Adaptively Secure Multi-Party Computation from LWE (via Equivocal FHE)

Ivan Damgård[1], Antigoni Polychroniadou[1], and Vanishree Rao[2]

[1] Department of Computer Science, Aarhus University
[2] PARC, a Xerox Company

Abstract. Adaptively secure Multi-Party Computation (MPC) is an essential and fundamental notion in cryptography. In this work, we construct Universally Composable (UC) MPC protocols that are adaptively secure against all-but-one corruptions based on LWE. Our protocols have a constant number of rounds and communication complexity dependant only on the length of the inputs and outputs (it is independent of the circuit size).

Such protocols were only known assuming an honest majority. Protocols in the dishonest majority setting, such as the work of Ishai et al. (CRYPTO 2008), require communication complexity proportional to the circuit size. In addition, constant-round adaptively secure protocols assuming dishonest majority are known to be impossible in the stand-alone setting with black-box proofs of security in the plain model. Here, we solve the problem in the UC setting using a set-up assumption which was shown necessary in order to achieve dishonest majority.

The problem of constructing adaptively secure constant-round MPC protocols against arbitrary corruptions is considered a notorious hard problem. A recent line of works based on indistinguishability obfuscation construct such protocols with near-optimal number of rounds against arbitrary corruptions. However, based on standard assumptions, adaptively secure protocols secure against even just all-but-one corruptions with near-optimal number of rounds are not known. However, in this work we provide a three-round solution based only on LWE and NIZK secure against all-but-one corruptions.

In addition, Asharov et al. (EUROCRYPT 2012) and more recently Mukherjee and Wichs (ePrint 2015) presented constant-round protocols based on LWE which are secure *only* in the presence of static adversaries. Assuming NIZK and LWE their static protocols run in two rounds where the latter one is only based on a common random string. Assuming adaptively secure UC NIZK, proposed by Groth et al. (ACM 2012), and LWE as mentioned above our adaptive protocols run in three rounds.

Our protocols are constructed based on a special type of cryptosystem we call equivocal FHE from LWE. We also build adaptively secure UC commitments and UC zero-knowledge proofs (of knowledge) from LWE. Moreover, in the decryption phase using an AMD code mechanism we avoid the use of ZK and achieve communication complexity that does not scale with the decryption circuit.

1 Introduction

Secure multi-party computation is an extremely strong and important tool for making distributed computing more secure. General solutions to the problem allows us to carry out any desired computation among a set of players, without compromising, the privacy of their inputs or the correctness of the outputs. This should even hold if some of the players have been corrupted by an adversary. An important issue in this connection is how the adversary chooses which players to target. In the static model, the adversary must choose who to corrupt before the protocol starts. A more general and also more realistic model is adaptive corruption where the adversary may corrupt new players during the protocol.

Of course efficiency of the protocol is also important, and important measures in this respect are the number of rounds we need to do, as well as the communication complexity (the total number of bits sent). Obviously, achieving a constant number of rounds and small communication complexity, while still getting the best possible security, is an important research goal.

Unconditionally secure protocols such as [BGW88] are typically adaptively secure. But these protocols are not constant round, and it is a major open problem if it is even possible to have unconditional security and constant number of rounds for secure computation of any function, see [DNP15] for a detailed discussion.

If we are willing to make computational assumptions, we can achieve constant round protocols, the first example of this is Yao's garbled circuits for two players, but on the other hand this does not give us adaptive security. Another class of protocols based on Fully Homomorphic Encryption (FHE) also naturally leads to constant round protocols, where we can tolerate that a majority of players are corrupted. Here we also get low communication complexity, that depends only on the lenght of inputs and outputs. But again, these protocols achieve only static security (see for instance [Gen09,AJLA$^+$12,LTV12]). More recently, the work of Mukherjee and Wichs [MW15] achieve a two-round static protocol assuming LWE and NIZK where additionally the protocol only assumes a random reference string (as opposed to being sampled form a specific distribution).

We can in fact get adaptive security in the computational setting, as shown in [CFGN96] by introducing the notion of Non-Commiting Encryption (NCE). Moreover, in [DN03], adaptive security was obtained as well, but much more efficiently using additively homomorphic encryption. However, neither [CFGN96] nor [DN03] run in a constant number of rounds.

If we assume honest majority we can get both constant round and adaptive security but the communication complexity will be propositional to the size of the evaluated circuit. This was shown in several papers [DI05,DI06,DIK$^+$08,IPS08]. The idea here is to use an unconditionally secure protocol to compute, for instance, a Yao garbled circuit, that is then used to compute the desired function in a constant number of rounds. Since the computation leading to the Yao circuit is easy to parallelise, this can be constant round as well and we inherit adaptive security from the unconditionally secure "preprocessing". On the other hand, as mentioned this requires communication that is proportional to the size of circuit to be securely evaluated. One may apply the IPS compiler to one of these protocols to get a solution for dishonest majority. This preserves the adaptive security and the constant number of rounds, but unfortunately also preserves the dependence of the communication complexity on the circuit size. Therefore, the question becomes:

Is it possible to construct constant round MPC protocols secure against an adaptive adversary that may corrupt all but one parties with communication complexity independent of the circuit size?

1.1 Contributions

We answer this in the affirmative. More specifically, we achieve an adaptive UC-secure protocol that tolerates corruption of $n-1$ of the n players with UC secure composition with protocols secure against $n-1$ corruptions. Our protocol requires a constant number of rounds and its communication complexity depends only on the length of inputs and outputs (and the security parameter), and not on the size of the evaluated circuit and the decryption circuit. The protocol is secure if the LWE problem is hard. Moreover, we *do not* consider the weaker model of secure erasures.

Theorem 1 (informal). *Assuming hardness of LWE, we show that arbitrary functions can be UC-securely computed in the presence of adaptive, active corruption of all-but-one parties within a constant number of rounds.*

Assuming adaptively secure UC NIZK, proposed by Groth et al. [GOS12], and LWE our adaptive protocols run in three rounds.

Theorem 2 (informal). *Assuming hardness of LWE and the existence of adaptively secure UC NIZK, we show that arbitrary functions can be UC-securely computed in the presence of adaptive, active corruption of all-but-one parties in three rounds of broadcast.*

In our construction we assume a broadcast channel where encryption is performed using what we call Equivocal FHE, a notion weaker than non-commiting encryption, presented in Section 3 which can be of independent interest. For example, using our equivocal scheme we also build adaptively secure UC commitment and UC zero-knowledge proofs (of knowledge) based on hardness of LWE (see Section 4).

Last but not least, in the standard ZK-based decryption used by approaches based on FHE, all the parties need to append a ZK proof , to prove that they decrypted correctly, whose communication complexity grows with the size of the decryption circuit. In this work using an AMD code mechanism [CDF$^+$08] we avoid the use of ZK and achieve communication complexity that does not scale with the decryption circuit. In particular, the total communication complexity of the decryption phase of our concrete protocol is $\mathcal{O}(n^2\lambda)$ where λ is the security parameter.

1.2 Technical Difficulties and New Ideas

To construct our adaptively secure protocol, we start from the well known blue-print for FHE-based MPC: players encrypt their inputs under a common public key, evaluate the desired function locally and then jointly decrypt the result. This is possible under an appropriate set-up assumption, which is always needed for UC security and dishonest majority. Namely, we assume that a public key has been distributed, and players have been given shares of the corresponding secret key.

This approach has been used before and usually leads to static security. One reason for this is that encryptions are usually committing, so we are in trouble if the sender of a ciphertext is corrupted later. This can be solved using a cryptosystem with equivocal properties and this would mean that the input and the evaluation phase of the protocol can be simulated, even for adaptive corruptions. Players need, of course, to prove that they know the inputs they contribute, but this can be done once we construct constant round adaptively secure UC commitment and ZK proofs from LWE.

An important tool we would like to get in order to achieve constant-round adaptively secure MPC protocols may be a Fully Homomorphic Encryption (FHE) scheme with equivocal properties.

Starting point – Fully Homomorphic NCE. It is tempting to consider a generic solution from FHE and Non-Commiting Encryption (NCE). In particular, in such a hypothetical construction, the secret key would be a secret key for an FHE scheme, the public key an FHE encryption of the NCE secret key and the NCE public key. Encryption would be performed using the NCE, and homomorphic evaluation and decryption would be performed as expected. However, there are fundamental caveats with this approach.

It does not seem to buy us any efficiency at all. In particular, NCE schemes are interactive, in that the receiver must send fresh (public-)key material for each new message to be encrypted. There is even a result by Nielsen saying that this is inherent for NCE [Nie02]. It will be hard for an interactive scheme to fit the above suggestion. Indeed, the public key material would run out after encrypting some number of inputs. Therefore, in generic NCE the public-key cannot be reused, and has to be updated for each new message. Moreover, one may go around this issue by having an NCE public-key for each party where the FHE encryption in the public key will include all the public keys. However, such a solution is highly inefficient since it is not the number of parties that matter but the amount of data to be encrypted. The amount of public-key material has to be proportional to size of the plaintext data. For instance, if only a constant number of parties had input, but a lot of, we would have a significant problem.

Another suggestion is to always regenerate this setup afresh using a constant round adaptive protocol prior to each new execution. This might work but unfortunately set-up data are considered reasonable if its size does not depend on the function to be computed (otherwise we are in the preprocessing model which is a completely different ball game). Hence, one would in fact always need this key regeneration step per execution.

It turns out that the motivation of considering NCE in this context is very weak.

Our approach – Starting afresh. Towards minimising the above caveat we propose a scheme we call Equivocal FHE. An equivocal FHE scheme is a fully homomorphic encryption scheme with additional properties. Most importantly, it is possible to generate "fake" public keys that look like normal keys but where encryption leads to ciphertexts that contain no information on the plaintext. This is similar to the known notion of meaningful/meaningless keys, but in addition we want that fake public keys come with a trapdoor that allows to "explain" (equivocate) a ciphertext as an encryption of any desired plaintext. This is similar to (but not the same as) what is required for NCE: for NCE one needs to equivocate a ciphertext even if the decryption key is also given (say, by corrupting the receiver), here we only need to give the adversary valid looking randomness for the encryption. In order to achieve such a cryptosystem the main properties we require from an FHE scheme is formula privacy, invertible sampling and homomorphishm over the randomness. Given this, we managed to obtain the required equivocation directly with much less overhead compared to a possible NCE solution.

We give a concrete instantiation of equivocal FHE based on LWE, starting from the FHE scheme by Brakerski et al. [BV11b].

Adaptive UC commitments and ZK from LWE. A second tool we need is constant-round UC-secure commitments and zero-knowledge proofs. For the commitments we start from a basic construction appeared in [CLOS02], which was originally based on claw-free trapdoor permutations (CFTP). We show that it can be instantiated based on LWE (which is not known to imply CFTP). Zero-knowledge then follows quite easily from known techniques.

Achieving a simulatable protocol. A harder problem is how to simulate the output phase in which ciphertexts containing the outputs are decrypted. In the simulation we cannot expect that these ciphertexts are correctly formed and hold the actual outputs, so the simulator needs to "cheat". However, each player holds a share of the secret key which we have to give to the adversary if he is corrupted. If this happens after some executions of the decryption protocol, we (the simulator) may already be committed to this share. It is therefore not clear how the simulator can achieve the desired decryption results by adjusting the shares of the secret key. To get around this, we adapt an idea from Damgård and Nielsen [DN03], who proposed an adaptively secure protocol based on additively homomorphic threshold encryption but in the honest majority scenario. The idea is to add a step to the protocol where each ciphertext is re-randomised just before decryption. This gives the simulator a chance to cheat and turn the ciphertext into one that contains the correct result, and one can therefore simulate the decryption without having to modify the shares of the secret key. The re-randomisation from [DN03] only works for honest majority, we show a different method that works for dishonest majority and augment our Equivocal FHE scheme with the *ciphertext randomisation* property to achieve our goal.

General purpose Equivocal FHE. We mention for completeness that there is also a more generic approach which will give us adaptive security based only on our Equivocal FHE: namely, we follow the same blueprint as before, with input, evaluation and output phases. However, we implement the verification of ciphertexts in the input phase and the decryptions in the output phase using generic adaptively secure MPC a la [CLOS02,IPS08]. This way, the communication and the number of rounds do not depend on the size of circuit to be computed securely. However, it would not be genuinely constant round, and the communication complexity would depend on the circuits computing the encryption and decryption functions of the underlying cryptosystem. Hence, unlike our protocol, any such solution would have communication complexity proportional to the Boolean circuit complexity of the decryption function (which seems inherent since one needs Yao garbling underneath). We measure the round and communication complexity of such a possible solution based on the IPS compiler. The bottom line is that using IPS generically would yield a larger (constant) number of rounds (20-30 rounds) and worse dependence on the security parameter. A concise estimate can be found in Appendix D. Clearly the above estimate should be taken with large grains of salt. We have tried to be optimistic on the part of IPS, to not give our concrete protocol an unfair advantage. Thus, actual numbers could be larger. On the other hand, we propose a three-round solution.

AMD code solution to replace ZK. However, contrary to the above generic IPS solution, our approach allows for significant optimization of the decryption as follows. Instead of using ZK proofs to prove that the player's evaluation shares to the decryption phase are correct, we change the evaluation phase of the protocol. In particular, instead of having ciphertexts containing the desired output z, the evaluation phase computes encryptions containing a codeword $c = (z, \alpha)$ in an algebraic manipulation detection code, where z is the data and α is the key/randomness. In the decryption stage, players commit to their decryption shares (recall that we have UC commitment available), and then all shares are opened. If decryption fails, or decoding the codeword fails, we abort, else we output the decoded z. If z and α are thought of as elements in a (large) finite field, then the codeword can just be $(z, \alpha, \alpha z)$. According to our optimization, the communication complexity of our protocol is not only independent of the the size of the evaluated circuit but also independent of the circuit size of the decryption circuit.

Impossibility results? In the following we mention two impossibility results which apply to adaptively secure MPC and mention why they do not apply in our setting.

Motivated by ruling out one possible approach to achieving adaptive security, Katz et al. [KTZ13] showed that FHE with security against adaptive corruption of the receiver is impossible. In our setting, we distribute the private key of an FHE scheme among n parties; since we allow only $n-1$ of the parties to be corrupted, the impossibility result from [KTZ13] does not apply. Note that if an FHE scheme is to be of use in MPC, it seems to be necessary that the players are able to decrypt, if not by themselves, then at least by collaborating. But if corruption of all n players was allowed, the adversary would necessarily learn all secret keys, and then the impossibility result from [KTZ13] would apply. This suggests that our result with $n-1$ corruptions is the best we can achieve based only on FHE.

We note that in [GS12], adaptive security in constant number of rounds in the plain model was obtained using a non-blackbox proof in the stand-alone setting. Also a solution with a blackbox proof was shown to be impossible, but this does not, of course, apply to our case, where we go for UC security, and therefore require a set-up assumption.

Security against arbitrary corruptions: Round complexity of all known adaptively secure protocols secure against n corruptions grows (see, e.g. [CLOS02], [KO04,GS12,DMRV13]) linearly in the depth of the evaluated circuit. Recent independent works [GP15,CGP15,DKR15], have been shown that MPC protocols with security against n corruptions in a constant number of rounds can be achieved using indistinguishability obfuscation (IO) [GGH+13].

While the above results on constant round MPC using IO are exciting, the focus of this work is to avoid indistinguishability obfuscation altogether and to achieve adaptive security against corruption of $n-1$ of the n players, (with communication complexity depended only on the length of inputs and outputs and not on the size of the circuit to be computed securely), using simpler tools with simple standard assumptions involving them. In particular, our construction only requires FHE based on the hardness of LWE and avoids the use of IO which also incurs a cost in efficiency. Also as we have already mentioned, our result with $n-1$ corruptions is the best we can achieve based only on FHE.

Roadmap. In section 3 we define our *Equivocal fully homomorphic encryption* scheme and its properties. A concrete instantiation based on the scheme of [BV11b] is given in Appendix E. In Section 4 we give our construction for UC commitments and ZKPoK. Next in Section 5, we proceed by presenting our MPC protocol. The simulator and the security proof of our protocol can be found in Appendix C. In Section 6 we show how AMD codes can be used in order to avoid the use of ZK.

2 Notation

Throughout the paper $\lambda \in \mathbb{N}$ will denote the security parameter. We use $d \leftarrow \mathcal{D}$ to denote the process of sampling d from the distribution $\mathcal{D}$ or, if $\mathcal{D}$ is a set, a uniform choice from it. We say that a function $f : \mathbb{N} \to \mathbb{R}$ is negligible if $\forall c\ \exists n_c$ s.t. if $n > n_c$ then $f(n) < n^{-c}$. We will use $\mathrm{negl}(\cdot)$ to denote an unspecified negligible function. We often use $[n]$ to denote the set $\{1, ..., n\}$. We write $\boxplus$ and $\boxdot$ to denote operations over encrypted data including multiplication of a ciphertext with a non encrypted string. If $\mathcal{D}_1$ and $\mathcal{D}_2$ are two distributions, then we denote that they are statistically close by $\mathcal{D}_1 \approx_{\mathrm{s}} \mathcal{D}_2$; we denote that they are computationally indistinguishable by $\mathcal{D}_1 \approx_{\mathrm{c}} \mathcal{D}_2$; and we denote that they are identical by $\mathcal{D}_1 \equiv \mathcal{D}_2$. For a randomized algorithm A, we use $a \leftarrow A(x; r)$ to denote running A on input x and uniformly random bits $r \in \{0,1\}^*$, producing output a.

Invertible Sampling [OPW11]: We recall the notion of invertible sampling, which is closely connected to adaptive security in simulation models where erasures are not allowed. We say that an algorithm A with input space X has invertible sampling if there exists a PPT inverting algorithm, denoted by Inv_A, such that for all input $x \in X$, the outputs of the following two experiments are either computationally, or statistically indistinguishable:

$$\begin{array}{l|l} y \leftarrow A(x, r) & y \leftarrow A(x, r) \\ & r' \leftarrow \mathsf{Inv}_A(y, x) \\ \text{Return } (x, y, r) & \text{Return } (x, y, r') \end{array}$$

3 Equivocal Fully Homomorphic Encryption Scheme

We start by recalling the notions of (fully) homomorphic encryption. Next we define the new notion of Equivocal FHE and we specify the properties needed for such an instantiation. We give a concrete instantiation of our Equicocal FHE scheme from the LWE assumption, based on Brakerski and Vaikutanathan [BV11b] FHE scheme, described in Appendix E.

3.1 Homomorphic Encryption

A homomorphic encryption scheme $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ is a quadruple of PPT algorithms. In this work, the message space M of the encryption schemes will be some (modulo 2) ring, and the functions to be evaluated will be represented as arithmetic circuits over this ring, composed of addition and multiplication gates. The syntax of these algorithms is given as follows.

- *K*ey-Generation. The algorithm KeyGen, on input the security parameter 1^λ, outputs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, where pk is a public encryption key and sk is a secret decryption key.
- *E*ncryption. The algorithm Enc, on input pk and a message $m \in M$, outputs a ciphertext $\mathrm{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$.
- *D*ecryption. The algorithm Dec on input sk and a ciphertext ct, outputs a message $\tilde{m} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\mathrm{ct})$.
- *H*omomorphic-Evaluation. The algorithm Eval, on input pk, an arithmetic circuit ckt, and a tuple of ℓ ciphertexts $(\mathrm{ct}_1, \ldots, \mathrm{ct}_\ell)$, outputs a ciphertext $\mathrm{ct}' \leftarrow \mathsf{Eval}_{\mathsf{pk}}\big(\mathrm{ckt}(\mathrm{ct}_1, \ldots, \mathrm{ct}_\ell)\big)$.

We note that we can treat the evaluation key as a part of the public key. The security notion needed in this work is security against chosen plaintext attacks (IND-CPA security), defined as follows.

Definition 1 (IND-CPA security). *A scheme* HE *is IND-CPA secure if for any PPT adversary* $\mathcal{A}$ *it holds that:*

$$\mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{HE}}[\lambda] := |Pr[\mathcal{A}(\mathsf{pk}, \mathsf{Enc}_{\mathsf{pk}}(0)) = 1] - Pr[\mathcal{A}(\mathsf{pk}, \mathsf{Enc}_{\mathsf{pk}}(1)) = 1]| = \mathrm{negl}(\lambda),$$

where, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

3.2 Fully Homomorphic Encryption

A scheme HE is fully homomorphic if it is both compact and homomorphic with respect to a class of circuits. More formally:

Definition 2 (Fully homomorphic encryption). *A homomorphic encryption scheme* $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* fully homomorphic *if it satisfies the following properties:*

1. Homomorphism: *Let* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *be the set of all polynomial sized arithmetic circuits.* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*,* $\forall \mathrm{ckt} \in \mathcal{C}_\lambda$*,* $\forall (m_1, \ldots, m_\ell) \in M^\ell$ *where* $\ell = \ell(\lambda)$*,* $\forall (\mathrm{ct}_1, \ldots, \mathrm{ct}_\ell)$ *where* $\mathrm{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_i)$*, it holds that:*
$$Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{pk}}(\mathrm{ckt}, \mathrm{ct}_1, \ldots, \mathrm{ct}_\ell)) \neq \mathrm{ckt}(m_1, \ldots, m_\ell)] = \mathrm{negl}(\lambda)$$
2. Compactness: *There exists a polynomial* $\mu = \mu(\lambda)$ *such that the output length of* Eval *is at most* μ *bits long regardless of the input circuit* ckt *and the number of its inputs.*

3.3 Equivocal Fully Homomorphic Encryption Scheme

Our *Equivocal fully homomorphic encryption scheme* consists of a tuple $(\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Rand}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{Equiv})$ of algorithms where the syntax of the procedures $(\mathsf{KeyGen}, \mathsf{QEnc}, \mathsf{Eval}, \mathsf{Dec})$ is defined as in the above FHE scheme. Our scheme is augmented with two algorithms $(\mathsf{KeyGen}^*, \mathsf{Equiv})$ used for equivocation. Jumping ahead, in this paper we are interested in building adaptively secure n-party protocols generically using an equivocal QFHE scheme and gain in terms of round and communication efficiency. Two extra properties needed for the MPC purpose, are distributed decryption and ciphertext randomisation where the latter one guarantees simulatable decryption [3]. If the purpose of our Equivocal scheme is not MPC then these properties are not required, see Section 4 for QFHE based UC commitment schemes. In the sequel, we will use blue color to stress whether a part is relevant to the ciphertext randomisation property.

[3] Ciphertext randomisation is needed in order to force the output in the simulation.

Definition 3 (Equivocal fully homomorphic encryption). *An Equivocal fully homomorphic encryption scheme* $\mathsf{QFHE} = (\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Rand},$ $\mathsf{Eval}, \mathsf{Dec}, \mathsf{Equiv})$ *with message space M is made up of the following PPT algorithms:*

- $(\mathsf{KeyGen}, \mathsf{QEnc}, \mathsf{Eval}, \mathsf{Dec})$ *is an FHE scheme with the same syntax as in section 3.1.*
- *The* Equivocal *key generation algorithm* $\mathsf{KeyGen}^*(1^\lambda)$*, outputs an* equivocal *public-key secret-key pair* $(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}})$.
- *The* Equivocation *algorithm* $\mathsf{Equiv}(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, \mathrm{ct}, r_{\mathrm{ct}}, m)$*, given* $\widetilde{\mathsf{PK}}$*,* $\widetilde{\mathsf{SK}}$*, a plaintext m, a ciphertext* ct *and random coins r_{ct}, outputs a value e in the randomness space.*
- *The* Ciphertext Randomisation *algorithm* $\mathsf{Rand}(\mathrm{ct}, \mathrm{ct}'_1, \ldots, \mathrm{ct}'_n)$*, given ciphertexts* $\mathrm{ct}, \mathrm{ct}'_1, \ldots, \mathrm{ct}'_n$ *generated by the procedure* QEnc *outputs a ciphertext* CT.
  We require the following properties:
  1. Indistinguishability of equivocal keys. *We say that the scheme has indistinguishability of equivocal keys if the distributions of* PK *and* $\widetilde{\mathsf{PK}}$ *are computationally indistinguishable, where* $(\mathsf{PK}, \cdot) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and* $(\widetilde{\mathsf{PK}}, \cdot) \leftarrow \mathsf{KeyGen}^*(1^\lambda)$.
  2. Indistinguishability of equivocation. *Let* $\mathcal{D}_{rand}(1^\lambda)$ *denote the distribution of randomness used by* QEnc. *Let* $\mathcal{O}(\widetilde{\mathsf{PK}}, m)$ *and* $\mathcal{O}'(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, m)$ *be the following oracles:*

Let $\mathcal{O}(\widetilde{\mathsf{PK}}, m)$:	Let $\mathcal{O}'(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, m)$:
$r_{\mathrm{ct}} \leftarrow \mathcal{D}_{rand}(1^\lambda)$	$r_{\mathrm{ct}} \leftarrow \mathcal{D}_{rand}(1^\lambda)$
$\mathrm{ct} = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}},(m; r_{\mathrm{ct}})$	$\mathrm{ct} = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(\widetilde{m}; r_{\mathrm{ct}})$
	$e = \mathsf{Equiv}(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, \mathrm{ct}, r_{\mathrm{ct}}, m)$
Output $(\widetilde{\mathsf{PK}}, \mathrm{ct}, r_{\mathrm{ct}})$	*Output* $(\widetilde{\mathsf{PK}}, \mathrm{ct}, e)$

  There exists $\widetilde{m} \in M$ *such that for any PPT adversary $\mathcal{A}$ with oracle access to* $\mathcal{O}(\widetilde{\mathsf{PK}}, \cdot)$ *and* $\mathcal{O}'(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, \cdot)$ *the following holds.*

$$\left| \Pr\left[\begin{matrix} (\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}) \leftarrow \mathsf{KeyGen}^*(1^\lambda) \\ 1 \leftarrow \mathcal{A}^{\mathcal{O}(\widetilde{\mathsf{PK}}, \cdot)} \end{matrix} \right] - \Pr\left[\begin{matrix} (\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}) \leftarrow \mathsf{KeyGen}^*(1^\lambda) \\ 1 \leftarrow \mathcal{A}^{\mathcal{O}'(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, \cdot)} \end{matrix} \right] \right| \leq \mathrm{negl}(\lambda)$$

  3. Ciphertext Randomisation. *Let* PK *be the public key used in the procedure* QEnc *for generating ciphertexts* $\mathrm{ct}, \mathrm{ct}'_1 \ldots \mathrm{ct}'_n$ *from the plaintexts* $m, m'_1, \ldots, m'_n \in M$*, respectevely. If* $Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathrm{ct}) = m] = 1 - \mathrm{negl}(\lambda)$ *and for all* $i \in [n]$*,* $Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathrm{ct}'_i) = m'_i] = 1 - \mathrm{negl}(\lambda)$ *then it holds that*

$$Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Rand}(\mathrm{ct}, \mathrm{ct}'_1 \ldots \mathrm{ct}'_n)) = m] = 1 - \mathrm{negl}(\lambda).$$

  On the other hand, let $\widetilde{\mathsf{PK}}$ *be the public key used in the procedure* QEnc *for generating ciphertexts* $\mathrm{ct}, \mathrm{ct}'_1 \ldots \mathrm{ct}'_n$*, respectevely. If* $Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathrm{ct}) = m] = 1 - \mathrm{negl}(\lambda)$ *and for all* $i \in [n]$*,* $Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathrm{ct}'_i) = m'_i] = 1 - \mathrm{negl}(\lambda)$ *then it holds that*

$$Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Rand}(\mathrm{ct}, \mathrm{ct}'_1 \ldots \mathrm{ct}'_n)) = m'_1 + \ldots + m'_n] = 1 - \mathrm{negl}(\lambda).$$

In the sequel for simplicity of exposition, we call the ciphertexts $\mathrm{ct}'_1 \ldots \mathrm{ct}'_n$ `redundant` in case they are generated by $\mathsf{QEnc}_{\mathsf{PK}}$ and `non − redundant` if they are generated by $\mathsf{QEnc}_{\widetilde{\mathsf{PK}}}$. Analogously, we call the ciphetext ct `non − redundant` or `redundant` if it is generated by $\mathsf{QEnc}_{\mathsf{PK}}$ or $\mathsf{QEnc}_{\widetilde{\mathsf{PK}}}$, respectively [4].

In order to construct our equivocal QFHE scheme we use the following *special* FHE scheme with some additional properties.

Definition 4. *[Special fully homomorphic encryption] We call a fully homomorphic encryption scheme* $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *a* special *FHE scheme, if it is IND-CPA secure and satisfies the following properties: Let* $\mathcal{D}_{rand}(1^\lambda)$ *denote the distribution of randomness used by* Enc.

[4] By the ciphertext randomisation property, the reader can think of the `redundant` messages as encryptions of zeros.

1. Additive homomorphism over random coins*: $\forall r_1, r_2 \in \mathsf{Supp}(\mathcal{D}_{rand}(1^\lambda))$ and $\forall m \in M$, it holds that* $\big(m \boxdot \mathsf{Enc}_{\mathsf{pk}}(0; r_1)\big) \boxplus \mathsf{Enc}_{\mathsf{pk}}(0; r_2) = \mathsf{Enc}_{\mathsf{pk}}(0; m \cdot r_1 + r_2)$.
2. E-Hiding*: There exists $\mathcal{D}'_{rand}(1^\lambda)$ such that $\forall m \in M$, if $r^{blind} \leftarrow \mathcal{D}_{rand}(1^\lambda)$ and $r^K \leftarrow \mathcal{D}'_{rand}(1^\lambda)$ then the distribution of $(r^{blind} - m \cdot r^K)$ is statistically close to $\mathcal{D}_{rand}(1^\lambda)$.* [5]
3. Invertible Sampling*: The distribution $\mathcal{D}_{rand}(1^\lambda)$, has invertible sampling via the algorithm $\mathsf{Inv}_{\mathcal{D}_{rand}}$.*

Recall that we defined an invertible sampler of an algorithm A in Section 2 as an algorithm Inv_A that takes as inputs the input x and output y with consistent random coins. In our case, $x = 1^\lambda$ and y is a sample from the range of $\mathcal{D}_{rand}$. Next, in Figure 1, we show how to build an equivocal FHE scheme using a special FHE scheme. The high level intuition is as follows. In order to achieve equivocality we modify an FHE scheme satisfying the properties of Definition 4 as follows: The public key contains an encryption of 1 and an encryption of 0. More specifically, $\mathsf{PK} = (\mathsf{pk}, K = \mathsf{Enc}_{\mathsf{pk}}(1), R = \mathsf{Enc}_{\mathsf{pk}}(0))$ where pk is the public key of an FHE scheme. An encryption of a message m in the real world is computed using K as $(m \boxdot K \boxplus \mathsf{Enc}_{\mathsf{pk}}(0))$ and encryption for re-randomisation is computed using R as $(z \boxdot R \boxplus \mathsf{Enc}_{\mathsf{pk}}(0))$ for a random value z. In the simulation, the values encrypted in K and R are switched, in particular, $K = \mathsf{Enc}_{\mathsf{pk}}(0)$ and $R = \mathsf{Enc}_{\mathsf{pk}}(1)$. Therefore, normal encryption leads to encryption of 0 with the guarantee of equivocation. However, encryption for re-randomisation actually encrypts non-zero values i.e., z, in order to force the output.

Theorem 3. *Let* FHE *be a* special *fully homomorphic encryption scheme. Then* $\mathsf{QFHE} = (\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Rand}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{Equiv})$ *in Figure 1 is an equivocal* QFHE *scheme.*

Proof. *Indistinguishability of equivocal keys.* Let $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}) \leftarrow \mathsf{KeyGen}^*(1^\lambda)$, then the indistinguishability of the two pairs of public keys follows from the IND-CPA security of the FHE scheme.

Indistinguishability of equivocation. Without loss of generality, we will show that indistinguishability of equivocation holds for $\widetilde{m} = 0$. Let $\mathcal{A}$ be an adversary that breaks indistinguishability of equivocation; then we construct a PPT algorithm R such that $R^{\mathcal{A}}$ breaks *E-hiding*. R simulates the oracle for every query m_i as follows. R invokes $\mathcal{A}$ and receives some message m_i and forwards it to the *E-hiding* challenger. Next it receives the challenge r_{ct_i} and computes $\mathrm{ct}_i = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0, m_i; r_{\mathrm{ct}_i})$ and forwards $(r_{\mathrm{ct}_i}, \mathrm{ct}_i)$ to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ does. Now, if $r_{\mathrm{ct}_i} \leftarrow \mathcal{D}_{rand}(1^\lambda)$ then $\mathrm{ct}_i \leftarrow \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0, m_i; r_{\mathrm{ct}_i})$, namely, the view of $\mathcal{A}$ follows the distribution which corresponds to the left game in Definition 3 of indistinguishability of equivocation. On the other hand, if $r_{\mathrm{ct}_i} = (r_i^{blind} - m_i \cdot r^{\widetilde{K}})$; then $\mathrm{ct}_i = (m_i \boxdot \widetilde{K}) \boxplus \mathsf{Enc}_{\mathsf{pk}}(0; r_i^{blind} - m_i \cdot r^{\widetilde{K}}) = \mathsf{Enc}_{\mathsf{pk}}(0; r_i^{blind}) = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0, 0; r_i^{blind})$ which implies that in this case the view of $\mathcal{A}$ follows the distribution of the right game in Definition 3 of indistinguishability of equivocation. This means that the distinguishing advantage of R is the same as that of $\mathcal{A}$ which leads to a contradiction.

Ciphertext Randomisation. The algorithm Rand adds the ciphertexts $(\mathrm{ct}, \mathrm{ct}'_1 \ldots \mathrm{ct}'_n)$. If ct is a ciphertext generated by $\mathsf{QEnc}_{\mathsf{PK}}$ for $b = 0$ and $(\mathrm{ct}'_1 \ldots \mathrm{ct}'_n)$ are ciphertexts generated by $\mathsf{QEnc}_{\mathsf{PK}}$ for $b = 1$ then

$$Pr[\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Rand}(\mathrm{ct}, \mathrm{ct}'_1 \ldots \mathrm{ct}'_n)) = m] = 1 - \mathrm{negl}(\lambda)$$

since it is easy to see that the ciphertexts $(\mathrm{ct}'_1 \ldots \mathrm{ct}'_n)$ contain encryptions of zeros due to the fact that $R = \mathsf{Enc}_{\mathsf{pk}}(0)$. An analogous argument holds for ct and $\mathrm{ct}'_1 \ldots \mathrm{ct}'_n$ generated by $\mathsf{QEnc}_{\widetilde{\mathsf{PK}}}$ for $b = 0$ and $b = 1$, respectively, since in this case the ciphertext ct contain an encryption of a zero (because in this case $\widetilde{K} = \mathsf{Enc}_{\mathsf{pk}}(0)$) and ciphertexts $(\mathrm{ct}'_1 \ldots \mathrm{ct}'_n)$ contain encryptions of the corresponding m'_i since $\widetilde{R} = \mathsf{Enc}_{\mathsf{pk}}(1)$.

□

[5] Intuitively, E-Hiding can be argued in the same way as formula privacy for some FHE schemes. This requires *dwarfing* in the sense that r^{blind} should be *large* enough to dwarf mr^K where $\mathcal{D}_{rand}(1^\lambda)$ and $\mathcal{D}'_{rand}(1^\lambda)$ are Gaussian distributions. Hence, $r^K \leftarrow \mathcal{D}'_{rand}(1^\lambda)$ and $r^{blind} \leftarrow \mathcal{D}_{rand}(1^\lambda)$ such that the noise of $\mathcal{D}_{rand}(1^\lambda)$ is super-polynomially larger than the noise of $\mathcal{D}'_{rand}(1^\lambda)$.

QFHE

Let FHE $=$ ($\mathsf{KeyGen}_{\mathsf{FHE}}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}$) be a *special* fully homomorphic encryption scheme. QFHE $=$ ($\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Eval}, \mathsf{Rand}, \mathsf{Dec}, \mathsf{Equiv}$) is defined as follows:

$\mathsf{KeyGen}(1^\lambda)$**:**

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$.
2. $K = \mathsf{Enc}_{\mathsf{pk}}(1; r^K)$ where $r^K \leftarrow \mathcal{D}'_{rand}(1^\lambda)$ and $R = \mathsf{Enc}_{\mathsf{pk}}(0; r^R)$ where $r^R \leftarrow \mathcal{D}'_{rand}(1^\lambda)$
3. Return as public key $\mathsf{PK} = (\mathsf{pk}, K, R)$ and secret key $\mathsf{SK} = \mathsf{sk}$.[a]

$\mathsf{KeyGen}^*(1^\lambda)$**:**

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$.
2. $\widetilde{K} = \mathsf{Enc}_{\mathsf{pk}}(0; r^{\widetilde{K}})$ where $r^{\widetilde{K}} \leftarrow \mathcal{D}'_{rand}(1^\lambda)$ and $\widetilde{R} = \mathsf{Enc}_{\mathsf{pk}}(1; r^{\widetilde{R}})$ where $r^{\widetilde{R}} \leftarrow \mathcal{D}'_{rand}(1^\lambda)$.
3. Return as public key $\widetilde{\mathsf{PK}} = (\mathsf{pk}, \widetilde{K}, \widetilde{R})$ and secret key $\widetilde{\mathsf{SK}} = (\mathsf{sk}, r^{\widetilde{K}}, r^{\widetilde{R}})$.

$\mathsf{QEnc}_{\mathsf{PK}}(b, m)$:

1. Compute $\mathrm{ct}^{\mathsf{blind}} = \mathsf{Enc}_{\mathsf{pk}}(0; r^{\mathsf{blind}})$ where $r^{\mathsf{blind}} \leftarrow \mathcal{D}_{rand}(1^\lambda)$.
2. If $b \notin \{0,1\}$ then output $\perp$.
3. If $b = 0$ then output $\mathrm{ct} = (m \boxdot K) \boxplus \mathrm{ct}^{\mathsf{blind}}$ *otherwise* output $\mathrm{ct} = (m \boxdot R) \boxplus \mathrm{ct}^{\mathsf{blind}}$.

$\mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(b, \widetilde{m})$:

1. Compute $\widetilde{\mathrm{ct}}^{\mathsf{blind}} = \mathsf{Enc}_{\mathsf{pk}}(0; \widetilde{r}^{\mathsf{blind}})$ where $\widetilde{r}^{\mathsf{blind}} \leftarrow \mathcal{D}_{rand}(1^\lambda)$.
2. If $b \notin \{0,1\}$ then output $\perp$.
3. If $b = 0$ then output $\widetilde{\mathrm{ct}} = (\widetilde{m} \boxdot \widetilde{K}) \boxplus \widetilde{\mathrm{ct}}^{\mathsf{blind}}$ *otherwise* output $\widetilde{\mathrm{ct}} = (\widetilde{m} \boxdot \widetilde{R}) \boxplus \widetilde{\mathrm{ct}}^{\mathsf{blind}}$.

$\mathsf{Equiv}(b, \widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, \widetilde{\mathrm{ct}}, \widetilde{r}^{\mathsf{blind}}, m, \widetilde{m})$**:**

1. If $b = 0$ compute $r^{\mathsf{blind}} := \widetilde{r}^{\mathsf{blind}} + (\widetilde{m} - m) \cdot r^{\widetilde{K}}$ *otherwise* $r^{\mathsf{blind}} := \widetilde{r}^{\mathsf{blind}} + (\widetilde{m} - m) \cdot r^{\widetilde{R}}$
2. Run $r_{state} \leftarrow \mathsf{Inv}_{\mathcal{D}_{rand}}(r^{\mathsf{blind}})$ and output r_{state}.

$\mathsf{Rand}(\mathrm{ct}, \mathrm{ct}'_1 \ldots, \mathrm{ct}'_n)$: Output $\mathrm{CT} = \mathrm{ct} \boxplus \mathrm{ct}'_1 \boxplus \ldots \boxplus \mathrm{ct}'_n$.

Procedures ($\mathsf{Eval}, \mathsf{Dec}$) are as defined in normal FHE schemes.

[a] Note that procedure Dec, given sk, runs as in normal FHE schemes (see Section 3.1), so there is no need to provide r^K in SK. We also enhance the notation of QEnc to include a bit b which indicates whether the encryption is performed using the key K or R, respectively. In addition, the plaintext $\widetilde{m}$ is usually set to zero.

Fig. 1. Instantiation of a QFHE scheme

Distributed Decryption: As we mentioned above, we need distributed decryption to implement our MPC protocol. To this end, we assume that the common public key pk has been set up where the secret key sk has been secret-shared among the players in such a way that they can collaborate to decrypt. Notice that some setup assumption is always required to show UC security in the dishonest majority setting. Roughly, we assume that a functionality is available which generates a key pair and secret-shares the secret key among the players using a secret-sharing scheme that is assumed to be given as part of the specification of the cryptosystem. Since we allow corruption of all but one player, the maximal unqualified sets must be all sets of $n-1$ players. We point out that we could make a weaker set-up assumption, such as a common reference string, and using a general UC secure multiparty computation protocol for the common reference string model to implement the above functionality. While this may not be very efficient, one only needs to run this protocol once in the life-time of the system. The properties needed for the distributed decryption and its protocol are specified later.

4 UC Adaptive Commitments and ZKPoK from LWE

Commitment schemes that satisfy both equivocality and extractability form useful tools in achieving adaptive security. In this section, we show how using a QFHE scheme, one can build equivocal and extractable commitments. Having realized a QFHE scheme based on the LWE assumption, we consequently get equivocal and extractable commitments assuming the hardness of LWE. Note that such commitments based on LWE can be of independent interest. We remark that any encryption scheme that satisfies the properties specified in Definition 4 would have sufficed for our purposes in this section – the multiplicative homomorphic property of our QFHE scheme will not be of use here; however, since we are using our commitment scheme as a tool in our adaptive MPC protocol based on LWE, we use the same QFHE scheme in our commitment scheme too.

Since we are interested in UC security against adaptive adversaries, our commitment scheme is in the CRS model. The scheme must satisfy the following two properties, polynomial equivocality and simulation extractability. The former guarantees that the simulator $\mathcal{S}$ needs to be able to produce polynomially many equivocal commitments using the same CRS. More specifically, $\mathcal{S}$ can open the equivocal commitments to any value of its choice and give consistent randomness to adversary $\mathcal{A}$. The latter property says that the simulator $\mathcal{S}$ needs to be able to extract the contents of any valid commitment generated by adversary $\mathcal{A}$, even after $\mathcal{A}$ obtains polynomially many equivocal commitments generated by $\mathcal{S}$. Note that there is only an apparent conflict between equivocality and the binding property and between the extractability and the hiding property, as the simulator is endowed with additional power (trapdoors) in comparison with the parties in the real world execution. In the following we elaborate how our commitment scheme satisfies the above properties.

Our construction. Equivocation in our scheme is achieved via QFHE. In particular, the commitment algorithm is the algorithm QEnc, defined in Figure 1. In order to add extractability we must enhance our scheme in such a way that we do not sacrifice equivocality. A failed attempt is to include a public key for an encryption scheme secure against CCA2 attacks in the CRS. In this case, the committer will send an encryption of the decommitment information along with the commitment itself. Then, as the simulator has the associated decryption key, it can decrypt the decommitment information and hence extract the committed value from any adversarially prepared commitment. However, notice that such an encryption is binding even to the simulator, so equivocality cannot be achieved.

The solution to the problem is to send the commitment along with two pseudorandom ciphertexts. One ciphertext is an encryption of the decommitment information and the other ciphertext is a uniformly random string. In this way, the simulator can encrypt both decommitment values and later show that it only knows the decryption to one and that the other was uniformly chosen.

For the security of our construction, the encryption scheme used to encrypt the decommitment information has to be a CCA2-secure encryption scheme with the property that any produced ciphertext is pseudorandom and has deterministic decryption. To this end, the CCA2 encryption scheme of Micciancio and Peikert [MP12] based on LWE satisfies the above properties. They obtain their result via relatively generic assumptions using either strongly unforgeable one-time signatures [DDN00], or a message authentication code and a weak form of commitment [BCHK07]. The first assumption does not yield pseudorandom ciphertexts, thus another encryption producing pseudorandom ciphertexts on top of the scheme of [MP12] could have been used, resulting in a double encryption scheme. However, it turns out that their construction with the latter set of assumptions has pseudorandom ciphertexts.

The reader might have observed that this bears some resemblance with the trick used in the seminal work of [CLOS02], referred to as CLOS hereafter, to achieve extractability. Their scheme is based on enhanced trapdoor permutations, also needed in order to get double encryption CCA2 security. Moreover, in order to build equivocal commitments they need an NP reduction to graph Hamiltonicity since the CRS of their commitment scheme consists of a graph G sampled from a distribution such that it is computationally hard to tell if G has a Hamiltonian cycle. Interestingly, the CLOS commitment scheme does not give an instantiation based on LWE and to begin with, there are no known trapdoor permutations based on LWE. On the other hand, assuming the hardness of LWE, we propose an extractable and equivocal commitment *with no need of an NP reduction*, leading to a huge improvement in efficiency.

More formally, given a $\mathsf{QFHE} = (\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{Equiv})$[6] scheme, a CCA2-secure scheme $\mathsf{E}_{\mathsf{CCA}}$ with encryption algorithm $\mathsf{ENC}_{\mathsf{CCA}}$ based on LWE [MP12], with the property that any ciphertext is pseudorandom and has deterministic decryption, we construct the following equivocal and extractable UC

[6] Algorithms $\mathsf{QEnc}', \mathsf{Rand}$ scheme are not needed for the constructtion of UC Commitments.

bit-commitment scheme Π_{Com}. For simplicity of exposition, we will use $\mathsf{E}_{\mathsf{CCA}}$ in a black box manner. We note that the scheme naturally extends to a setting where commitments are defined over strings instead of just bits.

Common Reference String: The CRS consists of the public key (PK) of the QFHE scheme and the public key for the encryption scheme $\mathsf{ENC}_{\mathsf{CCA}}$.

Commit Phase:

1. On input $(\text{Commit}, sid, ssid, P_i, P_j, b)$ where $b \in \{0,1\}$, party P_i computes $z = \mathsf{QEnc}_{\mathsf{PK}}(b; r)$ where $r \leftarrow \mathcal{D}_{rand}(1^\lambda)$. Next, P_i computes $C_b = \mathsf{ENC}_{\mathsf{CCA}}(P_i, P_j, sid, ssid, r; s)$ using random coins s, and sets C_{1-b} to a random string of length $|C_b|$. Then, P_i records $(sid, ssid, P_j, r, s, b)$, and sends $c = (sid, ssid, P_i, z, C_0, C_1)$ to P_j.
2. P_j receives and records c, and outputs $(\text{Receipt}, sid, ssid, P_i, P_j)$. P_j ignores any later commit messages from P_i with the same $(sid, ssid)$.

Reveal Phase:

1. On input $(\text{Reveal}, sid, ssid)$, party P_i retrieves $(sid, ssid, P_j, r, s, b)$ and sends $(sid, ssid, r, s, b)$ to P_j.
2. Upon receiving $(sid, ssid, r, s, b)$ from P_i, P_j checks that it has a tuple $(sid, ssid, P_i, z, C_0, C_1)$. If yes, then it checks that $z = \mathsf{QEnc}_{\mathsf{PK}}(b; r)$ and that $C_b = \mathsf{ENC}_{\mathsf{CCA}}(P_i, P_j, sid, ssid, r; s)$. If both these checks succeed, then P_j outputs $(\text{Reveal}, sid, ssid, P_i, P_j, b)$. Otherwise, it ignores the message.

Proposition 1. *Assuming hardness of LWE, Protocol Π_{Com} UC realizes $\mathcal{F}_{\text{MCom}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model.*

The above commitment scheme UC realizes the multi-session ideal commitment functionality $\mathcal{F}_{\text{MCom}}$, described in Figure 2 in Appendix F, which reuses the public string for multiple commitments. The proof can be found in Appendix F. Next, we show how our UC commitment scheme serves towards the realization of a commit-and-prove functionality $\mathcal{F}_{\text{Com-ZK}}$ based on LWE.

4.1 Adaptive UC ZKPoK from LWE

Our UC commitment scheme serves towards the realization of a commit-and-prove functionality $\mathcal{F}_{\text{Com-ZK}}$ based on LWE. Such a functionality is generic and hence is quite useful – it allows a party to prove NP statements relative to its commitment value in the setting where parties commit to their inputs but they never decommit. The functionality $\mathcal{F}_{\text{Com-ZK}}$ is presented in Figure 3 and is comprised of two phases. In the first phase, a party commits to a specific value. In the second phase, this party proves NP statements in zero-knowledge relative to the committed value. It allows the committer to commit to multiple secret values w_i, and then have the relation $\mathcal{R}$ depend on all these values in a single proof. In addition, the committer may ask to prove multiple statements with respect to the same set of secret values. Hence, once a committer gives a new (Commit, sid, w) command, $\mathcal{F}_{\text{Com-ZK}}$ adds the current w to the already existing list $\overline{w}$ of committed values. Then, on receiving a $(\text{Proof}, sid, \mathcal{R}, x)$ request, $\mathcal{F}_{\text{Com-ZK}}$ evaluates $\mathcal{R}$ on x and the current list $\overline{w}$.

Using the power of the UC commitment scheme we constructed in Section 4, we show how it can be used to first construct UC Zero-Knowledge protocols from LWE. Canetti and Fischlin [CF01, Theorem 5], show that in the $\mathcal{F}_{\text{Com}}$-hybrid model there exists a 3-round protocol that securely realizes $\mathcal{F}_{\text{ZK}}$ with respect to any NP relation without any computational assumptions. Using the composition theorem and [CF01, Theorem 5], we can instantate $\mathcal{F}_{\text{Com}}$ with the UC commitment protocol from LWE (see Section 4) in the CRS model and realize $\mathcal{F}_{\text{ZK}}$ from LWE. Also, as it is noted by [CF01] we can replace $\mathcal{F}_{\text{Com}}$ by the functionality $\mathcal{F}_{\text{MCom}}$.

We next obtain a protocol for UC realizing functionality $\mathcal{F}_{\text{Com-ZK}}$ in the $\mathcal{F}_{\text{ZK}}$-hybrid model, in the presence of adaptive adversaries. In [CLOS02, Proposition 7.2], a protocol for UC realizing $\mathcal{F}_{\text{Com-ZK}}$ in the $\mathcal{F}_{\text{ZK}}$-hybrid model, based on any one-way function is proposed. To guarantee security against adaptive adversaries, they need equivocal and extractable commitments which they instantiate assuming the existence of enhanced trapdoor permutations. Using [CLOS02, Proposition 7.2] we can get such an instantiation assuming the hardness of LWE via our extractable and equivocal commitment scheme described above and instantiation of the $\mathcal{F}_{\text{ZK}}$ functionality from LWE.

Functionality $\mathcal{F}_{\text{MCom}}$

The functionality $\mathcal{F}_{\text{MCom}}$ runs with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. It proceeds as follows:

Commit Phase:

Upon receiving a message $(\text{Commit}, sid, ssid, P_i, P_j, b)$ from Pi, where $b \in \{0, 1\}$, record the tuple $(ssid, P_i, P_j, b)$ and send the message $(\text{Receipt}, sid, ssid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any future commit messages with the same ssid from P_i to P_j.

Prove Phase:

Upon receiving a message $(\text{Reveal}, sid, ssid)$ from P_i: If a tuple $(ssid, P_i, P_j, b)$ was previously recorded, then send the message $(\text{Reveal}, sid, ssid, P_i, P_j, b)$ to Pj and $\mathcal{S}$. Otherwise, ignore.

Fig. 2. The ideal functionality $\mathcal{F}_{\text{MCom}}$.

Functionality $\mathcal{F}_{\text{Com-ZK}}$

The functionality $\mathcal{F}_{\text{Com-ZK}}$ runs with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. It proceeds as follows:

Commit Phase:

Upon receiving a message $(\text{Commit}, sid, cid, \mathcal{P}, w)^a$ from P_i where $\mathcal{P}$ is a set of parties and $w \in \{0, 1\}^*$, append the value w to the existing list $\overline{w}$, record $\mathcal{P}$, and send the message $(\text{Receipt}, sid, cid, P_i, \mathcal{P})$ to the parties in $\mathcal{P}$ and $\mathcal{S}$. (Initially, the list $\overline{w}$ is empty. Also, if a commit message has already been received, then check that the recorded set of parties is $\mathcal{P}$. If it is a different set, then ignore this message.)

Prove Phase:

Upon receiving a message $(\text{Prover}, sid, \mathcal{R}, x)$ from P_i, where $x \in \{0, 1\}^{poly(k)}$, compute $\mathcal{R}(x, w)$: If $\mathcal{R}(x, w) = 1$, then send the message $(\text{Proof}, sid, \mathcal{R}, x)$ to the parties in $\mathcal{P}$ and $\mathcal{S}$. Otherwise, ignore.

[a] Note that in the protocol we use one command for two cid's. In particular we use cid_1 to commit to the encrypted value and cid_2 to commit to the randomness used for the corresponding encryption

Fig. 3. Ideal functionality $\mathcal{F}_{\text{Com-ZK}}$.

5 Our Protocol

Since we established all the primitives needed we are ready to present our MPC protocol. Our protocol is based on any equivocal QFHE scheme which comes together with a statistically secure distributed function sharing scheme. In addition, the protocol assumes access to the $\mathcal{F}_{\text{Com-ZK}}$ functionality which we build from any equivocal QFHE, see Section 4. In Figure 4 we describe our protocol Π_{MPC} realizing the functionality $\mathcal{F}_{\text{AMPC}}$ in Figure 6, in the $(\mathcal{F}_{\text{broadcast}}, \mathcal{F}_{\text{Key-Dist}}, \mathcal{F}_{\text{Com-ZK}})$-hybrid model. The functionality $\mathcal{F}_{\text{Key-Dist}}$ is described in Figure 5 and the functionality $\mathcal{F}_{\text{Com-ZK}}$ is described in Figure 3.

During the Load phase, players encrypt their inputs x_i under a common public key PK and give a ZKPoK. In the evaluation phase, players evaluate the desired function locally and obtain the ciphertext $\mathsf{enc}(z)$. In the output phase they jointly decrypt the result calling the decryption protocol Π_{DDec} together with the ciphertext randomisation technique as is abstracted by the algorithm Rand of the QFHE, see Section 3. In the protocol Π_{DDec} parties use ZK to prove that their evaluation shares are correct. However, as discussed in the introduction we optimise the output phase avoiding the expensive use of ZK proofs to prove that the player's evaluation shares to the decryption protocol are correct, changing the evaluation phase of the protocol and avoiding the ZK proofs. For details see Section 6.

Protocol Π_{MPC}

Protocol Π_{MPC} uses an equivocal $\mathsf{QFHE} = (\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Rand}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{Equiv})$ scheme and runs in the $(\mathcal{F}_{\text{BROADCAST}}{}^a, \mathcal{F}_{\text{KEY-DIST}}, \mathcal{F}_{\text{COM-ZK}})$-hybrid model with parties $(P_1, \ldots, P_n)$. It proceeds as follows:

Initialize:

On input $(\text{init}, 1^\lambda)$ from all parties, invoke the functionalities $\mathcal{F}_{\text{BROADCAST}}$, $\mathcal{F}_{\text{KEY-DIST}}$ and $\mathcal{F}_{\text{COM-ZK}}$. The invocation of $\mathcal{F}_{\text{KEY-DIST}}$ results in every party P_i receiving $\big((\mathsf{PK}, c_1, \ldots, c_n), (\mathsf{sk}_i, r_i)\big)$.

Load:

To encrypt its input x_i, P_i does the following:

- P_i computes $X_i = \mathsf{QEnc}_{\mathsf{PK}}(0, x_i; r_{x_i})$, where $r_{x_i} \leftarrow \mathcal{D}_{rand}(1^\lambda)$, and broadcasts X_i via $\mathcal{F}_{\text{BROADCAST}}$.
- For $i \neq j$, P_i sends $(\text{Commit}, sid, cid_1, cid_2, P_i, P_j, x_i, r_{x_i})$ to $\mathcal{F}_{\text{COM-ZK}}$. At this point all other parties P_j receive message $(\text{Receipt}, sid, cid_1, cid_2, P_i, P_j)$ from $\mathcal{F}_{\text{COM-ZK}}$.
- For $j \neq i$, P_i sends $(\text{Prover}, sid, (cid_1, cid_2), \mathcal{R}_{eq}, X_i)$ to $\mathcal{F}_{\text{COM-ZK}}$ for the relation

$$\mathcal{R}_{eq} = \{((\mathsf{PK}, X_i), (x_i, r_{x_i})) : X_i = \mathsf{QEnc}_{\mathsf{PK}}(0, x_i; r_{x_i})\}$$

  whereupon P_j receives $(\text{Proof}, sid, P_i, \mathcal{R}_{eq}, (\mathsf{PK}, X_i))$.
- If all the proofs are accepted then the parties define $\mathsf{enc}(x_i) = X_i$, otherwise output $\perp$.

Evaluation Phase:

Let ckt be the arithmetic circuit to be computed on inputs $(x_1, \ldots, x_n)$ by n parties. Every party executes the deterministic algorithm Eval and obtains $\mathsf{enc}(z) \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\text{ckt}, \mathsf{enc}(x_1), \ldots, \mathsf{enc}(x_n))$.

Output Phase:

- P_i generates $y_i \leftarrow \mathcal{D}_{rand}(1^\lambda)$ and **Load**s it into variable $\mathsf{enc}(y_i)$ via $\mathsf{QEnc}_{\mathsf{PK}}$ for $b = 0$. Let cid_1 and cid_2 be the identifiers of the commitment phase of this **Load**.
- P_i computes $\widetilde{\mathsf{enc}(y_i)} = \mathsf{QEnc}_{\mathsf{PK}}(1, y_i; \tilde{r}_{y_i})$, where $\tilde{r}_{y_i} \leftarrow \mathcal{D}_{rand}(1^\lambda)$, and broadcasts $\widetilde{\mathsf{enc}(y_i)}$ via $\mathcal{F}_{\text{BROADCAST}}$. Next, for $j \neq i$ party P_i sends $(\text{Commit}, sid, cid_3, P_i, P_j, \tilde{r}_{y_i})$ to $\mathcal{F}_{\text{COM-ZK}}$ and $(\text{Prover}, sid, (cid_1, cid_3), \mathcal{R}_{eq}, \widetilde{\mathsf{enc}(y_i)})$ to $\mathcal{F}_{\text{COM-ZK}}$, where cid_1 is the identifier of the commitment phase of the **Load** of the above Step 1, where P_i commits to y_i.
- Let J be the set of indices of P_j's having defined $\mathsf{enc}(y_i)$ and $\widetilde{\mathsf{enc}(y_i)}$. Then compute $\text{CT} = \mathsf{Rand}(\mathsf{enc}(z), \{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J})$.
- Every party P_i runs $\Pi_{\text{DDEC}}{}^b$ with the rest of the parties to decrypt CT.

[a] Since we have (potential) dishonest majority, note that we cannot guarantee termination. For a concrete implementation of the broadcast functionality we refer to [DPSZ12].

[b] The protocol Π_{DDEC} is described in Subsetion 5.1 and Figure 7.

Fig. 4. Π_{MPC} Protocol.

Functionality $\mathcal{F}_{\text{KEY-DIST}}$

The functionality $\mathcal{F}_{\text{KEY-DIST}}$ runs with parties $P_1, \ldots, P_n$ and is parameterized by a statistically hiding commitment scheme with commitment function Com. It proceeds as follows:

Generate:

On input $(\text{init}, 1^\lambda)$ from all honest parties, run $\mathsf{KeyGen}(1^\lambda)$ of the QFHE scheme and obtain PK, SK and then additively secret-share sk to obtain $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.

1. For $i = 1, \ldots, n$, commits to the share sk_i by computing $c_i = \mathsf{Com}(\mathsf{sk}_i; r_i)$ where $r_i \leftarrow \mathcal{D}_{rand}(1^\lambda)$.
2. In a round specified by the adversary, output $\big((\mathsf{PK}, c_1, \ldots, c_n), (\mathsf{sk}_i, r_i)\big)$ to P_i.

Incorrect inputs:

If in the first round an honest party inputs a non-trivial value and does not input init, abort. Moreover, abort if an honest party inputs init twice or any other value than init.

Fig. 5. Ideal functionality $\mathcal{F}_{\text{KEY-DIST}}$.

Functionality $\mathcal{F}_{\text{AMPC}}$

The functionality $\mathcal{F}_{\text{AMPC}}$ runs with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$ and is parametrised by an arithmetic circuit ckt. It proceeds as follows.

Initialize:

On input $(\text{init}, 1^\lambda)$ from all parties, the functionality generates a random FHE key $(\mathsf{SK}, \mathsf{PK})$. It outputs PK to all parties.

Load Phase:

On input $(\text{Input}, P_i, \mathit{varid}, x)$ from P_i and $(\text{Input}, P_i, \mathit{varid}, ?)$ from all other parties, with varid a fresh identifier, the functionality stores (varid, x) and outputs $(\mathit{cid}, \mathit{varid}, \text{Defined})$ to all parties. If P_i is corrupted before $(\mathit{cid}, \mathit{varid}, \text{Defined})$ is output, and if the adversary outputs $(\mathit{cid}, \mathit{varid}, \text{Fail})$, then output $(\mathit{cid}, \mathit{varid}, \text{Fail})$ to all parties.

Evaluation Phase:

On input $(\text{Evaluation}, \mathit{varid}_1, \ldots, \mathit{varid}_n, \mathit{varid}_{n+1})$ from all parties (if $\mathit{varid}_1, \ldots, \mathit{varid}_n$ are present in memory and varid_{n+1} is not), the functionality retrieves (varid_1, x_1), $\ldots$,(varid_n, x_n) and stores $(\mathit{varid}_{n+1}, \text{ckt}(x_1, \ldots, x_n))$.

Output Phase:

On input $(\text{Output}, \mathit{varid}_{n+1})$ from all honest parties (if varid_{n+1} is present in memory), the functionality retrieves $(\mathit{varid}_{n+1}, x)$ and outputs it to the environment. If the environment inputs OK then x is output to all players. Otherwise $\perp$ is output to all players.

Fig. 6. Ideal functionality for Arithmetic MPC.

5.1 Distributed Function Evaluation

In order to achieve distributed decryption, we assume, as a set up assumption, that a common public key pk has been set up where the secret key sk has been secret-shared between n parties in such a way that they can compute their corresponding decryption evaluation shares and then collaborate to decrypt while the sk is kept secret. We also need to enforce honest computation of the evaluation shares of a ciphertext. Commitments to the shares of the secret key are also made public, along with pk. Using these commitments, when parties are distributedly decrypting a ciphertext, they can then prove (via $\mathcal{F}_{\text{Com-ZK}}$) that the evaluation shares were computed honestly using the secret-key shares initially delegated to them.

To this end, the functionality $\mathcal{F}_{\text{Key-Dist}}$ generates a key pair $(\mathsf{pk}, \mathsf{sk})$[7] and secret-shares the secret key sk among the players using a secret-sharing scheme that is assumed to be given as part of the specification of the cryptosystem. The validity of the evaluation shares is tested inside the protocol Π_{DDec} calling the functionality $\mathcal{F}_{\text{Com-ZK}}$. In order to describe our protocol Π_{DDec}, we next define the following distributed sharing scheme.

Definition 5. *We call* $(\mathsf{ShareSK}, \mathsf{ShareEval}, \mathsf{Combine})$ *a distributed function sharing scheme for an encryption scheme* $(\mathsf{KeyGen}_{\mathsf{FHE}}, \mathsf{Enc}, \mathsf{Dec})$*, with construction threshold* c *and privacy threshold* t*, if for a triple* $(\mathsf{ShareSK},$
$\mathsf{ShareEval}, \mathsf{Combine})$ *of PPT algorithms the following hold:*

Key sharing: *The algorithm* $\mathsf{ShareSK}$ *on input* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$ *and a construction threshold* c*, outputs a tuple* $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{ShareSK}(\mathsf{sk})$.

Evaluation sharing: *The evaluation function* $\mathsf{ShareEval}$ *on input* $(\mathsf{pk}, \mathsf{sk}_i)$ *and a ciphertext* $\mathsf{Enc}_{\mathsf{pk}}(z)$*, outputs an evaluation share*

$$ev_i = \mathsf{ShareEval}(\mathsf{pk}, \mathsf{sk}_i, \mathsf{Enc}_{\mathsf{pk}}(z); r_{ev_i})$$

for $i \in [n]$ *where* $r_{ev_i} \leftarrow \mathcal{D}_{rand}(1^\lambda)$.

Share combining: *The algorithm* $\mathsf{Combine}$ *on input correctly computed evaluation shares* $\{ev_i\}_{i\in[n]}$ *of the same ciphertext* $\mathsf{Enc}_{\mathsf{pk}}(z)$*, constructs the output* $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(z)) = \mathsf{Combine}(\{ev_i\}_{i\in[n]})$.

For our purposes, the construction threshold $\mathsf{c} = n$ and the corruption threshold $\mathsf{t} = n - 1$. In Figure 7, we describe our protocol Π_{DDec}, parameterized by $(\mathsf{ShareSK}, \mathsf{ShareEval}, \mathsf{Combine})$.

[7] In the description of our protocol we choose to explicitly refer to the keys $(\mathsf{pk}, \mathsf{sk})$ since it helps in the description of the decryption protocol.

Protocol Π_{DDec}

The protocol runs in the $(\mathcal{F}_{\text{BROADCAST}}, \mathcal{F}_{\text{KEY-DIST}}, \mathcal{F}_{\text{COM-ZK}})$-hybrid model with parties $P_1, \ldots, P_n$ and it is parametrized by $(\mathsf{ShareEval}, \mathsf{Combine})$, as defined in Definition 5. It proceeds as follows:

Key Sharing: On input $(\text{init}, 1^\lambda)$ from all parties, invoke the functionalities $\mathcal{F}_{\text{BROADCAST}}, \mathcal{F}_{\text{KEY-DIST}}$ and $\mathcal{F}_{\text{COM-ZK}}$. The invocation of $\mathcal{F}_{\text{KEY-DIST}}$ results in every party P_i receiving $\big((\mathsf{PK}, c_1, \ldots, c_n), (\mathsf{sk}_i, r_i)\big)$.

Evaluation Sharing:

1. For $i \neq j$, P_i samples $r_{ev_i} \leftarrow \mathcal{D}_{rand}(1^\lambda)$ and sends $(\text{Commit}, sid, cid, P_i, P_j, r_{ev_i})$ to $\mathcal{F}_{\text{COM-ZK}}$. At this point all other parties P_j receive message $(\text{Receipt}, sid, P_i, P_j)$ from $\mathcal{F}_{\text{COM-ZK}}$.
2. Party P_i, on input ciphertext CT, computes its evaluation share $ev_i \leftarrow \mathsf{ShareEval}(\mathsf{PK}, \mathsf{sk}_i, \text{CT}; r_{ev_i})$ and broadcasts ev_i via $\mathcal{F}_{\text{BROADCAST}}$.
3. For $j \neq i$, P_i sends $(\text{Prover}, sid, P_i, P_j, \mathcal{R}_{eval}, (c_i, \mathsf{PK}, \mathsf{enc}(z), ev_i))$ to $\mathcal{F}_{\text{COM-ZK}}$ for the relation

$$\mathcal{R}_{eval} = \{((c_i, \mathsf{PK}, \text{CT}, ev_i), (\mathsf{sk}_i, r_i, r_{ev_i})) : c_i = \mathsf{Com}(\mathsf{sk}_i; r_i) \wedge$$

$$ev_i = \mathsf{ShareEval}(\mathsf{PK}, \mathsf{sk}_i, \text{CT}; r_{ev_i})\}$$

   where Com is the commitment scheme used in $\mathcal{F}_{\text{KEY-DIST}}$.
4. For $i \neq j$, P_j sends the message $(\text{Proof}, sid, \mathcal{R}_{eval}, (c_i, \mathsf{PK}, \text{CT}, ev_i))$.

Share Combining: If any party P_i outputs reject for a proof given by any party P_j, then output Abort. $\mathsf{Otherwise, output\ Combine}(\{ev_i\}_{i \in [n]})$.

A concrete instantiation of the protocol Π_{DDec} based on LWE is given in Appendix E.

Fig. 7. Distributed decryption protocol.

Theorem 4. *Let* $\mathsf{QFHE} = (\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Eval}, \mathsf{Rand}, \mathsf{Dec}, \mathsf{Equiv})$ *be an equivocal fully homomorphic encryption scheme; let it be associated with a distributed function sharing scheme* $(\mathsf{ShareSK}, \mathsf{ShareEval}, \mathsf{Combine})$. *Then the constant-round protocol* Π_{MPC} *UC-securely realises the ideal functionality* $\mathcal{F}_{\text{AMPC}}$ *in the* $(\mathcal{F}_{\text{BROADCAST}}, \mathcal{F}_{\text{KEY-DIST}}, \mathcal{F}_{\text{COM-ZK}})$*-hybrid model with computational security against any adaptive, active adversary corrupting at most all-but-one parties.*

For the proof of Theorem 4 see Appendix C. Replacing *UC* ZK with *UC* NIZK leads to a three-round protocol.

High level idea of the security proof. Our simulator uses the properties of the QFHE scheme such as the indistingusability of equivocation, according to Definition 3. Furthermore, as we discussed in Section 1, the simulator will not be able to cheat in the distributed decryption protocol by decrypting a given ciphertext to any desired value. The key setup for the decryption protocol fixes the shares of the private key even in the simulation. Thus, a ciphertext can only be decrypted to the value it actually contains. Of course, when decrypting the outputs, the correct results should be produced both in simulation and real life, and so we have a problem since all ciphertexts in the simulation generated with respect to the honest parties will contain encryptions of 0. For this issue we use the ciphertext randomisation property. Notice that the ciphertext ct in the ciphertext randomization property as per Definition 3 corresponds to the real output $\mathsf{enc}(z)$ of the protocol Π_{MPC} and the ciphertexts $\text{ct}'_1, \ldots, \text{ct}'_n$ correspond to the ciphertexts $\{\mathsf{enc}(y_i)\}_{i \in J}$. In the real-world the ciphertexts $\{\mathsf{enc}(y_i)\}_{i \in J}$ are `redundant`. On the other hand, in the ideal-world the final ciphertext CT decrypts to a value contributed only by the ciphertexts $\{\mathsf{enc}(y_i)\}_{i \in J}$. In this case we will call the ciphertexts $\{\mathsf{enc}(y_i)\}_{i \in J}$ `non − redundant`. This implies that an honest execution of the **Output** stage is not possible with the ciphertexts of $\{\mathsf{enc}(y_i)\}_{i \in J}$ being `non − redundant`. Analogously, the ciphertext $\mathsf{enc}(z)$ can be either `redundant` or `non − redundant`. In other words, it is pertinent that before we get to a hybrid where the **Output** stage is performed honestly, we need a hybrid where $\{\mathsf{enc}(y_i)\}_{i \in J}$ turn to `redundant` ciphertexts. However, with both ciphertexts $\{\mathsf{enc}(y_i)\}_{i \in J}$ and $\mathsf{enc}(z)$ `redundant`, we can not hope to get the final output CT to decrypt to the actual output value. Thus, even before turning $\{\mathsf{enc}(y_i)\}_{i \in J}$ to `redundant` ciphertexts, we need a hybrid where we can cheat in the final decryption. That is, we first need to have a hybrid that, instead of running the distributed decryption protocol, runs what we abstract as the simulator for the distributed decryption. Moreover, we also based on the semantic security of the FHE scheme in interchangeably switching the keys K and R to encryptions of 0 and 1, respectively.

A full proof is given in the Appendix.

6 On the Communication Complexity of Distributed Decryption

Our protocol as described in Section 5 assumes that the QFHE scheme comes with a semi-honest secure distributed decryption protocol: from the ciphertext and shares of the secret key players can compute decryption shares which, if correct, allow the reconstruction of the plaintext. We then augment the distributed decryption with ZK proofs so that players prove that their contributions to the decryption are correct. This solution has communication complexity proportional to the circuit complexity of the decryption function.

However, our approach allows for a significant optimization of the decryption procedure compared to generic solutions. More specifically, we tweak our protocol Π_{MPC} such that the communication complexity of the decryption becomes independent of its circuit complexity.

To this end, we modify the evaluation phase of our protocol presented in Section 5. Note that our original protocol allows us to securely compute any (randomized) function. In particular, any randomized function allows the parties to encrypt randomized shares and then add up them together. Therefore, instead of computing the original function, we compute a new function, which for each output z of the original function also outputs α and $w = \alpha z$ where α is randomly chosen in some large field, and where the multiplication αz also takes place in that field. Of course if we can compute this function securely then we can also compute the original function securely. Observe that this new function comes along with an extra property which allows to check if the output is correct or not based on whether $w = \alpha z$.

In order to incorporate the above, the modification to the protocol is as follows. Instead of having a single ciphertext $\mathsf{enc}(z)$ containing z, we will have two extra ciphertexts, namely $\mathsf{enc}(\alpha)$ and $\mathsf{enc}(w)$. The ciphertext $\mathsf{enc}(\alpha)$ is computed as follows. Each party randomly selects a one-time a_i and encrypts it according to the Load phase of our protocol Π_{MPC} in Figure 5. Once each party has loaded and broadcasted $\mathsf{enc}(a_i)$, each party computes $\mathsf{enc}(\alpha) = \mathsf{enc}(a_1) \boxplus \ldots \boxplus \mathsf{enc}(a_n)$ and $\mathsf{enc}(w) = \mathsf{enc}(\alpha) \boxdot \mathsf{enc}(z)$. Thus, instead of calling the output phase of our protocol only on input $\mathsf{enc}(z)$ we call it on three different ciphertexts $\mathsf{enc}(z), \mathsf{enc}(\alpha), \mathsf{enc}(w)$. This means that now the decryption protocol will generate three sets of evaluation shares.

The modification in the decryption protocol is as follows. Before we first broadcast the shares and then we prove in ZK that they were correct. Instead, we are *not* going to broadcast all the evaluation shares immediately due to the adversary who may see the contributions from the honest parties to α before his broadcast enabling him to forge. We need to guarantee that the adversary cannot forge the output by making sure that he should output his share before he sees α. In order to avoid the above complication, we first commit to the evaluation shares and then we open them. In particular, all players compute their evaluation shares for z, α and w and commit to them. If opening fails or if the decrypted values do not satisfy $\alpha z = w$, we abort. This solution avoids the use of ZK proofs yielding a solution which is independent of the circuit complexity of the decryption.

Since there is an encryption of α available, the new aspect in the proof is to show that this does not help the adversary to learn α unless he can break CPA security. We can argue this in the proof in Appendix C where we turn the ciphertext $\mathsf{enc}(z)$ to `redundant`. Therefore, the same proof still applies but instead we will have three `redundant` ciphetexts $\mathsf{enc}(\alpha z), \mathsf{enc}(\alpha), \mathsf{enc}(w)$. In this hybrid the outputs cannot be forged since the ciphertext $\mathsf{enc}(\alpha)$ is `redundant` and it does not contain information about α. Thus, an advesary that he cannot forge he cannot distinguish in the real world and break CPA-security.

7 Acknowledgements

The authors would like to thank Nico Döttling, Yuval Ishai and Chris Peikert for helpful discussions. We also thank Jonathan Katz for pointing out his result [KTZ13]. Ivan Damgård and Antigoni Polychriniadou acknowledge support from the Danish National Research Foundation and the National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation and from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish

Strategic Research Council. In addition, the research was supported by the MPCPRO project funded by the ERC.

References

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT*, pages 483–501, 2012.

[BCHK07] Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *SIAM Journal on Computing*, 36(5):1301–1328, 2007.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press.

[CDF+08] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 471–488, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 19–40, 2001.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[CGP15] Ran Canetti, Shafi Goldwasser, and Oxana Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 557–585, 2015.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 494–503, 2002.

[DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *SIAM Journal on Computing*, pages 542–552, 2000.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006.

[DIK+08] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, pages 241–261, 2008.

[DKR15] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 586–613, 2015.

[DMRV13] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Muthuramakrishnan Venkitasubramaniam. Adaptive and concurrent secure computation from new adaptive, non-malleable commitments. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 316–336, Bengalore, India, December 1–5, 2013. Springer, Berlin, Germany.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *CRYPTO*, pages 247–264, 2003.

[DNP15] Ivan Damgård, Jesper Buus Nielsen, and Antigoni Polychroniadou. On the communication required for unconditionally secure multiplication. *IACR Cryptology ePrint Archive*, 2015:1097, 2015.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11, 2012.

[GP15] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 614–637, 2015.

[GS12] Sanjam Garg and Amit Sahai. Adaptively secure multi-party computation with dishonest majority. In *CRYPTO*, pages 105–123, 2012.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 335–354, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Berlin, Germany.

[KTZ13] Jonathan Katz, Aishwarya Thiruvengadam, and Hong-Sheng Zhou. Feasibility and infeasibility of adaptively secure fully homomorphic encryption. In *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*, pages 14–31, 2013.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1219–1234, 2012.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 700–718, 2012.

[MW15] Pratyay Mukherjee and Daniel Wichs. Two round mutliparty computation via multi-key FHE. Cryptology ePrint Archive, Report 2015/345, 2015. http://eprint.iacr.org/.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 111–126, 2002.

[OPW11] Adam O'Neill, Chris Peikert, and Brent Waters. Bi-deniable public-key encryption. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 525–542, 2011.

[Pei14] Chris Peikert. personal communication, September 2014.

A Universally Composable Security

The universally composable (UC) security framework was introduced by Canetti [Can01]. The strength of this framework relies on the universally composable theorem, which states that if a protocol is secure in the UC model, then this protocol will preserve the same security even if composed with an arbitrary number of copies of itself or with other protocols. The UC framework gives us also a way to design our protocols in a modular way: we can design sub-protocols for simpler tasks and then combine them in more complex protocols, and still we can prove the security of the sub-protocols independently. In order to develop interesting protocols in the UC model we need some kind of setup assumptions, like a common reference string (CRS) available to the parties, or a key registration authority, that checks that the parties know their secret keys and the public keys are well-formed, or many other different assumptions.

Adversarial model. A *static* adversary $\mathcal{A}$ chooses the set of corrupted parties before the protocol starts, as opposed to an *adaptive* adversary that can corrupt the players during the protocol. We say that the adversary is *passive* or *semi-honest* if $\mathcal{A}$ follows the protocol but tries to extract some information about the other parties' inputs from his view of the protocol. We say that the adversary is *active* or *malicious* if $\mathcal{A}$ is allowed to deviate arbitrarily from the protocol specifications. We will say that a protocol is *passive-secure* if it is secure against a passive adversary and *active-secure* (or malicious secure) if it is secure against an active adversary. In the UC model the adversary, as well as all the other parties involved, are modeled as probabilistic polynomial time (PPT) interactive Turing machine (ITM). In this paper we consider *active-security* against an *adaptive* adversary.

The real world. We model a real world execution of a cryptographic protocol in the UC model by defining a PPT ITM $\mathcal{Z}$ called the environment, that gives inputs and gets outputs from the parties $P_1, \ldots, P_n$ running the protocol. Moreover, $\mathcal{Z}$ communicates with $\mathcal{A}$ giving instructions on how to attack the protocol. The parties and the adversary usually also have access to some ideal functionality $\mathcal{H}$.

The ideal world. We define also an ideal world, where the parties $P_1, \ldots, P_n$ interact with an ideal functionality $\mathcal{F}$, that captures the properties we expect from our protocol. Here the parties get their inputs from the environment $\mathcal{Z}$ and simply forward them to $\mathcal{F}$, therefore they are usually referred as the dummy parties. There is also an ideal adversary $\mathcal{S}$, called the simulator, that communicates with the environment $\mathcal{Z}$ and with the ideal functionality.

Indistinguishability. At the beginning of the protocol all parties, the environment and the adversary are given a security parameter λ. The environment is also given an auxiliary input z. At some point the environment stops and outputs a bit. We use $\mathrm{REAL}^{\mathcal{H}}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, r, z)$ to denote the output of $\mathcal{Z}$ in the real world and $\mathrm{IDEAL}^{\mathcal{H}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, r, z)$ in the ideal word where we take r to be uniformly random. This defines the Boolean distribution ensembles $\{\mathrm{REAL}^{\mathcal{H}}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$ and $\{\mathrm{IDEAL}^{\mathcal{H}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$

Definition 6. *We say that π securely implements $\mathcal{F}$ in the $\mathcal{H}$-hybrid model if $\forall PPT\ \mathcal{A}, \exists PPT\ \mathcal{S}$ such that* $\mathrm{REAL}^{\mathcal{H}}_{\pi,\mathcal{A},\mathcal{Z}}$ *and $IDEAL^{\mathcal{H}}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable in λ.*

B Security Definition of Distributed Function Evaluation

Definition 7. *A sharing scheme* $(\mathsf{ShareSK}, \mathsf{ShareEval}, \mathsf{Combine})$ *for an encryption scheme* $(\mathsf{KeyGen}_{\mathsf{FHE}}, \mathsf{Enc}, \mathsf{Dec})$ *is called* a statistically secure distributed scheme for corruption threshold $n-1$ *if there exist PPT algorithm,* $\mathcal{S}_{\mathsf{KeyDist}}, \mathcal{S}_{\mathsf{Eval}}$ *such that the following hold:*

Key distribution simulation: *The algorithm $\mathcal{S}_{\mathsf{KeyDist}}$ on input $(\mathsf{pk}, \mathrm{C})$, where $\mathrm{C} \subseteq [n]$, outputs $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\mathrm{C}})$. We require that $\forall \mathrm{C}$ with $|\mathrm{C}| \leq n-1$, the following two experiments are statistically close.*

$(\mathsf{pk}, \cdot) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$	$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$
$\{\mathsf{sk}_i\}_{i\in\mathrm{C}} \leftarrow \mathcal{S}_{\mathsf{KeyDist}}(\mathsf{pk}, \mathrm{C})$	$\{\mathsf{sk}_i\}_{i\in[n]} \leftarrow \mathsf{ShareSK}(\mathsf{sk})$
Return $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\mathrm{C}})$	*Return* $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\mathrm{C}})$

Evaluation simulation: *The algorithm $\mathcal{S}_{\mathsf{Eval}}$ on input $\{\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\mathrm{C}}, \mathrm{CT}, z, \{ev_i\}_{i\in\mathrm{C}})$, where $\mathrm{C} \subseteq [n]$, outputs $\{ev_i\}_{i\in[n]\setminus\mathrm{C}}$. We require that $\forall \mathrm{C}$ with $|\mathrm{C}| \leq n-1$, the following two experiments are statistically close.*

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$	$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$
$\{\mathsf{sk}_i\}_{i\in[n]} \leftarrow \mathsf{ShareSK}(\mathsf{sk})$	$\{\mathsf{sk}_i\}_{i\in[n]} \leftarrow \mathsf{ShareSK}(\mathsf{sk})$
$\{ev_i\}_{i\in[n]} \leftarrow \mathsf{ShareEval}(\mathsf{pk}, \mathsf{sk}_i, \mathrm{CT})$	$\{ev_i\}_{i\in\mathrm{C}}, \leftarrow \mathsf{ShareEval}(\mathsf{pk}, \mathsf{sk}_i, \mathrm{CT})$
	$\{ev_i\}_{i\in[n]\setminus\mathrm{C}} \leftarrow \mathcal{S}_{\mathsf{Eval}}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\mathrm{C}}, \mathrm{CT}, z, \{ev_i\}_{i\in\mathrm{C}})$.
Return $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\mathrm{C}}, \mathrm{CT}, z, \{ev_i\}_{i\in[n]})$	*Return* $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\mathrm{C}}, \mathrm{CT}, z, \{ev_i\}_{i\in[n]})$

Remark 1. The existence of $\mathcal{S}_{\mathsf{KeyDist}}$ in essence says that the values seen by at most t ($n-1$ corrupted) parties could have been generated from pk alone.

Remark 2. The existence of $\mathcal{S}_{\mathsf{Eval}}$ in essence says that if one knows the values that $n-1$ parties are entitled to see, and if one knows $z = \mathsf{Dec}_{\mathsf{sk}}(\mathrm{CT})$, then one can compute the evaluation shares of all parties. It is of course trivial to compute $ev_i \leftarrow \mathsf{ShareEval}(\mathsf{pk}, \mathsf{sk}_i, \mathrm{CT})$ for the $n-1$ values $\{\mathsf{sk}_i\}_{i\in\mathrm{C}}$ one knows; but what the evaluation simulation property says is that $ev_i \leftarrow \mathsf{ShareEval}(\mathsf{pk}, \mathsf{sk}_i, \mathrm{CT})$ can be computed even for $i \notin \mathrm{C}$, provided, the plaintext z, the rest of the secret-key shares $\{\mathsf{sk}_i\}_{i\in\mathrm{C}}$, and the rest of the decryption shares $\{ev_i\}_{i\in\mathrm{C}}$ are known.

C Proof of Security

Theorem 5. *Let* $\mathsf{QFHE} = (\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Eval}, \mathsf{Rand}, \mathsf{Dec}, \mathsf{Equiv})$ *be an equivocal fully homomorphic encryption scheme; let it be associated with a distributed function sharing scheme* $(\mathsf{ShareSK}, \mathsf{ShareEval}, \mathsf{Combine})$*. Then the constant-round protocol* Π_{MPC} *UC-securely realises the ideal functionality* $\mathcal{F}_{\mathrm{AMPC}}$ *in the* $(\mathcal{F}_{\textsc{broadcast}}, \mathcal{F}_{\textsc{Key-Dist}}, \mathcal{F}_{\textsc{Com-ZK}})$*-hybrid model with computational security against any adaptive, active adversary corrupting at most all-but-one parties.*

Proof. We begin by giving a high-level intuition for the proof. As we shall see, the crucial aspects of our protocol that we exploit in the proof are the properties listed in Definition 3.

The proof is carried out by a sequence of hybrids. We shall begin with a hybrid that is identical to the ideal-world execution. Note that the main difference from the real-world execution is the ciphertext randomisation. Naturally, when we start from one world, in order to move to the other, we would need to employ the indistinguishability of equivocal keys property. Hence, the natural direction would be to eliminate the steps which use the secret key and, roughly speaking, somehow simulate these steps without the secret key. To this end, firstly, we rely on the simulatable evaluation property of the distributed function sharing scheme. Here, the simulator first learns the evaluation shares that the corrupted parties might send by intercepting the commit commands sent by the corrupted parties. Then, as a function of these evaluation shares and the supposed output, the hybrid would compute the evaluation shares of honest parties. Observe that there is a possibility that the corrupted parties may not send the evaluation shares consistent with the randomness it would have sent earlier through $\mathcal{F}_{\textsc{Com-ZK}}$; however, in this case, by the security of $\mathcal{F}_{\textsc{Com-ZK}}$, the corrupted party could not have given convincing proofs, leading to an abort. Thus, whenever conditioned on no abort, with this modification of simulating the evaluation shares of honest parties, the deviation introduced in the view of the adversary is computationally indistinguishable, by applying the simulatable evaluation property. Next, we can also sample all the secret-key shares sk_i to be uniformly random. This introduces no deviation in the view of the adversary for the following reason: the commitments made to the secret-key shares are using a statistically hiding scheme; furthermore, no longer at any point in the execution we use the secret key. With this, we can turn $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ to `redundant` ciphertexts. With this, we can switch to a hybrid where we can start executing as prescribed by the protocol, except for simulating the evaluation shares. Next, we move to a hybrid which is given the actual inputs of parties; therein, the hybrid would load the actual inputs. Next, we may turn $\mathsf{enc}(z)$ to a $\texttt{non}-\texttt{redundant}$ ciphertext and switch to loading the random coins used to distributedly decrypt the ciphertext honestly. In this step, we need to deploy the indistinguishability of equivocation and the indistinguishability of equivocal keys in various steps. Finally, we additively secret share the secret key instead of choosing all shares at random and performing the distributed decryption as prescribed by $\Pi_{\textsc{DDec}}$. We now proceed to provide a formal proof. We shall provide our proof at a low level for clarity, while implementing certain generic algorithms such as $\mathsf{ShareSK}$ (with additive secret sharing of the secret key).

Let $\mathcal{A}$ be an adaptive adversary who operates against the Protocol Π_{MPC} in the $(\mathcal{F}_{\textsc{broadcast}}, \mathcal{F}_{\textsc{Key-Dist}},$ $\mathcal{F}_{\textsc{Com-ZK}})$-hybrid model. Our objective is to construct an ideal-process adversary, called simulator, $\mathcal{S}_{\mathrm{AMPC}}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running Protocol Π_{MPC} in the $(\mathcal{F}_{\textsc{broadcast}},$ $\mathcal{F}_{\textsc{Key-Dist}}, \mathcal{F}_{\textsc{Com-ZK}})$-hybrid model or with $\mathcal{S}_{\mathrm{AMPC}}$ in the ideal process for $\mathcal{F}_{\mathrm{AMPC}}$. Generally, the challenging aspect in constructing a simulator for an adaptive adversary is the following. Since $\mathcal{A}$ corrupts parties adaptively as the protocol progresses, the simulator $\mathcal{S}_{\mathrm{AMPC}}$ must deal with instructions from $\mathcal{A}$ to corrupt parties also as the simulation progresses. More specifically, to begin with, $\mathcal{S}_{\mathrm{AMPC}}$ must simulate to the adversary the messages generated by honest parties, without knowing the inputs of the honest parties. Then, if and when an honest party gets corrupted, the simulator learns the input (and possibly the output also) of this party; then, it needs to be able to equivocate and generate the state of the corrupted party in a way that is consistent with the revealed input/output and with the already simulated messages. Below, we construct our simulator $\mathcal{S}_{\mathrm{AMPC}}$.

At a high level, the simulator $\mathcal{S}_{\mathrm{AMPC}}$ will run a simulated copy of $\mathcal{A}$ and will use $\mathcal{A}$ in order to interact with $\mathcal{S}_{\mathrm{AMPC}}$ and $\mathcal{F}_{\mathrm{AMPC}}$. For this purpose, $\mathcal{S}_{\mathrm{AMPC}}$ will "simulate for $\mathcal{A}$" an interaction with parties running

Protocol Π_{MPC}, where the interaction will match the inputs and outputs seen by $\mathcal{Z}$ in its interaction with $\mathcal{S}_{\mathrm{AMPC}}$ in the ideal process for $\mathcal{F}_{\mathrm{AMPC}}$.

More specifically, $\mathcal{S}_{\mathrm{AMPC}}$ behaves as follows. In the following we use Π_{MPC} as a shorthand of $\Pi_{\mathrm{MPC}}{}^{\mathcal{F}_{\text{BROADCAST}},\mathcal{F}_{\text{KEY-DIST}},\mathcal{F}_{\text{COM-ZK}}}$.

Simulating the communication with $\mathcal{Z}$: Every input value that $\mathcal{S}_{\mathrm{AMPC}}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment). Likewise, every output value written by $\mathcal{A}$ on its output tape is copied onto $\mathcal{S}_{\mathrm{AMPC}}$'s own output tape (to be read by $\mathcal{S}_{\mathrm{AMPC}}$'s environment $\mathcal{Z}$).

Simulated CRS: The common reference string is chosen by $\mathcal{S}$ in the following manner (recall that $\mathcal{S}$ chooses the CRS for the simulated $\mathcal{A}$ as we are in the $\mathcal{F}_{\text{KEY-DIST}}$-hybrid model):

- $\mathcal{S}_{\mathrm{AMPC}}$ simulates the **Key-Generation** phase of the QFHE scheme as follows. It samples $(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}) \leftarrow \mathsf{KeyGen}^*(1^\lambda)$ where $\widetilde{\mathsf{SK}}$ includes sk secret shared among the parties.
- $\mathcal{S}_{\mathrm{AMPC}}$ computes additive secret shares $(\mathsf{sk}_1, \dots, \mathsf{sk}_n)$ of sk. $^{(\mathsf{sk1})}$. For $i = 1, \dots, n$, compute $c_i = \mathsf{Com}(\mathsf{sk}_i; r_i)$.
- Moreover, $\mathcal{S}_{\mathrm{AMPC}}$ generates randomness $(r_1^0, \dots, r_{2n}^0) \leftarrow \mathcal{D}_{rand}(1^\lambda)$ which may be used in the **Load** commands for equivocation.

$\mathcal{S}_{\mathrm{AMPC}}$ sets the CRS equal to $(\widetilde{\mathsf{PK}})$ and locally stores $(\widetilde{\mathsf{SK}}, \{\mathsf{sk}_i\}_{i\in[n]})$.

Simulating actual protocol messages in $\Pi_{\mathbf{MPC}}$: Note that there might be multiple sessions executing concurrently. Let sid be the session identifier for one specific session. We will specify the simulation strategy corresponding to this specific session. The simulator strategy for all other sessions will be the same. Let $\mathcal{P} = \{P_1, \dots, P_n\}$ be the set of parties participating in the execution of Π corresponding to the session identified by the session identifier sid. Some of the parties may be corrupted. Also, recall that we are in the setting of adaptive corruption so more parties could be corrupted as the protocol proceeds. At any point $\mathcal{S}$ only generates messages on behalf of the honest parties.

Simulation of the Load stage: Note that we describe how to simulate the **Load** stage for the encryption algorithm $\mathsf{QEnc}_{\widetilde{\mathsf{PK}}}$, since for all loads with the algorithm $\mathsf{QEnc}'_{\widetilde{\mathsf{PK}}}$, the simulator actually would be knowing the values to load, and hence, loading in the latter case can be performed honestly as per the protocol.

Load stage messages $\mathcal{S}_{\mathrm{AMPC}} \to \mathcal{A}$: In this stage the simulator $\mathcal{S}_{\mathrm{AMPC}}$ must generate messages on behalf of the honest parties. Therefore, $\mathcal{S}_{\mathrm{AMPC}}$ for every honest party P_i proceeds as follows:

X_i: Computes $X_i = \mathsf{QEnc}_{\mathsf{pk},\widetilde{K}}(0, 0; r_i^0)$ (where, r_i^0 was generated by $\mathcal{S}_{\mathrm{AMPC}}$ in the Initialize stage) and broadcasts X_i.

Commitment phase in $\mathcal{F}_{\text{COM-ZK}}$: Recall that in the protocol, if P_i is honest then, for every $j \neq i$, it would send $(\text{Commit}, sid, cid_1, cid_2, P_i, P_j, x_i, r_{x_i})$ to $\mathcal{F}_{\text{COM-ZK}}$ upon which P_j receives $(\text{Receipt}, sid, cid_1, cid_2, P_i, P_j)$; $\mathcal{S}_{\mathrm{AMPC}}$ simulates this interaction simply by sending the latter Receipt message to P_j. Moreover, for $j \neq i$, P_i sends $(\text{Prover}, sid, \mathcal{R}_{eq}, (\widetilde{\mathsf{PK}}, X_i))$ to $\mathcal{F}_{\text{COM-ZK}}$ for the relation $\mathcal{R}_{eq} = \{((\widetilde{\mathsf{PK}}, X_i), (x_i, r_{x_i})) : X_i = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}$
$(0, x_i; r_{x_i})\}$.

Prove phase in $\mathcal{F}_{\text{COM-ZK}}$: $\mathcal{S}_{\mathrm{AMPC}}$ simulates this interaction by sending $(\text{Proof}, sid, P_i, \mathcal{R}_{eq}, (\widetilde{\mathsf{PK}}, X_i))$ to P_j.

Load stage messages $\mathcal{A} \to \mathcal{S}_{\mathrm{AMPC}}$: Also in the load stage the adversary $\mathcal{A}$ generates the messages on behalf of corrupted parties. For each corrupted party P_i our simulator proceeds as follows: Let us consider the case when P_i is corrupted before the honest parties output $(cid, X_i, \text{Defined})$. If some proof is not accepted, then input (cid, X_i, Fail) to $\mathcal{F}_{\mathrm{AMPC}}$. On the other hand, if the proofs are accepted, P_i must have sent to $\mathcal{F}_{\text{COM-ZK}}$ the messages $(\text{Commit}, sid, cid_1, cid_2, P_i, P_j, x_i, r_{x_i})$. $\mathcal{S}_{\mathrm{AMPC}}$, which intercepts these messages, learns x_i and inputs $(\text{Input}, P_i, X_i, x_i)$ to $\mathcal{F}_{\mathrm{AMPC}}$.

Simulating corruption of parties in Load stage: When $\mathcal{A}$ corrupts a real world party P_i, then $\mathcal{S}_{\mathrm{AMPC}}$ first corrupts the corresponding ideal world party P_i and obtains its input x_i. Next $\mathcal{S}_{\mathrm{AMPC}}$ prepares the internal state on behalf of P_i such that it will be consistent with the message X_i that it had provided to $\mathcal{A}$ earlier. In particular, it needs to present to $\mathcal{A}$ the random coins $r_{state_i} \leftarrow \mathsf{Inv}_{\mathcal{D}_{rand}}(r_i^{\mathsf{blind}})$ that it can claim as the ones used in generating X_i and that is consistent with x_i as plaintext, e.i. $X_i = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0, x_i; r_i^{\mathsf{blind}})$. Specifically, $\mathcal{S}_{\mathrm{AMPC}}$ proceed as follows:

X_i: $\mathcal{S}_{\mathrm{AMPC}}$ runs the algorithm $\mathsf{Equiv}(0, \widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, X_i, r^0, x_i)$ where $X_i = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0, 0; r_i^0)$ in order to obtain randomness e_i. Furthermore, if P_i is corrupted before **Load** begins, then $\mathcal{S}_{\mathrm{AMPC}}$ inputs $(\mathrm{Input}, P_i, X_i, 0)$ to $\mathcal{F}_{\mathrm{AMPC}}$ on behalf of P_i and simulates the honest parties for this **Load** by following the protocol. If any of the simulated honest parties outputs $(cid, X_i, \mathrm{Defined})$, then the simulator must at the end of the **Load** input (Change, x_i') to $\mathcal{F}_{\mathrm{AMPC}}$ to define X_i. The value of x_i' is determined as follows: Since $\mathcal{F}_{\mathrm{AMPC}}$ has output $(cid, X_i, \mathrm{Defined})$, P_i must have input $(\mathrm{Input}, P_i, X_i, s)$. The interface $\mathcal{S}_{\mathrm{AMPC}}$ thus learns s and sets x_i' to be s.

Commitment phase in $\mathcal{F}_{\mathrm{COM\text{-}ZK}}$: If P_i gets corrupted after $\mathcal{S}_{\mathrm{AMPC}}$ sends $(\mathrm{Receipt}, sid, cid_1, cid_2, P_i, P_j)$ to P_j, then $\mathcal{S}_{\mathrm{AMPC}}$ learns x_i, and sends x_i to $\mathcal{A}$. Moreover, $\mathcal{S}_{\mathrm{AMPC}}$ would run the algorithm Equiv to patch r_{x_i} and send it to $\mathcal{A}$. If P_i gets corrupted before $\mathcal{S}_{\mathrm{AMPC}}$ sends $(\mathrm{Receipt}, cid_1, cid_2, P_i, P_j)$ to P_j, then $\mathcal{A}$ specifies x_i', and $(sid, cid_1, P_i, P_j, x_i')$ is recorded. Furthermore, $\mathcal{A}$ specifies r_{x_i}', and $(sid, cid_1, cid_2, P_i, P_j, x_i', r_{x_i}')$ is recorded. For $j \neq i$, P_i sends $(\mathrm{Prover}, sid, \mathcal{R}_{eq}, (\widetilde{\mathsf{PK}}, X_i))$ to $\mathcal{F}_{\mathrm{COM\text{-}ZK}}$ for the relation $\mathcal{R}_{eq} = \{((\widetilde{\mathsf{PK}}, X_i), (x_i, r_{x_i})) : X_i = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0, x_i; r_{x_i})\}$.

Prove phase in $\mathcal{F}_{\mathrm{COM\text{-}ZK}}$: If P_i gets corrupted after $\mathcal{S}_{\mathrm{AMPC}}$ sends $(\mathrm{Proof}, sid, P_i, \mathcal{R}_{eq}, (\widetilde{\mathsf{PK}}, X_i))$ to P_j, then $\mathcal{S}_{\mathrm{AMPC}}$ would learn x_i and run the algorithm $\mathsf{Equiv}(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, X_i, r^0, x_i)$ to obtain e_i. Here, it would send (x_i, e_i) to $\mathcal{A}$.

Simulation of the Evaluation stage: Recall that the **Evaluation** stage does not require any interaction among the parties. Let ckt be the arithmetic circuit to be computed on the n inputs of the parties. On behalf of every honest party, the simulator $\mathcal{S}_{\mathrm{AMPC}}$ computes $\mathsf{enc}(z) \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathrm{ckt}, X_1, \ldots, X_n)$.

Simulation of the Output stage: In this stage the functionality $\mathcal{F}_{\mathrm{AMPC}}$ outputs (Output, z). Now, in order for the simulator to be able to enforce the final output to be z, the simulator exploits the rerandomization step. More specifically, the simulator will cheat in the randomization step in such a way that the resultant 'rerandomized' ciphertext is an encryption of z. With this, the simulator can then simulate the distributed decryption protocol by simply behaving honestly. In detail, $\mathcal{S}_{\mathrm{AMPC}}$ proceeds as follows.

Output stage messages $\mathcal{A} \to \mathcal{S}_{\mathrm{AMPC}}$: On behalf of every corrupted party P_i, the simulator sends $(\mathrm{Output}, \mathsf{enc}(z))$ to $\mathcal{F}_{\mathrm{AMPC}}$. Then, $\mathcal{F}_{\mathrm{AMPC}}$ returns $(\mathsf{enc}(z), z)$. $\mathcal{S}_{\mathrm{AMPC}}$ thus learns z.

Output stage messages $\mathcal{S}_{\mathrm{AMPC}} \to \mathcal{A}$: $\mathcal{S}_{\mathrm{AMPC}}$ proceeds as follows.

1. For every honest party P_i, compute $Y_i = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0, 0; r_{n+i}^0)$ (where, r_{n+i}^0 was generated in the Initialize stage).
   Simulating corruption of parties in this step: If P_i gets corrupted soon after, then $\mathcal{S}_{\mathrm{AMPC}}$ patches P_i's state to y_i running the algorithm $\mathsf{Equiv}(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, Y_i, r_{n+i}^0, y_i)$. Also, from the **Load** performed by every corrupted party P_j, learn y_j as in the simulation of the **Load** stage.
2. Let P_k be an honest party. For every other honest party, assign $y_i \in M$. Then for P_k, set $y_k' = z - \sum_{i \in [n] \setminus \{k\}} y_i$. Then, for every honest party P_i, proceed as per the protocol to load $\widetilde{\mathsf{enc}(y_i)}$.
   Simulating corruption of parties in this step: If any honest party gets corrupted, then the random coins used for generating the encryption using key R are patched running the algorithm Equiv so that the value encrypted is y_i.
3. Now randomize the ciphertext $\mathsf{enc}(z)$ from the **Evaluation** stage as per the protocol. That is, compute $\mathrm{CT} = \mathsf{Rand}(\mathsf{enc}(z), \{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J})$ where $\mathsf{enc}(z)$ is the resultant encryption from the **Evaluation** stage.
4. Run Π_{DDEC} as in the protocol to decrypt CT. At a high level, note that as every party is required to prove the correctness in computing the evaluation shares, then with high probability, all the evaluation shares correspond to being computed using a set of valid shares of the secret key. This ensures towards correctness of the value decrypted and output at the end.[(sk2)]

This completes the description of our simulator. We shall now prove via a hybrid argument that the environment's view generated by the simulator is indistinguishable from its view in the real world. We begin by giving a high-level intuition of the proof.
Let us begin with the ideal world and then via hybrids migrate to the real world. In other words, we will modify the simulator hybrid-by-hybrid such that we finally reach a modified simulator that, on behalf of the honest parties, just honestly runs the protocol. Before we embark on actual proof, we shall first list the obstacles in this migration that shall guide us in designing the sequence of the hybrids. Notice that the ciphertext ct in the ciphertext randomization property as per Definition 3 corresponds to the real output $\mathsf{enc}(z)$ of the protocol Π_{MPC} and the ciphertexts $\text{ct}'_1, \ldots, \text{ct}'_n$ correspond to the ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$. Moreover, by the ciphertext randomization property observe the fact that, in the real-world the ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ generated by the algorithm $\mathsf{QEnc}_{\mathsf{PK}}$ for $b = 1$ implicitly do not contribute to the decryption of the final ciphertext CT (since it decrypts to the plaintext derived only by the ciphertext ct generated by $\mathsf{QEnc}_{\mathsf{PK}}$). In this case without loss of generality we will call the ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ `redundant`. On the other hand, in the ideal-world the final ciphertext CT decrypts to a value contributed only by the ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$. In this case we will call the ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ $\texttt{non} - \texttt{redundant}$.
This implies that an honest execution of the **Output** stage is not possible with the ciphertexts of $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ being $\texttt{non} - \texttt{redundant}$. Analogously, the ciphertext $\mathsf{enc}(z)$ can be either `redundant` or $\texttt{non} - \texttt{redundant}$.
In other words, it is pertinent that before we get to a hybrid where the **Output** stage is performed honestly, we need a hybrid where $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ turn to `redundant` ciphertexts. However, with both ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ and $\mathsf{enc}(z)$ `redundant`, we can not hope to get the final output CTto decrypt to the actual output value. Thus, even before turning $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ to `redundant` ciphertexts, we need a hybrid where we can cheat in the final decryption. That is, we first need to have a hybrid that, instead of running the distributed decryption protocol, runs what we abstract as the simulator for the distributed decryption. Finally, in order to ensure indistinguishability between the hybrids where $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ are `redundant` or not, we need a reduction to the IND-CPA security of the $\widetilde{\mathsf{QFHE}}$ scheme. In light of this, we also need to ensure that, by the time we reach the hybrid where $\{\widetilde{\mathsf{enc}(y_i)}\}_{i \in J}$ turns to `redundant`, the modified simulator does not crucially use the secret key in any part of the execution. With this as our guide, we have the following sequence of hybrids.

Hyb_0**:** This hybrid is identical to the ideal-world.
Trivially,

Lemma 1. $\text{IDEAL}^{\mathcal{F}_{\text{BROADCAST}}, \mathcal{F}_{\text{COM-ZK}}}_{\mathcal{F}_{\text{AMPC}}, \mathcal{S}_{\text{AMPC}}, \mathcal{Z}} \equiv \text{Hyb}_0$.

Hyb_1**:** This hybrid is the same as Hyb_0, except for the following modification in the way the hybrid computes the evaluation shares for the final rerandomized ciphertext. Recall that in Hyb_0, Π_{DDEC} was run. Now, we introduce certain changes in the steps of execution of Π_{DDEC}. Recall that every party P_i is first required to commit through $\mathcal{F}_{\text{COM-ZK}}$, the secret-key share sk_i and the commitment information r_i, where $c_i = \mathsf{Com}(\mathsf{sk}_i; r_i)$. The hybrid first intercepts these commit messages from the corrupted parties and learns sk_i, r_i. If these values do not correspond to the actual values provided to the corrupted party P_i during the onset of the execution, then, the hybrid aborts. Otherwise, it proceeds as follows. Recall that every party P_i is also required to commit to r_{ev_i} through $\mathcal{F}_{\text{COM-ZK}}$. The hybrid intercepts this commit command from every corrupted P_i and learns r_{ev_i}. Then it computes by itself the evaluation share that would result by using sk_i and randomness r_{ev_i} on the final ciphertext. Let these values be $\{ev_i\}_{i \in \text{C}}$. Having also learnt the output z of the loaded inputs of all the parties, the hybrid computes $\{ev_i\}_{i \in [n] \setminus \text{C}} \leftarrow \mathcal{S}_{\mathsf{Eval}}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i \in \text{C}}, \text{CT}, z, \{ev_i\}_{i \in \text{C}})$, where, $\mathcal{S}_{\mathsf{Eval}}$ simulates the evaluation in the distributed decryption as per Definition 7 where an actual implementation of it can be derived since we are using additive secret sharing and the adversary can corrupt at most $n-1$ parties. In the meanwhile, it simply simulates the commit messages it needs to send by sending to the adversary, the corresponding Receipt messages. Then, $\{ev_i\}_{i\in[n]\setminus \mathrm{C}}$ is presented as the evaluation shares of the honest parties.

Lemma 2. $\mathrm{Hyb}_0 \approx_{\mathrm{s}} \mathrm{Hyb}_1$.

Proof. Before we proceed, we shall analyze the potential abort by Hyb_1 when it intercepts the commit messages by a corrupted party P_i to sk_i, r_i. Recall that if these values do not match the corresponding values provided to P_i at the onset of the execution, then the hybrid aborts. We argue that even in Hyb_1, this would have resulted in a premature abort, since, applying the security of $\mathcal{F}_{\text{COM-ZK}}$, the corrupted party P_i could not have provided a convincing proof as the statement would be invalid. Here, we note that until this point in the course of execution, both the hybrids in question are identical. Next, note that if the corrupted parties provide the evaluation shares computed indeed using the randomness r_{ev_i} committed via $\mathcal{F}_{\text{COM-ZK}}$, then, we have the following by applying the property of $\mathcal{S}_{\mathsf{Eval}}$. The evaluation shares of the honest parties computed using $\mathcal{S}_{\mathsf{Eval}}$, jointly with the evaluation shares of the corrupted parties, are distributed statistically close to their values in Hyb_0. On the other hand, that is if the evaluation shares computed by the corrupted parties do not correspond to the values it had committed earlier through $\mathcal{F}_{\text{COM-ZK}}$, then the execution would anyway have aborted, again by applying the security of $\mathcal{F}_{\text{COM-ZK}}$. Thus, the modification introduced in hybrid Hyb_1 introduces only statistical distance in the view generated by the simulator, thus proving the lemma.

Hyb_2**:** This hybrid is the same as Hyb_1, except for the way public key and the secret-key shares are computed. Before we proceed, recall that this is performed in the same way as $\mathcal{F}_{\text{KEY-DIST}}$ in Hyb_1. Now in the current hybrid, the public key and the secret-key shares are computed as follows. Firstly, run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$. Then sample at random $\mathsf{sk}_1, \ldots, \mathsf{sk}_n \leftarrow \{0,1\}^*$ of appropriate length. Then, the hybrid commits to these secret-key shares, as in Hyb_1, using Com to obtain $c_1, \ldots, c_n$. The rest of the hybrid remains the same as Hyb_1.

Lemma 3. $\mathrm{Hyb}_1 \approx_{\mathrm{s}} \mathrm{Hyb}_2$.

Proof. Recall that the adversary can corrupt at most $n-1$ parties. Hence, the values received by an adversary from $\mathcal{F}_{\text{KEY-DIST}}$ in Hyb_1 are: $\big((\mathsf{pk}, c_1, \ldots, c_n), \{(\mathsf{sk}_i, r_i)\}_{i\in \mathrm{C}}\big)$. Firstly, we observe that $(\mathsf{pk}, \{(\mathsf{sk}_i)\}_{i\in \mathrm{C}})$ as output by $\mathcal{F}_{\text{KEY-DIST}}$ are distributed identically to the output of the following process: $(\mathsf{pk}, \cdot) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$ and $\forall i \in \mathrm{C}$, $\mathsf{sk}_i \leftarrow \{0,1\}^*$ of appropriate length. Furthermore, we recall that Com is a statistically hiding commitment. Thus, clearly, the distribution of $\big((\mathsf{pk}, c_1, \ldots, c_n), \{(\mathsf{sk}_i, r_i)\}_{i\in \mathrm{C}}\big)$ as output by $\mathcal{F}_{\text{KEY-DIST}}$ is statistically close to the joint distribution of these values as generated by Hyb_2. Hence, the lemma.

Hyb_3**:** This hybrid is the same as Hyb_2, except that the simulator turn the ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i\in J}$ to `redundant`.

Lemma 4. $\mathrm{Hyb}_2 \approx_{\mathrm{c}} \mathrm{Hyb}_3$.

Proof. Note that in an earlier hybrid, we have introduced the modification from crucially using the fact that ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i\in J}$ are $\texttt{non}-\texttt{redundant}$ (to be able to cheat in the rerandomization of the final ciphertext) to just running $\mathcal{S}_{\mathsf{Eval}}$ for which the actual value of the plaintexts in $\{\widetilde{\mathsf{enc}(y_i)}\}_{i\in J}$ will not matter. Furthermore, we have also introduced the modification from obtaining the secret-key shares by simply sampling them uniformly at random, a process that does need the secret key. Thus, in the current hybrid Hyb_3, the simulator does not make use of the secret key anywhere in its execution. Thus, an algorithm that distinguishes hybrids Hyb_2 and Hyb_3 is directly reducible to an adversary against threshold IND-CPA security of the QFHE scheme. Hence, the lemma.

Hyb_4**:** This hybrid is the same as Hyb_3, except that the simulator is given the actual inputs of the honest parties and the simulator simply executes the protocol to compute the messages that the honest parties are supposed to send to the adversary.

Lemma 5. $\mathrm{Hyb}_3 \equiv \mathrm{Hyb}_4$.

Proof. Observe that in both the hybrids Hyb_3 and Hyb_4, both the ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i\in J}$ and $\mathsf{enc}(z)$ are $\texttt{redundant}$. Hence, no matter what the input values of the parties are, the **Load**ed variables all contain 0. Hence, the views generated by the simulator in these two hybrids are identical.

Hyb_5**:** This hybrid is the same as Hyb_4, except that the simulator now runs the algorithm KeyGen instead of KeyGen^*. In particular, the ciphertrxt $\mathsf{enc}(z)$ becomes $\texttt{non}-\texttt{redundant}$. Furthermore, in computing the loaded variables X_i in the **Load** stage, the simulator honestly follows the protocol; namely, for every honest party P_i, it samples $r_{x_i} \leftarrow \mathcal{D}_{rand}(1^\lambda)$ and computes $X_i \leftarrow \mathsf{QEnc}_{\mathsf{PK}}(0, x_i; r_{x_i})$, where x_i is the input of P_i. Also, when an honest party P_i gets corrupted, unlike Hyb_4 which patched the state of P_i in the computation of X_i by running the algorithm Equiv, the simulator in this hybrid simply presents $r_{state_i} \leftarrow \mathsf{Inv}_{\mathcal{D}_{rand}}(r_{x_i})$ as the state of P_i.

Lemma 6. $\mathrm{Hyb}_4 \approx_c \mathrm{Hyb}_5$.

Proof. The proof of this lemma immediately follows from Theorem 3 and more specifically, from the *indistinguishability of equivocal keys* and the *indistinguishability of equivocation.*
In particular:

- In hybrid Hyb_4, X_i is computed as: $X_i = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0, 0; r_i^0)$. Then for patching the state of P_i for input x_i, the simulator runs the algorithm $\mathsf{Equiv}(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}, X_i, r^0, x_i)$ to obtain e_i. Furthermore, in the output phase, while loading the random coins used to rerandomize the final ciphertext, Y_i is computed as $Y_i = \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0, 0; r_{n+i}^0)$. Then for patching the state of P_i for y_i, the simulator runs the algorithm $\mathsf{Equiv}(\widetilde{\mathsf{PK}}, \widetilde{\mathsf{SK}}), Y_i, r_{n+i}^0, y_i)$ to obtain e_{n+1}.
- On the other hand, in hybrid Hyb_5, X_i is computed as: $X_i = \mathsf{QEnc}_{\mathsf{PK}}(0, x_i; r_{x_i})$ where $r_{x_i} \leftarrow \mathcal{D}_{rand}(1^\lambda)$ and the ciphertext $\mathsf{enc}(z)$ is $\texttt{non}-\texttt{redundant}$. Also, Y_i is computed as: $Y_i = \mathsf{QEnc}_{\mathsf{PK}}(0, y_i; r_{y_i})$ where $r_{y_i} \leftarrow \mathcal{D}_{rand}(1^\lambda)$.

Now to show indistinguishability between the two hybrids, we can immediately use the proof of Theorem 3 and the *indistinguishability of Equivocation.*
Next, to argue indistinguishability between the $\texttt{non}-\texttt{redundant}$ and $\texttt{redundant}$ $\mathsf{enc}(z)$, we again directly use Theorem 3 and the *indistinguishability of equivocal keys.* In detail, we need to be able to obtain a reduction to threshold IND-CPA security of the QFHE scheme. Thus, we need to ensure that the reduction does not need the secret key in simulating either of the hybrids Hyb_4 and Hyb_5. To see this, we recall

the aspects of Hyb_4 due to which the secret key will not be needed by the reduction. Observe that in the hybrid Hyb_4, ciphertexts $\{\widetilde{\mathsf{enc}(y_i)}\}_{i\in J}$ are `redundant` just like in the protocol Π_{MPC}. We also emphasize that the simulator does not use the secret key for QFHE anywhere in its execution. Hence, given an adversary that distinguishes the hybrids Hyb_4 and Hyb_5 with some non-negligible property ϵ, we have an adversary that breaks threshold IND-CPA security of the QFHE scheme with probability negligibly close to ϵ. Hence, the lemma.

Hyb_6**:** This hybrid is the same as Hyb_5, except that, in computing the shares of the secret key at point ($\mathsf{sk}\mathbf{1}$), the simulator now switches back from using $\mathsf{KeyGen}_{\mathsf{FHE}}$ and uniformly sampling the secret-key shares to using $\mathsf{KeyGen}_{\mathsf{FHE}}$ and additively secret-sharing the secret key like in $\mathcal{S}_{\text{AMPC}}$.

Lemma 7. $\text{Hyb}_5 \equiv \text{Hyb}_6$.

Proof. This proof is similar to the proof of indistinguishability of the hybrids Hyb_1 and Hyb_2 (Lemma 3), where the simulator switched the other way; i.e., from using $\mathsf{KeyGen}_{\mathsf{FHE}}$[8] and additively secret-sharing the secret key to using $\mathsf{KeyGen}_{\mathsf{FHE}}$ and uniformly sampling the secret-key shares. On the same lines as in Lemma 3, we have Lemma 7.

Hyb_7**:** This hybrid is the same as Hyb_6, except that, in decrypting the final rerandomized ciphertext at point ($\mathsf{sk}\mathbf{2}$), the simulator at this hybrid switches back to decrypting as in Π_{DDEC}.

Lemma 8. $\text{Hyb}_6 \equiv \text{Hyb}_7$.

Proof. This proof is similar to the proof of indistinguishability of the hybrids Hyb_0 and Hyb_1 (Lemma 2), where the simulator switched the other way; i.e., from using Π_{DDEC} to using $\mathcal{S}_{\mathsf{Eval}}$. Namely, the property of simulatable evaluation ensures that the evaluation shares for the honest parties generated using $\mathcal{S}_{\mathsf{Eval}}$ are distributed statistically close to the evaluation shares generated using Π_{DDEC}, thus proving the lemma.

Observe that the view in the final hybrid Hyb_7 is identical to the real-world view. Hence, we have that $\text{Hyb}_7 \equiv \text{REAL}^{\mathcal{F}_{\text{BROADCAST}},\mathcal{F}_{\text{COM-ZK}}}_{\Pi_{\text{MPC}},\mathcal{Z}}$.
In summary, we have that,

$$\text{IDEAL}_{\mathcal{F}_{\text{AMPC}},\mathcal{S}_{\text{AMPC}},\mathcal{Z}} \approx_{\text{c}} \text{REAL}^{\mathcal{F}_{\text{BROADCAST}},\mathcal{F}_{\text{COM-ZK}}}_{\Pi_{\text{MPC}},\mathcal{Z}}$$

D Performance of General Solution based on IPS Compiler

Simulating corruption of parties in Load stage: The following should be taken with large grains of salt. We have tried to be optimistic on the part of the IPS compiler, to not give our concrete protocol an unfair advantage. Thus, actual numbers could be larger.

We estimate that using the best known outer and inner protocols in the IPS compiler, one invocation of IPS would require $10-15$ rounds. For the generic suggestion one needs two invocations, one to generate key material for NCE (see below) and one for decryption. On top of that one needs a few rounds for distributing inputs and proving knowledge of them in ZK or NIZK. So we estimate at least 30 rounds for the complete protocol.

The computation and communication overhead is even harder to estimate. We looked at communication since that is a lower bound on computation and made a crude estimate that equates statistical and computational security parameters. To do the FHE decryption generically, one needs to write it as a binary circuit,

[8] Recall that now $\mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$ is part of the $\mathsf{KeyGen}(1^\lambda)$ algorithm.

say of size s and then use the IPS compiler. For n players and security parameter λ, we get communication $\Omega(n^4\lambda^2 s)$ where s depends on the FHE scheme but can be expected to be at least quadratic in λ. This is based on a very optimistic assumption on what the outer protocol can do while also minimizing the number of rounds. If this is not true, then such a protocol yields an $\Omega(n^6\lambda^3 s)$ overhead.

In comparison the total communication of the decryption phase of our concrete protocol is $O(n^2\lambda)$. We used the IPS paper and there are likely ways to optimize, but it does seem that the difference is very significant nevertheless.

E Concrete Instantiation of our Equivocal QFHE Scheme

In this section we describe the concrete QFHE scheme, which is based on the somewhat homomorphic encryption scheme of Brakerski and Vaikuntanathan (BV) [BV11b]. The scheme is secure under the polynomial LWE (PLWE) assumption, which is a simplified version of the Ring-LWE assumption.

Definition 8 (PLWE Assumption). *For all $\lambda \in \mathbb{N}$, let $F(X) = F_\lambda(X) \in \mathbb{Z}[X]$ be a polynomial of degree $N = N(\lambda)$, let $q = q(\lambda) \in \mathbb{Z}$ be a prime integer, let $R = \mathbb{Z}[X]/\langle F(X)\rangle$ and $R_q = R/qR$, and let χ denote a distribution over the ring R. The* polynomial LWE assumption $\mathsf{PLWE}_{f,q,\chi}$ *states that for any $l = poly(\lambda)$ it holds that*

$$\{(a_i, a_i \cdot s + e_i)\}_{i\in[l]} \approx_c \{(a_i, u_i)\}_{i\in[l]}$$

where s is sampled from the distribution χ, and a_i, u_i are uniformly random in R_q. We require computational indistinguishability to hold given only l samples, for some $l = poly(\lambda)$.

Now we present the fully homomorphic encryption scheme QFHE and its threshold decryption procedure. Later we show that it indeed satisfies the properties listed in Definition 3. To begin with, the scheme is associated with the following parameters:

- A cyclotomic polynomial $F(X) := \Phi_m(X) = X^N + 1$ of degree $N := \phi(m)$, where $m = 2N$ and where the dimension N is a power of 2 and lower bounded by some function of the security parameter λ.
- The modulus q, which is a prime such that $q \equiv (mod\ 2N)$. Together, N, q and $F(X)$ define rings $R := \mathbb{Z}[X]/\langle F(X)\rangle$ and $R_q := R/qR = \mathbb{Z}_q[X]/\langle F(X)\rangle$. Addition in these rings is done component-wise in their coefficients (thus, their additive group is isomorphic to Z^N and Z_q^N, respectively). Multiplication is simply a polynomial multiplication modulo $F(X)$ (and also modulo q, in the case of the ring R_q). The two operations in R will be denoted by $+$ and $\cdot$.
- The error parameter σ, which defines a discrete Gaussian error distribution $\chi = \mathcal{D}_{\mathbb{Z}^N,\sigma}$ over the ring $R_q = \mathbb{Z}_q[X]/\langle F(X)\rangle$ with standard deviation σ. We usually refer to $\mathcal{D}_{\mathbb{Z}^N,\sigma}$ as $\mathcal{D}_{rand}(\lambda)$ used by the encryption algorithm to select the random coins needed during the encryption.
  The parameters λ, F, q and χ are public and we assume that given λ, there exist PPT algorithms that output F and q, and sample from the error distribution χ.
- A prime $p < q$, for some integer $p = p(\lambda)$ and rel. prime to q, which defines the message space M of the scheme as $R_p = \mathbb{Z}_p[X]/\langle F(X)\rangle$, i.e. the ring of integer polynomials modulo $F(X)$ and modulo p. Moreover, we encode messages from M to R_q. Namely, we encode our messages as elements in R_q with coefficients modulo p. More specifically, to transform a message $\mathbf{m} \in M$ into some $\mathbf{x} \in R_q$, we assume that there is an injective encoding function $\mathsf{encode} : M \to R_q$ which takes elements in M to elements in a ring R_q which is equal $\mathbb{Z}^N$ (as a $\mathbb{Z}$-module). We also assume a decoding function $\mathsf{decode} : R_q \to M$ which takes an *arbitrary* element in $\mathbb{Z}^N$ and returns an element in M. We require that the following conditions hold:
  1. $\forall m \in M : \mathsf{decode}(\mathsf{encode}(m)) = m$.
  2. $\forall x \in R_q : \mathsf{decode}(x) = \mathsf{decode}(x \mod p)$.
  3. $\forall m \in M : \|\mathsf{encode}(m)\|_\infty \le \tau$ where $\tau = p/2$.
  4. $\forall m_1, m_2 \in M : \mathsf{decode}(\mathsf{encode}(m_1) + \mathsf{encode}(m_2)) = m_1 + m_2$ and $\mathsf{decode}(\mathsf{encode}(m_1) \cdot \mathsf{encode}(m_2)) = m_1 \cdot m_2$.
- A number $D > 0$, which defines a bound on the maximum number of multiplications that can be performed correctly using the scheme.

The above parameters depend on the security parameter λ in a way to guarantee correctness and security. Our *special* FHE scheme consists of a tuple $(\mathsf{KeyGen}_{\mathsf{FHE}}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ of algorithms defined below, and parametrized by a security parameter λ.

$\underline{\mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)}$: Sample ring elements $a \leftarrow R_q$ and $s, e \leftarrow \mathcal{D}_{rand}(\lambda)$, s, e are rounded such that they can be seen as $s, e \in R_q$. Then compute $b \leftarrow ((a \cdot s) + (p \cdot e))$. The public and private keys are then set to be $\mathsf{pk} \leftarrow (a, b)$ and $\mathsf{sk} \leftarrow \mathbf{s}$ where $\mathbf{s} = (1, s, s^2, \ldots, s^D) \in R_q^{D+1}$.

$\underline{\mathsf{Enc}_{\mathsf{pk}}(x; r)}$:On input $x = \mathsf{encode}(m)$ where $m \in M$, and $r \leftarrow \mathcal{D}_{rand}(\lambda)$, we proceed as follows: The element r is parsed as $(u, v, w) \in R_q^3$. Then it computes $c_0 \leftarrow (b \cdot v) + (p \cdot w) + x$ and $c_1 \leftarrow (a \cdot v) + (p \cdot u)$ and returns the ciphertext $\mathbf{ct} = (c_0, c_1) \in R_q^2$. The algorithm only generates ciphertexts $\mathbf{ct} \in R_q^2$, but homomorphic operations might add more elements to the ciphertext. Thus the most generic form of a decryptable ciphertext in our scheme is $\mathbf{ct} = (c_0, \ldots, c_d)$ for $d \leq D$. [9] When applying this algorithm one would obtain $x = \mathsf{encode}(m)$. This is what we mean when we write $\mathsf{Enc}_{\mathsf{pk}}(m, r)$, where $m \in M$.
$\underline{\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct})}$: Given a secret key $\mathsf{sk} = \mathbf{s}$ and a ciphertext $\mathbf{ct} = (c_0, \ldots, c_D) \in R_q^{D+1}$, the decryption algorithm computes $\tilde{t} = \langle \mathbf{s}, \mathbf{ct} \rangle = \sum_{i=0}^{D} c_i s^i \mod q \in R_q$. Then the decryptor simply reduces $t = \tilde{t} \mod p$, which can then be decoded to m. Note that the condition for correct decryption is that $\|\tilde{t}\|_\infty$ is smaller than $q/2$.

$\underline{\mathsf{Eval}_{\mathsf{pk}}(\text{ckt}, \mathbf{ct}, \mathbf{ct}')}$: To compute an arbitrary function homomorphically, we construct an arithmetic circuit ckt (made of addition and multiplication operations over Z_t), and then use Add and Multiply to iteratively evaluate ckt on encrypted inputs. To this end, we show how to homomorphically add and multiply two elements in $\mathbb{Z}_t$.

– $\underline{\text{Add}_{\mathsf{pk}}(\mathbf{ct}, \mathbf{ct}')}$:: Let $\mathbf{ct} = (c_0, \ldots, c_\delta)$ and $\mathbf{ct}' = (c'_0, \ldots, c'_\gamma)$ be the two ciphertexts (If $\gamma \neq \delta$, we pad the shorter ciphertext with zeroes). Then compute and output
  $\mathbf{ct}_{Add} = (c_0 + c'_0, \ldots, c_{max(\gamma,\delta)} + c'_{max(\gamma,\delta)}) \in R_q^{max(\gamma,\delta)+1}$
– $\underline{\text{Multiply}_{\mathsf{pk}}(\mathbf{ct}, \mathbf{ct}')}$:: Let $\mathbf{ct} = (c_0, \ldots, c_\delta)$ and $\mathbf{ct}' = (c'_0, \ldots, c'_\gamma)$ be the two ciphertexts. Here, we do not pad either of the ciphertexts with zeroes. Let h be a symbolic variable and consider the expression $(\sum_{i=1}^{\delta} c_i h^i) \cdot (\sum_{i=1}^{\gamma} c'_i h^i)$ over R_q. We can (symbolically, treating h as an unknown variable) open the parentheses to compute $\hat{c}_0, \ldots, \hat{c}_{\delta+\gamma}$ such that $(\sum_{i=1}^{\delta} c_i h^i) \cdot (\sum_{i=1}^{\gamma} c'_i h^i) = (\sum_{i=1}^{\delta+\gamma} \hat{c}_i h^i)$. Therefore, output $\mathbf{ct}_{Mult} = (\hat{c}_0, \ldots, \hat{c}_{\delta+\gamma})$.

In order to achieve full homomorphism one can use Gentry's "bootstrapping" and "squashing" techniques. Another way, as an alternative to squashing, is the "re-linearization" technique. See [BV11b,BV11a] for more details.

Distributed Decryption: We now extend the scheme above to enable distributed decryption. The functionality $\mathcal{F}_{\text{KEY-DIST}}$ generates a key pair and secret-shares the secret key among the players using an additive secret-sharing scheme. Hence, each party P_i will receive a share $\mathsf{sk}_i = s_i$, chosen uniformly such that $s = s_1 + \cdots + s_n$. More specifically, the decryption protocol is described in Figure 8.

[9] Padding with zeros does not effect the ciphertext. More specifically, $(c_0, \ldots, c_d) \equiv (c_0, \ldots, c_d, 0, \ldots, 0)$.

Protocol Π_{DDec}

The distributed decryption proceeds as follows:

Key Sharing:

The invocation of $\mathcal{F}_{\text{Key-Dist}}$ results in every party P_i receiving $\big((\mathsf{pk}, c_1, \ldots, c_n), (\mathsf{sk}_i, r_i)\big)$, where, $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ are shares of the secret key s corresponding to the public key $\mathsf{pk} = (a, b)$ and $(c_1, \ldots, c_n)$ are commitments on the corresponding shares. In particular, $\mathsf{sk}_i = s_i$, chosen uniformly such that $s = s_1 + \cdots + s_n$.

Evaluation Sharing:

1. Given the ciphertext $\mathbf{ct} = (c_0, c_1) \in R_q^2$, party P_1 computes $v_i \leftarrow c_0 - (s_i \cdot c_1)$ and each other party P_i computes $v_i \leftarrow -(s_i \cdot c_1)$.
2. Compute $ev_i \leftarrow v_i + p \cdot r_{ev_i}$ where $r_{ev_i} \in R_q$ is a random element with $\|r_{ev_i}\|_\infty \leq B_{dec}$.
3. Each party P_i broadcasts ev_i.
4. P_i sends $(\text{Prover}, sid, P_i, P_j, \mathcal{R}_{eval}, (c_i, \mathsf{pk}, \mathbf{ct}, ev_i))$ to $\mathcal{F}_{\text{Com-ZK}}$ for the relation $\mathcal{R}_{eval} = \{((c_i, \mathsf{pk}, \mathbf{ct}, ev_i), (\mathsf{sk}_i, r_i, r_{ev_i})) : c_i = \mathsf{Com}(\mathsf{sk}_i; r_i) \wedge (ev_i = v_i + p \cdot r_{ev_i} \wedge (v_i = c_0 - (s_i \cdot c_1),\ \text{if } i = 1 \vee v_i = -(s_i \cdot c_1),\ \text{if } i = 0))\}$.

Share Combining:

1. All players compute $t' \leftarrow ev_1 + \cdots + ev_n$ and obtain a message $m' \leftarrow \mathsf{decode}(t' \bmod p)$.

Fig. 8. The threshold decryption protocol.

Equivocal FHE*:* Given the above *special* FHE scheme, we can define our $\mathsf{QFHE} = (\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{Equiv})$ scheme where the algorithms $(\mathsf{KeyGen}, \mathsf{KeyGen}^*, \mathsf{QEnc}, \mathsf{Equiv})$ are as described in Figure 1. Note that *indistinguishability of equivocation* and *indistinguishability of equivocal keys* are shown in Theorem 3.

E-Hiding: We should point out that the scheme of [BV11b] enjoys formula privacy. The idea is that adding to a given ciphertext an encryption of zero with an error super-polynomially larger than the error used in usual ciphertexts results in a ciphertext that still decrypts to the same result but statistically hides which ciphertext was initially given. Such a property is typically used to blind a ciphertext after a computation so that the final ciphertext only provides information about the result of the computation and not about how this result is obtained. Hence, it is easy to show that the *E-Hiding* property defined in Definition 3 can be argued as formula privacy for the above scheme.

Next, the only thing we need to argue is privacy and correctness of the distributed decryption protocol. In particular, we need to guarantee correct and private distributed decryption computing the bound B_{dec} as a function of all the other parameters. In order to make a choice for B_{dec} one can follow the line of analysis in [DPSZ12], however, in our case a simpler analysis can be followed since we do not need the SIMD approach, used by [DPSZ12], to handle many values in parallel in a single ciphertext.

Invertible Sampling: It is known how to do invertible sampling for Gaussian distributions suitable for our case using rejection sampling over the effective support of the distribution [OPW11,Pei14].

Homomorphism over random coins. Next we prove the property of *homomorphism over random coins* property defined in definition 4.

Lemma 9. (Homomorphism over random coins). $\forall (x_0, x_1, x_2) \in R_q^3$, $\forall (r_1, r_2) \in \mathcal{D}_{rand}^2$ *and* $\forall \mathsf{pk} = (a, b) \leftarrow \mathsf{KeyGen}_{\mathsf{FHE}}(1^\lambda)$ *it holds that:*

$$\big(x_0 \boxdot \mathsf{Enc}_{\mathsf{pk}}(x_1; r_1)\big) \boxplus \mathsf{Enc}_{\mathsf{pk}}(x_2; r_2) = \mathsf{Enc}_{\mathsf{pk}}(x_0 \cdot x_1 + x_2; x_0 \cdot r_1 + r_2)$$

Proof. By definition $\mathsf{Enc}_{\mathsf{pk}}(x_i; r_i) = (c_{0,i}, c_{1,i}) = (b \cdot v_i + p \cdot w_i + x_i, a \cdot v_i + p \cdot u_i)$ for $i = 0, 1$ where r_i is parsed as $(u_i, v_i, w_i) \in R_q^3$.

$$\begin{aligned}
&\big(x_0 \boxdot \mathsf{Enc}_{\mathsf{pk}}(x_1; r_1)\big) \boxplus \mathsf{Enc}_{\mathsf{pk}}(x_2; r_2) \\
&\quad = \big(x_0 \boxdot (b \cdot v_1 + p \cdot w_1 + x_1, a \cdot v_1 + p \cdot u_1)\big) \boxplus (b \cdot v_2 + p \cdot w_2 + x_2, a \cdot v_2 + p \cdot u_2) \\
&\quad = (x_0 \cdot b \cdot v_1 + x_0 \cdot p \cdot w_1 + x_0 \cdot x_1, x_0 \cdot a \cdot v_1 + x_0 \cdot p \cdot u_1) \boxplus (b \cdot v_2 + p \cdot w_2 + x_2, a \cdot v_2 + p \cdot u_2) \\
&\quad = \big(b \cdot (x_0 \cdot v_1 + v_2) + p \cdot (x_0 \cdot w_1 + w_2) + x_0 \cdot x_1 + x_2, a \cdot (x_0 \cdot v_1 + v_2) + p \cdot (x_0 \cdot u_1 + u_2)\big) \\
&\quad = \mathsf{Enc}_{\mathsf{pk}}(x_0 \cdot x_1 + x_2; x_0 \cdot r_1 + r_2).
\end{aligned}$$

F Security Proof of the UC Adaptive Commitments from LWE

Proposition 2. *Assuming the hardness of LWE, Protocol Π_{COM} UC realizes $\mathcal{F}_{\mathrm{MCOM}}$ in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model.*

Proof. Let $\mathcal{A}$ be an active, adaptive adversary that interacts with parties running the protocol Π_{COM} in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model. We construct a simulator $\mathcal{S}$ (the ideal world adversary) with access to the ideal functionality $\mathcal{F}_{\mathrm{MCOM}}$, which simulates a real execution of Π_{COM} with $\mathcal{A}$ such that no environment $\mathcal{Z}$ can distinguish the ideal world experiment with $\mathcal{S}$ and $\mathcal{F}_{\mathrm{MCOM}}$ from a real execution of Π_{COM} with $\mathcal{A}$.

$\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\mathrm{MCOM}}$ and with the environment $\mathcal{Z}$. The ideal adversary $\mathcal{S}$ starts by invoking a copy of $\mathcal{A}$ and running a simulated interaction of $\mathcal{A}$ with the environment $\mathcal{Z}$ and the parties running the protocol. We refer to the interaction of $\mathcal{S}$ in the ideal process as external interaction. The interaction of $\mathcal{S}$ with the simulated $\mathcal{A}$ is called *internal interaction.* The committing party is denoted by P_i and the receiver party P_j. Moreover, let sid be the session identifier and $ssid$ the sub-session identifier.

Our simulator $\mathcal{S}$ proceeds as follows:

Simulating CRS: The common reference string is chosen by $\mathcal{S}$ in the following manner (recall that $\mathcal{S}$ chooses the CRS for the simulated $\mathcal{A}$ as we are in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model):

1. $\mathcal{S}$ runs the setup algorithm $\mathsf{KeyGen}^*(1^\lambda)$ of the equivocal QFHE encryption scheme obtaining a public key $\widetilde{\mathsf{PK}}$ and secret key $\widetilde{\mathsf{SK}}$.
2. $\mathcal{S}$ runs the setup algorithm for the CCA2-secure encryption scheme $\mathsf{E}_{\mathsf{CCA}}$, obtaining a public key pk_{cca} and a secret key sk_{cca}.

$\mathcal{S}$ sets the CRS to be $(\widetilde{\mathsf{PK}}, \mathsf{pk}_{cca})$ and locally stores $(\widetilde{\mathsf{SK}}, \mathsf{sk}_{cca})$.

Simulating the communication with $\mathcal{Z}$: Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Similarly, every output value written by $\mathcal{A}$ on its own output tape is directly copied to the output tape of $\mathcal{S}$.

Simulating Commit *commands where the committer P_i is uncorrupted* : The honest committer P_i on input $(\text{Commit}, sid, ssid, P_i, P_j, b)$ from the environment, writes this message on its outgoing tape for $\mathcal{F}_{\mathrm{MCOM}}$. Then $\mathcal{S}$ simulates P_i writing the Commit message of Protocol Π_{COM} on its outgoing tape for P_j. In particular, $\mathcal{S}$ knowing $\widetilde{\mathsf{SK}}$ computes $z \leftarrow \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0,0)$ along with two strings r_0 and r_1 (running the algorithm Equiv) such that r_b constitutes a decommitment of z to b. Next, $\mathcal{S}$ computes $C_0 \leftarrow \mathsf{ENC}_{\mathsf{CCA}}(P_i, P_j, sid, ssid, r_0)$ using random coins s_0, and $C_1 \leftarrow \mathsf{ENC}_{\mathsf{CCA}}(P_i, P_j, sid, ssid, r_1)$ using random coins s_1. Then, $\mathcal{S}$ stores (c, r_0, s_0, r_1, s_1) and simulates P_i writing $c = (sid, ssid, P_i, z, C_0, C_1)$ on its outgoing tape for P_j. When $\mathcal{A}$ delivers c from P_i to P_j in the internal simulation, then $\mathcal{S}$ delivers the message from the ideal process P_i's outgoing tape to $\mathcal{F}_{\mathrm{MCOM}}$. Furthermore, $\mathcal{S}$ also delivers the $(\text{Reveal}, sid, ssid, P_i, P_j, b)$ message from $\mathcal{F}_{\mathrm{MCOM}}$ to P_j. If $\mathcal{A}$ passively corrupts P_i, then $\mathcal{S}$ carries out the simulation as described here. If $\mathcal{A}$ corrupts P_i before delivering c and then changes c before delivering it, then $\mathcal{S}$ proceeds by following the instructions for a corrupted committer.

Simulating Reveal *commands where the committer P_i is uncorrupted* : The honest committer P_i on input $(\text{Reveal}, sid, ssid)$ from the environment, writes this message on its outgoing tape for $\mathcal{F}_{\mathrm{MCOM}}$ $\mathcal{S}$ then delivers this message to $\mathcal{F}_{\mathrm{MCOM}}$ and gets the message $(\text{Reveal}, sid, ssid, P_i, P_j, b)$ from $\mathcal{F}_{\mathrm{MCOM}}$. Then $\mathcal{S}$ given the value b, generates a simulated decommitment message $(sid, ssid, r_b, s_b, b)$, where r_b and s_b are as generated above. $\mathcal{S}$ then internally simulates for $\mathcal{A}$ the event where P_i writes this message on its outgoing tape for P_j . When $\mathcal{A}$ delivers this message from P_i to P_j in the internal interaction, then $\mathcal{S}$ delivers the $(\text{Reveal}, sid, ssid, P_i, P_j, b)$ message from $\mathcal{F}_{\mathrm{MCOM}}$ to P_j.

Simulating corruption of parties : When a command 'corrupt P_i' is issued, $\mathcal{S}$ first corrupts P_i and obtains the values of all its unopened commitments and prepares the internal state of P_i to be consistent with these commitment values in the same way as shown above.

Simulating Commit *commands where the committer P_i is corrupted* : When a corrupted party P_i sends a commitment message $(sid, ssid, P_i, z,$
$C_0, C_1)$ to an uncorrupted party P_j in the simulated interaction, then $\mathcal{S}$ checks if the commitment with identifiers $(sid, ssid)$ was sent before. If this is the case then $\mathcal{S}$ ignores the message. Otherwise, $\mathcal{S}$ must extract the commitment bit committed to by $\mathcal{A}$. To this end, $\mathcal{S}$ decrypts C_0 and C_1 and acts as follows depending on the decrypted values:

– If C_b for some $b \in \{0,1\}$ decrypts to $(P_i, P_j, sid, ssid, r)$ such that r is the decommitment information for z as a commitment to b, and C_{1-b} does not decrypt to a decommitment of $1-b$, then $\mathcal{S}$ stores the value b and sends $(\text{Commit}, sid, ssid, P_i, P_j, b)$ to $\mathcal{F}_{\text{MCom}}$, and sends $\mathcal{F}_{\text{MCom}}$'s Receipt message to P_j.
– If neither of C_0 and C_1 decrypt to $(P_i, P_j, sid, ssid, r)$ such that r is the decommitment information for z, then $\mathcal{S}$ does not store the value b since it will never be opened correctly, sends $(\text{Commit}, sid, ssid, P_i, P_j, 0)$ to $\mathcal{F}_{\text{MCom}}$ and sends $\mathcal{F}_{\text{MCom}}$'s Receipt message to P_j.
– If C_0 decrypts to $(P_i, P_j, sid, ssid, r_0)$ and C_1 decrypts to $(P_i, P_j, sid, ssid, r_1)$, where r_0 and r_1 are the decommitment information for z for the values 0 and 1, respectively and the identifiers in the decryption information are the same then $\mathcal{S}$ outputs a special $failure$ symbol.

Simulating Reveal commands where the committer is corrupted : When a corrupted party P_i sends a Reveal message $(sid, ssid, r, s, b)$ to an uncorrupted party P_j in the simulated interaction, then $\mathcal{S}$ checks if $(sid, ssid, P_i, z, C_0, C_1)$ is stored and that r and s are the decommitment information to b. If this is the case, then $\mathcal{S}$ sends $(\text{Reveal}, sid, ssid, P_i, P_j)$ to $\mathcal{F}_{\text{MCom}}$ and the Reveal message from $\mathcal{F}_{\text{MCom}}$ to P_j. Otherwise, $\mathcal{S}$ ignores the message.

Via a sequence of hybrids, we will prove that no environment can distinguish an interaction of Π_{Com} with $\mathcal{A}$ from an interaction in the ideal world with $\mathcal{F}_{\text{MCom}}$ and $\mathcal{S}$(as defined above). The sequence of hybrids follows the lines of the [CLOS02] proof since in place of their trapdoor commitment scheme we use our equivocal scheme Π_{Com} and we also send along with the commitment ciphertexts C_0 and C_1 containing the decommitment information. For more details we refer the reader to [CLOS02].

Hyb_0: This hybrid is identical to the real world.
Hyb_1: This hybrid is similar to the real world except that we consider partially fake commitments. In particular, the secret key is not revealed upon corruption and in honest party commitments, a commitment to b is generated as in the simulation by computing $z \leftarrow \mathsf{QEnc}_{\widetilde{\mathsf{PK}}}(0,0)$ and strings r_0 , r_1 such that r_0 and r_1 are correct decommitments to 0 and 1, respectively. Then, C_b is computed as an encryption to $C_b \leftarrow \mathsf{ENC}_{\mathsf{CCA}}(P_i, P_j, sid, ssid, r_b)$. On the other hand, C_{1-b} is still chosen as a uniformly distributed string where this modification is not revealed upon corruption.
Hyb_2: This hybrid is similar to Hyb_1 except that in commitments generated by honest parties, the ciphertext C_{1-b} equals $C_{1-b} \leftarrow \mathsf{ENC}_{\mathsf{CCA}}(P_i, P_j, sid, ssid, r_{1-b})$ as generated by the simulator, rather than being chosen uniformly. So in this hybrid we consider completely fake commitments
Hyb_3: This hybrid is identical to the ideal world.

The indistinguishability between Hyb_0 and Hyb_1 follows immediately from the pseudorandomness/ CPA-security of the underlying commitment scheme.
The indistinguishability between Hyb_1 and Hyb_2 follows from the pseudorandomness of encryptions under $\mathsf{E}_{\mathsf{CCA}}$.
Next, the only difference between hybrids Hyb_2 and Hyb_3 is that in Hyb_2 the checks causing the simulator to output failure are not carried out. If the simulator never outputs failure then the two hybrids are identical. However considering the failure, the proof is carried out based on the CCA2 security of the $\mathsf{E}_{\mathsf{CCA}}$ and assuming that the simulator is given the true values of the inputs for all honest parties.