau_j,\ j \in (\mathbb{Z}/2d)^*\}$, each being defined by: $X \to X^j$. In [CDKS19], it is observed that homomorphic evaluation of each τ_j can be realized with the auxiliary key, which we denote as $\big(\mathbf{h}_0(j),\ \mathbf{h}_1\big)$ whose technical purpose is "key-switching". This *bootstrapping key* consists of a collection of keys, indexed by $j \in (\mathbb{Z}/2d)^*$, constructed as shown in Figure 3.8 for ℓ-BFV.

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

Algorithm 2 Adaptation of Key Switching: KeySwitch

Input $\mathsf{c} = (\mathsf{c}[0], \mathsf{c}[1]) \in R_q^2, \mathbf{bk} = [\mathbf{h}_0 | \mathbf{h}_1] \in (R_q^l)^2$
Output $\mathsf{c}' = (\mathsf{c}'[0], \mathsf{c}'[1]) \in R_q^2$

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

Remark. *To add bootstrapping in the UC simulation, we need to extend Assumption 42 described in Appendix B to account for these new keys. This is done implicitly in [CDKS19]. Based on the expanded assumption, we derive an analogue of Corollary 28, w.r.t.* $\mathbf{h}_1 \leftarrow U(R_q^l)$.

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

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

as desired.

3.4 Thresholdizing ℓ-BFV into trBFV

In this section, we outline how the ℓ-BFV scheme presented in Section 3.3.2 is transformed into a new robust threshold scheme, which we refer to as trBFV. The latter scheme will be presented as an end-to-end MPC protocol in Section 3.7.

As previously discussed in Section 3.1.2, existing protocols for distributed generation of a relinearization key were not robust. Therefore, after describing our DKG protocol in Section 3.4.1, we detail in Section 3.4.2 the robust distributed generation of the relinearization key introduced in Section 3.3.1. Finally, we justify in Section 3.4.3 simulatability of our distributed relinearization key generation.

3.4.1 Biased DKG, and its no-impact on the MPC security of our scheme

Let us remind from Section 3.1.1, that in order to emulate the ℓ-BFV.Keygen algorithm, we follow the classical pattern of previous DKGs in one broadcast [FS01; Gro21; BDO23] and leverage the linearity in the encryption key generation algorithm. For later use in our MPC protocol, we wrap this linearity through the use of the functionality $\mathcal{F}_{\mathsf{LSS}}$ introduced in Chapter 2. Therefore, we go one step further from Figure 3.3, and present the generation of a common ℓ-BFV encryption key in the $(\mathcal{F}_{\mathsf{LSS}}, \overline{\mathcal{G}}_{\mathsf{URS}})$-hybrid model in Figure 3.9.

Concretely, the first step is an all-to-all broadcast, as follows. Provided with a fixed public uniform random string (URS) denoted $\mathbf{a}$ as input, each party P_i generates a key pair $(\mathbf{ek}_i = (\mathbf{b}_i, \mathbf{a}), \mathsf{sk}_i)$, sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSS}}$ and broadcasts $\mathbf{b}_i$. In the second step, parties set a threshold encryption key pair without any interaction, as follows. Denote S the set of indices of non-aborting parties, i.e., the ones that have broadcast a contribution $\mathbf{b}_i$ and sent an input to $\mathcal{F}_{\mathsf{LSS}}$. Then, the common threshold encryption key is defined as the sum of the contributions.

Each party P_j owns a share of sk, consisting in the sum over $i \in S$ of its shares $\mathsf{sk}_i^{(j)}$ of the sk_is. In our formalism, each contribution is accessible via $\mathcal{F}_{\mathsf{LSS}}$ through a label $\overline{\mathsf{sk}_i}$. Thus, the sum of contributions: sk is also accessible via $\mathcal{F}_{\mathsf{LSS}}$, via a label which we denote $\overline{\mathsf{sk}}$. This will later prove useful for threshold decryption in Section 3.5.

Bias in DKG***.*** The ℓ-BFV.Keygen algorithm requires that the secret key sk is sampled from $\mathscr{X}_q$, which induces an almost-uniform distribution of the encryption key $\mathbf{ek} = (\mathbf{b}, \mathbf{a})$ in $R_q^{2\times l}$. Hence, ideally, we would like to have a DKG that outputs an encryption key of this form. However, this is not the case since the adversary first sees the broadcast contributions $\mathbf{b}_i$ of honest parties, before it decides on the contributions of corrupt parties. So the adversary has the power to set to almost any value the final threshold encryption key $\mathbf{ek} = (\mathbf{b} = \sum_{i\in S} \mathbf{b}_i, \mathbf{a})$.

More formally, the key pair obtained as output of the Π_{DKG} protocol presented in Figure 3.9, can be seen as generated by using what [BDO23, Fig. 2] formalized as the BiasKeyGen subroutine. Although the latter is formalized in the discrete-log setting, the formalism in our case would be unchanged thanks to the additivity of keys (modulo noises). A long line of works [GJKR07; GJM+21] studied how the security of single-key primitives is modified when the generation of the key is biased by the adversary. Let us now provide the intuition of the two main properties of our DKG, that our proof will use:

Protocol Π_{DKG}

Participants: n parties $P_1, \ldots, P_n$

Bulletin-board PKI setup and CRS setup. Each P_i:

- Sends (Setup) to $\mathcal{F}_{\mathsf{LSS}}$;
- Obtains common uniform strings $(\mathbf{a}) \leftarrow \overline{\mathcal{G}}_{\mathsf{URS}}$.

Round (1): Upon ready from $\mathcal{F}_{\mathsf{LSS}}$, each P_i:

- *Distributed Keys Generation:*
  - Computes $(\mathsf{sk}_i, (\mathbf{b}_i, \mathbf{a})) \leftarrow \ell\text{-BFV.Keygen}(\mathbf{a})$.
  - Sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSS}}$ and broadcasts $(\mathbf{b}_i)$.

Local Computation: Each party:

- *Reception of broadcasts:* Initializes an empty list $S \leftarrow \{\}$ of indices of non-aborting parties. $\forall j \in [n]$, checks if the data received from the broadcast of P_j parses as: $(\mathbf{b}_j)$ then stores this data and adds j to S.
- *Adding the contributions of non-aborting parties*: Computes:

  $$\mathbf{b} = \Sigma_{j \in S} \mathbf{b}_j \ , \tag{3.9}$$

  and sets the common threshold encryption key: $\mathsf{ek} = (b, a) = (\mathbf{b}[0], \mathbf{a}[0])$. //$(b, a)$ will serve for threshold encryption. The other components > 0 are only used during the relinearization.

  Sets the secret key as $\mathsf{sk} = \Sigma_{i \in S} \mathsf{sk}_i$. //accessible through $\mathcal{F}_{\mathsf{LSS}}$, via the label $\overline{\mathsf{sk}}$.

Figure 3.9: Biased Distributed Key Generation Protocol

1. The first is the so-called IND-CPA *under joint keys* property, detailed in Figure 3.10 below. It is an adaption of the one of [AJL+12], in our RLWE context. It states that even if the adversary adds any bias $\mathbf{b}'$ to the encryption key $\mathbf{ek} = (\mathbf{b}, \mathbf{a})$, then as long as the (semi-malicious) adversary can explain $\mathbf{b}'$ with a corresponding secret key sk' and small noise, then IND-CPA under the biased key: $\mathbf{b} + \mathbf{b}'$ is preserved.
2. The second property is that, due to the use of $\mathcal{F}_{\mathsf{LSS}}$ in the DKG, the threshold opening of any evaluation of a linear map over the secret key sk is as secure as (in the UC sense) if the evaluation came directly from $\mathcal{F}_{\mathsf{LSS}}$. In Section 3.5, we will use such threshold opening of well-chosen linear maps, enabling simulatability of the threshold decryption.

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 is encrypted under a key of the form $\mathbf{ek}' = (\mathbf{b} + \mathbf{b}', \mathbf{a})$, 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:

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

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 t 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 $\left\{\mathbf{b}' = -\mathbf{a}\cdot\mathsf{sk}' + \mathbf{e}^{(\mathsf{ek})}, (\mathsf{sk}_i')_{i\in\mathcal{I}}, (\mathbf{e}_i^{(\mathsf{ek})})_{i\in\mathcal{I}}\right\}$ to the challenger, along with some $m \in R_k$ of its choice.

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

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

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

Figure 3.10: IND-CPA under Joint Keys Game

The aim of this result is to be used later in Section 3.7 in the broader context of MPC. That is the reason why we consider that the honest key $\mathbf{ek} = (\mathbf{b}, \mathbf{a})$ is generated uniformly at random, instead of generated by ℓ-BFV.Keygen. Moreover, this specific use-case has an impact on the choice of parameters, which will be discussed in Corollary 28 and Equation (3.23).

Lemma 26. *Let* $\mathsf{pp} = (R_q, l, \mathscr{X}_q, R_k, \Psi_q)$ *be parameters such that Corollary 28 holds and* $\mathscr{B}_{\mathsf{Enc},q}$

that satisfies Equation (3.23)[7]*. Then for any* PPT *adversary* $\mathcal{A}$*, we have:*

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

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 Figure 3.2 in Section 3.1.1. 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 ℓ-BFV.Enc$(\mathbf{ek} = (\mathbf{b}[0], \mathbf{a}[0]), 0)$ $(\beta = 1)$ or is a sample in $U(R_q^2)$ $(\beta = 0)$.
2. Then $\mathcal{A}'$ gives $\mathbf{b}$ to $\mathcal{A}$ and gets back $\big(\mathbf{b}' = -\mathbf{a} \cdot \mathsf{sk}' + \mathbf{e}^{(\mathsf{ek})}, \mathsf{sk}', \mathbf{e}^{(\mathsf{ek})}, m\big)$ from $\mathcal{A}$, where m is a challenge plaintext.
3. Finally, $\mathcal{A}'$ sets $(\mathsf{c}_0^*, \mathsf{c}_1^*) = (\mathsf{c}_0 - \mathsf{c}_1 \cdot \mathsf{sk}', \mathsf{c}_1) \in R_q^2$, 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 (3.1), 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 $\frac{2dnB}{B_{\mathsf{Enc}}} = negl(\lambda)$ (cf Equation (3.23) later presented in Section 3.7.1). This conclusion can be formalized as the "smudging Lemma 27" below, which implies that, in the sum $e_0^{(\mathsf{Enc})} - e_1^{(\mathsf{Enc})} \cdot \mathsf{sk}' - u \cdot e^{(pk)}$, 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 24, thus has the same advantage as $\mathcal{A}$.

□

Lemma 27 (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.*

In brief, the above lemma states that two distributions differing by a small noise, can be made indistinguishable by adding an exponentially larger "smudging" noise to both.

Remark. *Although our proof is done at a low level, it could be done at a higher level by using that* ℓ-BFV *has the property known as key-homomorphism [AHI11]. Roughly speaking, this means that there is a public map which, given an offset (roughly:* sk'*) on the secret key, maps encryptions under* sk *into ciphertexts which have the same distribution as fresh encryptions under* $\mathsf{sk} + \mathsf{sk}'$*. It is a commonplace property in* FHE *schemes [BV11].*

[7] The choice of parameters will be discussed in the broader context of MPC later in Section 3.7.

3.4.2 Robust Distributed Relinearization Key Generation.

To distributively generate a common $\mathbf{rlk}$, one can leverage the additional linearity provided by our alternative algorithm presented in Section 3.3.1. In short, to build a RlkGen protocol, we let each party P_i compute an additive contribution to the relinearization key $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \ell\text{-RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$ and broadcast it. From the set S of indices of parties who have correctly broadcast their additive contributions to the relinearization key and to the threshold encryption key $\mathbf{ek}$ as in Equation (3.6), one can then compute:

$$\mathbf{rlk} := (\Sigma_{i\in S}\mathbf{d}_{0,i}, \mathbf{d}_1, \Sigma_{i\in S}\mathbf{d}_{2,i}) \tag{3.10}$$

3.4.3 Construction Details & Security

We now justify simulatability of our $\mathbf{rlk}$ generation introduced in Section 3.4.2, firstly by giving the reasoning behind its construction, before proving its security.

Detailed Construction Explanation: From multikey to single-key relinearization key. Let us slightly abuse future notation and denote RelinKeyGen 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 parties, so we roughly need a robust protocol that generates $\mathbf{rlk} := \mathsf{RelinKeyGen}(\mathsf{sk})$. 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 and $\mathbf{ek}_i$ to the threshold encryption key, each party P_i broadcasts the corresponding contribution: $\mathbf{rlk}_i = (\mathbf{d}_{0,i}, \mathbf{d}_{1,i}, \mathbf{d}_{2,i}) = \mathsf{RelinKeyGen}(\mathsf{sk}_i)$ to the relinearization key. Then, after it computed the threshold encryption key ek as in Equation (4.1), each party sets $\mathbf{rlk} = \sum_{i\in S}\mathbf{rlk}_i$, where S is the *same* set, i.e., of indices of parties not aborting in the first round, as the one used to set ek. However, one hurdle remains in that in the linear map RelinKeyGen defined in [CDKS19], the coefficient of sk_i, denoted $\mathbf{d}_{1,i}$, actually *depends* on the party making the contribution, since $\mathbf{d}_{1,i}$ is sampled by P_i. Hence, this prevents additivity between contributions from different parties (in [CDKS19], no additivity was needed) .

To solve this, we specify instead that $\mathbf{d}_1$ is in common and given by a 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 28, we will show a reduction from the pseudorandomness of our common $\mathbf{rlk}$ defined in Equation (3.10), 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 $(\mathbf{a},\ \mathbf{a}\cdot\mathsf{sk}_i + \mathbf{e}_i^{(\mathsf{ek})})_{i\in[n]}$, all with the same public uniform randomness $\mathbf{a}$ but with *different independently sampled* secrets sk_i. So the idea is that the reduction to RLWE, upon receiving *one* RLWE challenge sample: $(\mathbf{a},\ \mathbf{a}\cdot\mathsf{sk}_n + \mathbf{e}_n^{(\mathsf{ek})})$, simply generates itself $n-1$ other challenges: $(\mathbf{a},\ \mathbf{a}\cdot\mathsf{sk}_i + \mathbf{e}_i^{(\mathsf{ek})})_{i\in[n-1]}$ with the same $\mathbf{a}$, and handles them to our adversary.

Security. In Corollary 28, we prove that, despite our specification of a common $\mathbf{d}_1$, the concatenation of all the honestly generated contributions $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \ell\text{-RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$ 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 B.

Consider a public sampling of an 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 $(\mathsf{sk}_m, \mathbf{ek}_m)$ by using ℓ-BFV.Keygen, all using the common public $\mathbf{a}$. Each machine m generates $(\mathbf{d}_{0,m}, \mathbf{d}_{2,m}) \leftarrow \ell\text{-RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_m)$. 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 28 (Security with Common Public Randomness)**.** *Consider:*

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

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 B). □

3.5 Threshold Decryption

In this section, we first recall the blueprint of threshold decryption and in Section 3.5.1, the mainstream instantiation used in previous works. We then present in Section 3.5.2 a second improved protocol that enables n times shorter ciphertexts.

Threshold Decryption. Recall from Definition 7 that the decryption of a ℓ-HE ciphertext c can be seen as a two steps process: (i) the evaluation of a linear map $\Lambda^{\mathsf{c}}_{\mathsf{Dec}}$ applied to the secret key sk, with public coefficients equal to the ciphertext c, (ii) followed by the 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 is the case for RLWE-based FHE schemes. Indeed, the output μ_{dec} of (i) actually leaks information about the secret key (see also, e.g., [BGG+18, §2.1]), which ruins the security of the encryption scheme.

To circumvent this issue, Asharov et al. [AJL+12] introduced the technique of adding some extra noise to the output of (i) before it can be opened. 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 fixed upper-bound on the decryption noise of a ciphertext after evaluation of circuit C (formally defined in Definition 32 in Equation (3.20) for trBFV). The

choice of B_{sm} is crucial for both the security and correctness of our MPC protocol. This translates into the following two requirements:

1. First, there is a requirement that the output of (i), i.e. $\mu_{\text{dec}} = \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk})$, is statistically close enough to the (scaled) plaintext circuit evaluation $\Delta \cdot y$. Then, there should exist some level of noise B_{sm}, so that adding a uniform noise $e_{\text{sm}} \in [-B_{sm}, B_{sm}]$ to both μ_{dec} and $\Delta \cdot y$, makes them indistinguishable, while keeping the result correct: $y = \Omega_{\mathsf{Dec}}(\mu_{\text{dec}} + e_{\text{sm}})$. As stated in the Smudging Lemma 27, the indistinguishability requirement imposes a level of noise high enough so that $B_{\mathsf{C}}/B_{sm} \leqslant \text{negl}(\lambda)$.
2. Second, the correctness requirement imposes that B_{C} added with this smudging noise stays small, i.e. such that $B_{\mathsf{C}} + n \cdot B_{sm} \leq \Delta/2$[8].

In Sections 3.5.1 and 3.5.2, we give two methods for opening μ_{dec} added with such noise, that are illustrated in Figure 3.11.

Example of ℓ-BFV**.** In the specific case of ℓ-BFV defined in Figure 3.6, we have that the linear map used in (i) is defined as:

$$\Lambda^{\mathsf{c}}_{\mathsf{Dec}} : \mathsf{sk} \to \mathsf{c}[0] + \mathsf{c}[1] \cdot \mathsf{sk}, \tag{3.11}$$

applied to the secret key sk, with public coefficients equal to the ciphertext c.

The decryption is followed (ii) by the local computation of a non-linear decoding function defined as $\Omega_{\mathsf{Dec}}(.) = [\left\lfloor \frac{k}{q}(.) \right\rceil]_k$. The decryption noise of a ciphertext after evaluation of an arithmetic circuit C is formally defined in Definition 32 in Section 3.6.4 for trBFV.

3.5.1 Mainstream Threshold Decryption method

This first method follows the approach of [AJL+12] and has been used in most other works [BGG+18; KJY+20]. Each party P_i locally samples a so-called smudging noise $e_{\text{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ 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 [ABV+12] [BGG+18, §2.1]). Following the previous notation and explanations, the bound B_{sm} is chosen such that: $B_{\mathsf{C}}/B_{sm} = \text{negl}(\lambda)$ (for indistinguishability), and such that $B_{\mathsf{C}} + n.n!^3.B_{sm} < \Delta/2$ for correctness of Lagrange reconstruction-then-rounding (with $\Delta = q/2$ in the instantiation of [BGG+18] 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 size.

Remark. *Let us note that if this approach were to be used for* CKKS*, then it could not be applied by simply sending the decryption shares over asynchronous point-to-point channels. Indeed, in* CKKS *there is no rounding in decryption. Thus, if each party naively applies reconstruction on the first* $t+1$ *decryption shares which it asynchronously receives, since the added smudging noises are* different *in each different batch of* $t+1$ *shares, the resulting output*

[8] Where Δ denotes here a scheme-dependent scaling factor.

Threshold Decryption protocol

Participants: n parties P_i, $i = 1, \ldots, n$;
Inputs: a public ciphertext c; shared secret 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 2^{nd} 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).

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

- Generates its decryption share of the decryption, i.e. $\mu_{\mathrm{dec},i} = \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk}_i)$. Samples a smudging noise $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$. Then, sends over P2P channels its "noisy decryption share" $\mu_i = \mu_{\mathrm{dec},i} + n!^2 e_{\mathrm{sm},i}$. // The $n!^2$ factor is not needed if $\{0,1\}$-LSSD is used [JRS17; BS23; CCK23] ;
- Each party P_i waits until it receives noisy decryption shares from a subset $\mathcal{U} \subset [n]$ of $t+1$ parties: $(\mu_j)_{j \in \mathcal{U}}$. Denote $(\lambda_j^{\mathcal{U}})_{j \in \mathcal{U}}$ the Lagrange reconstruction coefficients corresponding to the subset $\mathcal{U}$ (see Definition 4). It sets $\mu = \sum_{j \in \mathcal{U}} \lambda_j^{\mathcal{U}} \mu_j$ the smudged decryption. Then it outputs $\Omega_{\mathsf{Dec}}(\mu)$

(2^{nd} method, with smaller noise): Each party P_i:

- Given labels $(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})$, and a ciphertext c, sends $\left(\mathsf{LCOpen}, \Lambda^{\mathsf{c}}_{\mathsf{Dec}+sm}(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})\right)$ (see Equation (3.12)) to $\mathcal{F}_{\mathsf{LSS}}$, obtains $\widetilde{\mu}$, and outputs $m = \Omega_{\mathsf{Dec}}(\widetilde{\mu})$.

Figure 3.11: Threshold Decryption Protocol

would also be different between parties. Therefore, the second approach presented below in Section 3.5.2 should be adopted in this case.

3.5.2 Improved Threshold Decryption Method

To keep the ciphertext size small, the improved method follows the forgotten approach, which we credit to [GLS15]. Parties do not anymore smudge their decryption share of μ_{dec}, which we recall is the result of the evaluation of a linear map $\Lambda^{\mathsf{c}}_{\mathsf{Dec}}$ applied to the secret key sk, with a public coefficient equal to the ciphertext. Instead, they now open all at once the decryption μ_{dec} and a common shared noise e_{sm}, i.e., they open the linear map defined as:

$$\Lambda^{\mathsf{c}}_{\mathsf{Dec}+sm} : (\mathsf{sk}, e_{\mathrm{sm}}) \to \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk}) + e_{\mathrm{sm}} \tag{3.12}$$

The distributed generation of the noise is simply by adding secret-shared contributions $e_{\mathrm{sm},i}$, each sampled in $[-B_{sm}, B_{sm}]$. As a result, the correctness constraint now imposes only $B_{\mathsf{C}} + n.B_{sm} < \Delta/2$. Hence, the ciphertext expansion factor Δ has a dependency in n which is only linear (n), instead of $n.n!$[3] in the previous method. We defer the security argument to the proof of our MPC protocol in Section 3.8, where these formulas show up in Lemma 34.

Since the noise can be used only for one threshold decryption, parties 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 party P_i secret-shares a contribution $e_{\mathrm{sm},i} \leftarrow [-B_{sm}, B_{sm}]$ in the form of a PVSS, i.e. a vector of encrypted shares, in the first step. Then, parties define the common shared smudging noise as the sum of the contributions of the parties which did not abort: $e_{\mathrm{sm}} = \sum_{i \in S} e_{\mathrm{sm},i}$.

3.6 Noise Analysis

We now analyze the noise introduced by the different protocols and algorithms presented so far. Specifically, we consider the changes made in ℓ-BFV with the relinearization algorithm introduced in Figure 3.5, compared to the original BFV cryptosystem [FV12]. In Sections 3.6.1 and 3.6.2, we detail the noise introduced in the keys and in the ciphertexts respectively. For them, the original analysis [FV12] largely applies, albeit with a dependency on the number n of parties. Finally in Section 3.6.3, we depart from the original scheme by analyzing the impact of the changes made to the relinearization, before concluding in Section 3.6.4 with an analysis of the noise introduced when computing an arithmetic circuit. The latter analysis will be of prime importance in selecting the practical parameters of the MPC protocol presented in Section 3.7.1.

Let us first introduce some general definitions:

Definition 29 (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* $e^{(\mathsf{Dec})}(\mathsf{c}, \mathsf{sk}, m) := \Lambda^{\mathsf{c}}_{\mathsf{Dec}}(\mathsf{sk}) - \Delta \cdot m$*.*

From there, we derive the following correctness property for ℓ-BFV.

Proposition 30 (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,* ℓ-BFV.Dec$(\mathsf{sk}, \mathsf{c}) = m$.

3.6.1 Decryption Noise of Distributed Key Generation

Following Π_{DKG} presented in Figure 3.9, the noise of the threshold encryption key is the result of the sum of additive contributions and therefore we have $\|\mathbf{e}^{(\mathbf{ek})}\| \leq n \cdot B$, where B is the worst-case norm for an error term from Ψ_q. Moreover, as each additive share sk_i is typically sampled in $\mathcal{R}_3$, we have that $\|\mathsf{sk}\| \leq n$.

3.6.2 Fresh Encryption

We now formalize the set in which belong the outputs of ℓ-BFV.Enc. For any $m \in R_k$, we denote as a "Fresh ℓ-BFV Encryption of m" under a key $\mathsf{ek} = (b, a)$, 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 $\left\|e_1^{(\mathsf{Enc})}\right\| \leq B$.

Following Definition 29, 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. Recall that by definition we have that $\mathsf{c}[0] + \mathsf{c}[1] \cdot \mathsf{sk} = \Delta m + e^{(fresh)}$. With $e^{(\mathsf{ek})} = \mathbf{e}^{(\mathbf{ek})}[0]$, we have:

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

$$\left\|e^{(fresh)}\right\| \leq B_{\mathsf{Enc}} + d\|e^{(\mathsf{ek})}\| + dB\,\|\mathsf{sk}\| = B_{\mathsf{Enc}} + 2d \cdot n \cdot B := B_{fresh}. \tag{3.13}$$

3.6.3 Decryption Noise of Homomorphic Operations

We now analyze the noise introduced by additions and multiplications.

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 $\mathsf{c}_{\mathsf{Add}} = \ell$-BFV.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,

$$\|e^{(\mathsf{Add})}\| = \|e_1^{(\mathsf{Dec})} + e_2^{(\mathsf{Dec})} + r_k(q) \cdot r\| \leq \|e_1^{(\mathsf{Dec})}\| + \|e_2^{(\mathsf{Dec})}\| + r_k(q) \tag{3.14}$$

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

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 3.3.1 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 the secret key sk and plaintext $m_1\, m_2$", and denote it $e^{(tens)}$, as:

$$\hat{\mathsf{c}}[0] + \hat{\mathsf{c}}[1] \cdot \mathsf{sk} + \hat{\mathsf{c}}[2] \cdot \mathsf{sk}^2 = \Delta [m_1\, m_2]_k + e^{(tens)} \tag{3.15}$$

Using [FV12, Lemma 2], we conclude that

$$\|e^{(tens)}\| \leq d \cdot k \left(\|e_1^{(\mathsf{Dec})}\| + \|e_2^{(\mathsf{Dec})}\| \right) (d \cdot \|\mathsf{sk}\| + 1) + 2k^2 \cdot d^2 \left(\|\mathsf{sk}\| + 1 \right)^2. \tag{3.16}$$

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

2. Relinearization. Second, a relinearization algorithm is executed using a key, denoted $\mathbf{rlk}$, generated distributively using our ℓ-$\mathsf{RlkKeygen}$ algorithm detailed in Figure 3.5. Recall that $\mathbf{rlk} = (\sum_{i \in n} \mathbf{d}_{0,i}, \mathbf{d}_1, \sum_{i \in n} \mathbf{d}_{2,i})$ where $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \ell\text{-}\mathsf{RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$, the latter being defined as: for $(\mathbf{e}_{0,i}^{(\mathbf{rlk})}, \mathbf{e}_{2,i}^{(\mathbf{rlk})}) \xleftarrow{\$} (\Psi_q^l)^2$ and $r_i \xleftarrow{\$} \mathscr{X}_q$, output $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) = \left(-\mathsf{sk}_i.\mathbf{d}_1 + \mathbf{e}_{0,i}^{(\mathbf{rlk})} + r_i.\mathbf{g},\ r_i.\mathbf{a} + \mathbf{e}_{2,i}^{(\mathbf{rlk})} + \mathsf{sk}_i.\mathbf{g} \right)$.

Consider a degree two ciphertext $\hat{\mathsf{c}}$ with decryption noise $e^{(tens)}$ with respect to plaintext m ($m_1 m_2$ in our context) and secret key sk. Let us now recall that algorithm Relin presented in Algorithm 1, takes as input $\hat{\mathsf{c}} = (\hat{\mathsf{c}}[0], \hat{\mathsf{c}}[1], \hat{\mathsf{c}}[2]) \in R_q^3$, $\mathbf{rlk} = \left(\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2 \right) \in (R_q^l)^3, \mathbf{b} \in R_q^l$, 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:

$$\hat{\mathsf{c}}[0] + \hat{\mathsf{c}}[1]\, \mathsf{sk} + \hat{\mathsf{c}}[2]\, \mathsf{sk}^2 = \mathsf{c}'[0] + \mathsf{c}'[1]\, \mathsf{sk} + e^{(relin)} \tag{3.17}$$

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

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

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

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

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

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

$$\|e^{(relin)}\| \leq \|err_1\| + \|err_2\| \leq d \cdot l \cdot n \cdot B_g \cdot B + 2d^2 \cdot l^2 \cdot n^2 \cdot B_g \cdot B \tag{3.18}$$

From Equations (3.16) and (3.18) we deduce the following proposition:

Proposition 31 (Decryption noise of a product)**.** *Consider two ciphertexts* c_1 *and* c_2 *of* m_1 *and* m_2 *respectively under a key* $\big(\boldsymbol{b} = -\mathbf{a} \cdot \mathsf{sk} + \mathbf{e}^{(\mathsf{ek})}, \mathbf{a}\big) \in R_q^{2\times l}$*, with decryption noises (Definition 29) 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* $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \ell\text{-RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$*, denote* $\mathsf{c}' := \ell\text{-BFV.Mult}(\mathsf{c}_1, \mathsf{c}_2, \mathbf{rlk}, \boldsymbol{b})$*, then* $e^{(\mathsf{Dec})}(\mathsf{c}', \mathsf{sk}, m_1 m_2)$ *is dominated by* $k \cdot d^2 \cdot n(\|e_1^{(\mathsf{Dec})}\| + \|e_2^{(\mathsf{Dec})}\|) + 2d^2 \cdot l^2 \cdot n^2 \cdot B_g \cdot B$*.*

3.6.4 Correctness of Threshold Decryption after Homomorphic evaluation of a Circuit

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

Definition 32 (Decryption noise of a circuit: B_{C})**.** *For any arithmetic circuit* $\mathsf{C} : R_k^n \to R_k$ *of depth* L*, with input gates indexed by* n*, as introduced in Section 1.4, 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* ℓ-BFV *Encryptions of them* $(\mathsf{c}_i)_{i\in[n]}$*; denoting* $\mathsf{c} := \ell\text{-BFV.Eval}(\mathsf{C}, (\mathsf{c}_i)_i, \mathbf{rlk}, \boldsymbol{b})$ *and* $y := \mathsf{C}((m_i)_i)$*.*

From Definition 29 and Figure 3.11, 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:*

$$B_{\mathsf{C}} + n \cdot B_{sm} < \frac{\Delta}{2} \tag{3.19}$$

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

The noise introduced by evaluating C is dominated by the one introduced by multiplications rather than additions, unless the width is much larger than L, which we do not assume in this estimation. Thus we neglect, comparatively, the impact of n. Using Proposition 31, we estimate an upper bound on the decryption noise of the evaluated ciphertext as:

(3.20) $$C_1^L \cdot B_{fresh} + C_2 \sum_{i=0}^{L-1} C_1^i \leq C_1^L \cdot B_{fresh} + L \cdot C_2 \cdot C_1^{L-1}$$

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

3.7 MPC Protocol

In Figure 3.12, we formalize our trBFV scheme as an end-to-end MPC protocol. For simplicity we describe and prove it in the $\mathcal{F}_{\mathsf{LSS}}$-hybrid model, hence, the protocol is called $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$. Then, by UC composability, replacing $\mathcal{F}_{\mathsf{LSS}}$ by its UC implementation detailed in Section 2.6, yields an actual MPC protocol that is as secure as $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$. Concretely it starts by a distributed key and smudging noise generation which is performed in (1) followed by the local computation of the keys, and the broadcast of encrypted inputs in (2). Finally, after the local evaluation of the circuit C, the output is decrypted through a threshold decryption protocol in (3).

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

We instantiate protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ from ℓ-BFV, leveraging the alternative relinearization key generation presented in Section 3.3.1. Note that this requires uniform random strings $\mathbf{a}$ and $\mathbf{d}_1$ to be in R_q^l, and to be common parameters in order to allow some form of additivity (they were previously sampled locally in [FV12] and in [CDKS19] respectively) .

The choice of practical parameter values is discussed in Section 4.7. For security and correctness, we require Equation (3.19); and:

(3.23) $$\frac{B_{\mathsf{C}}}{n \cdot B_{sm}} = \mathsf{negl}(\lambda) \text{ and } \frac{2 \cdot d \cdot n \cdot B}{B_{\mathsf{Enc}}} = \mathsf{negl}(\lambda) \ .$$

where B_{C} is a fixed upper-bound on the decryption noise of a ciphertext after the evaluation of circuit C as given by Equation (3.20) and B_{Enc} a bound on the encryption randomness. //the former will be used in Lemma 34, the latter in Lemma 35.

Note that following Section 3.5, two possibilities exist for threshold decryption: (1) either by using the mainstream threshold decryption method (Section 3.5.1), which does not require pre-shared noises, (2) or by using the second improved method (Section 3.5.2), in which parties distributively generate a pre-shared noise in parallel with the DKG. We use the latter in Figure 3.12.

Remark. *Note that in the protocol, malicious input-owners may decide not to send their encrypted inputs. Thus, we define a set $S_{\mathsf{c}} \subseteq [n]$ of indices of non-aborting input-owners, and assume that we can define a restriction of* C *to the received inputs with indices in S_{c} that we denote* C_{c}*.*

3.8 Security

By Proposition 19, Π_{LSS} UC implements $\mathcal{F}_{\mathsf{LSS}}$. Thus, the following Theorem 33 implies Theorem 25 presented in Section 3.2, i.e. that $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ UC implements $\mathscr{F}_{\mathsf{C}}$.

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

Participants: n parties $\mathscr{P} = \{P_1, \ldots, P_n\}$, and a set $\mathcal{Q}$ of n[a] input-owners, each with input a plaintext m_i.

Bulletin-board PKI setup and CRS setup. Each P_i:

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

(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 $(\mathsf{sk}_i, (\mathbf{b}_i, \mathbf{a})) \leftarrow \ell\text{-BFV.Keygen}(\mathbf{a})$ and $(\mathbf{d}_{0,i}, \mathbf{d}_{2,i}) \leftarrow \ell\text{-RlkKeygen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk}_i)$.
  - 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 with DKG broadcasts):*
  - Samples $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ and sends $(\mathsf{input}, \overline{e_{\mathrm{sm},i}}, e_{\mathrm{sm},i})$ to $\mathcal{F}_{\mathsf{LSS}}$. //Once for each subsequent threshold decryptions, if multiple circuits.

Local Computation: Each party:

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

  $$\mathbf{b} = \Sigma_{j \in S} \mathbf{b}_j \ , \tag{3.21}$$

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

  $$\mathbf{rlk} = (\Sigma_{j \in S} \mathbf{d}_{0,j}, \mathbf{d}_1, \Sigma_{j \in S} \mathbf{d}_{2,j}) \tag{3.22}$$

(2) Broadcast of encrypted inputs: Each $\mathsf{Q}_i \in \mathcal{Q}$:

- 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.

Evaluation (local): Each P_i sets $S_{\mathsf{c}} \subseteq [n]$ the subset of indices of owners from which it received a broadcast ciphertext c_j.

- Then it computes $\mathsf{c} \leftarrow \ell\text{-BFV.Eval}(\mathsf{C}, \{\mathsf{c}_j\}_{j \in S_{\mathsf{c}}}, \mathbf{rlk}, \mathbf{b})$.[b]

(3) Threshold Decryption: Each party P_i:

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

[a] For simplicity, we consider n input-owners, but any arbitrary value could be chosen instead.
[b] Without loss of generality, C sets to $\perp$ the non received inputs in $[n] \setminus S_{\mathsf{c}}$.
[c] Defined in Equation (3.12)

Figure 3.12: MPC Protocol $\Pi^{\mathcal{F}_{\mathsf{LSS}}}_{\mathsf{MPC}}$

Theorem 33. $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ *presented in Figure 3.12 UC implements the ideal functionality* $\mathscr{F}_{\mathsf{C}}$ *for any semi-malicious adversary, in the* $(\mathcal{F}_{\mathsf{LSS}}, \mathsf{BC})$*-hybrid model with external resource* $\overline{\mathcal{G}}_{\mathit{URS}}$*.*

To prove Theorem 33, we describe in Section 3.8.1 a simulator Sim of $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ that simulates honest parties following the protocol, and ideal functionalities behaving as specified. We first convey the main ideas of Sim by describing it via a sequence of incremental changes, starting from a real execution. In the last hybrid obtained, the view of Env is generated solely by interaction with $\mathscr{F}_{\mathsf{C}}$, hence what we are describing is a simulator. The hybrids and the proofs of indistinguishability are given in Section 3.8.2.

First, in Hybrid_1, we simulate decryption by modifying the behavior of $\mathcal{F}_{\mathsf{LSS}}$ in the threshold decryption. There it, incorrectly, outputs $\mu^{\mathsf{Sim}} := \Delta.y + \Sigma_{j\in S} e_{\mathrm{sm},j}$, where $y := \mathsf{C}((m_i)_{i\in S_c})$ is the evaluation in clear of the circuit on the actual inputs. Indistinguishability follows from the "smudging" Lemma 27, as detailed in Lemma 34.

Then, in Hybrid_2, the additive contributions $(\mathbf{b}_i, \mathbf{d}_{0,i}\mathbf{d}_{2,i})_{i\in\mathcal{H}}$ of honest parties 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 28.

Finally, in Hybrid_3, we replace the actual inputs m_i of simulated honest owners by $\widetilde{m_i} := 0$. 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 $y := \mathsf{C}((m_i)_{i\in S_c})$ is still the evaluation of the circuit on the *actual* inputs. Thanks to the modifications so far, the secret keys of the honest parties are no longer used in any computation. Furthermore, since honest parties sample their contributions $\mathbf{b}_i$ to the common threshold 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 26 "IND-CPA under Joint Keys" presented in Section 3.4.1. This enables us to conclude that the distributions are indistinguishable.

In conclusion, we arrived at a view produced by a machine that interacts only with Env and $\mathscr{F}_{\mathsf{C}}$.

3.8.1 Description of the Simulator Sim of $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$.

We describe in Figure 3.13 the simulator Sim for an honest $\mathcal{L}$ (the case where the output learner is corrupt is easy. Namely, the simulator plays $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ honestly, then indistinguishability follows from correctness of $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$).

Sim initiates in its head: sets of n parties and n input-owners, and may initially receive corruption requests from Env for arbitrarily many owners and up to t parties, 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 $\mathcal{F}_{\mathsf{LSS}}$ in the Output computation step. For instance, when receiving a message from Env intended for some functionality, Sim internally sends it to the functionality and then simulates the steps taken by the functionality accordingly. Upon every output from a simulated functionality to a simulated corrupt party, or, upon every message from a simulated functionality to Sim acting as $\mathcal{A}$, then Sim immediately forwards it to Env, as would have done the actual dummy $\mathcal{A}$.

3.8.2 Hybrids, and proofs of indistinguishabilities

We go through a series of hybrid games, starting from the real execution REAL_{Π_c}. The view of Env consists of its interactions with $\mathcal{A}/\mathsf{Sim}$, and of the outputs of the actual honest parties.

Sim

- **Setup.**
  - (0) Simulates the setup of $\mathcal{F}_{\mathsf{LSS}}$.
  - (0) Retrieves $(\mathbf{a}, \mathbf{d}_1)$ from $\overline{\mathcal{G}}_{\mathsf{URS}}$ and sends it to all on behalf of $\overline{\mathcal{G}}_{\mathsf{URS}}$
- **Distributed Key and smudging noise generation:** Simulates a correct behavior of $\mathcal{F}_{\mathsf{LSS}}$. For every simulated honest party $(P_i)_{i\in\mathcal{H}}$:
  - (1) Samples $\mathsf{sk}_i \xleftarrow{\$} \mathscr{X}_q$ (never used) and $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$, and sends them to $\mathcal{F}_{\mathsf{LSS}}$.
  - (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 $S \subset [n]$ the indices of parties, 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), defines $\overline{\mathsf{sk}} = \sum_{i\in S} \overline{\mathsf{sk}_i}$, and:
  - (2) $\forall$ simulated honest owners owners $\mathsf{Q}_\ell \in \mathcal{Q}$: sets $\widetilde{m_\ell} := 0$ 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 $\widetilde{\mathsf{c}_\ell} = (\Delta\widetilde{m_\ell} + u\cdot b + e_0^{(\mathsf{Enc})}, u\cdot a + e_1^{(\mathsf{Enc})})$ over $\mathsf{BC}^{\mathsf{Q}_\ell}$.
  - (2) $\forall$ simulated corrupt owners $\mathsf{Q}_\ell \in \mathcal{Q}$, upon receiving (c_ℓ) from Env, uses $\overline{\mathsf{sk}}$ to decrypt c_ℓ into m_ℓ and sets $\widetilde{m_\ell} := 0$ if $m_i = \perp$ or $\widetilde{m_\ell} := m_\ell$ otherwise, and sends $(\mathsf{input}, \mathsf{Q}_\ell, m_\ell)$ to $\mathscr{F}_{\mathsf{C}}$.
- **Threshold Decryption:** Upon being leaked the evaluation y from $\mathscr{F}_{\mathsf{C}}$, where by definition $y = \mathsf{C}(\{m_i\}_{i\in S_c})$, then Sim simulates the following incorrect behavior:
  - $\mathcal{F}_{\mathsf{LSS}}$ eventually-outputs $\left(\Lambda^{\mathsf{c}}_{\mathsf{Dec}+sm},\ \mu^{\mathsf{Sim}} := \Delta y + \Sigma_{j\in S} e_{\mathrm{sm},j}\right)$.

Figure 3.13: Description of the simulator

We deal with the latter once and for all in Lemma 34.

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

Lemma 34. *The outputs of the actual honest parties are the same in* $\text{REAL}_{\Pi^{\mathcal{F}_{\text{LSS}}}_{\text{MPC}}}$ *and* $\text{IDEAL}_{\mathscr{F}_\mathsf{C},\mathsf{Sim},\mathsf{Env}}$. *Also, the views of* Env *in* $\text{REAL}_{\Pi^{\mathcal{F}_{\text{LSS}}}_{\text{MPC}}}$ *and* Hybrid_1 *are computationally indistinguishable.*

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

$$\mu = \mathsf{c}[0] + \mathsf{c}[1] \cdot \Sigma_{j\in S}\mathsf{sk}_j + \Sigma_{j\in S} e_{\text{sm},j}\ , \tag{3.24}$$

with $e_{\text{sm},j} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ for all $j \in S$. First, by Definition 32, we have, for some noise $e^{(\mathsf{Dec})}$, with $\|e^{(\mathsf{Dec})}\| \leq B_\mathsf{C}$:

$$\mathsf{c}[0] + \mathsf{c}[1] \cdot \Sigma_{j\in S}\mathsf{sk}_j = \Delta y + e^{(\mathsf{Dec})}\ . \tag{3.25}$$

Since $\|e_{\text{sm},j}\| \leq B_{sm}$ for all $j \in S$, it follows from the choice of parameters (that verify Equation (3.19)) and the final remark in Definition 32, that the output of honest parties in $\text{REAL}_{\Pi^{\mathcal{F}_{\text{LSS}}}_{\text{MPC}}}$ is $\Omega_{\mathsf{Dec}}(\mu) := y$, which proves our first claim.

Second, since we specified $\|e^{(\mathsf{Dec})}\|/(n \cdot B_{sm}) = negl(\lambda)$ (see Equation (3.23)), it follows that the distribution of μ, given by Equation (3.24) is computationally indistinguishable from the one of $\Delta y + \Sigma_{j\in S} e_{\text{sm},j}$, see the "smudging Lemma" 27 presented in Section 3.4.1 for a further formalization of this fact. But the latter is by definition μ^{Sim}, which is exactly the output of the functionality $\mathcal{F}_{\text{LSS}}$ 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 parties 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 28 presented in Section 3.4.3.

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

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

Lemma 35. Hybrid_2 *and* Hybrid_3 *are computationally indistinguishable.*

Proof. Since Hybrid_2, the secret keys of the honest parties ($\{P_i\}_{i\in\mathcal{H}}$) are no longer used in any computation. Furthermore, since honest parties sample their contributions $\mathbf{b}_i$ to the 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 26 "IND-CPA under Joint Keys" presented in Section 3.4.1. It considers a uniform value $\mathbf{b}$ in R_q^l, then the adversary can add to it the sum $(\mathbf{b}', \mathbf{a})$ of t 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. □

3.9 Chapter Summary

In this chapter, we introduced trBFV, a (n, t)-threshold FHE scheme based on RLWE. The construction improves upon the previous works of [MTBH21; KJY+20; Par21] by making the distributed key generation robust, including for generating the relinearization key, and by using an improved threshold decryption to obtain a smaller modulus size. Combining them together, we built a robust MPC protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ in 2 broadcasts + 1 asynchronous P2P rounds.

One limitation of our proposed protocol, as well as all previous ThFHE-based ones, is the need for two broadcasts following the overall DKG-then-Input-Distribution approach. In the following chapter, we show how to reduce the number of initial broadcast rounds to an optimal value of one.

Chapter 4

Share & Shrink: Delegated MPC with GOD from one broadcast

Contents

In this chapter, we address the shortcomings of the threshold FHE-based MPC protocol presented in Chapter 3, to make it more suitable for computation delegation. More specifically, the MPC protocol presented in Section 3.7.1 that entails Theorem 25 requires two initial broadcast rounds before the evaluation: a first one for generating the keys and a second to distribute the encrypted inputs. However, although handy abstractions for designing simple MPC protocols, broadcast (BC, see Section 1.8.2) and round-by-round synchronous communication, are costly to implement in practice as previously discussed in Section 1.8. Moreover, due to the loss of security when messages are not delivered within a round, protocol implementations must set the duration of a round very high (typically [AMN+20] $50\times$ larger than the actual network delay, in order to tolerate slowdowns). Therefore, our aim in this chapter is to reduce the number of broadcast rounds to a minimum.

Significant effort is currently put to minimize the use of BC in MPC protocols [FN09; BHN10; GGOR13; PR18; CGZ20; GJPR21; DMR+21; DRSY23]. We push further this line of work by requiring only one initial call to BC i.e. a *broadcast-optimal* protocol as studied in [CGZ20; DMR+21; GJPR21; BMMR23], without any trusted DKG setup, followed by a fully asynchronous protocol. Since we aim at the (arguably gold standard) of guaranteed output delivery (GOD) under honest majority, this initial call to BC is necessary, as shown by an elementary split-brain attack [BHN10].

Overall, two main approaches exist to design round-efficient FHE-based MPC protocols:

Threshold FHE-based: A mainstream tool for round-efficient MPC is based on (n, t)-threshold-FHE schemes as previously discussed in Chapter 3. Overall, it proceeds with the following generic approach, or close variations as shown in Figure 4.1a: it starts (1) with the distributed generation (DKG) of a common *threshold encryption key* ek, along with the private assignment to each party of a secret *key share*, followed (2) by the broadcast of encrypted inputs under the threshold encryption key. Then, parties locally evaluate a circuit through an algorithm Eval on the encrypted inputs, that outputs a ciphertext. Finally, parties (3) perform an asynchronous threshold decryption protocol using their secret key shares, so that any set of $t + 1$ of them can recover the output.

Overall, this approach allows input-owners to distribute compact input ciphertexts, i.e. of size independent of the number of inputs, and an efficient evaluation of a circuit on them. However, it also inherently requires two initial broadcasts: a first one for the DKG and a second one for the input distribution.

Multikey FHE-based: Another popular tool to reduce the number of broadcast rounds is to use a (n, n)-Multikey-FHE scheme as previously discussed in Section 1.12.2. By using such schemes, no DKG is needed before the distribution of the inputs, effectively reducing the number of broadcasts in an MPC protocol to just one as shown in Figure 4.1b : the former starts (1) with the local generation of keys by the computation parties, directly followed by the broadcast of inputs encrypted under the different keys. Then (2), parties proceed with the multikey evaluation of a circuit through an algorithm MultikeyEval, before performing a (n, n)-threshold decryption of the evaluated ciphertext.

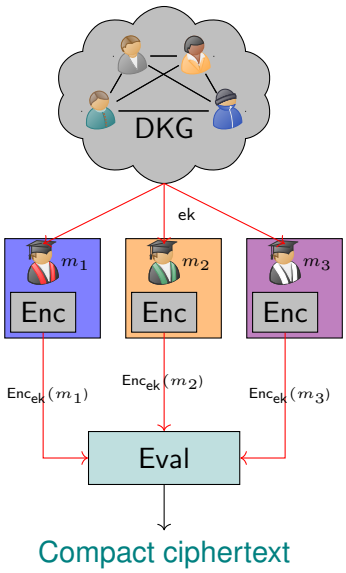
Overall, this approach makes possible the design of MPC protocols using one broadcast. However, the multikey evaluation is then done on multikey ciphertexts that are *non-compact*, i.e. that depend at least linearly on the number of inputs, which makes it very expensive in practice.

In short, these two approaches highlight a tradeoff: one allows an efficient evaluation but requires an additional broadcast round, while the other allows to obtain an MPC protocol in an optimal number of broadcasts at the cost of an expensive evaluation. In this chapter, our goal is to propose a new approach that offers the best of both worlds: *an MPC protocol with an efficient evaluation that requires only one broadcast*.

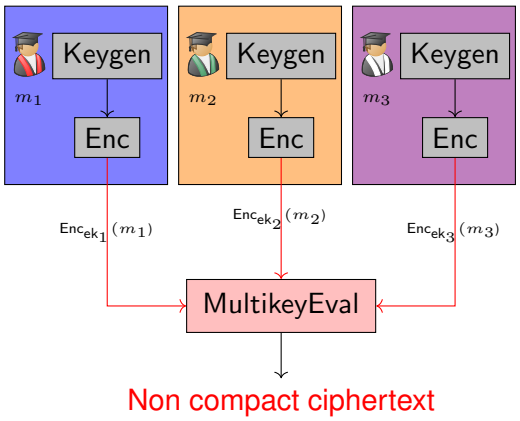
This chapter is organized as follows: we first detail the two main FHE-based approaches to build round-efficient MPC protocols and detail the main efficiency challenges (Section 4.1). Then, we present our contributions (Section 4.2) and compare them with previous related works. In Section 4.3 we give examples of multiple ℓ-HE schemes, before introducing our new approach in Section 4.4. We describe our MPC protocol and its security in Section 4.5, and prove in Section 4.6 some feasibility results. Finally, we detail in Section 4.7 some experiments to assess the practicability of the new protocol.

4.1 Preliminaries

We first recap in Sections 4.1.1 and 4.1.2 the two main approaches for designing round efficient FHE-based MPC protocols. In particular, we highlight the challenges to be overcome in order to achieve efficient protocol for delegated computation.



(a) Threshold-FHE based MPC. Parties first run a DKG protocol to obtain, in one BC, a common threshold encryption key ek and secret key shares. Then parties encrypt their inputs and broadcast them. A circuit can later be efficiently evaluated on them as they remain compact, i.e. independent from the number of inputs.



(b) Mulitkey-FHE based MPC. Parties first locally Keygen a key pair ek/sk and encrypt their inputs with their own key before broadcasting them. A circuit can later be evaluated on them even if they are not encrypted under the same key using a MultikeyEval algorithm. Unfortunately, ciphertexts are now non-compact, i.e. their sizes depend on the number of inputs, and as a result, the evaluation is inefficient.

Figure 4.1: Comparison of the threshold-FHE-based approach in 2 BC, and the multikey-FHE based approach in 1 BC for evaluating a circuit on encrypted inputs.

4.1.1 Threshold FHE

We briefly recall the approaches based on threshold-FHE schemes, following Chapter 3 and Figure 4.1a. Overall, starting from [AJL+12], an MPC protocol can be built from a threshold-FHE scheme following an approach that can be called "DKG-then-Input-distribution". It consists of two distinct steps:

1. First, a DKG protocol to generate a common threshold encryption key[1] as well as the assignment to each party of a secret key share. As seen in Chapter 3, this is a well-studied problem, and many works [MTBH21; KJY+20; Par21] have recently been proposed for efficient RLWE-based FHE schemes[2], or our robust protocol leveraging ℓ-BFV presented in Sections 3.4.1 and 3.4.2.
2. Second, the distribution by input-owners of encrypted inputs under the common threshold encryption key.

This approach, however, raises two major challenges for our goal of building a round-efficient protocol.

Challenge 1: Number of Broadcasts. Implementing DKG requires at the very least one broadcast [FS01]. When adding the input distribution, this results in at least two broadcasts, exceeding the desired goal of just one. To remove one broadcast, it would be possible to further open the box of DKG under honest majority [SBKN21]. However, inside are several synchronous rounds, due to the consensus (or MVBA) primitive, which is essentially equivalent to broadcast. So we aim for a different approach without a DKG before the input distribution.

Challenge 2: Support for Lightweight Input-Owners. An inherent limitation of the DKG-then-Input-distribution approach is that the common threshold encryption key ek produced by the DKG is not published explicitly: it is the result of a local computation made by each party. So in order for external input-owners to learn this key, it would require another intermediary step after the DKG, in which parties would notify the former of ek. Moreover, in the case of malicious parties, they would furthermore need to check NIZKs of the correctness of broadcasts in the DKG, which we want to avoid to keep them as *lightweight* as possible.

4.1.2 Multikey FHE

We briefly recall the approaches based on multikey-FHE schemes, following Section 1.12.2 and Figure 4.1b. Overall, starting from [LTV12], a multikey-FHE scheme allows to perform homomorphic operations directly over ciphertexts encrypted under different keys, which removes the need for an initial DKG before the distribution of the inputs. However, if at first sight their use seems well suited for delegated MPC, it actually comes with its own challenges.

Challenge 1: Robust Threshold Decryption. Multikey-FHE schemes inherently suffer from a lack of robustness. Indeed, the threshold decryption requires all n parties to participate,

[1] And possibly additional relinearization and/or bootstrapping keys.

[2] Which do not, however, allow to obtain an MPC protocol with GOD by lack of a robust distributed generation of relinearization & bootstrapping keys

otherwise as soon as one party aborts, the decryption fails. One could imagine a compilation of such threshold decryption protocols into ones guaranteeing GOD, at the cost of $\Omega(t)$ consecutive broadcasts and re-evaluations of the circuit. Namely: require parties to broadcast their decryption shares; Then, if some instances of broadcast for some parties returned $\perp$, discard them and restart the whole protocol with the remaining parties. Thus, in the case of t consecutively aborting parties in the threshold decryption, the execution is restarted t times. Another approach is then needed to design a robust constant-round MPC protocol.

Challenge 2: Efficient Evaluation & Delegation. What we observed from the previous DKG-then-Input-distribution approach, is that call to BC are used to provide parties with *a common view on threshold-encrypted inputs*. In a breakthrough approach, [GLS15] proposed the first (n, t)-threshold robust multikey-FHE scheme requiring one single BC. However, it suffers from two practical issues. First, because it is based on GSW [GSW13], the sizes of the ciphertexts are $d\times$ larger than those under more recent RLWE-based FHE scheme. Unfortunately, the approach of [GLS15] is not portable to RLWE, since it relies on the subset-sum Lemma of [Reg05] (see Section 4.2.2). Second and more importantly, their construction cannot enable delegation (see Section 4.2.2).

To remedy the former issue, [BJMS20] built an MPC protocol based on multikey-FHE using a more generic construction. However, their protocol still suffers from a major drawback: the sizes of the (multikey) ciphertexts of [BJMS20] are quadratic in the number $|\mathcal{Q}|$ of input-owners and so is the multikey evaluation. Part of this inefficiency is due to the use of the GSW-based multikey scheme from [MW16; BHP17]. However, even using recent more efficient multikey schemes [KKL+23; KÖA23] that reduced the overhead in homomorphic evaluation complexity to linear in $|\mathcal{Q}|$, the multikey evaluation is still $|\mathcal{Q}|\times$ less efficient than when using their threshold-FHE counterparts.

In this chapter, we then ask the following question:

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

4.2 Our Contributions

Before moving to our main contribution in Section 4.2.2, we complete in Section 4.2.1 the theoretical picture of honest majority MPC with GOD from one initial round of broadcast across various setup settings.

4.2.1 Contribution 1: Feasibility of 1 BC + asynch P2P MPC with GOD under honest majority with a bulletin board PKI.

We assume one initial access to a broadcast functionality BC (see Section 1.8.2), which guarantees an eventual delivery whatever the (non)behavior of the sender.

Theorem 36. *There exists an MPC protocol with guaranteed output delivery (GOD) under honest majority $(n = 2t + 1)$, under the sole setup of a bulletin board PKI; which furthermore*

<table>
<tr><th>Setup
Comm.</th><th>Trusted DKG [4]</th><th>bPKI + URS</th><th>bPKI</th><th>No setup</th></tr>
<tr><td>1 BC + 1 Sync P2P</td><td rowspan="2">[BHN10] ✓</td><td>[GLS15]+[DMR+21] ✓</td><td rowspan="2">✓ [BJMS20] +Thm. 36 ✓</td><td>[GIKR02] ✗</td></tr>
<tr><td>1 BC + ∞ Asynch P2P</td><td></td><td>[UR22] ✗</td></tr>
</table>

Table 4.1: Feasibility and impossibility of MPC with GOD under honest majority with different setups and communication patterns. URS stands for a public uniform random string, and bPKI for a bulletin-board PKI.

enjoys (i) termination in 1BC*-then-*1 *step of asynchronous P2P messages; (ii) allow inputs from external owners, i.e., which do not take part in the computation; (iii) is reusable, i.e. distributed inputs can be reused to compute different functions*[3]*.*

The baseline is the protocol of [BJMS20]. As stated, it does not have properties (i) nor (ii). We show in Section 4.6 how the three-round MPC protocol of [BJMS20] in the plain model can be modified to work in a BC-only model and support external input-owners. We note that our proof differs from the one of Goel et al. [GJPR21], because as stated in [DRSY23], *"we consider [...] communication patterns where broadcast is limited to one of the two rounds"*, while broadcast is assumed in both rounds in [GJPR21]. Moreover, their protocol is not reusable, i.e. given the transcript of the input distribution phase of the protocol, the computation phase cannot be reused across an unbounded polynomial number of executions to compute different functions on the same fixed joint inputs of all the parties. We refer to Appendix D for more details.

Related Works. A long line of recent works [DRSY23; DMR+21; GJPR21] have undertaken to fully characterize what MPC protocols with one initial broadcast round allow achieving across various setup settings (such as with or without PKI, with or without a URS, ...). For instance, [DRSY23] studied the feasibility and impossibility of two-round MPC with different guarantees and broadcast patterns considering a model in which only a URS is available but no PKI nor correlated randomness. Damgård et al [DMR+21] provided more details when a PKI is also available.

In Table 4.1, we complete the picture by studying the case where a bulletin board bPKI is available but no URS.

4.2.2 Contribution 2: Share&Shrink Generic Protocol for delegated MPC in one Broadcast.

We answer positively the main question by proposing a new generic protocol in the bulletin-board PKI (bPKI) model, called *Share&Shrink*. It *simultaneously* addresses the three main problems raised in Section 4.1:

> **Problem 1: MPC with GOD in one Broadcast**, i.e. the ability for our MPC protocol to achieve GOD using no more than one initial broadcast.

[3] see Appendix D for more details
[4] Threshold encryption key, secret-shared secret key

Problem 2: Delegability, i.e. the ability for an input-owner to outsource the computation to a set of parties while maintaining the privacy of its input.

Problem 3: Efficient Evaluation, i.e. the complexity of the homomorphic evaluation should not depend on the number of input-owners.

Overall, Share&Shrink performs *in parallel*: a DKG protocol, and a distribution of ciphertexts of inputs under a common threshold encryption key (thus of sizes independent of $|\mathcal{Q}|$). The broadcast BC is used **only once** simultaneously by both parties and input-owners. The second (and last) step is performed over asynchronous peer-to-peer channels.

It leverages two main ingredients:

- First, any linear homomorphic encryption scheme as defined in Definition 7 in Section 1.10.4. This includes, among others, the schemes or close variations known as CL [CL15], GSW [MW16], or the ℓ-BFV scheme detailed in Chapter 3. Intuitively, this linearity property is important because it allows us to express these schemes as a set of linear maps, which can be used as inputs to the ideal functionality $\mathcal{F}_{\mathsf{LSS}}$ introduced in Chapter 2.
- The second ingredient is precisely the linear secret sharing functionality $\mathcal{F}_{\mathsf{LSS}}$ detailed in Figure 2.2. Indeed, we recall that the latter has a reusability property, meaning among other things, that the linear map to be evaluated should not necessarily be known at the time of input sharing. This is precisely what we are going to use to enable us to perform the DKG in parallel with the input distribution. For simplicity of the following description, let us recall that sharing an input in $\mathcal{F}_{\mathsf{LSS}}$ corresponds roughly to broadcasting a PVSS[GV22; KMM+23], as defined in Definition 8 in Section 2.3.

In more detail, consider any linear homomorphic encryption scheme satisfying Definition 7 in Section 1.10.4, represented by a tuple of PPT algorithms ℓ-HE $= (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$ formalized by a set of linear maps $\Lambda^{a}_{\mathsf{EKeyGen}}, \Lambda^{\mathsf{ek}}_{\mathsf{Enc}}, \Lambda^{\mathsf{c}}_{\mathsf{Dec}}$ as well as a non-linear decoding function Ω_{Dec}. The Share&Shrink protocol is presented in Figure 4.2, and can be described as follows:

0. Setup. Each party non-interactively generates, then publishes a public key, for any public-key encryption scheme (PKE, see Definition 5 in Section 1.10.2). Parties also retrieve a uniform random string (URS), as needed in most of the ℓ-HE schemes considered.

1. Share. Parties run a DKG protocol in one round of broadcast. The pattern is the same as in [FS01] and in Section 3.4.1. Namely, each party P_i generates an additive contribution sk_i to the secret key, and ek_i to the threshold encryption key, which is the image of sk_i (and possibly some noise) by a *fixed public linear map* $\Lambda^{a}_{\mathsf{EKeyGen}}$ (e.g., for ℓ-BFV: $\mathsf{ek}_i \leftarrow (-a \cdot \mathsf{sk}_i + e_i^{(\mathsf{pk})}, a)$, where a is a URS and $e_i^{(\mathsf{pk})}$ some noise). It broadcasts ek_i and a PVSS of sk_i. We leave here implicit the necessary NIZK's proving that the public ek_i is derived from the shared sk_i (with possibly some noise $e_i^{(\mathsf{pk})}$).

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

2. Shrink. Each party locally sets the *threshold encryption key* ek as, roughly, the sum of the ek_i's for which the sk_i's were correctly shared. Parties then perform the threshold encryption of the shared inputs under ek, thereby "shrinking" them down to the sizes of ciphertexts encrypted under this single ek. What makes the threshold encryption work in *one step of*

peer-to-peer asynchronous messages, is that it simply consists in the opening of a *linear map* $\Lambda^{\mathsf{ek}}_{\mathsf{Enc}}$, parametrized by ek, evaluated over the shared secret inputs (and the shared encryption randomnesses), leveraging $\mathcal{F}_{\mathsf{LSS}}$.

In the end, parties obtain a common view on inputs encrypted under a common threshold encryption key ek. They can thus proceed as in the remaining of the DKG-then-Input-distribution approach, namely the evaluation and later the threshold decryption, which requires no further broadcast.

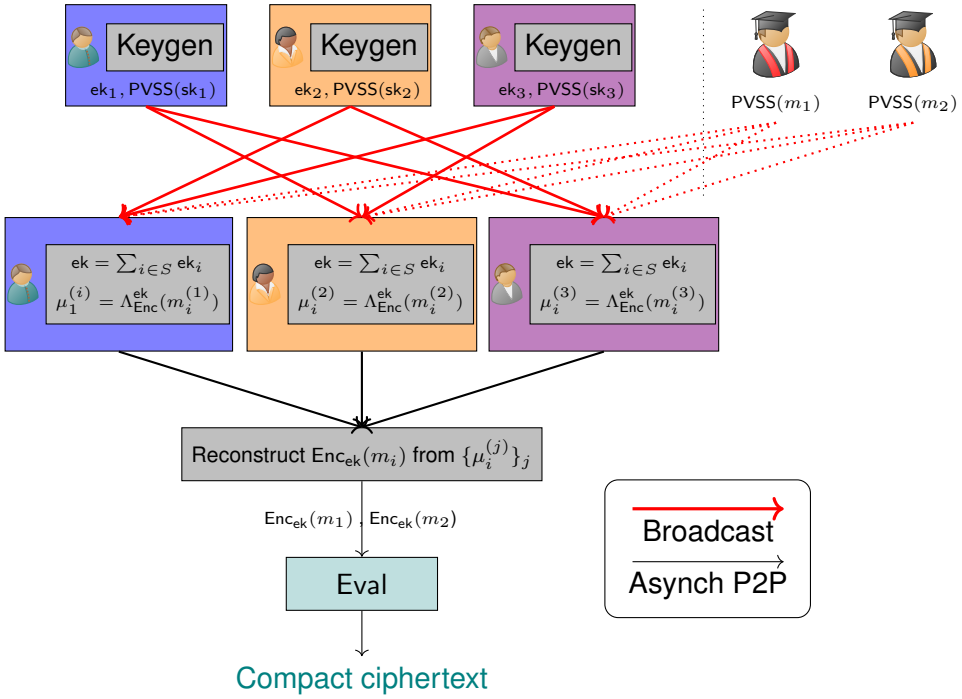


Figure 4.2: Overview of the Share&Shrink protocol for DKG & Encrypted Inputs Distribution in one BC. In broad terms, parties run a DKG in one broadcast, i.e. each party P_i Keygens a key pair $\mathsf{ek}_i/\mathsf{sk}_i$ and **shares** a PVSS of its secret keys and the contribution to a common threshold encryption key. *In parallel*, input-owners broadcast PVSSs of their inputs and leave the system. In the second step, parties locally set the *threshold encryption key* ek as roughly the sum of the ek_i's for which the sk_i's were correctly shared. Parties perform the threshold encryption of the shared inputs under ek, in one opening of the *linear map* $\Lambda^{\mathsf{ek}}_{\mathsf{Enc}}$, parametrized by ek and evaluated over the shared secret inputs, in one step of P2P messages. In more details, each party P_i evaluates $\Lambda^{\mathsf{ek}}_{\mathsf{Enc}}$ over its share $m^{(i)}$ of a shared input m, and sends its opening share $\mu^{(i)}$ of the evaluation. Upon receiving $t+1$ opening shares from any $(t+1)$-set of parties, the ciphertext can be reconstructed. This effectively **shrinks** the size of the shared encrypted inputs down to the sizes of ciphertexts encrypted under a common ek, which can be efficiently evaluated. (We leave implicit the distribution of encryption randomness as it is scheme-dependent)

Remark. *As previously discussed in Section 1.10.4, the definition of a ℓ-HE scheme encompasses schemes with various homomorphic capabilities. In particular, this includes schemes that are not fully homomorphic, for which we describe alternative evaluation methods over peer-to-peer asynchronous channels in Section 4.5.1.*

Summary. In short, as shown in Table 4.2, Share&Shrink enables building a delegated MPC protocol with GOD in one broadcast with the following properties:

- ***Generic* (RLWE compatible)**, i.e. it is scheme-independent, and in particular can be built from efficient RLWE-based schemes.
- ***Short ciphertexts size***, i.e. the size of the ciphertexts that undergo homomorphic evaluation does not depend on the number of inputs or parties. This matters in practice as this is directly linked to the computation complexity. Notably, the multikey ciphertexts used in previous approaches [BJMS20] have sizes that depend on $|\mathcal{Q}|$, which leads to a homomorphic operation complexity that also grows with $|\mathcal{Q}|$.

Theorem 37 (Share & Shrink)**.** *For any linear homomorphic encryption scheme in the sense of Definition 7, and evaluation algorithm (or asynchronous protocol) over encrypted inputs in the sense of Section 4.5.1, there exists an MPC protocol under honest majority with GOD, in 1* BC *followed by asynchronous peer-to-peer messages. It furthermore allows inputs from* external lightweight input-owners *and is reusable. It operates on ciphertexts of inputs under a common threshold encryption key, and, in particular, their sizes are* independent *of the number* $|\mathcal{Q}|$ *of input-owners, and of* n*.*

Protocol	1 BC + asynch P2P	GOD for $t < n/2$	Delegability	Size of ciphertexts
[KJY+20][MTBH21] [Par21]	✗	✗	✓	$\|\mathscr{C}\|$
Chapter 3 (trBFV)	✗	✓	✓	$\|\mathscr{C}\|$
[CDKS19]	✓	✗	✗	$\|\mathcal{Q}\| \cdot \|\mathscr{C}\|$
[GLS15]	✓	✓	✗	$\|\mathscr{C}\|$
[BJMS20] + **Thm. 36**	✓	✓	✓	$\|\mathcal{Q}\| \cdot \|\mathscr{C}\|$
Thm. 37	✓	✓	✓	$\|\mathscr{C}\|$

Table 4.2: MPC for $n = 2t + 1$ parties and $|\mathcal{Q}|$ input-owners, using FHE with lattice dimension d and modulus q, and assuming a URS and a bulletin-board PKI. The "Size" is the one of the ciphertexts which undergo homomorphic evaluation (possibly with asynchronous interactions, cf Section 4.5.1). The "Delegability" refers to the ability for the protocol to allow input-owners to outsource the computation to a set of untrusted computation parties. We did not mark as GOD the protocols which must be completely restarted when one party aborts in the middle.

It has furthermore the *reusability property* (see Appendix D), i.e., messages from input-owners are independent of the circuit to be evaluated, and multiple circuits may be evaluated on a set of distributed inputs. [BHKL18] pointed out that the lack of handling inputs from external lightweight owners, like mobile phones or web browsers, is one of the main obstacles to the deployment of MPC in practice. We, therefore, believe that enabling delegation, and having a complexity that does not grow with the number of input-owners, are significant advantages of our protocol over previous ones in 1 BC then asynchrony [GLS15; BJMS20].

Related Works. Since we have already discussed in Section 4.1 approaches based on threshold-FHE schemes [KJY+20; MTBH21; Par21; MBH23], and those based on multikey-

FHE schemes [CDKS19], we now detail the protocol of [GLS15][5] that is shown in Table 4.2.

Approach of [GLS15]: Let us recap their construction. Parties first receive a uniform random string (denoted $\mathbf{B}$), and, (0) generate and publish GSW encryption keys on the PKI. To encrypt its input m_i, each party P_i (1) generates a ciphertext $\mathsf{c}_{i,i}$ of it under its own GSW encryption key, and concatenates to it, encryptions of 0 under the $n-1$ keys of other parties, all with the same encryption randomness (denoted $\mathbf{R}$). Such a vector of ciphertexts $\hat{\mathsf{c}}_i = (\mathsf{c}_{i,1}, \mathsf{c}_{i,2}, \ldots, \mathsf{c}_{i,n})$ is denoted as a *flexible ciphertext*, and is broadcast. *In parallel* (1), parties perform the second event of a DKG, establishing a common threshold encryption key. Because a flexible ciphertext is generated using the same secret randomness for all the n GSW ciphertexts contained in it, parties are able to Transform, locally and deterministically, a flexible ciphertext into a FHE ciphertext under the common threshold encryption key by linearity. Then parties proceed with the local evaluation of the circuit, and finally (2) with the threshold decryption, which can be done over asynchronous P2P channels, according to our observation. Because the same encryption randomness is used, and given that t of these ciphertexts are encrypted under GSW encryption keys which were generated by the adversary $\mathcal{A}$, this, a priori, gives $\mathcal{A}$ an extra advantage to guess the plaintext of $\mathsf{c}_{i,i}$. For the security of GSW to hold, their encryption keys are therefore scaled slightly larger ($m = \Omega((d+n)\log(q))$ vs $m = \Omega(d\log(q))$ in GSW), in order to apply the leftover-hash-lemma (LHL [GLS15, lemma 1]).

Overall, this protocol has two main limitations:

1. ***It is not delegable*!** The generation of the flexible ciphertext $\hat{\mathsf{c}}_i$ has, by construction the secret key sk_i known by the corresponding party P_i, i.e. $\hat{\mathsf{c}}_i$ reveals the encrypted input m_i to the party P_i that owns the GSW secret key sk_i. It is therefore impossible to accommodate external input-owners without losing privacy, which prevents delegation.

2. ***It is not generic*!** A second limitation is that, since this technique relies on the Leftover Hash Lemma (LHL) [ILL89], it is unknown how to port this construction over other FHE schemes. Let us illustrate this issue by taking the BFV scheme detailed in Chapter 3 as an example. Overall, their technique is not easily transposable to efficient RLWE-based cryptosystems, for the following reasons.

   Suppose that the adversary is given one (or several) BFV encryptions of 0 under semi-maliciously generated key(s) (a, b_i), i.e., $\left(u \cdot b_i + e_{0,i}^{(\mathsf{Enc})}, u \cdot a + e_{1,i}^{(\mathsf{Enc})}\right)$, which would all be generated with the same secret randomness $u \leftarrow \mathscr{X}_q$. Then this may provide it with a distinguishing advantage when given a BFV encryption of some m under some honest key (a, b_j) which would re-use the same randomness u. One could possibly think of a fix, e.g., adapting BFV by specifying that the first component of the encryption key, $a := \mathbf{a}[0]$, would instead be vectors with coordinates in R_q, and encryption randomness $\mathbf{u}$ that is equal to a random vector with entries in R_q. But this fails since [DGKS21, §1] presented a counterexample showing that the LHL does not hold in general (a leakage of $1/d$ of the secret randomness which would make the outcome far from indistinguishable from uniform). They also point that [LPR13b, Cor 7.5] showed that a weaker version of LHL, denoted "regularity", does apply in this setting, nonwithstanding the previous leakage issue, in the case where the distribution of secret randomness would be Discrete Gaussian

[5] Note that a complete description of [BJMS20] will be done in Section 4.6.

with sufficiently large parameters. Concretely, we would apply "regularity" to $\mathbf{A}.\mathbf{u}$, where $\mathbf{A}$ would be a matrix with n rows encoding all encryption keys. The problem is that, for the applicability of "regularity", it is required that $\mathbf{A}$ be sampled uniformly, whereas in our setting we have t keys in $\mathbf{A}$ which are semi-maliciously generated, furthermore possibly depending on the other $t+1$ honest keys. Thus, this situation could potentially leak substantial information on $\mathbf{u}$, and thus potentially enable distinguishing the outcome from uniformly random, such as mentioned above [DGKS21, §1].

4.3 Cryptographic Preliminaries

We now observe that a number of encryption schemes, or close variations, such as [Pai99; BGN05; GSW13; CL15] verify Definition 7 of ℓ-HE, i.e. whose key generation, encryption, and decryption algorithms can be expressed as linear maps. As previously stated, this is the main ingredient behind our new protocol. In the following Sections 4.3.1 to 4.3.3, we give more details about some ℓ-HE schemes.

4.3.1 CL [CL15].

The ℓ-HE of Castagnos-Laguillaumie (CL) [CL15] has plaintexts in $\mathbb{Z}/p\mathbb{Z}$ but ciphertexts in a group of hidden order, hence the operations are seen as $\mathbb{Z}$-linear (the law $*$ in the target group being multiplication). In [CCL+20, §3.2] it is described how to set-up the parameters for a public common prime p.

We refer to [BDO23, Fig.4] for details about how to perform a DKG for CL, including a suitable secret sharing over $\mathbb{Z}$. Note that the DKG can be made non-interactive in one round of BC, using PVSS.

4.3.2 ℓ-BFV.

We refer to Figure 3.6 in Chapter 3 for a detailed description of the ℓ-BFV scheme. Let us note that ℓ-BFV is the only known ℓ-FHE scheme based on RLWE.

ℓ-BFV will serve as our main example in this chapter, including for our security proof in Section 4.5.3 and experiments in Section 4.7. Importantly, recall that it has fully homomorphic capabilities.

4.3.3 GSW.

From a remote perspective, the original GSW [GSW13] public key FHE scheme falls short of our linearity requirements. Indeed the encryptor needs to secretly compute a *non-linear* function, which takes as input the encryption key and some encryption randomness (namely: $\mathsf{BitDecomp}(A.R)$). Then, [AP14] introduced a variation of GSW which is compatible with our syntax, since the encryption is now a linear map. Furthermore, they observe that their variation is lossless, i.e., a ciphertext under their variation can be transformed into a GSW ciphertext without knowing the secret key. This GSW-AP variation is explicitly spelled-out in [MW16] ([AP14] described only a symmetric-key simplification) and used in [BGG+18, Appendix B].

GSW Scheme

Public Input: URS $\mathbf{A} \in \mathbb{Z}_q^{(n-1)m}$

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

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

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

Figure 4.3: The GSW scheme

Then, [BHP17] and [BJMS20] used a dual version of the GSW-AP variation, which we call "GSW*". It differs from GSW only from the choices of dimensions and distributions. In Figure 4.3, we recall the GSW-AP (that we denote as GSW for simplicity), where $\mathscr{E}$ is a distribution over $\mathbb{Z}$, m an integer, $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ a fixed efficiently computable matrix and $\mathbf{G}^{-1}(.)$ an efficiently computable deterministic "short preimage" function as defined in [MW16, Lemma 2.1]. As in Section 3.1.1, for additivity reasons, we consider that the public uniform randomness $\mathbf{A}$ is fixed and drawn from a common URS.

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

We follow the model and the formalism introduced in Section 4.3 and assume a linear homomorphic encryption scheme as in Definition 7, represented by a tuple of PPT algorithms ℓ-HE $= (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$. Recall in particular that they are built from public fixed linear maps $\Lambda^{a}_{\mathsf{EKeyGen}}, \Lambda^{\mathsf{ek}}_{\mathsf{Enc}}, \Lambda^{\mathsf{c}}_{\mathsf{Dec}}$[6] (as well as a non-linear decoding function Ω_{Dec}).

We now describe a protocol in the $\mathcal{F}_{\mathsf{LSS}}$-hybrid model, called Share&Shrink and formalized in Figure 4.4, which performs a "DKG & Encrypted Input distribution" in 1 BC + asynch P2P by leveraging $\mathcal{F}_{\mathsf{LSS}}$ introduced in Chapter 2. Precisely, it allows parties to obtain all-at-once: (i) a common threshold encryption key ek[7], (ii) a secret-shared secret key sk (formally: in $\mathcal{F}_{\mathsf{LSS}}$), and (iii) a common view on ℓ-HE ciphertexts of the inputs encrypted under ek. The challenge is that the input-owners have access to the broadcast, to distribute their inputs, *only before* ek is known!

(0) Setup: Parties receive some public parameters pp, and a uniform random string $\mathbf{a}$.

[6] And possibly an extra $\Lambda_{\mathsf{RlkGen}}$ if the specific scheme requires a relinearization key.
[7] Possibly along with a common relinearization key $\mathbf{rlk}$.

Share $\&$ Shrink protocol

Participants: n parties P_i, $i = 0, \ldots, n$, and $|\mathcal{Q}|$ input-owners;
Inputs (for each input-owner $\mathsf{Q}_\ell \in \mathcal{Q}$): plaintext m_ℓ with label $\overline{m_\ell}$. // we assume only one plaintext per input-owner .

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

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

Figure 4.4: Share $\&$ Shrink Protocol

In parallel, they Setup $\mathcal{F}_{\mathsf{LSS}}$ as explained in Section 2.2 (concretely: generate and publish public keys on bPKI).

(1) **Broadcast:** Input-owners send (**Share**) their inputs & encryption randomnesses $\rho_{\mathsf{Enc}}^{\vec{}}$ to $\mathcal{F}_{\mathsf{LSS}}$ (concretely: broadcast PVSSs of them).
In parallel, based on public parameters pp, parties generate additive contributions sk_i to the secret key, which they input to $\mathcal{F}_{\mathsf{LSS}}$ (concretely: broadcast a PVSS of them); and ek_i to the threshold encryption key (using randomness ρ_i^{key}), which they broadcast (along with possibly extra material, e.g., relinearization keys for ℓ-BFV).

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

$$\mathsf{ek} = \Sigma_{i\in S}\Lambda^a_{\mathsf{EKeyGen}}(\mathsf{sk}_i, \rho_i^{\mathsf{key}}) = \Lambda^a_{\mathsf{EKeyGen}}(\Sigma_{i\in S}(\mathsf{sk}_i, \rho_i^{\mathsf{key}}))^{8} \tag{4.1}$$

(2) **Asynchronous step (Shrinking of the inputs):** Then, parties jointly compute threshold encryptions under the common threshold encryption key ek of the shared inputs. Let us recall that the encryption in a ℓ-HE scheme is a linear function in the input and some randomness. Thus, by leveraging $\mathcal{F}_{\mathsf{LSS}}$, this step can simply be done as the threshold opening of the images of the inputs (and of the shared encryption randomnesses) by the linear map $\Lambda^{\mathsf{ek}}_{\mathsf{Enc}}$. So this is expedited in one step of asynchronous P2P messages.

The outlined protocol is fundamentally different from related threshold-FHE works [Par21; KJY+20; MTBH21] which roughly follow the pattern DKG-then-Input-distribution, since using our protocol, the Input Distribution is done without knowing a common key, which allows us $i)$ to reduce the number of broadcast rounds to just one while guaranteeing output delivery, and $ii)$ to keep short ciphertexts compared to [GLS15; BJMS20].

Remark. *Note that the security properties will come as a byproduct of the proof of MPC in Section 4.5.2, which presents a strictly harder setting, due to adversarial influence on the threshold encryption key (Lemma 44), re-use of random public parameters (Corollary 28), and public openings of smudged decryptions.*

4.5 MPC Protocol

We now present our delegated MPC protocol that operates in one single initial BC followed by asynchronous P2P messages, leveraging our novel Share&Shrink protocol introduced in

[8] Note that in the case of ℓ-BFV presented in section 4.3.2, the encryption key has another component which is the URS a, which is left outside of the sum. The same caveat applies to GSW (Section 4.3.3).

Section 4.4. First, we discuss the computation itself. When using a ℓ-HE scheme (see Definition 7) with fully homomorphic capabilities, i.e. a ℓ-FHE scheme, parties can locally evaluate a circuit on the encrypted inputs and produce an output ciphertext. While our primary focus is on evaluating circuits using FHE, we also explore alternative approaches for generating this output. To formalize this, we introduce a generic evaluation protocol Eval in Section 4.5.1 to evaluate a circuit on ℓ-HE-encrypted ciphertexts in various ways. This protocol, possibly interactive over asynchronous channels, must be simulatable. We then review several known evaluation protocols that meet these criteria. Next in section 4.5.2, we provide our generic MPC protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ in the $\mathcal{F}_{\mathsf{LSS}}$-hybrid model and discuss security. Finally in Section 4.5.3, we detail the security proof when our ℓ-BFV scheme reminded in Section 4.3.2 is used.

4.5.1 Asynchronous Evaluation of a Circuit

Consider a generic linear homomorphic encryption scheme ℓ-HE and n parties with inputs a common set of ℓ-HE ciphertexts $\{\mathsf{c}_\ell\}_{\ell\in[|\mathcal{Q}|]}$ of plaintexts $\{m_\ell\}_{\ell\in[|\mathcal{Q}|]}$ under a common threshold encryption key ek. We assume that there exists an asynchronous evaluation protocol Eval for (any) arithmetic circuit $\mathsf{C} : \mathscr{M}^{|\mathcal{Q}|} \to \mathscr{M}$ with $|\mathcal{Q}|$ input gates, that outputs a ciphertext of the evaluation of the circuit. Formally, we require that $\forall\ \mathsf{pp} \leftarrow \ell\text{-HE.Setup}(1^\lambda)$, there exists a distributed key generation in one broadcast that returns a common encryption key ek and, privately to the parties, shares sk_i of the corresponding secret key sk; $\forall\,(m_\ell)_{\ell\in[|\mathcal{Q}|]}$, $\mathsf{c}_\ell \leftarrow \ell\text{-HE.Enc}(\mathsf{pp}, \mathsf{ek}, m_\ell)$; then $\ell\text{-HE.Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{C}, \mathsf{c}_1, \ldots, \mathsf{c}_{|\mathcal{Q}|}, \mathsf{ek}^{9})) = \mathsf{C}(m_1, \ldots, m_{|\mathcal{Q}|})$. We furthermore require that Eval is simulatable, as exemplified below. In practice, there are different ways to implement Eval, among others:

- ℓ-FHE. When using a scheme with fully homomorphic capabilities as a particular kind of ℓ-HE scheme (as seen for ℓ-BFV in Section 4.3.2), there exists a built-in non-interactive algorithm Eval that comes as a property of the scheme. In particular, it is simulatable from the knowledge of the threshold encryption key and of the relinearization key.

- [CLO+13]. Choudhury et al. proposed, through a clever use of pre-processed masks, a protocol for evaluating a circuit based on an efficient interactive multi-party bootstrapping protocol for an encryption scheme that supports a limited number of homomorphic operations. Parties open threshold decryptions of masked intermediary evaluations, the simulator simply simulates the opening of a random value.

- [BHN10]. From a common view of ℓ-HE encrypted inputs under a common encryption key, and from secret key shares assigned to each party, it is possible to apply the asynchronous CDN-like ([CDN01]) protocol of [BHN10] to evaluate a circuit using a ℓ-HE scheme with only partial homomorphic properties. Parties open threshold decryptions of masked intermediary results. Since some masks can be deduced from each other by adversarially-chosen (but extractable) offsets, some extra care is paid by their simulator.

[9] Possibly along with an extra relinearization key.

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

Consider any linear homomorphic encryption scheme satisfying Definition 7, represented by a tuple of PPT algorithms ℓ-HE $= (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$. We show in Figure 4.5 how to build an MPC protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ from the Share&Shrink protocol instantiated from a ℓ-HE scheme, an evaluation algorithm Eval that depends on the homomorphic capacities of the scheme as discussed in Section 4.5.1, and a threshold decryption protocol as detailed in Section 3.5. Note that the number of shared smudging noises to be generated in parallel, for use in the 2^{nd} method of threshold decryption presented in Section 3.5.2, is equal to the number of distinct decryptions to be performed, i.e., of circuits to be evaluated.

Remark. *Note that we do not mention here the possible issues related to the generation of the relinearization keys, since they depend on the ℓ-HE scheme used. It will be done, along with a concrete instantiation of $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ from ℓ-BFV and a summarized proof of security in Section 4.5.3.*

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

Participants: n parties $P_1, \ldots, P_n$, and $|\mathcal{Q}|$ input-owners;
Inputs (for each input-owner $\mathsf{Q}_\ell \in \mathcal{Q}$): a plaintext m_ℓ with label $\overline{m_\ell}$.

- **Share** $\&$ **Shrink.** Parties and Input Owners play the Share&Shrink protocol of Figure 4.4 (with the added smudging noise sharing by parties), in which each input-owner $\mathsf{Q}_\ell \in \mathcal{Q}$ has an input m_ℓ.

  Denote $S_{\mathsf{c}} \subset [|\mathcal{Q}|]$ the indices of input-owners (resp. $S \subset [n]$ of parties), for which no instance returned $\perp$ i.e parties that broadcast a ek_i and for which $\mathcal{F}_{\mathsf{LSS}}$ has acknowledged the receipt of an additive contribution to the secret key, and input-owners for which $\mathcal{F}_{\mathsf{LSS}}$ has acknowledged the receipt for all variables of Q_ℓ's "input and randomness distribution".

  After Share $\&$ Shrink, parties have a common view on i) a common threshold encryption key ek[a], ii) a set of ciphertexts $\{\mathsf{c}_j\}_{j \in S_{\mathsf{c}}}$ encrypted under ek, iii) a shared secret key sk in $\mathcal{F}_{\mathsf{LSS}}$, as well as iv) a shared smudging noise e_{sm} in $\mathcal{F}_{\mathsf{LSS}}$ for threshold decryption (if the 2^{nd} method is used for decryption).

- **Evaluation.** To evaluate a circuit C, each party P_i runs $\mathsf{c} \leftarrow \mathsf{Eval}(\mathsf{C}, \{\mathsf{c}_j\}_{j \in S_{\mathsf{c}}}, \mathsf{ek}^{a})$.

- **Threshold Decryption.** Parties play the Threshold Decryption of Figure 3.11 with input the ciphertext c and the shared secret key sk in $\mathcal{F}_{\mathsf{LSS}}$.

  They output the plaintext m obtained.

[a] Possibly along with a common relineariation key $\mathbf{rlk}$.

Figure 4.5: MPC protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$

Sketch of UC Proof of Theorem 37. We now sketch a simulator for $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$. It initiates in its head sets $\mathscr{P}$ of n parties and $\mathcal{Q}$ of inputs-owners, and may initially receive corruption re-

quests from Env for arbitrarily many owners and up to t parties, indexed by $\mathcal{I}$. It simulates functionalities (BC, $\mathcal{F}_{\mathsf{LSS}}$) following a correct behavior, apart from the value returned by $\mathcal{F}_{\mathsf{LSS}}$ in the *threshold decryption* step[10]. We provide a full description in Section 4.5.3 in the case of an instantiation from ℓ-BFV.

We consider a ℓ-HE $= (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$ scheme as well as an Eval protocol as discussed in Section 4.5.1. To simulate $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$, the simulator Sim does the following:

- **Setup.** Simulates the setup of $\mathcal{F}_{\mathsf{LSS}}$.
- **Distributed Key and smudging noise generation:** Simulates a correct behavior of $\mathcal{F}_{\mathsf{LSS}}$. For every simulated honest party $(P_i)_{i\in\mathcal{H}}$:
  - Samples $\mathsf{sk}_i \xleftarrow{\$} \mathscr{X}$, and sends it to $\mathcal{F}_{\mathsf{LSS}}$. //possibly along with $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$
  - Samples $\mathsf{ek}_i \xleftarrow{\$} U(\mathscr{E}k)$, and sends it over BC^{P_i}. //possibly along with $\mathsf{rlk}_i \xleftarrow{\$} U(\mathscr{R}lk)$
- **Input and randomnesses distribution:** Simulates a correct behaviors of $\mathcal{F}_{\mathsf{LSS}}$, and:
  - $\forall$ simulated honest input-owners $\mathsf{Q}_\ell \in \mathcal{Q}$: sets $\widetilde{m_\ell} := 0$ and samples $\rho_{\vec{\mathsf{Enc}},\ell} \xleftarrow{\$} \overrightarrow{\mathscr{B}_{Enc}}$. Then sends $(\mathsf{input}, \{\overline{\widetilde{m_\ell}, \rho_{\vec{\mathsf{Enc}},\ell}}\}, \{\widetilde{m_\ell}, \rho_{\vec{\mathsf{Enc}},\ell}\})$ to $\mathcal{F}_{\mathsf{LSS}}$.
  - $\forall$ simulated corrupt owners $\mathsf{Q}_\ell \in \mathcal{Q}$, upon receiving (c_ℓ) from Env, use $\overline{\mathsf{sk}}$ to decrypt c_ℓ into m_ℓ and sets $\widetilde{m_\ell} := 0$ if $m_i = \perp$ or $\widetilde{m_\ell} := m_\ell$ otherwise, and sends $(\mathsf{input}, \mathsf{Q}_\ell, m_\ell)$ to $\mathscr{F}_{\mathsf{C}}$.

  As in the protocol, Sim sets $S_{\mathsf{c}} \subset [|\mathcal{Q}|]$ the indices of input-owners, resp. $S \subset [n]$ of the parties, for which no instance returned $\perp$.
- **Threshold Encryption:** Simulates correct behaviors to compute $\mathsf{ek} := \sum_{j\in S} \mathsf{ek}_j$[11], and to make $\mathcal{F}_{\mathsf{LSS}}$, for all $\ell \in S_{\mathsf{c}}$, eventually output: $\left(\Lambda_{\mathsf{Enc}}^{\mathsf{ek}}(\widetilde{m}_\ell, \rho_{\vec{\mathsf{Enc}},\ell})\right)$.
- **Circuit Computation:** Simulates Eval as discussed in Section 4.5.1.
- **Threshold Decryption:** Upon being leaked the evaluation y from $\mathscr{F}_{\mathsf{C}}$, where by definition $y = \mathsf{C}(\{m_i\}_{i\in S_{\mathsf{c}}})$, then Sim simulates the following incorrect behavior:
  - $\mathcal{F}_{\mathsf{LSS}}$ eventually-outputs $\left(\Lambda_{\mathsf{Dec}+sm}^{\mathsf{c}},\ y\right)$[12].

4.5.3 Protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ instantiated from ℓ-BFV

We describe in Figure 1 in Appendix C our MPC protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ instantiated from ℓ-BFV as introduced in Section 3.3.2.

By Proposition 19, Π_{LSS} UC implements $\mathcal{F}_{\mathsf{LSS}}$. Thus, the following Theorem 38 is a specific case of Theorem 37 introduced in Section 4.2.2 when ℓ-BFV is chosen as ℓ-HE scheme and the build-in Eval algorithm as the evaluation method.

[10] And possibly during the evaluation of the circuit depending on the method chosen as discussed in Section 4.5.1.
[11] And possibly an extra relinearization key $\mathbf{rlk}$.
[12] Possibly with an additional smudging noise which we omit here.

Theorem 38. $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ *implemented from* ℓ-BFV *detailed in Figure 1, UC implements the ideal functionality* $\mathscr{F}_{\mathsf{C}}$ *for any semi-malicious adversary, in the* $(\mathcal{F}_{\mathsf{LSS}}, \mathsf{BC})$*-hybrid model with external resource* $\overline{\mathcal{G}}_{URS}$*.*

Description of the Simulator Sim ***of*** $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$**.** To prove Theorem 38, we describe in Figure 4.6 a simulator Sim of $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ that simulates honest parties following the protocol, and ideal functionalities behaving as specified. More precisely, it initiates in its head: sets $\mathscr{P}$ of n parties and $\mathcal{Q}$ of input-owners, and may initially receive corruption requests from Env for arbitrarily many owners and up to t parties, indexed by $\mathcal{I}$. It simulates functionalities $\mathsf{BC}, \mathcal{F}_{\mathsf{LSS}}$ following a correct behavior, apart from the value returned by $\mathcal{F}_{\mathsf{LSS}}$ in the threshold decryption.

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 obtained, the view of Env is generated solely by interaction with $\mathscr{F}_{\mathsf{C}}$, thus what we are describing is a simulator. The full details of the proofs of indistinguishability are in Appendix C.1. Overall, it is largely similar to Section 3.8.

First, we simulate decryption by modifying the behavior of $\mathcal{F}_{\mathsf{LSS}}$ in the threshold decryption. There it, incorrectly, outputs $\mu^{\mathsf{Sim}} := \Delta.y + \Sigma_{j \in S} e_{\mathrm{sm},j}$, where $y := \mathsf{C}((m_\ell)_{\ell \in S_{\mathsf{c}}})$ is the evaluation in clear of the circuit on the actual inputs. Indistinguishability follows from the "smudging" Lemma 27, as detailed in Lemma 43.

Then, in Hybrid_2, the additive contributions $(\mathbf{b}_i, \mathbf{d}_{0,i}, \mathbf{d}_{2,i})_{i \in \mathcal{H}}$ of honest parties 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 28.

Finally, in Hybrid_3, we replace the actual inputs m_ℓ of simulated honest input-owners 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 $y := \mathsf{C}((m_\ell)_{\ell \in S_{\mathsf{c}}})$ is still the evaluation of the circuit on the *actual* inputs. Thanks to the modifications so far, the secret keys of the honest parties are no longer used in any computation. Furthermore, since honest parties sample their contributions $\mathbf{b}_i$ to the common threshold 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 26 "IND-CPA under Joint Keys" presented in Section 3.4.1. This enables us to conclude that the distributions are indistinguishable. In conclusion, we arrived at a view produced by a machine that interacts only with Env and $\mathscr{F}_{\mathsf{C}}$.

4.5.4 Semi-malicious Security to Malicious Security

At a high level, malicious security can be achieved by applying the compiler of [AJL+12, §E], i.e. by instructing parties to append NIZK proofs (see Definition 2) to their messages, to prove knowledge of a witness explaining them. But the compiler of [AJL+12] is designed for broadcast-based protocols, whereas in ours, parties in (3) also act based on previous outputs of $\mathcal{F}_{\mathsf{LSS}}$. This is why, in our adapted model in section 1.9.1, we also required semi-malicious parties to explain their messages based on these outputs. Namely, if a corrupt party P sends a message m, then it must have data equal to the internal state of an honest party sending this message m at this point of the protocol, i.e. the adversary must have a witness tape containing: an input value of P, coins explaining the random choices of P, and the set of all broadcast values so far. Compilation in the fully malicious model follows from instructing parties to prove knowledge of this witness tape in zero-knowledge, thereby allowing its extraction.

Sim

- **Setup.**
  - (0) Simulates the setup of $\mathcal{F}_{\mathsf{LSS}}$.
  - (0) Retrieves $(\mathbf{a}, \mathbf{d}_1)$ from $\overline{\mathcal{G}}_{\mathsf{URS}}$ and sends it to all on behalf of $\overline{\mathcal{G}}_{\mathsf{URS}}$
- **Input and randomnesses distribution:** Simulates a correct behavior of $\mathcal{F}_{\mathsf{LSS}}$. Moreover:
  - (1) $\forall$ simulated honest $\mathsf{Q}_\ell \in \mathcal{Q}$: sets $\widetilde{m_\ell} := 0$ 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 $(\mathsf{input}, \{\overline{\Delta\widetilde{m}_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(\mathsf{Enc})}}, \overline{e_{1,\ell}^{(\mathsf{Enc})}}\}, \{\Delta\widetilde{m}_\ell, u_\ell, e_{0,\ell}^{(\mathsf{Enc})}, e_{1,\ell}^{(\mathsf{Enc})}\})$ to $\mathcal{F}_{\mathsf{LSS}}$.
  - (1) $\forall$ simulated corrupt $\mathsf{Q}_\ell \in \mathcal{Q}$, upon $\Big(\mathsf{input}, \{\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(\mathsf{Enc})}}, \overline{e_{1,\ell}^{(\mathsf{Enc})}}\}, \{\Delta m_\ell, u_\ell, e_{0,\ell}^{(\mathsf{Enc})}, e_{1,\ell}^{(\mathsf{Enc})}\}\Big)$ from Env, sets $\widetilde{m_\ell} := 0$ if $m_\ell = \perp$ or $\widetilde{m_\ell} := m_\ell$ otherwise, and sends $(\mathsf{input}, \ell, m_\ell)$ to $\mathscr{F}_{\mathsf{C}}$.
- **Distributed Keys Generation:** Simulates a correct behavior of $\mathcal{F}_{\mathsf{LSS}}$. For every simulated honest $(P_i)_{i\in\mathcal{H}}$:
  - (1) Samples $\mathsf{sk}_i \xleftarrow{\$} \mathscr{X}_q$ (never used) and $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$, and sends them to $\mathcal{F}_{\mathsf{LSS}}$.
  - (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 $S_{\mathsf{c}} \subset [|\mathcal{Q}|]$ the indices of input-owners, resp. $S \subset [n]$ of the parties, for which no instance returned $\perp$.
- **Threshold encryption (Shrink) and evaluation** Simulates correct behaviors. For instance in (2), denoting $\mathbf{b} := \sum_{j\in S} \mathbf{b}_j$ and $\mathbf{rlk} := (\sum_{j\in S} \mathbf{d}_{0,j}, \mathbf{d}_1, \sum_{j\in S} \mathbf{d}_{2,j})$, the simulated $\mathcal{F}_{\mathsf{LSS}}$, $\forall \ell \in S_{\mathsf{c}}$, eventually-outputs: $\big(\Lambda_{\mathsf{Enc}}^{a,b}, \widetilde{\mathsf{c}_\ell} := (\Delta\widetilde{m}_\ell + u_\ell \cdot b + e_{0,\ell}^{(\mathsf{Enc})}, u_\ell \cdot a + e_{1,\ell}^{(\mathsf{Enc})})\big)$.
- **Threshold Decryption:** Upon being leaked the evaluation y from $\mathscr{F}_{\mathsf{C}}$, where by definition $y = \mathsf{C}(\{m_\ell\}_{\ell\in S_{\mathsf{c}}})$, then Sim simulates the following incorrect behavior:
  - $\mathcal{F}_{\mathsf{LSS}}$ eventually-outputs $\Big(\Lambda_{\mathsf{Dec}+sm}^{\mathsf{c}},\ \mu^{\mathsf{Sim}} := \Delta y + \Sigma_{j\in S} e_{\mathrm{sm},j}\Big)$.

Figure 4.6: Description of the simulator

Notice that our simulator will not perform any extraction (rewinding will be used, but only in the hybrids). The consequence is that our protocol can be instantiated with NIZKs with *non-necessary straight line* simulation extractability, which allows for more efficient NIZKs. Recent such examples are Bulletproofs and Spartan [DG23], of which the former are the ones used in [GV22] for our purpose.

We can also simplify the compiler presented in [AJL+12] by allowing parties to prove their statements with UC NIZKs, recalled in Section 1.9, since these can be set-up under honest majority from one initial call to bPKI, thanks to the technique denoted Multi-String CRS [GO14; BJMS20]. On the face of it, this call pre-pends one more step before the publication of keys on bPKI. However, we can actually have parties publish multi-strings *in parallel*. Indeed, in our semi-malicious model, we did not impose any conditions when publishing on bPKI. Multi-Strings instead of $\overline{\mathcal{G}}_{\mathsf{URS}}$-based NIZKs have the merits (i) to relieve parties from the need to access $\overline{\mathcal{G}}_{\mathsf{URS}}$ when constructing their NIZKs, and (ii) to preserve $\overline{\mathcal{G}}_{\mathsf{URS}}$ as a global resource, which would otherwise have needed to be simulated if used to produce NIZKs.

4.5.5 Proving Theorem 37 for other instantiations.

We believe that the previous proof of Theorem 38 conveys all the ideas of a generic proof of Theorem 37, since it is arguably the most complex instantiation. For instance, the instantiation with GSW is a strict simplification, since no relinearization key is needed. For the encryption scheme of CL, the IND-CPA property under a distributively generated key is not argued as in our lattice-based context (Lemma 26 "IND-CPA under Joint Keys"), so requires a minor change in the proof. Instead, in [BDO23], they provide a biased-but-simulatable DKG, which also returns a pair of encryptions: of 1 and of 0. Their simulator of [CDN01] uses the latter to replace the actual inputs by 0.

Finally, let us note that the adaptation of our proof of Theorem 37 to interactive evaluation protocols [BHN10; CLO+13], follows directly (in black box) from our simulatability requirement, exemplified in Section 4.5.1.

4.6 Proof of Theorem 36

The MPC protocol of [BJMS20] proceeds in 3 rounds of broadcasts that can roughly be described as follows:

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

Input Distribution: Let $S_1 \subseteq [n]$ be the set of indices of parties that sent a message in round 1, and let parties encrypt their inputs using a multikey-FHE scheme. Parties then broadcast the multikey ciphertexts along with PVSSs of their secret keys.

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

Now let us show how this outlined protocol can be cast in our model to obtain (i) termination in 1 BC-then-1 step of asynchronous P2P messages; and (ii) allows inputs from external owners.

We first note that the first broadcast round is input-independent. Hence, we replace their round 1 with the publication of public keys and uniform random strings on the PKI. Then, the broadcast in their 2^{nd} round consists of broadcasting one's input encrypted with a multikey-FHE scheme, along with a PVSS of one's secret key. The 3^{rd} broadcast is used to send one's decryption share of the evaluated multikey ciphertext of the output (with a suitable NIZK of correct decryption, as in [BJMS20], when compiled from semi-malicious to malicious security).

The modification to obtain (ii) is simply to allow any external input-owner to perform their round 2 broadcast, directed to the n parties. In more detail, consider the subset $S_2 \subset [|\mathcal{Q}|]$ of indices of input-owners which correctly shared their inputs and shares of multikey-FHE keys (in [BJMS20] S_2 is instead a subset of parties). From these secret shared keys, it is described in [BJMS20] how the parties still-honest-in-round-3 can *emulate* the multikey-FHE-reconstruction, as if it would have been performed by members of S_2 themselves (their participation *as input-owners* is not needed anymore).

Finally, to obtain (i), we need to replace the broadcast in their third round by one step of asynchronous P2P messages, in such a way that the output after round 3 is unchanged: parties still need to obtain the same common threshold decryption. We remark that their round 3 can actually be performed over asynchronous P2P channels. It is because the computation step performed by a party P at the end of round 3 is: *choose any* set S of $t+1$ valid decryption shares received in round-3 messages, then apply the local threshold decryption algorithm on them. So this step does not depend on whether all round-3 messages from honest parties were received by P or not, but it could very well be that t out-of-the $t+1$ valid decryption shares chosen by P, originate from corrupt parties, without any impact of the correctness of the decryption.

4.7 Experimental Evaluation

We present a proof-of-concept implementation to show that our protocol can lead to practical results. To this end, we instantiate Share&Shrink from ℓ-BFV as discussed in Section 4.3.2 (although, as seen in Section 4.3, we could also have used another ℓ-HE scheme such as GSW) and, for a fair comparison, we compare it to the most efficient multikey scheme [KKL+23] based on BFV, that we denote as MK-BFV. We consider inputs in R_k with $\log k = 16$, and parameters that achieve at least 128-bit of security level according to LWE-estimator [ACC+21]:

- For MK-BFV, we use the candidate parameter sets described in [KKL+23, Table 2], recalled in Table 4.3, that supports circuits of depth 6.
- For Share&Shrink instantiated from ℓ-BFV, we use the same parameters in our specific single-key case.

Our goal is to show that our Share&Shrink approach is effective in practice when many input-owners come together to compute a circuit on their inputs. Thus, we consider the following setting:

- a number of input-owners ranging from 1 to 128, each owning inputs in R_k,

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

Table 4.3: Experimental cryptographic parameters: Overview

- a cluster of $n = 11$ parties to perform the computation.

All experiments were performed on a MacBook Pro with a 3.1 GHz Intel i5 processor, using Lattigo[13] as well as the implementation of MK-BFV done by [KKL+23][14].

4.7.1 Experiment #1: Mult gate computation

We first compare in Figure 4.7a the running time of a Mult gate (followed by a relinearization) between two ciphertexts for different numbers of input-owners ranging from 1 to 64 (we are able to run MK-BFV with up to 64 input-owners, while the only limitation for 128 input-owners appears to be the size of RAM).

Overall, this verifies that the running time of the Mult algorithm is almost linear with the number of input-owners when using a multikey scheme, while Share&Shrink brings down this duration to a small constant, independent of the number of input-owners as expected.

4.7.2 Experiment #2: Broadcast size

We compare in Figure 4.7b the size of the broadcast for different numbers of input-owners ranging from 1 to 128 that each owns one or ten inputs in R_k, when using the MK-BFV scheme or Share&Shrink. It comprises:

- An input-independent part that consists of the encryption and relinearization keys as well as a PVSS of some secret key.
- An input-dependent part that consists either of MK-BFV ciphertexts or of PVSSs of an input $m \in R_k$ and randomnesses $e_0^{(\mathsf{Enc})}, e_1^{(\mathsf{Enc})}$, u over R_q as required in Figure 4.4.

Let us first provide some details about Figure 4.7b. For the PVSS, we use the class-group-based public-key encryption scheme recently employed in [KMM+24], while omitting zero-knowledge proofs in our semi-malicious corruption model. For n parties, the total bit-length of ciphertexts is $1752 \cdot (n+1)$ bits for a 256-bit plaintext, resulting in an asymptotic ciphertext expansion factor of 6.8.

Following Chapter 3 and denoting $|\#\text{inputs}|$ the number of inputs per input-owner, the broadcast for Share&Shrink is of size:

$$n \cdot |(3 \cdot l \cdot |R_q| + \mathsf{PVSS}(|R_q|))| + |\mathcal{Q}| \cdot |\#\text{inputs}| \cdot |\mathsf{PVSS}(3 \cdot |R_q| + |R_k|)|. \tag{4.2}$$

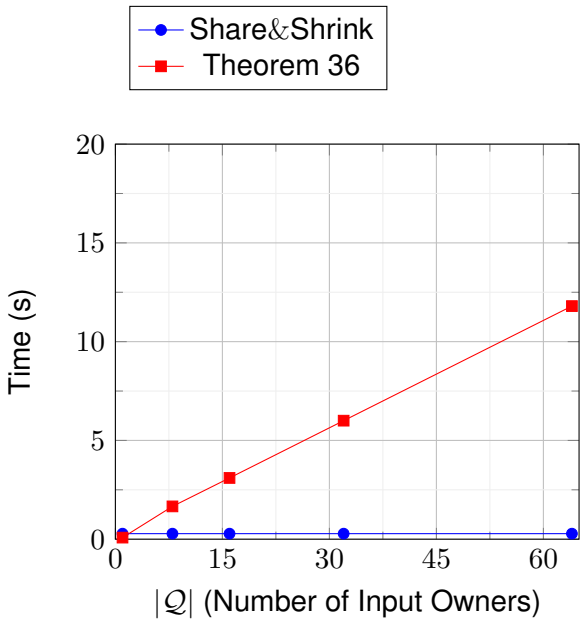
For Theorem 36 instantiated from the MK-BFV, the broadcast is of size:

$$|\mathcal{Q}| \cdot (4 \cdot l \cdot |R_q| + \mathsf{PVSS}(|R_q|) + |\mathcal{Q}| \cdot |\#\text{inputs}| \cdot 2|R_q|. \tag{4.3}$$

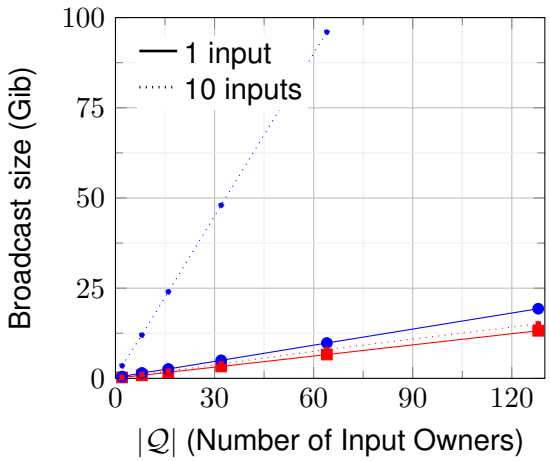
In Figure 4.7b, we remark that for a small number of inputs per owner, Theorem 36 requires broadcasting a comparable amount of data than Share&Shrink. However, the greater the

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

[14] Available at https://github.com/SNUCP/MKHE-KKLSS/



(a) Comparison of median Mult gate computation times of the Theorem 36 instantiated from the MK-BFV scheme [KKL+23] and our Share&Shrink method instantiated from ℓ-BFV. 100 runs for various number of input-owners were executed.



(b) Comparison of the size of initial broadcast between the Theorem 36 instantiated from the MK-BFV scheme [KKL+23] and our Share&Shrink protocol instantiated from ℓ-BFV, for various number of input-owners each owning one or 10 inputs.

Figure 4.7: Experimental Results

number of inputs per owner, the more the RHS of both Equations (4.2) and (4.3) dominates. Consequently, Share&Shrink requires a larger broadcast since a greater number of n-sized PVSSs are sent, which are larger than a MK-BFV ciphertext, whose size is $2|R_q|$.

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

4.8 Chapter Summary

In this chapter, we introduced a new generic protocol for designing MPC protocols in one single BC from ℓ-HE schemes. Notably, the construction improves on the previous works [GLS15; BJMS20] in that it simultaneously allows for an easy delegation and for an efficient evaluation that is independent of the number of input-owners.

Chapter 5

Conclusion and Future Research Directions

In this dissertation, we have proposed several MPC protocols that allow for easy delegation and efficient computation while being robust.

We have introduced a general framework for constructing threshold schemes using linear secret sharing for application to MPC. To do so, we proposed a new formalism, denoted (n, t)-LSSD, that serves as a wrapper for most of the (n, t)-LSS schemes used in practice, and that allows for a generic instantiation of inference and simulation of shares. This helps describe an ideal functionality $\mathcal{F}_{\mathsf{LSS}}$ in the simulation paradigm, making it much more versatile for use as a black box in complex protocols. Moreover, we detail how to implement a (n, t)-LSS scheme over polynomial rings to make it compatible with the latest, most efficient FHE schemes. This includes a variant of the Shamir scheme, denoted as R_{p^e}-Shamir, and defined over R_q, where q is a power $q = p^e$ of a prime, itself possibly small $p \leqslant n$. The latter case proves useful in practice [CH18; GIKV23] to speed-up homomorphic operations for multiple FHE schemes.

We have also introduced trBFV, the first robust (n, t)-threshold FHE scheme based on RLWE, and have instantiated this scheme in an efficient delegated MPC protocol. In doing so, we have improved on previous non-robust RLWE-based schemes [MTBH21; KJY+20; Par21], by proposing a new robust protocol for generating the relinearization key and detailing a more efficient threshold decryption protocol than the mainstream state-of-the-art approach of Boneh et al. [BGG+18], that enables smaller ciphertext sizes.

We have also improved our previous delegated MPC protocol by reducing the number of initial broadcasts to the optimal number of one while keeping an efficient evaluation. In doing so, we bridged the gap between previous approaches which were either based on threshold-FHE schemes with an efficient homomorphic evaluation but two broadcasts, or on multikey-FHE schemes which resulted in a protocol in one broadcast at the cost of a non-efficient evaluation.

Our new generic protocol, denoted Share&Shrink, presents several advantages compared to previous constructions in one broadcast [GLS15; BJMS20]: Our solution is generic and can be built from any ℓ-HE scheme, allowing for greater modularity. Moreover, the size of the ciphertexts that undergo homomorphic evaluation does not depend on the number of inputs or parties, which effectively enables scaling our MPC protocol to a very large number of *lightweight* input-owners. We have implemented and evaluated our protocol leveraging our new linear scheme, and we have demonstrated that it outperformed previous protocols in one

broadcast based on state-of-the-art multikey-FHE schemes.

Future Research Directions

Smudging Noise

The security of the threshold decryption protocol used in this work relies on a relatively crude approach consisting of hiding a distribution by adding a smudging noise from a distribution with exponentially larger variance. As a consequence, it requires the ciphertext modulus to be large enough to accommodate this extra noise, and in practice, to precisely keep track of the noise introduced during the evaluation to ensure the most accurate bounds possible [CCH+23].

To make up for these practical shortcomings, different research directions have been explored: first, replacing the statistical distance used to analyze the noise during the threshold decryption by the Rényi divergence. As recently shown by [CSS+22; BS23], this enables sampling a smudging noise from a distribution of only polynomially larger variance than the distribution to hide. However, these works are not yet usable for MPC, as they do not provide composability guarantees. Second, one can consider a relaxed model [LMSS22] where the decryption algorithm is modified by post-processing its output with a properly-chosen *differentially private* mechanism. When the use case allows it, this leads to a softer smudging mechanism.

Mobile Adversary Model

Throughout this dissertation, we have considered a model in which an adversary cannot change corruption during the protocol. However, regardless of the approach used to securely compute a circuit (FHE-based, LSS-based, ...), the evaluation may take some time. During this period, the adversary may try to disrupt the protocol and gain additional knowledge by changing the parties it corrupts. To account for that risk, Ostrovsky and Yung [OY91] introduced the notion of proactive security, in which the adversary becomes *mobile*, i.e. the set of corrupt parties may change over time. In this model, the life span of a protocol is therefore divided into separate time periods denoted “epochs”, and between each of them, all the private data is proactivized, i.e. made independent between epochs, to prevent leakage to an adversary that gradually compromises parties.

When applied to MPC, a recent line of research [GHK+21; CGG+21; RS22; AHKP22; AHKP24] studies MPC with specialized computation models, that support a *dynamically* evolving set of parties, i.e. where participants can join and leave the computation as desired, without interrupting the protocol. The rationale is that parties may devote only a limited amount of time (and computational resources) to a computation that can last a long period, or be unstably connected to the system due to an unreliable network [MCPT24]. This computation model was made even stronger by Gentry et al. [GHK+21] under the name **YOSO** (You Only Speak Once), where the computation is divided into a number of successive steps, each of them carried out by some committee of *ephemeral parties*, i.e. parties that compute one single computation step and publish a single message on a bulletin board, before vanishing from the system. An attractive consequence of this model is the drastic reduction of the window for adaptive corruption of these parties due to the unpredictable selection of committees of parties for the computation.

Unfortunately, this model does not seem to be easily adaptable to design a delegated MPC protocol, as it inherently seems to require some form of consistent terminating broadcast (BC) at the end of each computation step, for parties to perform their computation on the same intermediary values. An interesting research direction could therefore be to design a delegated MPC protocol in one broadcast in the YOSO model, which to our knowledge does not exist. This would open the way to practical scenarios in which some lightweight input-owners post some encrypted data on a public blockchain, while the main heavy computational tasks are performed off-chain by ephemeral parties.

Final Remarks

Delegated computation combined with robust cryptography has the power to efficiently unlock the valuable insights hidden within our data while safeguarding its confidentiality. Delegated MPC has the potential to be truly transformative by, paradoxically, bringing a certain form of centralization into a distributed domain. Indeed, although decentralization is often seen as the ultimate goal in MPC, this is often at the expense of performance. In the delegated model we have proposed in this dissertation, we separated the computation, which benefits from being outsourced to servers with significant resources, from the data sources, that are kept decentralized, enabling a wider spectrum of use cases. This could pave the way for an *MPC-as-a-service* system, in which improvements could be made to the evaluation that are not visible to end clients, making it easier to operate. Thus, the client's choice to carry out a certain processing operation on private data from various sources can be reduced solely to satisfying a number of properties, including robustness, which was at the heart of this dissertation and prevents unfair scenarios.

References

[21] *Google Apple. Exposure notification privacy-preserving analytics (enpa).* Online: `https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf`. 2021.

[23a] *Compilation of Public Comments on Multi-Party Threshold Cryptography Project*. Online: `https://csrc.nist.gov/files/pubs/ir/8214/c/ipd/docs/nistir-8214c-ipd-public-feedback.pdf`. 2023.

[23b] *Delegated Multi-key Private Matching for Compute: Improving match rates and enabling adoption*. Online: `https://research.facebook.com/blog/2023/1/delegated-multi-key-private-matching-for-compute-improving-match-rates-and-enabling-adoption/`. 2023.

[23c] *Lattigo v5*. Online: `https://github.com/tuneinsight/lattigo`. EPFL-LDS, Tune Insight SA. Nov. 2023.

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

[AAPP23] I. Abraham, G. Asharov, S. Patil, and A. Patra. "Detect, pack and batch: perfectly-secure MPC with linear communication and constant expected time". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques – EUROCRYPT*. 2023.

[ABT19] B. Applebaum, Z. Brakerski, and R. Tsabary. "Degree 2 is complete for the round-complexity of malicious MPC". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2019.

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

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

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

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

[ACGJ19] P. Ananth, A. R. Choudhuri, A. Goel, and A. Jain. “Two round information-theoretic MPC with malicious security”. In: *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques,* 2019.

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

[AHI11] B. Applebaum, D. Harnik, and Y. Ishai. “Semantic Security Under Related-Key Attacks and Applications”. In: *Innovations in Computer Science - ICS*. 2011.

[AHKP22] A. Acharya, C. Hazay, V. Kolesnikov, and M. Prabhakaran. “SCALES: MPC with Small Clients and Larger Ephemeral Servers”. In: *Theory of Cryptography Conference–TCC* (2022).

[AHKP24] A. Acharya, C. Hazay, V. Kolesnikov, and M. Prabhakaran. “Malicious Security for SCALES: Outsourced Computation with Ephemeral Servers”. In: *Advances in Cryptology–CRYPTO* (2024).

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

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

[ANOS24] B. Alon, M. Naor, E. Omri, and U. Stemmer. “MPC for tech giants (GMPC): enabling Gulliver and the Lilliputians to cooperate amicably”. In: *Advances in Cryptology–CRYPTO* (2024).

[AP14] J. Alperin-Sheriff and C. Peikert. “Faster Bootstrapping with Polynomial Error”. In: *Advances in Cryptology–CRYPTO*. 2014.

[BBB+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. “Bulletproofs: Short proofs for confidential transactions and more”. In: *IEEE S&P*. 2018.

[BBG+20] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. “Secure single-server aggregation with (poly) logarithmic overhead”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020.

[BBPT14] A. Beimel, A. Ben-Efraim, C. Padró, and I. Tyomkin. “Multi-linear secret-sharing schemes”. In: *Theory of Cryptography: 11th Theory of Cryptography Conference – TCC*. 2014.

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

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

[BDO23] L. Braun, I. Damgård, and C. Orlandi. “Secure multiparty computation from threshold encryption based on class groups”. In: *Annual International Cryptology Conference–CRYPTO*. 2023.

[Bea91] D. Beaver. “Efficient multiparty protocols using circuit randomization”. In: *Advances in Cryptology–CRYPTO*. 1991.

[BFLS91] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. “Checking computations in polylogarithmic time”. In: *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*. STOC’91. 1991.

[BFR13] M. Backes, D. Fiore, and R. M. Reischuk. “Verifiable delegation of computation on outsourced data”. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013.

[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: *Advances in Cryptology–CRYPTO*. 2018.

[BGN05] D. Boneh, E. Goh, and K. Nissim. “Evaluating Encryption Schemes for Pattern Matching”. In: *EUROCRYPT 2005*. Springer, 2005, pp. 40–62.

[BGV11] S. Benabbas, R. Gennaro, and Y. Vahlis. “Verifiable delegation of computation over large datasets”. In: *Annual Cryptology Conference–CRYPTO*. 2011.

[BGV12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. “(Leveled) fully homomorphic encryption without bootstrapping”. In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference–ITCS*. 2012.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. “Completeness theorems for non-cryptographic fault-tolerant distributed computation”. In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC ’88. 1988.

[BHKL18] A. Barak, M. Hirt, L. Koskas, and Y. Lindell. “An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security–CCS*. 2018.

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

[BHP17] Z. Brakerski, S. Halevi, and A. Polychroniadou. “Four round secure computation without setup”. In: *Theory of Cryptography Conference–TCC*. 2017.

[BIK+17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. “Practical secure aggregation for privacy-preserving machine learning”. In: *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017.

[BIP+22] C. Bonte, I. Iliashenko, J. Park, H. V. Pereira, and N. P. Smart. “FINAL: faster FHE instantiated with NTRU and LWE”. In: *International Conference on the Theory and Application of Cryptology and Information Security–ASIACRYPT*. 2022.

[BJMS20] S. Badrinarayanan, A. Jain, N. Manohar, and A. Sahai. “Secure MPC: Laziness Leads to GOD”. In: *International Conference on the Theory and Application of Cryptology and Information Security–ASIACRYPT*. 2020.

[BLPV18] F. Benhamouda, H. Lin, A. Polychroniadou, and M. Venkitasubramaniam. “Two-round adaptively secure multiparty computation from standard assumptions”. In: *Theory of Cryptography: 16th International Conference –TCC*. 2018.

[BLW08] D. Bogdanov, S. Laur, and J. Willemson. “Sharemind: A framework for fast privacy-preserving computations”. In: *Computer Security-ESORICS 2008: 13th European Symposium on Research in Computer Security, Proceedings 13*. 2008.

[BMMR23] S. Badrinarayanan, P. Miao, P. Mukherjee, and D. Ravi. “On the round complexity of fully secure solitary MPC with honest majority”. In: *Theory of Cryptography Conference*. 2023.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. “The round complexity of secure protocols”. In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 1990, pp. 503–513.

[Bol02] A. Boldyreva. “Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme”. In: *International Workshop on Public Key Cryptography*. 2002.

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

[BS23] K. Boudgoust and P. Scholl. “Simple Threshold (Fully Homomorphic) Encryption from LWE with Polynomial Modulus”. In: *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security*. 2023.

[BV11] Z. Brakerski and V. Vaikuntanathan. “Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages”. In: *Annual cryptology conference–CRYPTO*. 2011.

[Can01] R. Canetti. “Universally composable security: A new paradigm for cryptographic protocols”. In: *FOCS01*. We refer to eprint 2000/067 version 02/20/2020. 2001.

[CCH+23] A. Costache, B. R. Curtis, E. Hales, S. Murphy, T. Ogilvie, and R. Player. “On the precision loss in approximate homomorphic encryption”. In: *International Conference on Selected Areas in Cryptography*. 2023.

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

[CCKP19] S. Chen, J. H. Cheon, D. Kim, and D. Park. “Verifiable computing for approximate computation”. In: *Cryptology ePrint Archive* (2019).

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

[CCS19] H. Chen, I. Chillotti, and Y. Song. “Multi-key homomorphic encryption from TFHE”. In: *ASIACRYPT*. 2019.

[CD17] I. Cascudo and B. David. “SCRAPE: Scalable randomness attested by public entities”. In: *International Conference on Applied Cryptography and Network Security*. 2017.

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

[CD24] I. Cascudo and B. David. “Publicly verifiable secret sharing over class groups and applications to DKG and YOSO”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2024.

[CDD+98] R. Cramer, I. B. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. “Efficient multiparty computations with dishonest minority”. In: *BRICS Report Series* 36 (1998).

[CDGK22] I. Cascudo, B. David, L. Garms, and A. Konring. “YOLO YOSO: Fast and Simple Encryption and Secret Sharing”. In: *ASIACRYPT*. 2022.

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

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

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

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

[CF13] D. Catalano and D. Fiore. “Practical homomorphic MACs for arithmetic circuits”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2013.

[CGG+21] A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk. “Fluid MPC: secure multiparty computation with dynamic participants”. In: *Annual International Cryptology Conference–CRYPTO*. 2021.

[CGGI20] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. “TFHE: fast fully homomorphic encryption over the torus”. In: *Journal of Cryptology* 33.1 (2020), pp. 34–91.

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

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

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

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

[CKLS02] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. “Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems”. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. CCS ’02. 2002.

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

[CL17] R. Cohen and Y. Lindell. “Fairness versus guaranteed output delivery in secure multiparty computation”. In: *Journal of Cryptology* 30.4 (2017), pp. 1157–1186.

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

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. “Universally Composable Two-Party and Multi-Party Secure Computation”. In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing–STOC*. 2002.

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

[CSS+22] S. Chowdhury, S. Sinha, A. Singh, S. Mishra, C. Chaudhary, S. Patranabis, P. Mukherjee, A. Chatterjee, and D. Mukhopadhyay. “Efficient threshold FHE with application to real-time systems”. In: *Cryptology ePrint Archive, Paper 2022/1625* (2022).

[CSW23] R. Cohen, A. Shelat, and D. Wichs. “Adaptively Secure MPC with Sublinear Communication Complexity”. In: *Journal of Cryptology* 36.2 (2023), p. 11.

[DDOPS01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. “Robust non-interactive zero knowledge”. In: *Advances in Cryptology: 21st Annual International Cryptology Conference–CRYPTO*. 2001.

[DG23] Q. Dao and P. Grubbs. “Spartan and Bulletproofs are simulation-extractable (for free!)” In: *EUROCRYPT*. 2023.

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

[DHRW16] Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. “Spooky encryption and its applications”. In: *Annual International Cryptology Conference–CRYPTO*. 2016.

[Div22] S. K. and Divya Ravi and Sophia Yakoubov. *Constant-Round YOSO MPC Without Setup*. Cryptology ePrint Archive, Paper 2022/187. 2022.

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

[DPSZ12] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. “Multiparty computation from somewhat homomorphic encryption”. In: *Annual Cryptology Conference–CRYPTO*. 2012.

[DR85] D. Dolev and R. Reischuk. “Bounds on information exchange for Byzantine agreement”. In: *Journal of the ACM (JACM)* 32.1 (1985), pp. 191–204.

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

[Elg85] T. Elgamal. “A public key cryptosystem and a signature scheme based on discrete logarithms”. In: *IEEE Transactions on Information Theory* 31.4 (1985).

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

[FH06] M. Fitzi and M. Hirt. “Optimally efficient multi-valued byzantine agreement”. In: *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*. 2006, pp. 163–168.

[FKMV12] S. Faust, M. Kohlweiss, G. A. Marson, and D. Venturi. “On the Non-malleability of the Fiat-Shamir Transform”. In: *INDOCRYPT*. 2012.

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

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

[FNP20] D. Fiore, A. Nitulescu, and D. Pointcheval. “Boosting verifiable computation on encrypted data”. In: *Public-Key Cryptography–PKC*. 2020.

[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: *Cryptology ePrint Archive, Paper 2012/144* (2012).

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

[GGP10] R. Gennaro, C. Gentry, and B. Parno. “Non-interactive verifiable computing: Outsourcing computation to untrusted workers”. In: *Advances in Cryptology–CRYPTO*. 2010.

[GHK+21] C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov. “YOSO: You Only Speak Once: Secure MPC with Stateless Ephemeral Roles”. In: *CRYPTO*. 2021.

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

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

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

[GJKR07] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. “Secure distributed key generation for discrete-log based cryptosystems”. In: *Journal of Cryptology* 20 (2007), pp. 51–83.

[GJKR96] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. “Robust threshold DSS signatures”. In: *Advances in Cryptology—EUROCRYPT’96: International Conference on the Theory and Application of Cryptographic Techniques*. 1996.

[GJM+21] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. “Aggregatable Distributed Key Generation”. In: *EUROCRYPT*. 2021.

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

[GKR15] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. “Delegating Computation: Interactive Proofs for Muggles”. In: *J. ACM* 62.4 (Sept. 2015).

[GLOW21] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong. “Fully distributed verifiable random functions and their application to decentralised random beacons”. In: *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2021.

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

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

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. “The knowledge complexity of interactive proof-systems”. In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC ’85. 1985.

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

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

[GP21] C. Ganesh and A. Patra. “Optimal Extension Protocols for Byzantine Broadcast and Agreement”. In: *Distrib. Comput.* (2021).

[Gro21] J. Groth. “Non-interactive distributed key generation and key resharing”. In: *Cryptology ePrint Archive, Paper 2021/339* (2021).

[GSW13] C. Gentry, A. Sahai, and B. Waters. “Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based”. In: *Advances in Cryptology–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.

[HSS20] C. Hazay, P. Scholl, and E. Soria-Vazquez. “Low cost constant round MPC combining BMR and oblivious transfer”. In: *Journal of cryptology* 33.4 (2020), pp. 1732–1786.

[IKC+24] A. Ibarrondo, I. Kerenciler, H. Chabanne, V. Despiegel, and M. Önen. “Monchi: Multi-scheme optimisation for collaborative homomorphic identification”. In: *IH& MMSec 2024, 12th ACM Workshop on Information Hiding and Multimedia Security, 24-26 June 2024, Baiona, Spain / Also on ePrint, Paper 2024/654*. Ed. by ACM. Baiona, 2024.

[IKLP06] Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. “On combining privacy with guaranteed output delivery in secure multiparty computation”. In: *Advances in Cryptology-CRYPTO*. 2006.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. “Pseudo-random generation from one-way functions”. In: *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*. STOC ’89. 1989.

[ISO19] ISO/IEC. *ISO/IEC 18033-6:2019 IT Security techniques-Encryption algorithms-Part 6: Homomorphic encryption*. 2019. URL: https://www.iso.org/obp/ui/%5C#iso:std:iso-iec:18033:-6.

[JLP23] S. Jeon, H.-S. Lee, and J. Park. “Practical Randomized Lattice Gadget Decomposition With Application to FHE”. In: *Computer Security – ESORICS*. 2023.

[JNO14] T. P. Jakobsen, J. B. Nielsen, and C. Orlandi. “A framework for outsourcing of secure computation”. In: *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*. 2014, pp. 81–92.

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

[Kat24] J. Katz. “Round Optimal Fully Secure Distributed Key Generation”. In: *Annual Cryptology Conference–CRYPTO* (2024).

[Kel20] M. Keller. “MP-SPDZ: A versatile framework for multi-party computation”. In: *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 2020, pp. 1575–1590.

[KG21] C. Komlo and I. Goldberg. “FROST: Flexible round-optimized Schnorr threshold signatures”. In: *Selected Areas in Cryptography: 27th International Conference, October 21-23, 2020, Revised Selected Papers 27*. Springer. 2021, pp. 34–65.

[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).

[KK06] J. Katz and C.-Y. Koo. “On expected constant-round protocols for Byzantine agreement”. In: *Annual International Cryptology Conference*. Springer. 2006, pp. 445–462.

[KKL+23] T. Kim, H. Kwak, D. Lee, J. Seo, and Y. Song. “Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition”. In: *CCS*. 2023.

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

[KMM+24] A. Kate, E. V. Mangipudi, P. Mukherjee, H. Saleem, and S. A. K. Thyagarajan. *Non-interactive VSS using Class Groups and Application to DKG*. CCS. 2024.

[KMR11] S. Kamara, P. Mohassel, and M. Raykova. “Outsourcing multi-party computation”. In: *Cryptology ePrint, Paper 2011/272* (2011).

[KMS24] H. Kwak, S. Min, and Y. Song. “Towards Practical Multi-key TFHE: Parallelizable, Key-Compatible, Quasi-linear Complexity”. In: *IACR International Conference on Public-Key Cryptography–PKC* (2024).

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

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

[KOS16] M. Keller, E. Orsini, and P. Scholl. “MASCOT: faster malicious arithmetic secure computation with oblivious transfer”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 830–842.

[KPR18] M. Keller, V. Pastro, and D. Rotaru. “Overdrive: Making SPDZ great again”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pp. 158–189.

[KRDO17] A. Kiayias, A. Russell, B. David, and R. Oliynykov. “Ouroboros: A provably secure proof-of-stake blockchain protocol”. In: *Annual international cryptology conference*. 2017.

[Lin17] Y. Lindell. “How to Simulate It - A Tutorial on the Simulation Proof Technique”. In: *Tutorials on the Foundations of Cryptography*. Ed. by Y. Lindell. Springer International Publishing, 2017, pp. 277–346.

[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.

[LMSS22] B. Li, D. Micciancio, M. Schultz, and J. Sorrell. “Securing approximate homomorphic encryption using differential privacy”. In: *Annual International Cryptology Conference–CRYPTO*. 2022.

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

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

[LPSY19] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai. “Efficient constant-round multiparty computation combining BMR and SPDZ”. In: *Journal of Cryptology* 32 (2019), pp. 1026–1069.

[LSS16] Y. Lindell, N. P. Smart, and E. Soria-Vazquez. “More efficient constant-round multi-party computation from BMR and SHE”. In: *Theory of Cryptography: 14th International Conference –TCC*. 2016.

[LTV12] A. López-Alt, E. Tromer, and V. Vaikuntanathan. “On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption”. In: *Proceedings of the forty-fourth annual ACM symposium on Theory of computing–STOC*. 2012.

[MBH23] C. Mouchet, E. Bertrand, and J.-P. Hubaux. “An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption”. In: *Journal of Cryptology* 36.2 (2023), p. 10.

[MCPT24] C. Mouchet, S. Chatel, A. Pyrgelis, and C. Troncoso. *Helium: Scalable MPC among Lightweight Participants and under Churn*. Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security–CCS. 2024.

[MMT+24] D. Mouris, D. Masny, N. Trieu, S. Sengupta, P. Buddhavarapu, and B. Case. “Delegated Private Matching for Compute”. In: *Proceedings on Privacy Enhancing Technologies* (2024).

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

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

[OY91] R. Ostrovsky and M. Yung. “How to Withstand Mobile Virus Attacks (Extended Abstract)”. In: *PODC*. 1991.

[Pai99] P. Paillier. “Public-key cryptosystems based on composite degree residuosity classes”. In: *International conference on the theory and applications of cryptographic techniques*. Springer. 1999, pp. 223–238.

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

[Ped91] T. P. Pedersen. “A threshold cryptosystem without a trusted party”. In: *Advances in Cryptology—EUROCRYPT’91: Workshop on the Theory and Application of Cryptographic Techniques*. 1991.

[PHGR16] B. Parno, J. Howell, C. Gentry, and M. Raykova. “Pinocchio: Nearly practical verifiable computation”. In: *Communications of the ACM* 59.2 (2016), pp. 103–112.

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

[PRV12] B. Parno, M. Raykova, and V. Vaikuntanathan. “How to delegate and verify in public: Verifiable computation from attribute-based encryption”. In: *Theory of Cryptography: 9th Theory of Cryptography Conference –TCC*. 2012.

[Reg05] O. Regev. “On lattices, learning with errors, random linear codes, and cryptography”. In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing–STOC*. 2005.

[RRJ+22] T. Ruffing, V. Ronge, E. Jin, J. Schneider-Bensch, and D. Schröder. “ROAST: robust asynchronous schnorr threshold signatures”. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 2551–2564.

[RS22] R. Rachuri and P. Scholl. “Le mans: dynamic and fluid MPC for dishonest majority”. In: *Annual International Cryptology Conference–CRYPTO*. 2022.

[RSY21] L. Reyzin, A. D. Smith, and S. Yakoubov. “Turning HATE into LOVE: Compact Homomorphic Ad Hoc Threshold Encryption for Scalable MPC”. In: *CSCML*. 2021.

[RU21] M. Rambaud and A. Urban. *Almost-Asynchronous MPC under Honest Majority, Revisited*. Cryptology ePrint Paper 2021/503. 2021.

[SBKN21] N. Shrestha, A. Bhat, A. Kate, and K. Nayak. “Synchronous Distributed Key Generation without Broadcasts v1”. In: *ePrint 2021/1635 v1 of 2021-12-17* (2021).

[Sch99] B. Schoenmakers. “A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting”. In: *CRYPTO*. 1999.

[Set20] S. T. V. Setty. “Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup”. In: *CRYPTO*. 2020.

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

[Sho00] V. Shoup. “Practical threshold signatures”. In: *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques*. 2000.

[SS01] D. R. Stinson and R. Strobl. “Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates”. In: *Australasian Conference on Information Security and Privacy*. 2001.

[Sta96] M. Stadler. “Publicly Verifiable Secret Sharing”. In: *EUROCRYPT*. 1996.

[SV12] N. P. Smart and F. Vercauteren. “Fully homomorphic SIMD operations”. In: *Designs, Codes and Cryptography* (2012).

[SVV16] B. Schoenmakers, M. Veeningen, and N. de Vreede. “Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation”. In: *Applied Cryptography and Network Security: 14th International Conference –ACNS 2016*. 2016.

[TLC+23] F. Tang, G. Ling, C. Cai, J. Shan, X. Liu, P. Tang, and W. Qiu. “Solving small exponential ECDLP in EC-based additively homomorphic encryption and applications”. In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 3517–3530.

[UR22] A. Urban and M. Rambaud. “Share\& Shrink:(In-) Feasibility of MPC from one Broadcast-then-Asynchrony, and Improved Complexity”. In: *Cryptology ePrint Archive, Paper 2022/378* (2022).

[Val84] L. G. Valiant. “Short Monotone Formulae for the Majority Function”. In: *J. Algorithms* 5 (1984), pp. 363–366.

[WHZ24] W. Wu, S. Homsi, and Y. Zhang. “Confidential and Verifiable Machine Learning Delegations on the Cloud”. In: *Cryptology ePrint, Paper 2024/537* (2024).

[WJB+17] R. S. Wahby, Y. Ji, A. J. Blumberg, A. Shelat, J. Thaler, M. Walfish, and T. Wies. “Full accounting for verifiable outsourcing”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security–CCS*. 2017.

[WMR+23] J. Wan, A. Momose, L. Ren, E. Shi, and Z. Xiang. “On the Amortized Communication Complexity of Byzantine Broadcast”. In: *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*. 2023, pp. 253–261.

[WRK17] X. Wang, S. Ranellucci, and J. Katz. “Global-scale secure multiparty computation”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017.

[Yao82] A. C. Yao. “Protocols for secure computations”. In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE. 1982, pp. 160–164.

Appendices

A $(n,t)-$LSSD

Following [JRS17], we now define $(n,t)-$LSSD in Definition 39, presented as a subclass of $(n,t)-$LSS defined in Definition 3 in Section 1.10.1. This definition will help us in Proposition 41 to describe efficient simulation and inference strategies for instantiating a $(n,t)-$LSS schemes.

Definition 39 ($(n,t)-$LSSD adapted from [JRS17])**.** *A (n,t)-LSSD is defined by the following two algorithm:*

- LSSD.Share$(s \in R_q, n, t) \to (\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n)})$*: There exists a share matrix $\mathbf{M} \in R_q^{d \times l}$ with positive integers $d = \sum_{i=1}^{n} d_i, l$ and associate a partition T_i of $[d]$ of size $|T_i| = d_i$ to each party 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 $(\mathsf{sh}_1, \ldots, \mathsf{sh}_d)^T = \mathbf{M} \cdot (s, r_2, \ldots, r_l)^T$. The share for P_i is a set of entries $\mathbf{s}^{(i)} = \{\mathsf{sh}_j\}_{j \in T_i}$.*

- 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 $\{c_j^{\mathcal{U}}\}_{j \in \cup_{P_i \in \mathcal{U}} T_i}$ such that*

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

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

  (2) $$s = \sum_{j \in \cup_{P_i \in \mathcal{U}} T_i} c_j^{\mathcal{U}} \cdot \mathsf{sh}_j.$$

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

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

Definition 40. *Let $\mathscr{P} = \{P_1, \ldots, P_n\}$ be a set of parties. We define the following:*

- *A set of parties $S \subseteq \mathscr{P}$ is a* maximal invalid party set *if $|S| \leq t$ but for every $P_i \in \mathscr{P} \setminus S$, we have $|S \cup \{P_i\}| > t$.*
- *A set of parties $S \subseteq \mathscr{P}$ is a* minimal valid party 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* $\mathbf{M} \in \mathcal{M}^{d\times l}$*. For a set of indices* $T \subseteq [d]$*, we say that* T *is a* valid share set *if* $(1,0,\ldots,0) \in \mathsf{span}(\{\mathbf{M}[j]\}_{j\in T})$*, and an* invalid share set *otherwise. We also define the following:*

- *A set of indices* $T \subseteq [d]$ *is a* maximum invalid share set *if* T *is an invalid share set, but for any* $i \in [d] \setminus T$*, the set* $T \cup \{i\}$ *is a valid share set.*
- *A set of indices* $T \subseteq [d]$ *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 41. *From any* $(n,t)-$LSSD *scheme, there exists an efficient instantiation of* ShSim *and* ShInfer *following Definition 3.*

Proof. We leverage Definition 40 to outline simulation and inference strategies common to all $(n,t)-$LSSD schemes as defined in Definition 39, so that we can use functions ShSim and ShInfer as wrappers independent of the $(n,t)-$LSSD 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 $\mathbf{s}^{(i)} = \{\mathsf{sh}_j\}_{j\in T_i}$, 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:

     (3) $$\widetilde{\mathsf{sh}}_j = (c_j)^{-1}s - \sum_{j'\in T\setminus\{j\}} (c_j)^{-1}c_{j'} \cdot \mathsf{sh}_{j'}$$

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

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 $\mathbf{s}^{(1)},\ldots,\mathbf{s}^{(t)}$ 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 $\{\mathbf{s}^{(i)}\}_{i\in\mathcal{V}}$.

□

B Circular Security Hardness Assumption of [CDKS19].

The multikey-FHE 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 scheme 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 42. Consider an oracle $\mathcal{O}_{\mathcal{D}_0}$ which samples $\mathbf{a} \xleftarrow{\$} U(R_q^l)$ then Keygenerates one BFV key pair $(\mathsf{sk}, \mathbf{ek})$, then samples $\mathbf{d}_1 \xleftarrow{\$} U(R_q^l)$, then, using $\mathsf{RelinKeyGen}(\mathbf{a}, \mathbf{d}_1, \mathsf{sk})$, computes from it one public relinearization key $\mathbf{rlk} = (\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2)$ then outputs the pair $(\mathbf{ek}, \mathbf{rlk})$. Then, any adversary has a negligible advantage in distinguishing this *single* output from a single sampling in $U(R_q^{l\times 5})$.

Assumption 42. *Define the distribution:*

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

$$r \leftarrow \mathscr{X}_q,\ \boldsymbol{b} := -\mathbf{a}\cdot\mathsf{sk} + \mathbf{e}^{(\mathbf{ek})},\ \mathbf{d}_0 := -\mathsf{sk}\cdot\mathbf{d}_1 + \mathbf{e}_0^{(\mathbf{rlk})} + r\cdot\mathbf{g},\ \mathbf{d}_2 := r\cdot\mathbf{a} + \mathbf{e}_2^{(\mathbf{rlk})} + \mathsf{sk}\cdot\mathbf{g}\Big\}$$

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

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.

B.1 How Assumption 42 appears in [CDKS19]

Assumption 42 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 their 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 42. 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 Detailed Protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ when instantiated from ℓ-BFV

In Figure 1, we detail our MPC protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ when instantiated from ℓ-BFV. Notably, we adopt the same requirements as made in Section 3.7.1.

Protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ instantiated from ℓ-BFV

Participants: n parties $P_1, \ldots, P_n$ and a set $\mathcal{Q}$ of input owners;
Inputs (for each input owner $\mathsf{Q}_\ell \in \mathcal{Q}$): a plaintext Δm_ℓ with label $\overline{\Delta m_\ell}$.
Setup. Each party P_i:

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

Broadcast.

- *Input and Randomness Distribution:* Upon ready from $\mathcal{F}_{\mathsf{LSS}}$, each input owner $\mathsf{Q}_\ell \in \mathcal{Q}$:

  (1) Samples $u \xleftarrow{\$} \mathscr{X}_q$, $e_0^{(\mathsf{Enc})} \xleftarrow{\$} \mathscr{B}_{\mathsf{Enc},q}$ and $e_1^{(\mathsf{Enc})} \xleftarrow{\$} \Psi_q$ and sends $(\mathsf{input}, \{\overline{\Delta m_\ell}, \overline{u_\ell}, \overline{e_{0,\ell}^{(\mathsf{Enc})}}, \overline{e_{1,\ell}^{(\mathsf{Enc})}}\}, \{\Delta m_\ell, u_\ell, e_{0,\ell}^{(\mathsf{Enc})}, e_{1,\ell}^{(\mathsf{Enc})}\})$ to $\mathcal{F}_{\mathsf{LSS}}$. Then it goes offline.

- *Distributed Keys Generation:* Upon ready from $\mathcal{F}_{\mathsf{LSS}}$, each party P_i:

  (1) Computes $(\mathsf{sk}_i, (\mathbf{b}_i, \mathbf{a})) \leftarrow \ell\text{-BFV.Keygen}(\mathbf{a})$ and $(\mathbf{d}_{0,i}, \mathbf{d}_1, \mathbf{d}_{2,i}) \leftarrow \ell\text{-BFV.RelinKeyGen}(\mathbf{a}, \mathbf{d}_1)$. Sends $(\mathsf{input}, \overline{\mathsf{sk}_i}, \mathsf{sk}_i)$ to $\mathcal{F}_{\mathsf{LSS}}$ and $(\mathbf{b}_i, (\mathbf{d}_{0,i}, \mathbf{d}_{2,i}))$ over BC^{P_i}.

- *Distributed Smudging Noise Generation:* Upon ready from $\mathcal{F}_{\mathsf{LSS}}$, each party P_i:

  (1) Samples $e_{\mathrm{sm},i} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ and sends $(\mathsf{input}, \overline{e_{\mathrm{sm},i}}, e_{\mathrm{sm},i})$ to $\mathcal{F}_{\mathsf{LSS}}$. //Once for each subsequent threshold decryption, if multiple circuits

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

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

Asynchronous step. Each party P_i:

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

Evaluation: To evaluate a circuit C, each party P_i:

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

Threshold Decryption: Each party P_i:

(3) Given labels $(\overline{\mathsf{sk}}, \overline{e_{\mathrm{sm}}})$, and a ciphertext c, sends $\big(\mathsf{LCOpen}, \Lambda_{Dec+sm}^{\mathsf{c}}(\overline{\mathsf{sk}}, \overline{e_{sm}})\big)$ to $\mathcal{F}_{\mathsf{LSS}}$.

- Upon receiving $(\Lambda_{\mathsf{Dec}+sm} c^{\mathsf{c}},\ \mu)$ from $\mathcal{F}_{\mathsf{LSS}}$, outputs $m := \Omega_{\mathsf{Dec}}(\mu)$.

Figure 1: Protocol $\Pi_{\mathsf{MPC}}^{\mathcal{F}_{\mathsf{LSS}}}$ instantiated from ℓ-BFV

C.1 Proof of indistinguishability with a real execution

We go through a series of hybrid games, starting from the real execution $\mathrm{REAL}_{\Pi_\mathsf{C}}$. The view of Env consists of its interactions with $\mathcal{A}/\mathsf{Sim}$, and of the outputs of the actual honest parties. We deal with the latter once and for all in Lemma 43.

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

Lemma 43. *The outputs of the actual honest parties are the same in* $\mathrm{REAL}_{\Pi_\mathsf{C}}$ *and* $\mathrm{IDEAL}_{\mathscr{F}_\mathsf{C},\mathsf{Sim},\mathsf{Env}}$. *Also, the views of* Env *in* $\mathrm{REAL}_{\Pi_\mathsf{C}}$ *and* Hybrid_1 *are computationally indistinguishable.*

Proof. It is convenient to prove the two claims at once. The view of Env is identical in $\mathrm{REAL}_{\Pi_\mathsf{C}}$ and Hybrid_1 until (3) included. There, for all $\ell \in S_\mathsf{c}$, consecutively to an instance of Share&Shrink, $\mathcal{F}_\mathsf{LSS}$ outputs a fresh encryption c_ℓ of m_ℓ under $\mathsf{ek} = (b, a)$, following the terminology of Section 3.6.2. Thus, the evaluated $\mathsf{c} := \mathsf{Eval}(\mathsf{C}, \{\mathsf{c}_j\}_{j\in S_\mathsf{c}}, \mathbf{rlk}, \mathbf{b})$ is the same in both views. In the threshold decryption of $\mathrm{REAL}_{\Pi_\mathsf{C}}$, the output of $\mathcal{F}_\mathsf{LSS}$ is:

$$\mu = \mathsf{c}[0] + \mathsf{c}[1] \cdot \Sigma_{j\in S}\mathsf{sk}_j + \Sigma_{j\in S} e_{\mathrm{sm},j}\,, \tag{4}$$

with $e_{\mathrm{sm},j} \xleftarrow{\$} [-B_{sm}, B_{sm}]$ for all $j \in S$. First, by Definition 32, we have, for some noise $e^{(\mathsf{Dec})}$, with $\|e^{(\mathsf{Dec})}\| \leq B_\mathsf{C}$

$$\mathsf{c}[0] + \mathsf{c}[1] \cdot \Sigma_{j\in S}\mathsf{sk}_j = \Delta y + e^{(\mathsf{Dec})}\,. \tag{5}$$

Since $\|e_{\mathrm{sm},j}\| \leq B_{sm}$ for all $j \in S$, it follows from the choice of parameters (3.19) and the final remark in Definition 32, that the output of honest parties in $\mathrm{REAL}_{\Pi_\mathsf{C}}$ is $m := \Omega_{\mathsf{Dec}}(\mu) = y$, which proves our first claim. Second, since we specified $\|e^{(\mathsf{Dec})}\|/n.B_{sm} = negl(\lambda)$ (equation (3.23)), it follows that the distribution of μ, given by (4) is computationally indistinguishable from the one of $\Delta y + \Sigma_{j\in S} e_{\mathrm{sm},j}$, see the "smudging" Lemma 27 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 parties 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 28.

Hybrid_3 ***[Bogus Honest Inputs]*** This is the same as Hybrid_2 except that the input and randomness distribution on behalf of honest owners are computed with $\widetilde{m_\ell} := 0$, instead of with their actual inputs m_ℓ. 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 $y := \mathsf{C}((m_\ell)_{\ell\in S_\mathsf{c}})$ is still the evaluation of the circuit on the *actual* inputs.

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

Lemma 44. Hybrid_2 *and* Hybrid_3 *are computationally indistinguishable.*

Proof. Since Hybrid_2, the secret keys of the honest parties ($\{P_i\}_{i\in\mathcal{H}}$) are no longer used in any computation. Furthermore, since honest parties sample their contributions $\mathbf{b}_i$ to the common threshold 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 26 "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^l, then the adversary can add to it the sum $(\mathbf{b}', \mathbf{a})$ of t 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. □

D On Reusability and Comparison with [GJPR21]

In the MPC protocols we present in Chapter 4, we point out that they verify the *reusability* property of [BJMS20], defined as follows:

- **Reusability** (from [BJMS20]): Given the transcript of the input distribution phase of the protocol, the computation phase of the protocol should be able to be reused across an unbounded polynomial number of executions to compute different functions on the same fixed joint inputs of all the parties.

The latter is to be compared with the weaker *delayed-function* property used until now, e.g. in [ACGJ18], which roughly states that the first round messages of the honest parties are computed independent of the function and the number of parties.

Let us note that the concurrent work of [GJPR21] presented a result close to our Theorem 36, discovered independently and posted a few months before our work[1], which affirms the feasibility of MPC with GOD under honest majority with a bulletin board PKI. However, we believe that our result is stronger for several reasons. Notably, we identified a miscitation [GLS15] for their feasibility result (p10). Upon notification, they confirmed that their result was derived from [ACGJ18], which carries several important implications:

1. First, their MPC protocol does not have the *reusability* property (as observed by [BJMS20])
2. Second, the communication complexity of their protocol is, as observed in [BJMS20], linear in the circuit size, when it is only proportional to the circuit depth and the number of inputs in our Theorem 36.
3. Third, broadcast is assumed in both rounds in [GJPR21, Section 6], while our Theorem 36 requires only an asynchronous 2^{nd} round.
4. Finally, we believe that their protocol, in its current form, is not delegable. Indeed, the computation of both round 1 and round 2 messages by a party P_i requires the party's own input x_i. As a result, the only way an external input-owner could delegate a computation would be to reveal its input in clear text to one of the parties, which compromises security. However, we do not rule out the possibility of a more sophisticated solution that enables

[1] 28/05/2021 vs 11/2021

delegation, similar to approaches like [BJMS20] or [BGG+18], albeit with the additional overhead such methods entail.

ECOLE
DOCTORALE

Titre: Délégation efficace de calcul multipartite sécurisé

Mots clés: Cryptographie, MPC, Chiffrememt Homomorphe

Résumé: Avec l'essor du cloud, il est devenu plus simple de déléguer la gestion et l'analyse des données à des infrastructures externes, favorisant la combinaison de données variées pour en tirer des informations utiles. Toutefois, garantir la confidentialité des données sensibles reste un obstacle majeur. Le calcul sécurisé multipartite (MPC) répond à ce défi en permettant à plusieurs participants de collaborer pour effectuer des calculs sur leurs données sans révéler celles-ci. Cette thèse explore une approche où les propriétaires des données délèguent ces calculs à des serveurs non fiables, tout en préservant sécurité et confidentialité. Pour cela, nous nous appuyons sur le chiffrement complètement homomorphe (FHE), qui permet de calculer directement sur des données chiffrées. Nos contributions incluent un protocole robuste de MPC basé sur le FHE et une méthode générique réduisant les besoins en communication. Ces avancées rendent les calculs sécurisés plus efficaces et accessibles, même pour des projets impliquant de nombreux participants.

Title: Efficient Secure Delegated Computation

Keywords: Cryptography, MPC, Fully Homomorphic Encryption

Abstract: With the rise of cloud computing, it has become easier to delegate the management and analysis of data to external infrastructures, enabling the combination of diverse datasets to extract valuable insights. However, ensuring the confidentiality of sensitive data remains a significant challenge. Secure multiparty computation (MPC) addresses this issue by allowing multiple participants to collaborate on computations without revealing their private data. This thesis explores an approach where data owners delegate these computations to untrusted servers while maintaining security and confidentiality. To achieve this, we rely on fully homomorphic encryption (FHE), which allows computations to be performed directly on encrypted data. Our contributions include a robust MPC protocol based on FHE and a generic method that minimizes communication requirements. These advancements make secure computations more efficient and accessible, even for projects involving a large number of participants.

Institut Polytechnique de Paris
91120 Palaiseau, France