Secure Multiparty Computation from Threshold Encryption Based on Class Groups[⋆]

Lennart Braun, Ivan Damgård, and Claudio Orlandi

Aarhus University
{braun,ivan,orlandi}@cs.au.dk

Abstract. We construct the first actively-secure threshold version of the cryptosystem based on class groups from the so-called CL framework (Castagnos and Laguillaumie, 2015). We show how to use our threshold scheme to achieve general universally composable (UC) secure multiparty computation (MPC) with only transparent set-up, i.e., with no secret trapdoors involved.

On the way to our goal, we design new zero-knowledge (ZK) protocols with constant communication complexity for proving multiplicative relations between encrypted values. This allows us to use the ZK proofs to achieve MPC with active security with only a constant factor overhead.

Finally, we adapt our protocol for the so-called "You-Only-Speak-Once" (YOSO) setting, which is a very promising recent approach for performing MPC over a blockchain. This is possible because our key generation protocol is simpler and requires significantly less interaction compared to previous approaches: in particular, our new key generation protocol allows the adversary to bias the public key, but we show that this has no impact on the security of the resulting cryptosystem.

[⋆] This article is the full version of an article with the same name [BDO23], © IACR 2023, which appeared at Crypto 2023. The version published by Springer-Verlag is available at https://doi.org/10.1007/978-3-031-38557-5_20.

Table of Contents

1 Introduction

In secure multiparty computation (MPC), a set of N parties jointly compute an agreed function on inputs privately held by the parties. For security, we require that the result is correct and that the only new information revealed is the intended output. This should be true even if up to t parties are corrupted by an adversary, for some parameter $1 \leq t < N$.

1.1 The CDN Framework

In recent years, we have seen an explosion of results that improve the efficiency of general MPC protocols. One tool that is featured prominently in many recent works is Threshold Linearly Homomorphic Encryption (TLHE). In such a scheme there is a common public key pk, while the secret decryption key is shared among the parties using an appropriate secret-sharing scheme. It is assumed that we have a secure decryption protocol which on input a ciphertext $c = \mathsf{Enc}(\mathsf{pk}, m)$ returns the encrypted message m while revealing nothing else. Furthermore we assume that messages come from a ring, typically $\mathbb{Z}_n$ for some natural number n and that we have corresponding multiplication and exponentiation operations on ciphertexts such that for any messages a, b and public constant α in the ring, we have

$$\mathsf{Enc}(\mathsf{pk}, a) \cdot \mathsf{Enc}(\mathsf{pk}, b)^{\alpha} = \mathsf{Enc}(\mathsf{pk}, a + \alpha b).$$

This set-up, together with a set of appropriate zero-knowledge protocols, allows to implement MPC for general functions. This idea was first introduced by Cramer et al. in [CDN01]. The construction is known as the CDN framework because it works for any TLHE scheme, given the right set of zero-knowledge protocols. These include, for instance, proof of plaintext knowledge for a given ciphertext. While the required protocols can always be realized using generic techniques for zero-knowledge, some cryptosystems, such as Paillier encryption [Pai99], allow very efficient Σ-protocols for this purpose, such that the overhead required for the proofs is only a constant factor, compared to simply sending the required ciphertexts.

The CDN framework was defined for honest majority, but it is well-known in the folklore that it can also easily be adapted to work for dishonest majority, although only security with abort can then be obtained. In the years following the introduction of CDN, interest in the framework declined somewhat, for several reasons. For honest majority, it was realized that more efficient protocols can be obtained from techniques based on secret-sharing. For dishonest majority the BeDOZa/SPDZ protocols [Ben+11; Dam+12] introduced the idea of pushing the use of the expensive public key operations into a preprocessing phase where the inputs and function to compute need not be known. The preprocessing produces correlated randomness (typically so-called Beaver's multiplication triples [Bea92]) that is used once the function and inputs are known, and then, in the on-line phase, only very simple information-theoretic techniques are needed. The most practical instantiations of SPDZ perform preprocessing using somewhat-homomorphic Lattice-based encryption [KPR18; BCS19; GLM22]. This works well for relatively small plaintext spaces, say $\mathbb{F}_q$ for a 128-bit prime q, but gets very cumbersome for large q. While the preprocessing could also be done using the CDN approach based on Paillier encryption, this forces the plaintext ring to be $\mathbb{Z}_n$ for an RSA modulus n and this is not a good fit for all applications. In particular, the modulus, and hence the plaintext domain is not even known before the key generation is done, whereas other approaches can work for a predefined plaintext ring. Finally, one needs the modulus to be generated in a trusted way, which either requires a trusted authority or a (notoriously cumbersome) secure protocol for generating n. In summary, for large plaintext rings, there is a lack of a satisfactory approach to perform general preprocessing.

1.2 YOSO MPC

Interest in the CDN framework was recently renewed by the introduction of the You Only Speak Once (YOSO) model for MPC, by Gentry et al. in [Gen+21]. This model assumes a large universe of M parties, here referred to as servers, that are all willing to help execute a secure computation. The main idea is to have the secure computation be done by a committee of N servers where $N \ll M$, and where the committee will change over time. The goal is to hide from the adversary who is in the committee. If this can be done, we can hope to do MPC with communication complexity that scales sublinearly with M, even if the adversary can corrupt, say, just under $M/2$ of the servers – since then a randomly chosen but small committee will have honest majority with large probability.

However, the adversary is most likely able to tell who is in the committee as soon as these servers start sending messages and could then attack them. Hence the YOSO paradigm: a committee should only send one round of messages, and then the committee changes. The new committee will of course need to receive private information, so to realize this, one needs receiver anonymous communication channels (RACC). One can think of this as a primitive that outputs N public encryption keys such that the adversary does not know who they belong to (as long as the owner is not corrupt), and such that the majority of the owners are honest. In [Gen+21] a construction of RACC was proposed.

The bottleneck in a YOSO protocol is clearly the private communication between committees. The secret state one needs to maintain should therefore be as small as possible. This means that secret-sharing-based protocols and protocols in the preprocessing model are not well suited for YOSO, as parties need to remember a large number of shares. In contrast, a protocol following the CDN paradigm works much better: the only secret state that must be maintained consists of each party's share of the secret key. The rest of the state consists of public ciphertexts. This was pointed out already in [Gen+21], where a concrete construction was done from the Paillier-based instantiation of CDN. However, the construction assumed as set-up an honestly generated public key and a sharing of the secret key for the first committee. Doing CDN-style MPC in the YOSO model without trusted set-up was left as an open problem.

1.3 Our Contribution

The contributions of our work are as follows:

1. We construct the first actively-secure threshold version of the class-group-based LHE from the so-called CL framework [CL15]. This allows e.g., to address the problem related to the plaintext space mentioned above, since the CL cryptosystem can be used for plaintext space $\mathbb{F}_q$ where q can be (almost) any prime.
2. We propose a novel threshold key-generation protocol for the CL cryptosystem that is simpler and requires significantly less interaction than the state-of-the-art from [Gen+07]. This is essential to enable usage of the cryptosystem in the YOSO setting. The simplification is of independent interest as it also applies to other cryptosystems, including standard ElGamal based on DDH.
3. We identify a problem with Feldman-style secret sharing over a group of unknown order as introduced by Rabin [Rab98].[1] This is an essential tool used by our key-generation protocol. We fix the problem, patching the protocol (which becomes only slightly more complicated) and we can prove it secure under the assumption that the group order is hard to compute.
4. We design new zero-knowledge protocols for proving multiplicative relations between CL-encrypted values, with only constant factor overhead.

[1] We describe the issue in detail in Appendix C.

5. We introduce a new computational assumption, which we call the *rough order assumption*, stating that class groups with no small prime factors in their order are hard to distinguish from general class groups. Assuming this, we can considerably simplify our zero-knowledge protocols and their security proofs. We believe this is of independent interest as it will apply to other constructions based on class groups.
6. We build an UC-secure MPC protocol with transparent setup following the CDN paradigm. The protocol uses our novel TLHE scheme and ZK protocols described above. Transparent set-up in this context means that we do not need to assume a trusted party that generates the set-up and must keep some trapdoor information secret. Essentially, we just need to select some parameters from public randomness e.g., using a random oracle.
7. Finally, we adapt our MPC protocol to the YOSO setting, including the threshold key generation protocol. This solves the open problem mentioned above, from [Gen+21].

It would be easy to adapt our MPC protocol to the dishonest-majority setting, e.g., using the SPDZ paradigm (but we do not explicitly describe it in this paper). While this might not be so interesting in the YOSO context, it would have some benefit in the traditional MPC setting, since it would allow using any $\mathbb{F}_q$ (where q is a large prime) as plaintext ring, and as explained above, this can be an advantage over previous approaches for large q.

1.4 Technical Overview

As mentioned, we start from the CL cryptosystem as suggested in [CL15] and further investigated in [CLT18; Cas+19; Cas+20]. The cryptosystem is based on (subgroup of) a class group $G \simeq F \times H$ where F has known order q and the order of H is hard to compute. Such a group is specified by choosing initially a public number called the discriminant. This number is sampled transparently e.g., no hidden factorization is required.

A public key is a pair $\mathsf{pk} = (g, h)$ of elements in H, where the secret key is the discrete log of h base g. An encryption of $m \in \mathbb{F}_q$ with randomness r is of the form $\mathsf{Enc}(\mathsf{pk}, m; r) = (g^r, f^m h^r)$, where $f \in F$ is a generator of F or order q, and where in addition discrete logs base f are easy to compute. Decryption and the homomorphic properties follow almost immediately. CPA security follows from an appropriate assumption on subgroup indistinguishability. For the application of this system to MPC, there are a number of technical challenges to overcome, as we explain below.

Distributed Key Generation To avoid trusted set-up in the CDN paradigm, we need a distributed key generation protocol, where a public key is formed and parties obtain shares of the corresponding secret key. Now, encryption in the CL framework is essentially ElGamal encryption adapted to the class-group setting. So a natural first attempt is to have each player perform a Feldman-style verifiable secret sharing of their contribution to the key [Fel87], and then essentially add up all valid contributions.

In a Feldman VSS, the dealer publishes g^x and $\{g^{\alpha_k}\}_{k=1}^t$ where the α_k's are coefficients of a degree-t polynomial to be used for secret sharing x among the parties. Evaluating the polynomials in the exponent, it is possible to publicly compute values $y_j = g^{s_j}$, where s_j is the share of party P_j and this allows verification of the share that P_j receives in private. However, it turns out that there is a problem with this type of secret sharing in our setting: it takes place over a group of unknown order (as first done in [Rab98]) so parties cannot reduce modulo the order, meaning that we can only do integer arithmetic in the exponent. Therefore we must use the integer variant of Shamir secret sharing, i.e., polynomials with integer coefficients. Feldman's original suggestion was to use a group of known public order, such as a prime order subgroup of $\mathbb{Z}_p^*$ for a prime p. In such a setting, it is the case that from any $t+1$ shares $\{s_j\}$ that satisfy

$y_i = g^{s_i}$, one can reconstruct the correct secret x, and this can be concluded without relying on any computational assumptions. But this is not true for the case where integer secret sharing is used over an unknown order group. Intuitively, the reason is that we use integer secret sharing, but the verification of shares only guarantees that they are correct modulo the group order, not that they are the correct integers. It turns out that this can be used to mount a subtle attack against the protocols in the literature (which seems to have gone unnoticed since [Rab98]): As we describe in detail in Appendix C, a corrupt dealer who knows the group order can break reconstruction. This means that reconstruction cannot be shown secure unless we rely on the assumption that the group order is hard to compute. We fix this issue by having the dealer prove in zero-knowledge that they know the discrete log of g^x. Then, if any value is reconstructed that is different from what we extract from the dealer, we would get a multiple of the group order which we assume is infeasible to compute.[2]

Once we have a working version of Feldman VSS we can use the linearity properties and essentially add all valid contributions to the key to get the final key pair. However, this leads to a second challenge: as pointed out in [Gen+07], the approach based on Feldman VSS allows the adversary to bias the public key by selecting the contributions of the corrupt players after seeing what the honest players send. It is then not clear that the resulting encryption scheme is still secure. In [Gen+07] the problem was solved by designing a different protocol that guarantees a uniform public key, but this protocol is more complex, requires several rounds and is not suitable for the YOSO setting. Our approach is to instead take a closer look at the key generation using only Feldman VSS. We observe that the adversary's influence on this protocol can be shown to be equivalent to a modified CPA game where the adversary is first shown a public key pk, then selects a number δ, and the final public key is then $\mathsf{pk}' = \mathsf{pk} \cdot g^{\delta}$; the CPA game now continues as usual with pk' as the public key. Next, we show that for the cryptosystem in question, given an adversary who can win this modified CPA game, we can construct one that wins the standard CPA game. Hence, our encryption scheme is CPA secure, even if the public key is biased in the way our simpler protocol allows.

This observation is of independent interest, as it applies also to standard ElGamal encryption based on DDH (described in Appendix D). In [Gen+07], it was shown that a biased public key is good enough for some signature schemes. An earlier version of this paper stated that the case for encryption schemes was still open, however, it has been solved by Cortier et al. [Cor+13], who proved CPA security of ElGamal with biased key generation directly from the DDH assumption.

Zero-Knowledge Protocols An efficient zero-knowledge protocol for proof of plaintext knowledge was already suggested in [Cas+20]. In addition, we need a protocol for the following setting: given ciphertext $c = \mathsf{Enc}(\mathsf{pk}, a)$, a party P can choose b, compute $d = \mathsf{Enc}(\mathsf{pk}, b)$ and $e = c^b \cdot \mathsf{Enc}(\mathsf{pk}, 0) = \mathsf{Enc}(\mathsf{pk}, a \cdot b)$ and prove in zero-knowledge that d and e were correctly computed. Since $e = c^b \cdot \mathsf{Enc}(\mathsf{pk}, 0)$ and b also appears in the exponent in $\mathsf{Enc}(\mathsf{pk}, b)$, it turns out we can use a Schnorr-style Σ-protocol to prove this relation [CDS94].

Usually, one would want these protocols to have standard knowledge soundness, and it is well known that achieving this, as well as constant factor overhead protocols, leads to technical difficulties in groups of unknown order. More specifically, the extractor needs to compute inverses modulo the group order which cannot be done efficiently. This issue can sometimes be avoided by relying on the assumption that the adversary cannot compute non-trivial roots of a given group

[2] In follow-up work, we have observed that for the particular application to key generation, a slightly weaker reconstruction property suffices, namely that the product of x and a public constant modulo the group order can be reconstructed. This can be achieved without the dealer proving knowledge of x and hence the key generation can be made more efficient.

element. But even this will not work in our setting: we need to work with ciphertexts - and hence group elements - that are chosen by the adversary, and for those we can of course not assume that the adversary cannot compute roots. While it is possible to work around this using more complicated protocols, we observe that there is a much cleaner way to solve the problem:

We take a close look at the soundness property we need for our MPC protocol. It turns out to be sufficient that an adversarially generated ciphertext is proved to be well formed, and that we can extract the plaintext from the proof. This is weaker than full knowledge soundness that would equire us to also extract the ciphertext's randomness from the proof. It then turns out that we can prove the soundness property we need under the sole assumption that the order of the group we work in is "rough", i.e., has only large prime factors in its order. This is not necessarily true for class groups in general, but we introduce a new computational assumption stating that discriminants for class groups with rough order are indistinguishable from discriminants in general. This is then the only assumption we need for soundness. While this is admittedly a new and not yet well studied assumption, we believe that it will have impact in future work since it allows a much cleaner design and analysis of zero-knowledge protocols which would otherwise be quite cumbersome to analyse. (However, at the cost of more complicated protocols and analysis, we could instead use the more well-known strong root assumption.)

YOSO MPC All this leads to a general MPC protocol following the CDN paradigm, which we adapt to the YOSO setting in the final part of the paper. This solves the open problem from [Gen+21], in that we obtain a CDN-style MPC protocol without trusted set-up. We do this following the approach of [Gen+21] but replacing Pailler encryption by class-group-based encryption (which does not require any trusted set-up for key generation), however, the major difference is that the Feldman VSS used in the key generation must also be adapted to YOSO. In the original VSS, the dealer sends shares privately to the share holders. Instead, we let the dealer publish ciphertexts containing shares intended for the next committee, as well as zero-knowledge proof that the ciphertexts are correct. Now, the VSS consists of publishing one message which fits well the YOSO setting.

1.5 Other Related Work

Class groups have been used as tool to achieve transparent setups for other cryptographic primitives including: SNARKS [Aru+23], pseudorandom correlation functions [Abr+22], verifiable delay functions [Wes19; Pie19; BBF18], and much more.

In concurrent work, Kolby et al. [KRY22] show that it is possible to achieve guaranteed output delivery in a constant number of rounds without relying on trusted setup in the YOSO model. In particular, they propose two protocols based on garbled circuits and fully-homomorphic encryption which achieve constant round communication. They also propose a non constant-round protocol (which is used to bootstrap the constant-round protocols that require setup) which has some similarities with our work since it also uses linearly homomorphic encryption based on class groups in the YOSO context. However, the work in [KRY22] does not construct a TLHE scheme nor a YOSO key generation protocol. Instead, they assign a key pair from the basic scheme to each committee member and instruct committee members to send to the next committee by secret sharing their data and encrypting the shares for the receivers. This does achieve general MPC in YOSO without set-up, but this comes at a price: the amount of data that must be privately passed between committees is proportional to the size of the circuit being computed. In contrast, in a solution using a TLHE scheme, one can pass information by posting public ciphertexts encrypted under the global public key, and the data will then be accessible to any committee that has shares of the private key. Hence, these shares are the only private data we need to maintain. In [EFR21], Erwig et al. present a threshold cryptosystem for the YOSO setting based on the discrete logarithm problem in prime order groups.

Recently, in concurrent and independent work, Castagnos et al. [CLT22] created a threshold linearly homomorphic encryption scheme with $\mathbb{Z}_{2^k}$ as plaintext space using a new variant of the CL framework. While this enables new applications where arithmetic modulo a 2-power is needed, the construction requires a class group where the discriminant is an RSA modulus whose factors must not be known to the adversary. So this scheme does not allow transparent set-up.

2 Preliminaries

2.1 Notation

The following shorthands are used to describe ranges of integers: $[n] := \{1, \ldots, n\}$, $[a, b] := \{a, \ldots, b\}$, and $[a, b) := \{a, \ldots, b-1\}$. We use a computational security parameter λ and a statistical security parameter σ. Our protocols are defined for N parties $P_1, \ldots, P_N$ out of whom up to t parties can be corrupted. We use $\mathcal{C}$ and $\mathcal{H}$ to denote the subsets of corrupted and honest parties, respectively. We frequently use $\Delta := N! = 1 \cdot 2 \cdots N$.

2.2 The CL Framework for Unknown Order Groups

The framework was introduced by Castagnos et al. [CL15], and enhanced in [CLT18; Cas+19; Cas+20]. The following description is based on [Cas+19, Def. 4] and [Tuc20].

Group Structure The framework specifies two algorithms, CLGen and $\mathsf{CLSolve}$. CLGen takes a computational security parameter 1^λ and a prime $q > 2^\lambda$, and outputs a tuple $\mathsf{pp}_{\mathsf{cl}} := (q, \bar{s}, f, g_q, \widehat{G}, F; \rho) \leftarrow \mathsf{CLGen}(1^\lambda, q)$, where ρ denotes the randomness used by CLGen.[3] Here, $(\widehat{G}, \cdot)$ is a finite abelian group of order $\hat{n} := q \cdot \hat{s}$. The factor $\hat{s}$ remains unknown, but we are given an upper bound $\bar{s} \geq \hat{s}$. Moreover $\gcd(q, \hat{s}) = 1$, and the size of $\hat{s}$ depends on the security parameter λ. Taking the qth powers gives us the subgroup $\widehat{G}^q \subset \widehat{G}$, and $F = \langle f \rangle \subset \widehat{G}$ is the unique subgroup of order q. Hence, $\widehat{G}$ factors as $\widehat{G} \simeq \widehat{G}^q \times F$. While $\widehat{G}$ acts as base group, we are more interested in the cyclic subgroup $G \subset \widehat{G}$ of order $n := q \cdot s$. Whereas elements of $\widehat{G}$ are efficiently recognizable, this does not hold for elements of $G \subset \widehat{G}$. Again, $G^q = \langle g_q \rangle$ denotes the (cyclic) subgroup of qth powers, and G factors as $G \simeq G^q \times F$ with $g := g_q \cdot f$ being a generator of G. Given the output of CLGen, the second algorithm $\mathsf{CLSolve}$ deterministically and efficiently solves the discrete logarithm problem in the subgroup F.

Distributions Protocols in the CL framework make use of distributions $\mathcal{D}$ and $\mathcal{D}_q$ over the integers, such that $\{g^x \mid x \leftarrow \mathcal{D}\}$ and $\{g_q^x \mid x \leftarrow \mathcal{D}_q\}$ induce almost uniform distributions over G and G^q, respectively. E.g., $\mathcal{D}$ can be instantiated by sampling a uniform integer from the interval $\{0, \ldots, q\bar{s}/(4\delta) - 1\}$. Then $\{g^x \mid x \leftarrow \mathcal{D}\}$ is δ-close, i.e., has statistical distance at most δ, to the uniform distribution over G [Tuc20, S. 3.1.3]. To get a statistical distance of at most $2^{-\sigma}$, we can set the upper bound of the interval to $q\bar{s} \cdot 2^{\sigma-2} - 1$.

[3] [Cas+20, Sec. 3.2] remarks that the randomness used in their instantiation of CLGen is not crucial for security, but traditionally random discriminants are used for class-group-based crypto. Hence, we can use publicly known randomness $\rho \in \{0, 1\}^\lambda$.

Assumptions We make use of the well-known unknown order assumption (ORD) for class groups stating that it is hard to find a multiple of $\mathrm{ord}(h)$ for any $h \in (\widehat{G} \setminus F)$.

Definition 1 (Unknown Order Assumption [Cou+21]). *Let λ be a security parameter, $|q| \geq \lambda$ prime, and $\mathcal{A}$ be a PPT algorithm. The experiment generates public parameters $\mathsf{pp}_{\mathsf{cl}} := (q, \bar{s}, f, g_q, \widehat{G}, F; \rho) \leftarrow \mathsf{CLGen}(1^\lambda, q)$. It then runs $\mathcal{A}(\mathsf{pp}_{\mathsf{cl}}, \mathsf{CLSolve}(\cdot))$. We say $\mathcal{A}$ solves the* unknown order (ORD) problem *if it outputs a group element $h \in (\widehat{G} \setminus F)$*[4] *and an integer $e \neq 0$ such that $h^e = 1$, and define its advantage $\mathsf{Adv}_{\mathcal{A}}^{ORD}$ as its success probability. We say the ORD assumption holds if $\mathsf{Adv}_{\mathcal{A}}^{ORD}$ is negligible in λ for all PPT algorithms $\mathcal{A}$.*

We also use the hard subgroup membership assumption (HSM) by [CLT18] which says random elements of G and $\widehat{G}^q$ are indistinguishable.

Definition 2 (Hard Subgroup Membership Assumption [CLT18]). *Let λ be a security parameter, $|q| \geq \lambda$ prime, and $\mathcal{A}$ be a PPT algorithm. The experiment generates public parameters $\mathsf{pp}_{\mathsf{cl}} := (q, \bar{s}, f, g_q, \widehat{G}, F; \rho) \leftarrow \mathsf{CLGen}(1^\lambda, q)$, samples $x \leftarrow \mathcal{D}$, $x' \leftarrow \mathcal{D}_q$, and $b \in_R \{0,1\}$. It sets $Z_0 := g^x$ and $Z_1 := g_q^{x'}$, and runs $b^* \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{cl}}, Z_b, \mathsf{CLSolve}(\cdot))$.We say $\mathcal{A}$ solves the* hard subgroup membership (HSM) problem *if it outputs $b^* = b$, and define its advantage $\mathsf{Adv}_{\mathcal{A}}^{HSM}$ as its success probability. We say the HSM assumption holds if $\mathsf{Adv}_{\mathcal{A}}^{HSM}$ is negligible in λ for all PPT algorithms $\mathcal{A}$.*

The next assumption says that class groups with no small prime factors in their order are indistinguishable from class groups in general.

Definition 3 (Rough Order Assumption). *Let λ be a security parameter, $|q| \geq \lambda$ prime, $C \in \mathbb{N}$, and $\mathcal{A}$ be a PPT algorithm. Define $\mathcal{D}_C^{\mathsf{rough}}$ to be the uniform distribution over the set $\{\rho \in \{0,1\}^\lambda \mid (q, \bar{s}, f, g_q, \widehat{G}, F; \rho) \leftarrow \mathsf{CLGen}(1^\lambda, q; \rho) \wedge \forall$ prime $p < C \ : \ p \nmid \mathrm{ord}(\widehat{G})\}$, The experiment samples $\rho_0 \in_R \{0,1\}^\lambda$, $\rho_1 \leftarrow \mathcal{D}_C^{\mathsf{rough}}$, and $b \in_R \{0,1\}$, and runs $b^* \leftarrow \mathcal{A}(1^\lambda, \rho_b, \mathsf{CLSolve}(\cdot))$. We say $\mathcal{A}$ solves the C*-rough order (RO_C) problem *if it outputs $b^* = b$. We define its advantage $\mathsf{Adv}_{\mathcal{A}}^{RO_C}$ as its success probability. We say the RO_C assumption holds if $\mathsf{Adv}_{\mathcal{A}}^{RO_C}$ is negligible in λ for all PPT algorithms $\mathcal{A}$.*

This is a new assumption, and as such is not yet well studied. Our reasons for believing it is plausible are as follows: except for 2-powers, we do not know how to compute any non-trivial information on class group orders despite many years of research. Furthermore, Cohen and Lenstra [CL84] give heuristic arguments indicating that the fraction of discriminants less than some bound B for which a prime p divides the order of the corresponding class group is roughly the same as for random integers, as long as p is much smaller than B. This can be taken as evidence that random class group orders behave similarly to random integers w.r.t. the sizes of their prime factors. This would mean that a significant (and certainly non-negligible) fraction of class groups have C-rough order, as long as C is small compared to B, which is certainly the case for our parameters. Thus we are not comparing class groups in general to a vanishingly small (or even empty) subset. This also means that if our assumption would fail dramatically, so that a distinguisher with advantage essentially 1 was found, this would actually be great news, as this would allow us to sample class groups with rough order by trial and error.

Another thing to note is that the challenger in the security game corresponding to the assumption is not efficient, as we do not know how to sample class groups with C-rough order efficiently. This means the assumption has to be used with care, as indeed we do: the assumption is only used in the proof and nothing that would need to be implemented depends on the challenger. Incidentally, we are not the first to propose such an assumption: the gap-DDH assumption also has an inefficient challenger [OP01].

[4] This can by easily verified by checking $h^q \neq 1$.

Proofs and Arguments in the CL Framework [Cas+19] give a Σ-like protocol to prove knowledge of plaintext and randomness corresponding to ciphertexts of the HSM-CL encryption scheme of [CLT18] (see Section 3), that can be simplified to proofs for discrete logarithms. To allow 2-special soundness in the unknown order setting, these protocols are restricted to binary challenges and, thus, need to be repeated to obtain a negligible soundness error.

In [Cas+20], the authors overcome this efficiency issue by basing the soundness on computational assumptions. Even with a large challenge spaces they show how to either extract a witness or to break either the strong root or the low order assumption. Since we only need limited or no extraction, we base the soundness of our arguments on the RO_C assumption instead.

2.3 UC Functionalities

In our protocols, we make use of the following basic UC functionalities. $\mathcal{F}_{\mathsf{Rand}}$ is a functionality that on input (Rand, M) samples $x \in_R M$ and returns x to all parties. Since we consider the honest-majority setting, we can instantiate $\mathcal{F}_{\mathsf{Rand}}$ for many finite sets M without any further assumptions [BGW88; CCD88]. We also define $\mathcal{F}_{\mathsf{CL}}$ (Figure 1) that generates and distributes CL public parameters $\mathsf{pp}_{\mathsf{cl}}$. It can be implemented by a call to $\mathcal{F}_{\mathsf{Rand}}$ to generate randomness ρ followed by locally running $\mathsf{pp}_{\mathsf{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q; \rho)$.

CL Parameter Generation Functionality $\mathcal{F}_{\mathsf{CL}}$

Gen On input $(\mathsf{Gen}, 1^\lambda, q)$ from all parties, $\mathcal{F}_{\mathsf{CL}}$ runs the CL setup algorithm: $\mathsf{pp}_{\mathsf{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q)$. It stores and outputs the public parameters $\mathsf{pp}_{\mathsf{cl}} = (q, \bar{s}, f, g_q, \widehat{G}, F; \rho)$.

Fig. 1. CL Parameter Generation Functionality

3 The Linearly Homomorphic Encryption Scheme

In [CLT18], Castagnos et al. presented the HSM-CL linearly homomorphic encryption scheme, which we denote as $\Pi_{\mathsf{hsm\text{-}cl}}$ (see Figure 17). It provides IND-CPA security under the HSM assumption. A public key has the form $g_q^{\mathsf{sk}_{\mathsf{cl}}}$, where the secret key is sampled as $\mathsf{sk}_{\mathsf{cl}} \leftarrow \mathcal{D}_q$. A message $m \in \mathbb{F}_q$ is encrypted with randomness $r \leftarrow \mathcal{D}_q$ as $\mathsf{Enc}(\mathsf{pk}_{\mathsf{cl}}, m; r) = (g_q^r, f^m \cdot \mathsf{pk}_{\mathsf{cl}}^r)$.[5]

We use ct to denote a ciphertext and overload the $\cdot$ and $+$ operators to denote homomorphic operations: $a \cdot \mathsf{ct}_x + \mathsf{ct}_y + b$ denotes a sequence of scalar multiplication, and addition of ciphertexts and constants without randomization, and $a \cdot_R^{r_1} \mathsf{ct}_x +_R^{r_2} \mathsf{ct}_y +_R^{r_3} b$ is the same sequence randomized with r_1, r_2, r_3 (which we might omit if not explicitly needed).

We need to make some modification to the original HSM-CL encryption scheme $\Pi_{\mathsf{hsm\text{-}cl}}$ to make it work with our protocols and the proof strategy for our MPC protocol. The new procedures are specified in Figure 2. Starting from the original scheme $\Pi_{\mathsf{hsm\text{-}cl}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, we construct the new encryption scheme $\Pi^*_{\mathsf{hsm\text{-}cl}}$ in two steps:

1. The original KeyGen procedure samples key pairs that are distributed almost-uniformly in G^q. Our distributed key generation protocol, however, allows the adversary to bias the distribution of the generated keys, as we will discuss in Section 6. We show that despite this bias the encryption scheme $\Pi^1_{\mathsf{hsm\text{-}cl}} := (\mathsf{Setup}, \mathsf{BiasedKeyGen}^{\mathcal{A}}, \mathsf{Enc}, \mathsf{Dec})$ is still secure.

[5] The full specification of the original encryption scheme is given in Appendix A.1.

2. The proof strategy for our MPC protocol (given in Section 7) follows [Dam+12] and requires an encryption scheme with a particular property: There needs to be a second key generation algorithm that produces lossy public keys. These need to be indistinguishable to normal public keys, but encryptions under these keys should be statistically indistinguishable to encryptions of zero. Hence, we define a special key generation algorithm $\mathsf{BiasedSpecialKeyGen}_b^{\mathcal{A}}$ that outputs a public key $\widetilde{\mathsf{pk}_{\mathsf{cl}}}$ which includes a ciphertext ct_P which is an encryption of b. In the real protocol $b = 1$ so encryption with SEnc uses the homomorphic property to multiply the message with ct_P. In the proof, when $b = 0$, a lossy public key is produced instead. Therefore, encrypting an arbitrary message always results in an encryption of 0. Moreover, during the generation of ct_P the adversary can once more bias the distribution, which also needs to be handled by the security proof. Our final encryption scheme is defined as $\Pi^*_{\mathsf{hsm\text{-}cl}} := (\mathsf{Setup}, \mathsf{BiasedSpecialKeyGen}_1^{\mathcal{A}}, \mathsf{SEnc}, \mathsf{Dec})$.

```
$\underline{\mathsf{BiasedKeyGen}^{\mathcal{A}}(\mathsf{pp}_{\mathsf{cl}})}$

1. Sample $\alpha \leftarrow \mathcal{D}_q$.
2. Set $\mathsf{pk}^*_{\mathsf{cl}} := g_q^{\alpha}$.
3. $\delta \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{cl}}, \mathsf{pk}^*_{\mathsf{cl}})$.
4. $\mathsf{sk}_{\mathsf{cl}} := \alpha + \delta$ and $\mathsf{pk}_{\mathsf{cl}} := g_q^{\mathsf{sk}_{\mathsf{cl}}} = g_q^{\alpha+\delta}$.
5. Output $(\mathsf{pk}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}})$.
```

```
$\underline{\mathsf{SEnc}(\widetilde{\mathsf{pk}_{\mathsf{cl}}}, m \in \mathbb{F}_q; r)}$

1. Parse $\widetilde{\mathsf{pk}_{\mathsf{cl}}} = (\mathsf{pk}_{\mathsf{cl}}, \mathsf{ct}_P)$.
2. Output $\mathsf{ct} := m \cdot_R^r \mathsf{ct}_P$.
```

```
$\underline{\mathsf{BiasedSpecialKeyGen}_b^{\mathcal{A}}(\mathsf{pp}_{\mathsf{cl}})}$ for $b \in \{0, 1\}$

1. $(\mathsf{pk}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}}) \leftarrow \mathsf{BiasedKeyGen}^{\mathcal{A}}(\mathsf{pp}_{\mathsf{cl}})$.
2. Sample $\beta \leftarrow \mathcal{D}_q$.
3. Set $\mathsf{ct}^*_P := (g_q^{\beta}, f^b \cdot \mathsf{pk}_{\mathsf{cl}}^{\beta})$.
4. $\varepsilon \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{cl}}, \mathsf{pk}_{\mathsf{cl}}, \mathsf{ct}^*_P)$.
5. $\widetilde{\mathsf{pk}_{\mathsf{cl}}} := (\mathsf{pk}_{\mathsf{cl}}, (g_q^{\beta+\varepsilon}, f^b \cdot \mathsf{pk}_{\mathsf{cl}}^{\beta+\varepsilon}))$.
6. Output $(\widetilde{\mathsf{pk}_{\mathsf{cl}}}, \mathsf{sk}_{\mathsf{cl}})$.
```

Fig. 2. Modified key generation, and encryption algorithms that allow an adversary $\mathcal{A}$ to influence the distribution of public keys in a limited way.

The main result of this section is the following Theorem 1. In the following subsections, we prove a series of lemmas that, when combined, yield the theorem as a corollary.

Theorem 1 (Security of $\Pi^*_{\mathsf{hsm\text{-}cl}}$). *Under the HSM assumption, $\Pi^*_{\mathsf{hsm\text{-}cl}}$*

1. *provides indistinguishability under chosen-plaintext attacks (IND-CPA), and*
2. *has lossy public keys which are indistinguishable from real public keys.*

3.1 Handling Biased Keys

Our efficient distributed key generation protocol allows the adversary to bias the distribution of the keys. We prove that the bias does not affect the security, and exploit the homomorphic properties to reduce its security to that of $\Pi_{\mathsf{hsm\text{-}cl}}$.[6]

Lemma 2 (IND-CPA Security of $\Pi^1_{\mathsf{hsm\text{-}cl}}$). *If $\Pi_{\mathsf{hsm\text{-}cl}}$ is indistinguishabile under chosen-plaintext attacks (IND-CPA) then so is $\Pi^1_{\mathsf{hsm\text{-}cl}}$.*

Proof. We prove the claim by showing that any adversary $\mathcal{B}$ that wins the IND-CPA game for $\Pi^1_{\mathsf{hsm\text{-}cl}}$ with advantage $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{hsm\text{-}cl},1}$ can be transformed into an adversary $\mathcal{A}$ for $\Pi_{\mathsf{hsm\text{-}cl}}$ with the

[6] Independently of us, Stengele et al. [Ste+21] made a similar observation for ElGamal encryption (cf. Appendix D).

same advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl}} = \mathsf{Adv}_{\mathcal{B}}^{\mathsf{hsm\text{-}cl},1}$. The reduction is depicted in Figure 18 in Appendix A.2. Given an adversary $\mathcal{B}$, we create $\mathcal{A}$ as follows: Initially $\mathcal{A}$ receives public parameters $\mathsf{pp}_{\mathsf{cl}}$ and a public key $\mathsf{pk}_{\mathsf{cl}}$ from the challenger. It gives $\mathsf{pk}_{\mathsf{cl}}$ to $\mathcal{B}$, which responds with δ. Define the biased public key $\mathsf{pk}'_{\mathsf{cl}} := \mathsf{pk}_{\mathsf{cl}} \cdot g_q^{\delta}$. This phase corresponds to the $\mathsf{BiasedKeyGen}^{\mathcal{B}}(\mathsf{pp}_{\mathsf{cl}})$ procedure. Then $\mathcal{B}$ selects $m_0, m_1 \in \mathbb{F}_q$, which $\mathcal{A}$ forwards to the challenger. The challenger samples $b \in_R \{0,1\}$, encrypts m_b and sends the resulting $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ to $\mathcal{A}$. $\mathcal{A}$ sends $\mathsf{ct}' = (\mathsf{ct}'_1, \mathsf{ct}'_2) = (\mathsf{ct}_1, \mathsf{ct}_2 \cdot \mathsf{ct}_1^{\delta})$ to $\mathcal{B}$. Finally, $\mathcal{B}$ outputs a bit $b' \in \{0,1\}$, and $\mathcal{A}$ forwards this output to the challenger.

The challenger creates the ciphertext $\mathsf{ct} = (g_q^r, \mathsf{pk}_{\mathsf{cl}}^r \cdot f^{m_b})$, where r denotes the randomness used during the encryption. We have $\mathsf{pk}'_{\mathsf{cl}} = \mathsf{pk}_{\mathsf{cl}} \cdot g_q^{\delta}$ and

$$\mathsf{ct}' = \left(g_q^r, (\mathsf{pk}_{\mathsf{cl}}^r \cdot f^{m_b}) \cdot (g_q^r)^{\delta}\right) = \left(g_q^r, (\mathsf{pk}_{\mathsf{cl}} \cdot g_q^{\delta})^r \cdot f^{m_b}\right) = \left(g_q^r, (\mathsf{pk}'_{\mathsf{cl}})^r \cdot f^{m_b}\right).$$

Therefore, ct' is a valid encryption of m_b under the biased public key $\mathsf{pk}'_{\mathsf{cl}}$ with the same distribution as it would normally have. Overall, $\mathcal{A}$ wins the game iff $\mathcal{B}$ answers correctly, which happens with probability $1/2 + \mathsf{Adv}_{\mathcal{B}}^{\mathsf{hsm\text{-}cl},1}$. Hence, the advantage in the case of biased keys is the same as in the standard scheme. □

3.2 Special Public Keys

In Figure 2 we specify the key generation algorithm $\mathsf{BiasedSpecialKeyGen}_b^{\mathcal{A}}$ parametrized by a bit $b \in \{0,1\}$. $\mathsf{BiasedSpecialKeyGen}_1^{\mathcal{A}}$ produces working public keys, whereas $\mathsf{BiasedSpecialKeyGen}_0^{\mathcal{A}}$ produces lossy public keys. Each special public key $\widetilde{\mathsf{pk}}_{\mathsf{cl}}$ consists of a normal public key $\mathsf{pk}_{\mathsf{cl}}$ and a ciphertext ct_P that is an encryption of $b \in \{0,1\}$. First we show that the distributions of working public keys ($b = 1$) and lossy public keys ($b = 0$) produced by $\mathsf{BiasedSpecialKeyGen}_b^{\mathcal{A}}$ are indistinguishable.

Lemma 3 ($\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ has Indistinguishable Lossy Keys). *If $\mathit{\Pi}^1_{\mathsf{hsm\text{-}cl}}$ is IND-CPA secure, then $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ has lossy public keys which are indistinguishable from real public keys.*

Since working and lossy public keys are encryptions of 1 and 0, an efficient distinguisher would break IND-CPA security of $\mathit{\Pi}^1_{\mathsf{hsm\text{-}cl}}$. The complete proof is given in Appendix A.3. Now we show that the final encryption scheme $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ is actually IND-CPA secure.

Lemma 4 (IND-CPA Security of $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$). *If $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ has indistinguishable lossy public keys as defined in Lemma 3, then $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ provides indistinguishability under chosen-plaintext attacks (IND-CPA).*

IND-CPA security of $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ follows from the fact that real keys are indistinguishable from lossy keys (cf. Lemma A.3), and the fact that when lossy keys are used then the encryptions are (statistically) independent of the encrypted message. The full proof is given in Appendix A.4. Now Theorem 1 follows as a corollary of the lemmas in this section.

4 Secret Sharing over the Integers

For our threshold encryption scheme in Section 6, we need a threshold secret sharing scheme to share a secret key of the $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ encryption scheme. Any majority of at least $t + 1 > N/2$ parties needs to be able to (implicitly) reconstruct the secret key to decrypt ciphertexts.

Since we work in an unknown order setting, the secret key of $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ is an integer and not a field element. Hence, the standard Shamir's secret sharing scheme [Sha79] over a finite field does not work for us: We need a secret sharing scheme that is suitable for sharing integers (from a bounded interval). Moreover, Lagrange interpolation requires division, but we cannot simply divide modulo the unknown group order to reconstruct the secret key in the exponent.

Similar problems have appeared in the construction of threshold RSA systems, where the group order is also unknown unless the factorization of the modulus is revealed [DF92; Fra+97; Rab98]. It is usually solved by multiplying the Lagrange coefficients with a suitable factor to eliminate the denominator. We also use a Shamir-style secret sharing scheme over $\mathbb{Z}$ with a polynomial of degree t.

The next challenge is to make sure that the dealer distributes consistent shares that allow reconstruction of the secret, and that during reconstruction every party publishes its correct share. We use a variant of Feldman's verifiable secret sharing scheme [Fel87] adapted to the our class group setting in a similar way as [Rab98] for threshold RSA.

4.1 Shamir's Secret Sharing over $\mathbb{Z}$

Secret Sharing To share a secret from a bounded range $\alpha \in [0, 2^\ell)$ with $(t+1)$-out-of-N reconstruction, we could use a random degree-t polynomial $f(X)$ such that $f(0) = \alpha$ and giving P_i the share $y_i := f(i)$ for $i \in [N]$. First, note that $f(i)$ reveals $\alpha \bmod i$. Hence, we share $\tilde{\alpha} := \alpha \cdot \Delta$ instead, where $\Delta := N!$. Then, we also need to make sure that the random coefficients of f are sampled from a large enough interval to hide α.

Protocol 1 (Shamir's Secret Sharing over $\mathbb{Z}$)**.** Let N be the number of parties, t the corruption threshold, ℓ a bound on the secret size, and σ a statistical security parameter. Moreover, let $\ell_0 \in \mathbb{N}$ be a parameter. To share a secret $\alpha \in [0, 2^\ell)$ the dealer proceeds as follows:

1. Let $\tilde{\alpha} := \alpha \cdot \Delta$ with $\Delta := N!$.
2. Sample $\mathbf{r} = (r_1, \ldots, r_t) \in_R [0, 2^{\ell_0+\sigma})^t$
3. Set $f(X) := \tilde{\alpha} + r_1 \cdot X + \cdots + r_t \cdot X^t$.
4. Send $y_i := f(i)$ over a private channel to party P_i for $i \in [N]$.

We denote this as $\mathsf{Share}(\alpha, \mathbf{r})$ when α is shared using the random coins $\mathbf{r}$, and use $\mathsf{Share}_i(\alpha, \mathbf{r})$ to denote the share of P_i.

Privacy Statistically no information about the shared secret should be revealed by any set of up to t shares. We formally define and prove privacy for Protocol 1 in Appendix B.1, similarly to linear integer secret sharing schemes [CF02; DT06].[7]

Integer Reconstruction By Lagrange interpolation, any subset S of at least $t+1$ parties can combine their shares to reconstruct the secret. We have

$$f(X) = \sum_{i \in S} y_i \cdot \ell_i^S(X) \quad \text{with} \quad \ell_i^S(X) = \prod_{j \in S \setminus \{i\}} \frac{x_j - X}{x_j - x_i}. \tag{1}$$

We also write the Lagrange coefficients at 0 (jointly often referred to as reconstruction vector) as $\ell_i^S := \ell_i^S(0)$. Clearly reconstruction also works for integer polynomials f, but the Lagrange coefficients are not necessarily integers. If we need to reconstruct without division, then we can multiply Lagrange coefficients by Δ, since $\Delta \cdot \ell_i^S(X) \in \mathbb{Z}[X]$ for all i and S, although we are reconstructing the value $\Delta \cdot f(X)$ instead of $f(X)$. Hence, we can use Shamir's secret sharing

[7] We can see Protocol 1 as a linear integer secret sharing (LISS) [CF02; DT06] variant, with the difference that we reconstruct $\alpha \cdot \Delta^2$ The distribution matrix is a Vandermonde matrix, and the distribution vector is the vector of coefficients of the polynomial.

over the integers to reconstruct a secret $\alpha \cdot \Delta^2$ in the exponent of a group element with unknown order. Similarly, we can also recover the polynomial itself from a sufficient number of shares. In Appendix B.2 we show that we can reconstruct $\Delta \cdot f$ by using $\mathbb{Z}$ linear combinations of the shares:

Lemma 5 (Reconstruction of Sharing Polynomial). *Given $t+1$ pairs $(x_i, y_i) \in \mathbb{Z}^2$ for $i \in [0,t]$ with $x_0, \ldots, x_t \in [0,N]$ and $x_0 < x_1 < \cdots < x_t$, then we can efficiently compute a polynomial $f(X) = \sum_{i=0}^{t} a_i \cdot X^i \in \mathbb{Z}[X]$ of degree at most t so that $f(x_i) = \Delta \cdot y_i$ for $i = 0, \ldots, t$. Moreover, we can compute the coefficients a_i as linear combinations of y_i with integer coefficients.*

4.2 Feldman VSS

Verifiable Secret Sharing We now adapt Feldman's verifiable secret sharing scheme [Fel87] to the class group setting. In [Rab98], Rabin has already provided a variant of this for threshold RSA, but our techniques are slightly different.[8] In the following, we assume that public parameters $\mathsf{pp}_{\mathsf{cl}}$ are given together with a designated group element $g_{\mathsf{F}} \in (\widehat{G} \setminus F)$.

Protocol 2 (Feldman VSS)**.** To share a secret $\alpha \in [0, 2^{\ell})$, the dealer shares it as $(y_1, \ldots, y_N) \leftarrow \mathsf{Share}(\alpha, \mathbf{r})$ (see Protocol 1). It sends y_i privately to P_i, and additionally, broadcasts values $C_0 := g_{\mathsf{F}}^{\alpha}$ and $C_k := g_{\mathsf{F}}^{\Delta \cdot r_k}$ for $k \in [t]$, where $\Delta := N!$. Party P_i verifies its share y_i by checking that

$$g_{\mathsf{F}}^{\Delta \cdot y_i} \overset{?}{=} C_0^{\Delta^2} \cdot \prod_{k=1}^{t} (C_k)^{(i^k)} \tag{2}$$

holds, and broadcasts a complaint otherwise. Moreover, the dealer needs to prove in zero-knowledge that $C_0, \ldots, C_t \in \langle g_{\mathsf{F}} \rangle$, e.g., by proving knowledge of discrete logarithms of all C_i to base g_{F}. During reconstruction, Equation (2) can be used to verify that the parties publish the correct shares. For the sharing procedure we write $(y_1, \ldots, y_N; C_0, \ldots, C_t) \leftarrow \mathsf{F\text{-}Share}(\alpha; \mathbf{r}; g_{\mathsf{F}})$, and $\{\bot, \top\} \leftarrow \mathsf{F\text{-}Check}(i, y_i; C_0, \ldots, C_t; g_{\mathsf{F}})$ denotes the verification.

We prove that Protocol 2 has the desired reconstruction properties:

Lemma 6 (Feldman Reconstruction). *If $\gcd(\Delta, \mathrm{ord}(g_{\mathsf{F}})) = 1$, and under the (ORD) assumption, we have*

(i) Given $C_0, \ldots, C_t \in \langle g_{\mathsf{F}} \rangle$, a proof of knowledge of the discrete logarithm of C_0, and $t+1$ shares $y_{i_1}, \ldots, y_{i_{t+1}}$ such that Equation (2) holds. Then there exists an efficient algorithm which the parties can use to find (except with negligible probability) a value $\alpha \in \mathbb{Z}$ such that $C_0 = g_{\mathsf{F}}^{\alpha}$.

(ii) Moreover, any collection of $t+1$ shares which pass the check in Equation (2) can be used to recover the same value $\alpha \pmod{\mathrm{ord}(g_{\mathsf{F}})}$.

Proof. (i) Let $h \in \mathbb{Q}[X]$ be the unique polynomial of degree at most t such that $h(i_j) = y_{i_j}$ for $j \in [t+1]$. Let L_j be the Langrange interpolation coefficients that can be used to reconstruct $h(0)$ from its values in $i_1, .., i_{t+1}$, but multiplied by Δ, so the L_j's are integers. We can efficiently compute the integer $\Delta h(0) = \sum_{j=1}^{t+1} L_j y_{i_j}$. We will see that this number is divisible by Δ^2, so the reconstruction outputs $\alpha = \frac{\Delta h(0)}{\Delta^2}$.

[8] [Rab98] uses generators $g_0, g := g_0^{\Delta^2}$ of maximal order in $\mathbb{Z}_n^*$ and then g as base in the Feldman scheme. Rabin does not prove the reconstruction property, but claims it follows directly from Feldman's work. We could not reproduce the proof and, therefore, use a slightly different construction.

To see why this works, note that, by Equation (2), we have

$$g_{\mathsf{F}}^{\Delta \cdot h(i_j)} = C_0^{\Delta^2} \cdot \prod_{k=1}^{t} (C_k)^{(i_j^k)}, \ j \in [t+1]. \quad (\star)$$

Let $C_0' := C_0^{\Delta}$, but $C_k' := C_k^{\Delta^{-1}}$ for $k \in [t]$. Note that Δ^{-1} in the exponent refers to the inverse of Δ modulo the order of g_{F}, which exists by our assumption. Then we get the following

$$\left(g_{\mathsf{F}}^{h(i_j)}\right)^{\Delta} = g_{\mathsf{F}}^{\Delta \cdot h(i_j)} \overset{(\star)}{=} C_0^{\Delta^2} \cdot \prod_{k=1}^{t} (C_k)^{(i_j^k)} = C_0'^{\Delta} \cdot \prod_{k=1}^{t} (C_k'^{\Delta})^{(i_j^k)} = \left(\prod_{k=0}^{t} (C_k')^{(i_j^k)}\right)^{\Delta}.$$

By raising both sides to $\Delta^{-1} \pmod{\operatorname{ord}(g_{\mathsf{F}})}$ we obtain $g_{\mathsf{F}}^{h(i_j)} = \prod_{k=0}^{t} (C_k')^{(i_j^k)}$. Now we can write $C_k' = g_{\mathsf{F}}^{\gamma_k}$ for some $\gamma_k \in \mathbb{Z}$ and $k \in [0,t]$, hence, we can write $g_{\mathsf{F}}^{h(i_j)} = g_{\mathsf{F}}^{\sum_{k=0}^{t} \gamma_k \cdot (i_j^k)}$. If we define the polynomial $h'(X) = \sum_{k=0}^{t} \gamma_k \cdot X^k$, we can rewrite this as $g_{\mathsf{F}}^{h(i_j)} = g_{\mathsf{F}}^{h'(i_j)}$. This implies, by interpolation in the exponent using the L_j's as coefficients, that

$$g_{\mathsf{F}}^{\Delta h(0)} = g_{\mathsf{F}}^{\Delta h'(0)} = g_{\mathsf{F}}^{\Delta \gamma_0} = (C_0')^{\Delta} = C_0^{\Delta^2}$$

From this follows immediately that if $\Delta h(0)$ is divisible by Δ^2, then the reconstruction works correctly and returns $\alpha = \frac{\Delta h(0)}{\Delta^2}$ where $g_{\mathsf{F}}^{\alpha} = C_0$. Assume for contradiction that with non-negligible probability, it happens that the Dealer's proofs are accepted, but the reconstruction finds that $\Delta h(0)$ is not divisible by Δ^2. Then, from the prover's proof for C_0, we can extract s such that $C_0 = g_{\mathsf{F}}^{s}$. It now follows that $g_{\mathsf{F}}^{\Delta h(0)} = C_0^{\Delta^2} = g_{\mathsf{F}}^{s\Delta^2}$, hence $\Delta h(0)$ and $s\Delta^2$ are congruent modulo $\operatorname{ord}(g_{\mathsf{F}})$, but are not the same integer since one is divisible by Δ^2 and the other one is not. Hence $\Delta h(0) - s\Delta^2$ is a solution to the *ORD* problem.

(ii) By the argument above, it must be $\alpha = \gamma_0 \pmod{\operatorname{ord}(g_{\mathsf{F}})}$. Since γ_0 is fixed modulo $\operatorname{ord}(g_{\mathsf{F}})$ given C_0, the same goes for α. Suppose, we recover $\alpha \neq \alpha' \in \mathbb{Z}$ from different sets of shares (note that still $\alpha = \alpha' \pmod{\operatorname{ord}(g_{\mathsf{F}})}$). Then we have found a solution $M := \alpha - \alpha' \neq 0$ to the *ORD* problem. □

Remark 7. Why is the assumption $\gcd(\Delta, \operatorname{ord}(g_{\mathsf{F}}))$ sensible? To obtain such a g_{F}, one can raise a given element $g_{\mathsf{F}}' \in \widehat{G} \setminus F$ to large enough powers of all primes up to N. This eliminates all small prime factors in the order of g_{F}. Otherwise, if a base $g_{\mathsf{F}} \in G^q$ is already given, one can rely on the RO_{N+1} assumption (Definition 3). This allows us to pretend that $\gcd(\Delta, \operatorname{ord}(g_{\mathsf{F}})) = 1$ holds, because the adversary cannot distinguish the two cases.

Simulating VSS with Fixed C_0 In the simulation of higher level protocols (see Section 6), we need to fake a Feldman VSS towards the adversary where the checking value C_0 cannot be chosen freely, but needs to be a prescribed value. First, we use Protocol 1 to share an arbitrary value $\tilde{\alpha}$ such that the corrupted parties obtains shares $(x_1, y_1), \ldots, (x_t, y_t) \in \mathbb{Z}^2$, We use the following lemma to find values $C_1, \ldots, C_t \in \langle g_{\mathsf{F}} \rangle$ that matches the adversary's view:

Lemma 8 (Feldman Simulation). *Given $C_0 \in \langle g_{\mathsf{F}} \rangle$ and $(x_1, y_1), \ldots, (x_t, y_t) \in \mathbb{Z}^2$, $1 \leq x_1 < \cdots < x_t \leq N$, we can efficiently find $C_1, \ldots, C_t \in \langle g_{\mathsf{F}} \rangle$ such that F-Check$(x_j, y_j; C_0, \ldots, C_t, g_{\mathsf{F}})$ is satisfied for all (x_j, y_j), $j \in [t]$. Moreover, the values $C_1, \ldots, C_t \in \langle g_{\mathsf{F}} \rangle$ have, conditioned on C_0 and $(x_j, y_j)_{j \in [t]}$, the same distribution as produced by F-Share.*

Proof. Set $x_0 := 0$, and let $y_0' \in \mathbb{Z}$ denote the unknown discrete logarithm of C_0 such that $C_0 := g_{\mathsf{F}}^{y_0'}$. Let $y_0 := y_0' \cdot \Delta$ and write $\mathbf{y} := (y_0 \ \ldots \ y_t)$.

By Lemma 5 we can reconstruct a polynomial $f(X) = \sum_{k=0}^{t} a_k \cdot X^k \in \mathbb{Z}[X]$ such that $f(x_i) = \Delta \cdot y_i$ for $i \in [0,t]$. Moreover, there exists an integer matrix $W = (w_{i,j}) \in \mathbb{Z}^{(t+1)\times(t+1)}$ (namely $W = \Delta \cdot V_X^{-1}$, where V_X is the Vandermonde corresponding to $X := \{x_0, x_1, \ldots, x_t\}$) such that $\mathbf{a} = W \cdot \mathbf{y}$. Hence, we can write $a_j = \sum_{i=0}^{t} w_{j,i} \cdot y_i$. For $k \in [t]$, define $C_k := C_0^{\Delta \cdot w_{k,0}} \cdot \prod_{i=1}^{t} g_{\mathsf{F}}^{w_{k,i} \cdot y_i}$. We show that these values satisfy the F-Check equation for all (x_i, y_i), $i \in [t]$:

$$C_0^{\Delta^2} \cdot \prod_{k=1}^{t} (C_k)^{(x_i^k)} = C_0^{\Delta^2} \cdot \prod_{k=1}^{t} \left(C_0^{\Delta \cdot w_{k,0}} \cdot \prod_{i=1}^{t} g_{\mathsf{F}}^{w_{k,i} \cdot y_i} \right)^{(x_i^k)} = C_0^{\Delta^2} \cdot \prod_{k=1}^{t} \left(g_{\mathsf{F}}^{\sum_{i=0}^{t} w_{k,i} \cdot y_i} \right)^{(x_i^k)}$$

$$= g_{\mathsf{F}}^{\Delta \cdot y_0} \cdot \prod_{k=1}^{t} (g_{\mathsf{F}}^{a_k})^{(x_i^k)} = g_{\mathsf{F}}^{\Delta \cdot y_0 + \sum_{k=1}^{t} a_k \cdot x_i^k} = g_{\mathsf{F}}^{f(x_i)} = g_{\mathsf{F}}^{\Delta \cdot y_i}.$$

Since f the unique interpolation polynomial, and we set the C_k according to its coefficients, the C_k have the correct distribution. □

5 Zero-Knowledge

For our protocols we need zero-knowledge arguments for several different relations. First, we need arguments for the discrete logarithm and equal discrete logarithm relations, $\mathsf{R}_{\mathsf{DLog}}$ and $\mathsf{R}_{\mathsf{EqDLog}}$. These relations are defined with respect to CL public parameters $\mathsf{pp}_{\mathsf{cl}}$ which we leave implicit in the notation.

$$\mathsf{R}_{\mathsf{DLog}} := \left\{ (g,h); x \;\middle|\; g,h \in \widehat{G} \wedge x \in \mathbb{Z} \wedge h = g^x \right\} \tag{3}$$

$$\mathsf{R}_{\mathsf{EqDLog}} := \left\{ (g_1,g_2,h_1,h_2); x \;\middle|\; ((g_i,h_i);x) \in \mathsf{R}_{\mathsf{DLog}} \text{ for } i = 1,2 \right\} \tag{4}$$

We use $\Pi_{\mathsf{DLog}}^{\mathsf{bin}}$ and $\Pi_{\mathsf{EqDLog}}^{\mathsf{bin}}$ to denote the standard Σ protocols for these relations with binary challenges and parallel repetitions. Then, we also need proofs of plaintext knowledge (PoPK) for $\Pi_{\mathsf{hsm\text{-}cl}}^{*}$ ciphertexts, that allow the extraction of the plaintext, as well as proofs of correct multiplication (PoCM). We use Σ-like protocols [Cra97], which can be specified by three PPT algorithms $(\mathsf{Commit}, \mathsf{Respond}, \mathsf{Verify})$ and a challenge space. In this work we consider *zero-knowledge arguments* of the same shape that satisfy statistical special honest-verifier zero-knowledge (SHVZK) and completeness for witnesses of a bounded size, as well as computational soundness for language membership, since in most cases it is not necessary to extract the (full) witness. Formal definitions are given in Section 5.1. In Section 5.2 we obtain arguments for general relations of class group elements, and Section 5.3 covers the protocols for $\Pi_{\mathsf{hsm\text{-}cl}}^{*}$ ciphertexts.

5.1 Definitions

***Σ*-Protocols** A Σ-protocol [Cra97] is a three-move protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ for a witness relation R. The common input are the relation R and a statement x, and $\mathcal{P}$ additionally knows a witness w such that $(x,w) \in \mathsf{R}$. We can specify a Σ-protocol by a tuple of three PPT algorithms $(\mathsf{Commit}, \mathsf{Respond}, \mathsf{Verify})$ and a challenge space.

1. $\mathcal{P}$ computes $(\mathsf{stt}, \mathsf{com}) \leftarrow \mathsf{Commit}(x,w)$ and sends the “commitment” com.
2. $\mathcal{V}$ sends a uniformly sampled challenge $\mathsf{chl} \in_R [C]$ from a challenge set $[C]$.
3. $\mathcal{P}$ sends a response $\mathsf{res} \leftarrow \mathsf{Respond}(x, w, \mathsf{stt}, \mathsf{chl})$ to the challenge.

4. $\mathcal{V}$ accepts or rejects the proof depending on $\mathsf{Verify}(x, \mathsf{com}, \mathsf{chl}, \mathsf{res}) \in \{\bot, \top\}$.

Moreover, a Σ-protocol satisfies completeness, 2-special soundness, and special honest-verifier zero-knowledge. We will use zero-knowledge protocols of this shape, but we will relax the soundness condition: In the most cases regular soundness for language membership is sufficient, in other cases, we do not need to extract the full witness. Moreover, we are satisfied with computational soundness and statistical SHVZK.

Definition 4 (Zero-Knowledge Arguments). *A protocol $\Pi = (\mathcal{P}, \mathcal{V})$ that has the shape of a Σ-protocol (*Commit*,* $\mathsf{Respond}$*,* Verify*) for a witness relation* R *with witness domain* D *is*

(i) computationally sound *if no malicious $\mathcal{P}^*$ can make $\mathcal{V}$ accept a false statement x (i.e., there is no w such that $(x, w) \in \mathsf{R}$) with probability non-negligible in λ,*

(ii) statistically special honest-verifier zero-knowledge (SHVZK) *for* D *if there exists an algorithm* $\mathsf{Simulate}$ *that for a statement x and a given challenge* chl *produces matching* $(\mathsf{com}, \mathsf{res})$*, that is distributed statistically close (in σ) to real protocol transcripts with* chl *of $(\mathcal{P}, \mathcal{V})$ on input* $(x, w) \in \mathsf{R}$ *where* $w \in \mathsf{D}$*.*

(iii) complete *for* D *if in an honest interaction between $\mathcal{P}$ and $\mathcal{V}$ for $(x, w) \in \mathsf{R}$ with $w \in \mathsf{D}$, $\mathcal{V}$ always accepts.*

5.2 ZK Arguments for General Relations

Here we consider proofs for general relations R, which are conjunctions of n statements containing m secrets:

$$\mathsf{R} = \Big\{(Y_i, \mathbf{X}_i)_{i\in[n]}; \mathbf{w} \;\Big|\; \bigwedge_{i=1}^{n} \big[Y_i = \prod_{j=1}^{m} X_{i,j}^{w_j}\big] \wedge \text{ all } X \in \widehat{G}^{n(m+1)} \wedge \mathbf{w} \in \mathbb{Z}^m \Big\}. \tag{5}$$

The statement $X := (Y_i, \mathbf{X}_i)_{i\in[n]}$ consists of elements of $\widehat{G}$ either defined by the context, or chosen by $\mathcal{P}$, and the witness $\mathbf{w}$ usually consists of integers from a bounded range $w_j \in [-S, +S]$. The protocols are sound with respect to R, but completeness and SHVZK are only guaranteed for witnesses from this interval.[9]

For such a relation, we can write the canonical Σ protocol $\mathsf{C\Sigma P}(\mathsf{R})$ where $A, C \in \mathbb{N}$ are parameters:

- $\mathsf{Commit}(X, \mathbf{w})$: Sample $\mathsf{stt} := (r_1, \ldots, r_m) \in_R [A]^m$, compute $t_i := \prod_{j=1}^{m} X_{i,j}^{r_j}$ for $i \in [n]$, and output $(\mathsf{stt}, \mathsf{com} := (t_i)_{i\in[n]})$.
- The challenge is sampled $\mathsf{chl} \in_R [C]$.
- $\mathsf{Respond}(X, \mathbf{w}, \mathsf{stt}, \mathsf{chl})$: Compute $u_j := r_j + \mathsf{chl} \cdot w_j$ for $j \in [m]$, and output $\mathsf{res} := (u_1, \ldots, u_m)$.
- $\mathsf{Verify}(X, \mathsf{com}, \mathsf{chl}, \mathsf{res})$: If $Y_i, X_{i,j} \in \widehat{G}$ for $i \in [n], j \in [m]$, $u_j \in [-SC, +SC + A]$ for $j \in [m]$, and $t_i \cdot Y_i^{\mathsf{chl}} = \prod_{j=1}^{m} X_{i,j}^{u_j}$ for $i \in [n]$ output $\top$, otherwise $\bot$.
- $\mathsf{Simulate}(X, \mathsf{chl})$: Sample $\mathsf{res} := (u_1, \ldots, u_m) \in_R [-SC, +SC + A]^m$, compute $t_i := Y_i^{-\mathsf{chl}} \cdot \prod_{j\in[m]} X_{i,j}^{u_j}$ for $i \in [n]$, and output $\big(\mathsf{com} := (t_i)_{i\in[n]}, \mathsf{res}\big)$.

For this class of protocols, we can show the following properties:

Theorem 9. *If* R *be a relation as described in Equation 5, then:*

[9] For simplicity, we use the same bounds $[-S, +S]$ for each secret, but we could also specify separate bounds for each secret $s_j \in [-S_j, +S_j]$ and sample the randomness $r_j \in_R [A_j]$ s.t. $S_j C / A_j$ is negligible for every $j \in [m]$. This makes the protocol description more complicated, but it would be more efficient when we have secrets of different sizes.

(i) $\mathsf{C\Sigma P(R)}$ *is sound for* R *with soundness error* $1/C + \mathsf{negl}(\lambda)$ *under the* RO_C *assumption.*[10]

(ii) $\mathsf{C\Sigma P(R)}$ *is complete for* R *if* $\mathbf{w} \in [-S, +S]^m$.

(iii) $\mathsf{C\Sigma P(R)}$ *is statistical special honest-verifier zero-knowledge if* $\mathbf{w} \in [-S, +S]^m$ *and* SC/A *is negligible.*

Proof. While the proofs of completeness and SHVZK are standard, the soundness proof (i) is more involved and relies on our RO_C assumption:

(i) Consider a malicious PPT prover $\mathcal{P}^*$ that generates an instance X such that there is no witness $\mathbf{w}$ such that $(X, \mathbf{w}) \in \mathsf{R}$. Now $\mathcal{P}^*$ can guess $\mathcal{V}$'s random challenge $\mathsf{chl} \in_R [C]$ with probability $1/C$ and use the simulator (cf. (iii)) to obtain a transcript which makes $\mathcal{V}$ accept. Assume towards contradiction that $\mathcal{P}^*$ had a success probability that is strictly larger than $1/C$, and suppose $\mathrm{ord}(\widehat{G})$ was C-rough. Then, by averaging, there exists a first message com such that there exist third messages $\mathsf{res}, \mathsf{res}'$ for two different challenges $\mathsf{chl} \neq \mathsf{chl}'$ such that $\mathcal{V}$ accepts both $(\mathsf{com}, \mathsf{chl}, \mathsf{res})$ and $(\mathsf{com}, \mathsf{chl}', \mathsf{res}')$.
Note that all relevant group elements live in the subgroup $\langle Y_i, X_{i,j} \rangle_{i,j} \subseteq \widehat{G}$. Let $\ell := \mathrm{ord}(\langle Y_i, X_{i,j} \rangle_{i,j}) \mid \prod_i \mathrm{ord}(Y_i) \cdot \prod_{i,j} \mathrm{ord}(X_{i,j})$. Since $\mathrm{ord}(\widehat{G})$ is C-rough, ℓ must also be C-rough. Hence, as $|\mathsf{chl} - \mathsf{chl}'| < C$, $(\mathsf{chl} - \mathsf{chl}')$ is invertible modulo ℓ, i.e., we can find an integer $(\mathsf{chl} - \mathsf{chl}')^{-1} \in \mathbb{Z}$ such that $(\mathsf{chl} - \mathsf{chl}') \cdot (\mathsf{chl} - \mathsf{chl}')^{-1} = 1 \pmod{\ell}$. Therefore we have

$$\left[t_i \cdot Y_i^{\mathsf{chl}} = \prod_{i \in [m]} X_{i,j}^{u_j}\right] \wedge \left[t_i \cdot Y_i^{\mathsf{chl}'} = \prod_{i \in [m]} X_{i,j}^{u_j'}\right] \implies$$

$$Y_i^{\mathsf{chl}-\mathsf{chl}'} = \prod_{i \in [m]} X_{i,j}^{u_j - u_j'} \iff Y_i = \prod_{i \in [m]} X_{i,j}^{(u_j - u_j') \cdot (\mathsf{chl}-\mathsf{chl}')^{-1}},$$

where all the exponents are considered integers. By setting $w_j := (u_j - u_j') \cdot (\mathsf{chl} - \mathsf{chl}')^{-1}$ for $j \in [m]$ and $\mathbf{w} := (w_1, \ldots, w_m)$ we have found a witness such that $(X, \mathbf{w}) \in \mathsf{R}$ and, therefore, reached a contradiction.
So such a $\mathcal{P}^*$ cannot exist or $\mathrm{ord}(\widehat{G})$ is not C-rough after all. Hence, if we had such a $\mathcal{P}^*$, we could distinguish groups $\widehat{G}$ that do not have a C-rough order. By the RO_C assumption, such a distinguisher has at most negligible advantage. We conclude that $\mathsf{C\Sigma P(R)}$ has a soundness error of $1/C + \mathsf{negl}(\lambda)$.

(ii) Assuming the honest prover has secrets $w_j \in [-S, +S]$, then all the u_j are within the required range $[-SC, +SC + A]$ and the other checks passes as well:

$$t_i \cdot Y_i^{\mathsf{chl}} = \prod_{j=1}^{m} X_{i,j}^{r_j} \cdot \left(\prod_{j=1}^{m} X_{i,j}^{w_j}\right)^{\mathsf{chl}} = \prod_{j=1}^{m} X_{i,j}^{r_j + \mathsf{chl} \cdot w_j} = \prod_{j=1}^{m} X_{i,j}^{u_j}.$$

Hence, the honest verifier accepts.

(iii) To show special honest-verifier zero-knowledge, we use the $\mathsf{Simulate}$ algorithm: If SC/A is negligible, then the uniformly sampled u_j from the simulation are distributed statistically close to the u_j appearing in a real execution, where the honest prover uses a witness $\mathbf{w} \in [-S, +S]^m$. The t_i are chosen in the only possible way so that the verifier would accept the transcript. □

Corollary 10. *The protocols* $\Pi_{\mathsf{DLog}} := \mathsf{C\Sigma P(R_{DLog})}$ *and* $\Pi_{\mathsf{EqDLog}} := \mathsf{C\Sigma P(R_{EqDLog})}$ *are zero-knowledge arguments according to Definition 4 with* $\mathsf{D} := [-S, +S]$ *for the relations* $\mathsf{R_{DLog}}$ *and* $\mathsf{R_{EqDLog}}$, *respectively.*

[10] Note that the soundness property does not require the existence of a witness $\mathbf{w}$ such that each w_j is within the range $[-S, +S]$.

5.3 Proofs of Plaintext Knowledge and Correct Multiplication

For our MPC protocol in Section 7, we need additional protocols that are not only sound, but also allow for partial extraction of the witnesses. First, we need the parties to prove that whenever they publish a ciphertext, it is a) well-formed and b) they know the corresponding plaintext.

$$\mathsf{R}_{\mathsf{Enc}} = \{\mathsf{ct}; (m, s) \mid \mathsf{ct}_1 = \mathsf{ct}_{P1}^m \cdot g_q^s \wedge \mathsf{ct}_2 = \mathsf{ct}_{P2}^m \cdot \mathsf{pk}_{\mathsf{cl}}^s\} \tag{6}$$

The relation $\mathsf{R}_{\mathsf{Enc}}$ is parametrized by public parameters $\mathsf{pp}_{\mathsf{cl}}$ and a special public key $\widetilde{\mathsf{pk}_{\mathsf{cl}}} = (\mathsf{pk}_{\mathsf{cl}}, \mathsf{ct}_P)$, where $\mathsf{ct}_P = (\mathsf{ct}_{P1}, \mathsf{ct}_{P2})$ is an encryption of 1 (or 0 in the case of a lossy public key, see Section 3.2), but we generally omit them from the notation. We do not require a full argument of knowledge, since we only need to extract the encrypted message $a \in \mathbb{F}_q$, but not the randomness s. Therefore, we give a specialized definition:

Definition 5 (Proof of Plaintext Knowledge). *Let Π be a zero-knowledge argument as defined in Definition 4 for the relation $\mathsf{R}_{\mathsf{Enc}}$ (Equation (6)) (defined for a valid public key $\widetilde{\mathsf{pk}_{\mathsf{cl}}}$) with witness domain $\mathsf{D} := \mathbb{F}_q \times \mathrm{dom}(\mathcal{D}_q)$. Let $\mathsf{sk}_{\mathsf{cl}}$ be a secret key matching the public key in $\mathsf{R}_{\mathsf{Enc}}$. We say Π is a* Proof of Plaintext Knowledge (PoPK) *if additionally there exists an efficient algorithm* $\mathsf{Extract}$ *and the following holds: Given two accepting transcripts* $(\mathsf{com}, \mathsf{chl}, \mathsf{res}), (\mathsf{com}, \mathsf{chl}', \mathsf{res}')$ *corresponding to a ciphertext* ct *such that* $\mathsf{chl} \neq \mathsf{chl}'$*, then*

(i) ct *can be correctly decrypted* $\mathsf{Dec}(\mathsf{sk}_{\mathsf{cl}}, \mathsf{ct}) = m \in \mathbb{F}_q$*,*

(ii) and the same value m can be extracted from the transcripts:
$\mathsf{Extract}(\mathsf{ct}, (\mathsf{com}, \mathsf{chl}, \mathsf{res}), (\mathsf{com}, \mathsf{chl}', \mathsf{res}')) = m$*.*

We show that the canonical protocol $\mathsf{C\Sigma P}(\mathsf{R}_{\mathsf{Enc}})$ satisfies the above condition and define the corresponding extractor $\mathsf{Extract}$. Extraction of the message $m \in \mathbb{F}_q$ works, because it appears in the exponent of f of known order q. We cannot extract the randomness s, however, since we can neither invert $\mathsf{chl} - \mathsf{chl}'$ modulo the unknown order of g_q nor expect division over the integers to work. However, by using the RO_C assumption we are able to pretend that certain inverses exist modulo $\mathrm{ord}(g_q)$.

Theorem 11. *The protocol $\Pi_{\mathsf{PoPK}} := \mathsf{C\Sigma P}(\mathsf{R}_{\mathsf{Enc}})$ is a Proof of Plaintext Knowledge under the RO_C assumption.*

Proof. By Theorem 9, Π_{PoPK} is already a zero-knowledge argument according to Definition 4. Let $(\mathsf{com} := (t_1, t_2), \mathsf{chl}, \mathsf{res} := (u_1, u_2))$ and $(\mathsf{com} := (t_1, t_2), \mathsf{chl}', \mathsf{res}' := (u_1', u_2'))$ be two accepting transcripts. From the Verify equations we get

$$\begin{aligned} \mathsf{ct}_{P1}^{u_1} \cdot g_q^{u_2} &= t_1 \cdot \mathsf{ct}_1^{\mathsf{chl}} & \qquad \mathsf{ct}_{P2}^{u_1} \cdot \mathsf{pk}_{\mathsf{cl}}^{u_2} &= t_2 \cdot \mathsf{ct}_2^{\mathsf{chl}} \\ \mathsf{ct}_{P1}^{u_1'} \cdot g_q^{u_2'} &= t_1 \cdot \mathsf{ct}_1^{\mathsf{chl}'} & \qquad \mathsf{ct}_{P2}^{u_1'} \cdot \mathsf{pk}_{\mathsf{cl}}^{u_2'} &= t_2 \cdot \mathsf{ct}_2^{\mathsf{chl}'} \end{aligned}$$

Dividing the equation yields:

$$\mathsf{ct}_{P1}^{u_1 - u_1'} \cdot g_q^{u_2 - u_2'} = \mathsf{ct}_1^{\mathsf{chl} - \mathsf{chl}'} \qquad \mathsf{ct}_{P2}^{u_1 - u_1'} \cdot \mathsf{pk}_{\mathsf{cl}}^{u_2 - u_2'} = \mathsf{ct}_2^{\mathsf{chl} - \mathsf{chl}'}$$

Suppose, $\ell := \mathrm{ord}(\widehat{G})$ is C-rough. Since $|\mathsf{chl} - \mathsf{chl}'| < C$, it must be invertible modulo ℓ. Then we can write:

$$\begin{aligned} \mathsf{ct}_{P1}^{(u_1 - u_1') \cdot (\mathsf{chl} - \mathsf{chl}')^{-1}} \cdot g_q^{(u_2 - u_2') \cdot (\mathsf{chl} - \mathsf{chl}')^{-1}} &= \mathsf{ct}_1 \\ \mathsf{ct}_{P2}^{(u_1 - u_1') \cdot (\mathsf{chl} - \mathsf{chl}')^{-1}} \cdot \mathsf{pk}_{\mathsf{cl}}^{(u_2 - u_2') \cdot (\mathsf{chl} - \mathsf{chl}')^{-1}} &= \mathsf{ct}_2 \end{aligned}$$

Since $\widetilde{\mathsf{pk}_{\mathsf{cl}}}$ is assumed to be a valid public key and thus ct_P has the form $(g_q^\beta, \mathsf{pk}_{\mathsf{cl}}^\beta \cdot f)$, we can simplify to obtain:

$$\begin{aligned} g_q^{(\beta\cdot(u_1-u_1')+(u_2-u_2'))\cdot(\mathsf{chl}-\mathsf{chl}')^{-1}} &= \mathsf{ct}_1 \\ \mathsf{pk}_{\mathsf{cl}}^{(\beta\cdot(u_1-u_1')+(u_2-u_2'))\cdot(\mathsf{chl}-\mathsf{chl}')^{-1}} \cdot f^{(u_1-u_1')\cdot(\mathsf{chl}-\mathsf{chl}')^{-1}} &= \mathsf{ct}_2 \end{aligned}$$

We see that ct is a valid ciphertext with message $(u_1 - u_1') \cdot (\mathsf{chl} - \mathsf{chl}')^{-1} \bmod \ell$ and randomness $(\beta \cdot (u_1 - u_1') + (u_2 - u_2')) \cdot (\mathsf{chl} - \mathsf{chl}')^{-1} \bmod \ell$. While ℓ is unknown and hard to compute, we know that $\operatorname{ord}(f) = q \mid \ell$ and can efficiently compute $m := (u_1 - u_1') \cdot (\mathsf{chl} - \mathsf{chl}')^{-1} \bmod q$. Hence, we can define $\mathsf{Extract}$ to output m. Since ct is a valid ciphertext, this matches what $\mathsf{Dec}(\mathsf{sk}_{\mathsf{cl}}, \mathsf{ct})$ would produce. □

In the multiplication part of our MPC protocol, a party needs to multiply a publicly known ciphertext ct_b (which we assume to be valid) with a private value $a \in \mathbb{F}_q$ that is additionally encrypted as ct_a to produce a new ciphertext ct_c. We formalize this in the following relation, where s is the randomness used to encrypt a, and s' is the randomness used to randomize the resulting ciphertext.

$$\begin{aligned} \mathsf{R}_{\mathsf{Mult}} = \{(\mathsf{ct}_a, \mathsf{ct}_b, \mathsf{ct}_c); (a, s, s') \mid (\mathsf{ct}_a; (a, s)) \in \mathsf{R}_{\mathsf{Enc}} \wedge \\ \mathsf{ct}_{c,1} = \mathsf{ct}_{b,1}^a \cdot g_q^{s'} \wedge \mathsf{ct}_{c,2} = \mathsf{ct}_{b,2}^a \cdot \mathsf{pk}_{\mathsf{cl}}^{s'}\} \end{aligned} \tag{7}$$

Definition 6 (Proof of Correct Multiplication). *Let Π be a zero-knowledge argument as defined in Definition 4 for the relation $\mathsf{R}_{\mathsf{Mult}}$ (Equation (7)) (defined for a valid public key $\widetilde{\mathsf{pk}_{\mathsf{cl}}}$) with witness domain $\mathsf{D} := \mathbb{F}_q \times \operatorname{dom}(\mathcal{D}_q)^2$. We say Π is a* Proof of Correct Multiplication (PoCM) *if additionally it satisfies the PoPK extraction property (Definition 5) with respect to ct_a.*

Since $\mathsf{R}_{\mathsf{Mult}}$ is an extension of $\mathsf{R}_{\mathsf{Enc}}$, Theorem 11 immediately carries over:

Corollary 12. *The protocol $\Pi_{Mult} := \mathsf{C\Sigma P}(\mathsf{R}_{\mathsf{Mult}})$ is a Proof of Correct Multiplication under the RO_C assumption.*

6 Threshold Linearly Homomorphic Encryption

Here, we present our novel threshold encryption scheme based on the linearly homomorphic encryption scheme $\Pi_{\mathsf{hsm\text{-}cl}}$ of [CLT18; Cas+20]. To obtain a more efficient protocol we use the modified $\Pi^*_{\mathsf{hsm\text{-}cl}}$ encryption scheme (Section 3).

Bias in Distributed Key Generation The key generation KeyGen algorithm of the standard $\Pi_{\mathsf{hsm\text{-}cl}}$ scheme requires that the private key $\mathsf{sk}_{\mathsf{cl}}$ is sampled from $\mathcal{D}_q$, which induces an almost-uniform distribution of the public key $\mathsf{pk}_{\mathsf{cl}}$ in G^q. Hence, ideally, we would like to have a distributed key generation protocol that outputs a public key of this form. Gennaro et al. [Gen+07] noted that the classic distributed key generation protocol by Pedersen [Ped91] (with player elimination) allows an adversary to bias the distribution of the public key. They show that the generation of an unbiased public key is possible [Gen+07], however, it requires more rounds and additional setup. Gennaro et al. [Gen+07] also show that the bias can be tolerated in certain settings, for example in a threshold protocol for Schnorr signatures [Sch90]. In Section 3.1 we showed that the bias can also be tolerated in ElGamal-style encryption schemes, i.e., we showed that the $\Pi^*_{\mathsf{hsm\text{-}cl}}$ encryption scheme does still provide IND-CPA security even when the adversary is allowed to bias the distribution of the key. Hence, we use a simpler key generation based on Pedersen's [Ped91] protocol.

Special Public Keys Our protocol generates special public keys of the form $\widetilde{\mathsf{pk}}_{\mathsf{cl}} = (\mathsf{pk}_{\mathsf{cl}}, \mathsf{ct}_P)$ where ct_P is an encryption of $1 \in \mathbb{F}_q$ (see Section 3.2). Since our protocol might be useful for other applications, we note that if special public keys are not required, the protocol can easily be adapted to skip the generation of ct_P and just output the normal public key $\mathsf{pk}_{\mathsf{cl}}$.

Ideal Functionality We specify a UC functionality $\mathcal{F}_{\mathsf{TE}}$ in Figure 3. Apart from the Init procedure, it allows the parties to generate a key pair $(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}})$ such that the special public key $\widetilde{\mathsf{pk}}_{\mathsf{cl}}$ is made public, but nobody learns the corresponding secret key $\mathsf{sk}_{\mathsf{cl}}$. The KeyGen method allows the simulator $\mathcal{S}$ to influence the distribution of the generated key pair as specified by $\mathsf{BiasedSpecialKeyGen}_1^{\mathcal{S}}$ (Figure 2). Moreover, the functionality allows parties to decrypt $\mathit{\Pi}_{\mathsf{hsm\text{-}cl}}$ ciphertexts using the stored secret key. Defining a single functionality that contains both the key generation and the decryption simplifies the simulation proof, since the environment never sees the honest parties' shares of the secret key. A similar approach is taken, e.g., in [AF04] for UC threshold Schnorr signatures.

Ideal Threshold Encryption Functionality $\mathcal{F}_{\mathsf{TE}}$

Init On input $(\mathsf{Init}, 1^\lambda, q)$ from all parties, $\mathcal{F}_{\mathsf{TE}}$ generates $\mathsf{pp}_{\mathsf{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q)$. It stores $\mathsf{pp}_{\mathsf{cl}}$ and outputs them to all parties. This method must be called exactly once, and before any other call.

KeyGen On input (KeyGen) from all parties, $\mathcal{F}_{\mathsf{TE}}$ runs $(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}}) \leftarrow \mathsf{BiasedSpecialKeyGen}_1^{\mathcal{S}}(\mathsf{pp}_{\mathsf{cl}})$ with $\mathcal{S}$. When $\mathcal{S}$ responds with $\mathsf{continue}$, $\mathcal{F}_{\mathsf{TE}}$ sends $\widetilde{\mathsf{pk}}_{\mathsf{cl}}$ to all parties. This method must be called exactly once, and before any call to $\mathsf{Decrypt}$.

Decrypt On input $(\mathsf{Decrypt}, \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) \in \widehat{G}^2)$ from all parties, $\mathcal{F}_{\mathsf{TE}}$ computes $M := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}_{\mathsf{cl}}}$ and sends M to $\mathcal{S}$.[a] If $\mathcal{S}$ responds with abort, $\mathcal{F}_{\mathsf{TE}}$ aborts. Otherwise, if it responds with $\mathsf{continue}$, $\mathcal{F}_{\mathsf{TE}}$ sends $m \leftarrow \mathsf{CLSolve}(\mathsf{pp}_{\mathsf{cl}}, M)$ to all parties.

[a] We leak M instead of m to also be able to simulate the decryption in case ct is not a valid ciphertext since if $M \notin F \implies \mathsf{CLSolve}(\mathsf{pp}_{\mathsf{cl}}, M) = \perp$.

Fig. 3. Ideal Threshold Encryption Functionality

6.1 Distributed Key Generation and Threshold Decryption

Distributed Key Generation (DKG) We present our protocol in Figure 4. in the ($\mathcal{F}_{\mathsf{CL}}$, $\mathcal{F}_{\mathsf{Rand}}$)-hybrid model, where $\mathcal{F}_{\mathsf{Rand}}$ is a standard coin tossing functionality. The DKG protocol consists of two parts: First the parties generate a key pair $(\mathsf{pk}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}})$ such that the secret key $\mathsf{sk}_{\mathsf{cl}}$ is distributed among all parties. Then, they generate ct_P which is an encryption of 1 under the public key $\mathsf{pk}_{\mathsf{cl}}$.

Recall that a key pair is of the form $(\mathsf{sk}_{\mathsf{cl}}, \mathsf{pk}_{\mathsf{cl}} = g_q^{\mathsf{sk}_{\mathsf{cl}}})$, where $\mathsf{sk}_{\mathsf{cl}}$ is a sufficiently large random integer. Each party first samples their contribution α_i to the secret key, and then uses a variant of Feldman VSS (cf. Section 4.2) with base g_q to share it with all other parties. The properties of the VSS scheme and complaint resolution guarantee that for every P_i we have either a consistent secret sharing of α_i, or P_i is disqualified and ignored for the remainder of the key generation. The secret key is now well-defined as $\mathsf{sk}_{\mathsf{cl}} = \sum_{P_i \in Q} \alpha_i$, and each of these α_i can be reconstructed by a majority of all parties. Each party additionally obtains a share γ_i of $\mathsf{sk}_{\mathsf{cl}}$ such that any set of at

Protocol $\mathit{\Pi}_{\mathsf{TE}}$ (Part I)

The parties maintain a set $\mathcal{Q}$ of qualified parties initially containing all parties. After a party gets disqualified, it will be ignored by all honest parties.

Init Send $(\mathsf{Gen}, 1^\lambda, q)$ to $\mathcal{F}_{\mathsf{CL}}$ which returns $\mathsf{pp}_{\mathsf{cl}} = (q, \bar{s}, f, g_q, \widehat{G}, F; \rho)$.

KeyGen

1. Generation of a key pair $(\mathsf{pk}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}})$. All P_i proceed in parallel:
   (a) Sample the contribution to the secret key $\alpha_i \leftarrow \mathcal{D}_q$.
   (b) Share α_i as $(y_{i,1}, \ldots, y_{i,N}; C_{i,0}, \ldots, C_{i,t}) \leftarrow \mathsf{F\text{-}Share}(\alpha_i; \mathbf{r}_i; g_q)$ such that $y_{i,j}$ is privately sent to P_j, and $C_{i,0}, \ldots, C_{i,t}$ are broadcasted.
   (c) Compute $(\mathsf{com}^1_{i,k}, \mathsf{stt}^1_{i,k}) \leftarrow \mathit{\Pi}_{\mathsf{DLog}}.\mathsf{Commit}(C_{i,k}, r_{i,k})$ for $k \in [t]$ and $(\mathsf{com}^1_{i,0}, \mathsf{stt}^1_{i,0}) \leftarrow \mathit{\Pi}^{\mathsf{bin}}_{\mathsf{DLog}}.\mathsf{Commit}(C_{i,0}, \alpha_i)$. Broadcast $(\mathsf{com}^1_{i,k})_{k\in[0,t]}$.
   (d) Send $(\mathsf{Rand}, [C])$ to $\mathcal{F}_{\mathsf{Rand}}$ so that all parties receive $\mathsf{chl}^1 \in_R [C]$.
   (e) Broadcast $\mathsf{res}^1_{i,k} \leftarrow \mathit{\Pi}_{\mathsf{DLog}}.\mathsf{Respond}(C_{i,k}, r_{i,k}, \mathsf{stt}^1_{i,k}, \mathsf{chl}^1)$ for $k \in [t]$ and $\mathsf{res}^1_{i,0} \leftarrow \mathit{\Pi}^{\mathsf{bin}}_{\mathsf{DLog}}.\mathsf{Respond}(C_{i,0}, \alpha_i, \mathsf{stt}^1_{i,0}, \mathsf{chl}^1)$.
   (f) Verify shares received from $P_j \neq P_i$: Check if
      i. $\mathsf{F\text{-}Check}(i, y_{j,i}; C_{j,0}, \ldots, C_{j,t}; g_q) \overset{?}{=} \top$, and
      ii. $\mathit{\Pi}_{\mathsf{DLog}}.\mathsf{Verify}(C_{j,k}, \mathsf{com}^1_{j,k}, \mathsf{chl}^1, \mathsf{res}^1_{j,k}) \overset{?}{=} \top$ for all $k \in [t]$, and $\mathit{\Pi}^{\mathsf{bin}}_{\mathsf{DLog}}.\mathsf{Verify}(C_{j,0}, \mathsf{com}^1_{j,0}, \mathsf{chl}^1, \mathsf{res}^1_{j,0}) \overset{?}{=} \top$.

      If only Step 1(f)i failed, then broadcast a complaint against P_j.
   (g) For every complaint received by $P_j \neq P_i$, broadcast the value $y_{i,j}$.
   (h) If Step 1(f)ii failed for P_j, or if P_j broadcasted a value $y_{j,l}$ in response to P_l that does not satisfy Step 1(f)i, remove P_j from $\mathcal{Q}$.
2. Computing the public key and shares of the secret key
   (a) All parties compute the public key $\mathsf{pk}_{\mathsf{cl}} := \prod_{P_j \in \mathcal{Q}} C_{j,0}$ where the secret key is defined as $\mathsf{sk}_{\mathsf{cl}} := \sum_{P_j\in\mathcal{Q}} \alpha_j$.
   (b) Each P_i computes its share $\gamma_i := \sum_{P_j\in\mathcal{Q}} y_{j,i}$ of $\mathsf{sk}_{\mathsf{cl}}$.
   (c) All parties compute $\mathit{\Gamma}_i := \prod_{P_j\in\mathcal{Q}} C_{j,0}^{\Delta^2} \cdot \prod_{k=1}^{t} C_{j,k}^{(i^k)}$ for each P_i.
3. Continued in Figure 5.

Fig. 4. Distributed Key Generation and Decryption protocols for $\mathit{\Pi}_{\mathsf{hsm\text{-}cl}}$

least $(t+1)$ parties can reconstruct $\mathsf{sk}_{\mathsf{cl}}$. Moreover, each party has learned $\mathsf{pk}_i := g_q^{\alpha_i}$ for $P_i \in \mathcal{Q}$ and can compute the public key as $\mathsf{pk} = \prod_{P_i\in\mathcal{Q}} \mathsf{pk}_i$. Note that the Feldman VSS already reveals the parties' contributions to the public key pk_i. Hence, a rushing adversary is able to bias the resulting key based on the honest parties contributions.[11]

The aim of the second part is to generate a ciphertext $\mathsf{ct}_P = (\mathsf{ct}_1, \mathsf{ct}_2) = (g_q^\beta, f \cdot \mathsf{pk}_{\mathsf{cl}}^\beta)$, where β is again a large random value. Since ct_1 is basically an ephemeral public key, the protocol is very similar to the key generation part of $\mathit{\Pi}_{\mathsf{TE}}$: Every party samples their contribution β_i to the randomness β, and proves that their contribution $g_q^{\beta_i}$ has the right form. The difference is that now we also need to compute ct_2. Hence, each party also computes $\mathsf{pk}_{\mathsf{cl}}^{\beta_i}$ and uses $\mathit{\Pi}_{\mathsf{EqDLog}}$ to prove that their contributions to ct_1 and ct_2 are consistent. Moreover, parties that misbehave are not disqualified, but their contributions to ct_1 and ct_2 are essentially removed by setting them to 1. While it is not necessary that β_i can be reconstucted by a majority of the parties, we

[11] The protocol of [Gen+07] prevents this kind of bias in the setting of prime-order groups. We could instantiate their protocol also in the unknown order setting, but the setup would be significantly more complicated.

Protocol $\mathit{\Pi}_{\mathsf{TE}}$ (Part II)

KeyGen (continued) 3. Generation of ct_P. All P_i proceed in parallel:

(a) Sample the contribution to the randomness $\beta_i \leftarrow \mathcal{D}_q$.

(b) Share β_i as $(z_{i,1}, \dots, z_{i,N}; D_{i,0}, \dots, D_{i,t}) \leftarrow \mathsf{F\text{-}Share}(\beta_i; \mathsf{s}_i; g_q)$ such that $z_{i,j}$ is privately sent to P_j, and $D_{i,0}, \dots, D_{i,t}, D'_{i,0} := \mathsf{pk}_{\mathsf{cl}}^{\beta_i}$ are broadcasted.

(c) Compute $(\mathsf{com}^2_{i,0}, \mathsf{stt}_{i,0}) \leftarrow \mathit{\Pi}^{\mathsf{bin}}_{\mathsf{EqDLog}}.\mathsf{Commit}((D_{i,0}, \mathsf{pk}_{\mathsf{cl}}, D'_{i,0}), s_{i,0})$, and $(\mathsf{com}^2_{i,k}, \mathsf{stt}_{i,k}) \leftarrow \mathit{\Pi}_{\mathsf{DLog}}.\mathsf{Commit}(D_{i,k}, s_{i,k})$ for $k \in [1,t]$, and broadcast $\mathsf{com}^2_{i,0}, \dots, \mathsf{com}^2_{i,t}$.

(d) Send $(\mathsf{Rand}, [C])$ to $\mathcal{F}_{\mathsf{Rand}}$ so that all parties receive $\mathsf{chl}^2 \in_R [C]$.

(e) Broadcast $\mathsf{res}^2_{i,0} \leftarrow \mathit{\Pi}^{\mathsf{bin}}_{\mathsf{EqDLog}}.\mathsf{Respond}((D_{i,0}, \mathsf{pk}_{\mathsf{cl}}, D'_{i,0}), s_{i,0}, \mathsf{stt}^2_{i,0}, \mathsf{chl}^2)$, and $\mathsf{res}^2_{i,k} \leftarrow \mathit{\Pi}_{\mathsf{DLog}}.\mathsf{Respond}(D_{i,k}, s_{i,k}, \mathsf{stt}^2_{i,k}, \mathsf{chl}^2)$ for $k \in [1,t]$.

(f) Verify shares received from $P_j \neq P_i$: Check if

i. $\mathsf{F\text{-}Check}(j, z_{j,i}; D_{j,0}, \dots, D_{j,t}; g_q) \overset{?}{=} \top$, and

ii. $\mathit{\Pi}^{\mathsf{bin}}_{\mathsf{EqDLog}}.\mathsf{Verify}((D_{j,0}, \mathsf{pk}_{\mathsf{cl}}, D'_{j,0}), \mathsf{com}^2_{i,0}, \mathsf{chl}^2, \mathsf{res}^2_{i,0}) \overset{?}{=} \top$, and

$\mathit{\Pi}_{\mathsf{DLog}}.\mathsf{Verify}(D_{j,k}, \mathsf{com}^2_{j,k}, \mathsf{chl}^2, \mathsf{res}^2_{j,k}) \overset{?}{=} \top$ for all $k \in [1,t]$,

If only Step 3(f)i failed, then broadcast a complaint against P_j.

(g) For every complaint received by $P_j \neq P_i$, broadcast the value $z_{i,j}$.

(h) If Step 3(f)ii failed for P_j, or if P_j broadcasted a value $z_{j,l}$ in response to P_l that does not satisfy Step 3(f)i, reset $D_{j,0} := D'_{j.0} := 1$.

Finally, everyone computes $\mathsf{ct}_P := \left(\prod_{P_i \in \mathcal{Q}} D_{i,0}, f \cdot \prod_{P_i \in \mathcal{Q}} D'_{i,0}\right)$.

Decrypt To jointly decrypt a $\mathit{\Pi}_{\mathsf{hsm\text{-}cl}}$ ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) \in \widehat{G}^2$, all P_i proceed in parallel as follows:

1. Compute $w_i := \mathsf{ct}_1^{\gamma_i \cdot \mathit{\Delta}}$ and $(\mathsf{com}_i, \mathsf{stt}_i) \leftarrow \mathit{\Pi}_{\mathsf{EqDLog}}.\mathsf{Commit}((\mathsf{ct}_1, \mathit{\Gamma}_i, w_i), \gamma_i \cdot \mathit{\Delta})$. Broadcast w_i and com_i.
2. Send $(\mathsf{Rand}, [C])$ to $\mathcal{F}_{\mathsf{Rand}}$ so that all parties receive $\mathsf{chl} \in_R [C]$.
3. Broadcast $\mathsf{res}_i \leftarrow \mathit{\Pi}_{\mathsf{EqDLog}}.\mathsf{Respond}((\mathsf{ct}_1, \mathit{\Gamma}_i, w_i), \gamma_i \cdot \mathit{\Delta}, \mathsf{stt}_i, \mathsf{chl})$.
4. Define $S := \{P_i \mid \mathit{\Pi}_{\mathsf{EqDLog}}.\mathsf{Verify}((\mathsf{ct}_1, \mathsf{pk}_j, w_j), \mathsf{com}_j, \mathsf{chl}, \mathsf{res}_j) = \top\}$.
5. Compute $W := \prod_{P_j \in S} w_j^{(\ell_j^S \cdot \mathit{\Delta})}$ and $\overline{M} := \mathsf{ct}_2^{\mathit{\Delta}^3} \cdot W^{-1}$.
6. Output $m := \mathsf{CLSolve}(\mathsf{pp}_{\mathsf{cl}}, \overline{M}) \cdot \mathit{\Delta}^{-3} \bmod q$.

Fig. 5. Distributed Key Generation and Decryption protocols for $\mathit{\Pi}_{\mathsf{hsm\text{-}cl}}$

nevertheless share it with a Feldman VSS. This is only necessary for the proof, since it allows the simulator to reconstruct the β_i of the corrupted $P_i \in \mathcal{C}$. Alternatively, an online-extractible proof for the $\mathsf{R}_{\mathsf{EqDLog}}$ relation would suffice as well. Finally, the parties can multiply f into the result to get an encryption of 1.

Distributed Decryption To decrypt a ciphertext ct, each party P_i computes a partial decryption $w_i := \mathsf{ct}_1^{\gamma_i \cdot \mathit{\Delta}}$, and proves that they used their share of $\mathsf{sk}_{\mathsf{cl}}$ via $\mathit{\Pi}_{\mathsf{EqDLog}}$. Since at least $t+1$ parties do this honestly, they can reconstruct $\mathsf{sk}_{\mathsf{cl}}$ in the exponent of ct_1 and perform the remainder of the $\mathit{\Pi}_{\mathsf{hsm\text{-}cl}}$ decryption.

6.2 Proof of Security

We formally state the security of $\mathit{\Pi}_{\mathsf{TE}}$ in Theorem 13.

Theorem 13. *The protocol $\mathit{\Pi}_{\mathsf{TE}}$ (Figure 4) securely realizes the functionality $\mathcal{F}_{\mathsf{TE}}$ (Figure 3) in the $(\mathcal{F}_{\mathsf{CL}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model with secure channels and broadcast with static and computational* *security tolerating up to $t < N/2$ corruptions under the ORD and RO_{N+1} assumptions when $1/C = \mathsf{negl}(\lambda)$.*

Proof. We first show correctness of the protocol, and then prove security by simulation.

Correctness Let us first convince ourselves that a valid key pair $(\mathsf{pk}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}})$ is generated during the first part of the key generation.

Initially, each P_i samples α_i and shares it as $(y_{i,1}, \ldots, y_{i,N}; C_{i,0}, \ldots, C_{i,t}) \leftarrow \mathsf{F\text{-}Share}(\alpha_i; \mathbf{r}_i; g_q)$. The checks in Step 1f make sure that the sharing of each qualified $P_i \in \mathcal{Q}$ is valid. That means, for each $P_i \in \mathcal{Q}$, we have $y_{i,j} = f_i(j)$ for the polynomial f_i defined by $f_i(X) = \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_{i,k} \cdot X^k$ for all P_j, and $C_{i,0} = g_q^{\alpha_i}$ and $C_{i,k} = g_q^{r_{i,k} \cdot \Delta}$ for $k \in [t]$.

The public key is defined as $\mathsf{pk}_{\mathsf{cl}} = \prod_{P_j \in \mathcal{Q}} C_{j,0} = g_q^{\sum_{P_j \in \mathcal{Q}} \alpha_j}$ such that $\mathsf{sk}_{\mathsf{cl}} := \sum_{P_j \in \mathcal{Q}} \alpha_j$ is the corresponding secret key. Moreover, each P_i computes $\gamma_i := \sum_{P_j \in \mathcal{Q}} y_{j,i}$ which are shares of $\mathsf{sk}_{\mathsf{cl}}$ shared via a polynomial

$$\bar{f}(X) = \sum_{P_j \in \mathcal{Q}} f_j(X) = \left(\sum_{P_j \in \mathcal{Q}} \alpha_j \right) \cdot \Delta + \sum_{k=1}^{t} \left(\sum_{P_j \in \mathcal{Q}} r_{j,k} \right) \cdot X^k.$$

Then all parties also compute for each P_i

$$\begin{aligned} \Gamma_i := \prod_{P_j \in \mathcal{Q}} C_{j,0}^{\Delta^2} \cdot \prod_{k=1}^{t} C_{j,k}^{(i^k)} = \prod_{P_j \in \mathcal{Q}} (g_q^{\alpha_j})^{\Delta^2} \cdot \prod_{k=1}^{t} (g_q^{r_{j,k} \cdot \Delta})^{(i^k)} \\ = \prod_{P_j \in \mathcal{Q}} (g_q^{\alpha_j})^{\Delta^2} \cdot \prod_{k=1}^{t} (g_q^{r_{j,k} \cdot \Delta})^{(i^k)} = g_q^{\Delta \cdot \sum_{P_j \in \mathcal{Q}} \left(\alpha_j \cdot \Delta + \sum_{k=1}^{t} r_{j,k} \cdot (i^k) \right)} = g_q^{\Delta \cdot \gamma_j}. \end{aligned}$$

Consider what happens when a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = (g_q^s, (g_q^{\mathsf{sk}_{\mathsf{cl}}})^s \cdot f^m)$ is being decrypted. Every P_i publishes $w_i := \mathsf{ct}_1^{\gamma_i \cdot \Delta} = \mathsf{ct}_1^{\bar{f}(i) \cdot \Delta}$ and proves consistency w.r.t. Γ_i. The set S contains all parties that behave honestly in this step, so we have $|S| \geq t + 1$. Using the Lagrange coefficients ℓ_i^S, we reconstruct $\mathsf{sk}_{\mathsf{cl}}$ in the exponent:

$$W := \prod_{P_j \in S} w_j^{\ell_j^S \cdot \Delta} = \mathsf{ct}_1^{\sum_{P_j \in S} \bar{f}(j) \cdot \ell_j^S \cdot \Delta^2} = \mathsf{ct}_1^{\mathsf{sk}_{\mathsf{cl}} \cdot \Delta^3}.$$

Note that we accumulate three factors Δ:

- When sharing α_i as $\mathsf{Share}(\alpha_i; \mathbf{r}_i)$ then the resulting polynomial f_i has actually $\alpha_i \cdot \Delta$ as constant term.
- In the definition of w_i, the share γ_i is multiplied by Δ.
- Finally, to do Lagrange interpolation in the exponent, we need to multiply the Lagrange coefficients by Δ to obtain integer exponents.

We can adjust for this by also raising ct_2 to Δ^3:

$$\overline{M} := \mathsf{ct}_2^{\Delta^3} \cdot W^{-1} = g_q^{\mathsf{sk}_{\mathsf{cl}} \cdot s \cdot \Delta^3} \cdot f^{m \cdot \Delta^3} \cdot (g_q^{s \cdot \mathsf{sk}_{\mathsf{cl}} \cdot \Delta^3})^{-1} = f^{m \cdot \Delta^3}.$$

Finally, we take the discrete logarithm of $\overline{M}$ and also remove Δ^3 in the plaintext space: $m = \mathsf{CLSolve}(\mathsf{pp}_{\mathsf{cl}}, \overline{M}) \cdot \Delta^{-3} \bmod q$.

Simulation We set up the simulation as follows: The environment $\mathcal{Z}$ selects a set $\mathcal{C}$ of at most $t < N/2$ parties to corrupt. $\mathcal{Z}$ sends $(\mathsf{corrupt}, P_i)$ for each $P_i \in \mathcal{C}$ to the simulator $\mathcal{S}$ who forwards it to $\mathcal{F}_{\mathsf{TE}}$. Now $\mathcal{S}$ controls the communication of the P_i to $\mathcal{F}_{\mathsf{TE}}$. $\mathcal{S}$ sets up simulated copies of the all parties, and forwards the instructions of $\mathcal{Z}$ for the corrupted parties to their simulated counterparts.

To prove adaptive security for their DKG protocol, Abe et al. [AF04] used the single-inconsistent-player (SIP) variant of UC. We only aim for static security, but use the same underlying idea in the standard UC model: $\mathcal{S}$ selects a distinguished honest party $P_h \in \mathcal{H}$, and lets all other honest parties $\mathcal{H} \setminus \{P_h\}$ act according to Π_{TE}. The messages sent by P_h, however, are chosen such that the resulting view of the environment is consistent with the values output by $\mathcal{F}_{\mathsf{TE}}$. Hence, $\mathcal{S}$ does not always know the witnesses needed to perform the zero-knowledge arguments. Instead it uses the corresponding simulators and programs the $\mathcal{F}_{\mathsf{Rand}}$ functionality with the used challenges. Since $\mathcal{S}$ controls the honest majority of (simulated) parties, it knows enough shares to extract the contributions α_i of the corrupted parties from the VSS. The full $\mathcal{S}$ is given in Figures 6 and 7.

Simulator for Π_{TE} (Part I)

Init Simulate the generation of $\mathsf{pp}_{\mathsf{cl}}$ by running $\mathsf{pp}_{\mathsf{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q)$.

KeyGen 1. Simulate generation of $\mathsf{pk}_{\mathsf{cl}}$. Run $\mathsf{BiasedKeyGen}^{\mathcal{S}}$ with $\mathcal{F}_{\mathsf{TE}}$.

(a) Send (KeyGen) to $\mathcal{F}_{\mathsf{TE}}$ and receive the intermediate public key $\mathsf{pk}^*_{\mathsf{cl}}$.

(b) For all $P_i \in \mathcal{H} \setminus \{P_h\}$, simulate Steps 1a to 1c according to Π_{TE}.

(c) For P_h, instead send $C_{h,0}, \ldots, C_{h,t}$ computed as follows:

  i. Sample $\alpha_h \leftarrow \mathcal{D}_q$ and share it as $(y_{h,1}, \ldots, y_{h,N}) \leftarrow \mathsf{Share}(\alpha_h; \mathbf{r}_h)$.

  ii. Set $C_{h,0} := \mathsf{pk}^*_{\mathsf{cl}} \cdot \prod_{P_i \in \mathcal{H} \setminus \{P_h\}} C_{i,0}^{-1}$.

  iii. Use Lemma 8 to get $C_{h,1}, \ldots, C_{h,t}$ such that $\mathsf{F\text{-}Check}(j, y_{h,j}; C_{h,0}, \ldots, C_{h,t}; g_q)$ holds for all $P_j \in \mathcal{C}$.

  iv. Sample $\mathsf{chl}^1 \in_R [C]$, run $(\mathsf{com}^1_{h,0}, \mathsf{res}^1_{h,0}) \leftarrow \Pi^{\mathsf{bin}}_{\mathsf{DLog}}.\mathsf{Simulate}(C_{h,0}, \mathsf{chl}^1)$, and $(\mathsf{com}^1_{h,k}, \mathsf{res}^1_{h,k}) \leftarrow \Pi_{\mathsf{DLog}}.\mathsf{Simulate}(C_{h,k}, \mathsf{chl}^1)$ for $k \in [t]$. Send $(\mathsf{com}^1_{h,k})_{k \in [0,t]}$

(d) Receive shares and checking values $(y_{i,j}; C_{i,0}, \ldots, C_{i,t})$ and $(\mathsf{com}^1_{i,0}, \ldots, \mathsf{com}^1_{i,t})$ from all $P_i \in \mathcal{C}$ for all $P_j \in \mathcal{H}$.

(e) Simulate the call to $\mathcal{F}_{\mathsf{Rand}}$ and send chl^1 to all parties.

(f) For all $P_i \in \mathcal{H} \setminus \{P_h\}$, compute $\mathsf{res}^1_{i,0} \leftarrow \Pi^{\mathsf{bin}}_{\mathsf{DLog}}.\mathsf{Respond}(C_{i,0}, \alpha_i, \mathsf{stt}^1_{i,0}, \mathsf{chl}^1)$ and $\mathsf{res}^1_{i,k} \leftarrow \Pi_{\mathsf{DLog}}.\mathsf{Respond}(C_{i,k}, r_{i,k}, \mathsf{stt}^1_{i,k}, \mathsf{chl}^1)$ for $k \in [t]$. Send $(\mathsf{res}^1_{i,k})_{k \in [0,t]}$ for all $P_i \in \mathcal{H}$ and receive $(\mathsf{res}^1_{i,k})_{k \in [0,t]}$ from all $P_i \in \mathcal{C}$.

(g) Simulate the complaint resolution:

  i. Respond honestly to all complaints against honest parties.

  ii. Broadcast a complaint on behalf of $P_j \in \mathcal{H}$ if they received an invalid share $y_{i,j}$ from $P_i \in \mathcal{C}$.

  iii. If a $P_i \in \mathcal{C}$ responds to a complaint with a a corrected value $y'_{i,j}$: Remove P_i from $\mathcal{Q}$ if $y'_{i,j}$ is invalid. Otherwise, reset $y_{i,j} := y'_{i,j}$.

(h) Reconstruct α_i from the shares $\{(j, y_{i,j})\}_{P_j \in \mathcal{H}}$ for each $P_i \in \mathcal{C} \cap \mathcal{Q}$.

(i) Set $\delta := \sum_{P_i \in \mathcal{C} \cap \mathcal{Q}} \alpha_i$, and send δ to $\mathcal{F}_{\mathsf{TE}}$.

(j) Define $\mathsf{pk}_{\mathsf{cl}} := \prod_{P_j \in \mathcal{Q}} C_{j,0} = \mathsf{pk}^*_{\mathsf{cl}} \cdot g_q^{\delta}$, and set $\gamma_i := \sum_{P_j \in \mathcal{Q}} y_{j,i}$ and $\Gamma_i := g_q^{\gamma_i \cdot \Delta}$ for all P_i.

2. Continued in Figure 7.

Fig. 6. Simulator for the key generation protocol in Π_{TE}

Simulator for $\mathit{\Pi}_{\mathsf{TE}}$ (Part II)

KeyGen (continued) 2. Simulate generation of ct_P. Continue of $\mathsf{BiasedSpecialKeyGen}_1^{\mathcal{S}}$ with $\mathcal{F}_{\mathsf{TE}}$.

(a) Receive the intermediate ciphertext $\mathsf{ct}_P^* = (\mathsf{ct}_1^*, \mathsf{ct}_2^*)$.

(b) For all $P_i \in \mathcal{H} \setminus \{P_h\}$, Simulate Steps 3a to 3c according to $\mathit{\Pi}_{\mathsf{TE}}$:

(c) For P_h, instead send $D_{h,0}, \ldots, D_{h,t}, D'_{h,0}$ computed as follows:

  i. Sample $\beta_h \leftarrow \mathcal{D}_q$ and share it as $(z_{h,1}, \ldots, z_{h,N}) \leftarrow \mathsf{Share}(\beta_h; \mathbf{r}_h)$.

  ii. Set $D_{h,0} := \mathsf{ct}_1^* \cdot \prod_{P_i \in \mathcal{H}\setminus\{P_h\}} D_{i,0}^{-1}$.

  iii. Set $D'_{h,0} := \mathsf{ct}_2^* \cdot f^{-1} \cdot \prod_{P_i \in \mathcal{H}\setminus\{P_h\}} D'^{-1}_{i,0}$.

  iv. Use Lemma 8 to get $D_{h,1}, \ldots, D_{h,t}$ such that $\mathsf{F\text{-}Check}(j, z_{h,j}; D_{h,0}, \ldots, D_{h,t}; g_q)$ holds for all j such that $P_j \in \mathcal{C}$.

  v. Sample $\mathsf{chl}^2 \in_R [C]$, run

    A. $(\mathsf{com}_{h,0}^2, \mathsf{chl}^2, \mathsf{res}_{h,0}^2) \leftarrow \mathit{\Pi}_{\mathsf{EqDLog}}^{\mathsf{bin}}.\mathsf{Simulate}((D_{h,0}, \mathsf{pk}_{\mathsf{cl}}, D'_{h,0}), \mathsf{chl}^2)$, and

    B. $(\mathsf{com}_{h,k}^2, \mathsf{chl}^2, \mathsf{res}_{h,k}^2) \leftarrow \mathit{\Pi}_{\mathsf{DLog}}.\mathsf{Simulate}(D_{h,k}, \mathsf{chl}^2)$ for $k \in [1, t]$,

  and send $\mathsf{com}_{h,k}^2$ for $k \in [0, t]$.

(d) Receive shares and checking values $(z_{i,j}; D_{i,0}, \ldots, D_{i,t})$, $D'_{i,0}$, and $(\mathsf{com}_{i,0}^2, \ldots, \mathsf{com}_{i,t}^2)$ from all $P_i \in \mathcal{C}$ for all $P_j \in \mathcal{H}$.

(e) Simulate the call to $\mathcal{F}_{\mathsf{Rand}}$ and send chl^2 to all parties.

(f) For all $P_i \in \mathcal{H} \setminus \{P_h\}$, compute

  i. $\mathsf{res}_{i,0}^2 \leftarrow \mathit{\Pi}_{\mathsf{EqDLog}}^{\mathsf{bin}}.\mathsf{Respond}((D_{i,0}, \mathsf{pk}_{\mathsf{cl}}, D'_{i,0}), r_{i,0}, \mathsf{stt}_{i,0}^2, \mathsf{chl}^2)$, and

  ii. $\mathsf{res}_{i,k}^2 \leftarrow \mathit{\Pi}_{\mathsf{DLog}}.\mathsf{Respond}(D_{i,k}, r_{i,k}, \mathsf{stt}_{i,k}^2, \mathsf{chl}^2)$ for $k \in [1, t]$.

  Send $\mathsf{res}_{i,k}^2$ for all $P_i \in \mathcal{H}$ and receive $\mathsf{res}_{i,k}^2$ from all $P_i \in \mathcal{C}$ (for $k \in [0, t]$).

(g) Simulate the complaint resolution as in Step 1g above, but instead of disqualifying misbehaving $P_j \in \mathcal{C}$, reset $D_{j,0} := D'_{j.0} := 1$ and $\beta_j := 0$.

(h) Reconstruct β_i from the shares $\{(j, z_{i,j})\}_{P_j \in \mathcal{H}}$ for all other $P_i \in \mathcal{C} \cap \mathcal{Q}$.

(i) Set $\varepsilon := \sum_{P_i \in \mathcal{C}\cap\mathcal{Q}} \beta_i$, and send ε to $\mathcal{F}_{\mathsf{TE}}$.

(j) Define $\mathsf{ct}_P := \left(\prod_{P_j \in \mathcal{Q}} D_{j,0}, F \cdot \prod_{P_j \in \mathcal{Q}} D'_{j,0}\right) = \left(\mathsf{ct}_1^* \cdot g_q^{\varepsilon}, \mathsf{ct}_2^* \cdot \mathsf{pk}_{\mathsf{cl}}^{\varepsilon}\right)$. Send $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{TE}}$.

Decrypt 1. Send $(\mathsf{Decrypt}, \mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) \in \widehat{G}^2)$ on behalf of the corrupted parties to $\mathcal{F}_{\mathsf{TE}}$ and receive the partially decrypted message M.

2. Compute $\overline{M} := M^{\Delta^3}$ and $W := \mathsf{ct}_2^{\Delta^3} \cdot \overline{M}^{-1}$.
3. Set $A := \{0\} \cup \{i \mid P_i \in \mathcal{C}\}$ such that $|A| = t + 1$.
4. Compute $\overline{w}_i := \mathsf{ct}_1^{\gamma_i}$ and $w_i := \overline{w}_i^{\Delta}$ for $P_i \in \mathcal{C}$.
5. Set $\overline{w}_0 := \mathsf{ct}_2 \cdot M^{-1}$.
6. Compute $w_i := \overline{w}_0^{\ell_0^A(i)\cdot\Delta} \cdot \prod_{P_j \in \mathcal{C}} \overline{w}_j^{\ell_j^A(i)\cdot\Delta}$ for each $P_i \in \mathcal{H}$.
7. Sample $\mathsf{chl} \in_R [C]$.
8. For all $P_i \in \mathcal{H}$ run $(\mathsf{com}_i, \mathsf{chl}, \mathsf{res}_i) \leftarrow \mathit{\Pi}_{\mathsf{EqDLog}}.\mathsf{Simulate}((\mathsf{ct}_1, \Gamma_i, w_i), \mathsf{chl})$.
9. Send (w_i, com_i) for all $P_i \in \mathcal{H}$.
10. Simulate the call to $\mathcal{F}_{\mathsf{Rand}}$ and send chl to all parties.
11. Send res_i for all $P_i \in \mathcal{H}$.
12. Send $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{TE}}$.

Fig. 7. Simulator for the decryption protocol in $\mathit{\Pi}_{\mathsf{TE}}$

Set of Qualified Parties and Secret Key are Well-Defined All honest parties have computed the same set $\mathcal{Q}$ after Step 1h of $\mathit{\Pi}_{\mathsf{TE}}$, because disqualification of parties depend only on information that was broadcasted, and is thus consistent among all honest parties. Moreover, no honest party gets disqualified. Since honest parties adhere to the protocol, no other honest parties would complain against them. If a corrupted party complains against an honest party, it will respond with that party's share which satisfies F-Check, so no other honest party will consider it disqualified. Let $P_i \in \mathcal{Q}$ be still qualified party. If $P_i \in \mathcal{H}$, then the honest parties have $\geq t+1$ valid shares of α_i. If $P_i \in \mathcal{C}$ is corrupted and not disqualified, then it has responded with valid shares to all complaints, and, by soundness of Π_{DLog} and $\Pi^{\mathsf{bin}}_{\mathsf{DLog}}$, we have $C_{i,0}, \ldots, C_{i,t} \in G^q$ except with negligible probability. Thus, each honest party $P_j \in \mathcal{H}$ has either directly received a valid share $y_{i,j}$ from P_i in Step 1b, or a valid share $y_{i,j}$ has been broadcasted by P_i in Step 1g after P_j had issued a complaint in Step 1f. Hence, the honest parties know $\geq t+1$ consistent shares. By Lemma 6, this determines α_i.

We can apply Lemma 6 here, because we use the Feldman VSS with base g_q and we use with $\Pi^{\mathsf{bin}}_{\mathsf{DLog}}$ a proof of knowledge for C_0. By the RO_{N+1} assumption (Def. 3), the parameters $\mathsf{pp}_{\mathsf{cl}}$ are indistinguishable to parameters where g_q has an order which is co-prime to Δ.

Indistinguishability of the Simulation of KeyGen Following the KeyGen protocol, the simulation also consists of two phases: The first concerns simulating the generation of a key pair $(\mathsf{pk}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}})$; the second handles the generation of ct_P. In the following, we cover both parts separately.

In the first part, the messages sent by the simulated parties $P_i \in \mathcal{H} \setminus \{P_h\}$ are distributed exactly as in the real execution since $\mathcal{S}$ lets them behave according to protocol. The only deviation happens in Step 1c of the simulation:

1. $\mathcal{S}$ lets P_h send a value $C_{h,0}$ that is distributed correctly given the intermediate public key $\mathsf{pk}^*_{\mathsf{cl}}$ generated by $\mathcal{F}_{\mathsf{TE}}$. The other values $C_{h,1}, \ldots, C_{h,k}$ are produced using Lemma 8. Hence, the $C_{h,1}, \ldots, C_{h,k}$ have the correct distribution given $C_{h,0}$ and the corrupted parties' shares $\{y_j \mid P_j \in \mathcal{C}\}$; they are actually fully determined if $|\mathcal{C}| = t$).
   In the real protocol, it holds $C_{h,0} = g_q^{\alpha_h}$, where α_h is the value shared by P_h, but in the simulation, $C_{h,0}$ and the shared α_h are unrelated. Since the Feldman VSS reveals no other information about α_h and the environment sees at most t shares of the corrupted parties $P_i \in \mathcal{C}$, it cannot detect the difference.
2. Additionally, the proofs by P_h are simulated which is possible, since $\mathcal{S}$ can sample the challenge chl^1 in advance and then program $\mathcal{F}_{\mathsf{Rand}}$ to output it in Step 1e. By SHVZK of Π_{DLog} and $\Pi^{\mathsf{bin}}_{\mathsf{DLog}}$, the simulated proofs are indistinguishable from normally computed proofs.

Since $\mathcal{S}$ controls the honest parties, it has $t+1$ shares of α_i for each qualified $P_i \in \mathcal{Q}$. Thus, $\mathcal{S}$ can extract the contributions of the corrupted, but still qualified parties' contributions in Step 1h, and submit their sum δ in Step 1i to $\mathcal{F}_{\mathsf{TE}}$ to complete the $\mathsf{BiasedKeyGen}^{\mathcal{S}}$ procedure.

Note that when $\mathcal{S}$ defines the values γ_i and Γ_i for each parties, then, by construction, $(\gamma_1, \ldots, \gamma_N)$ is a secret sharing of $\sum_{P_i \in \mathcal{Q}} \alpha_i \neq \mathsf{sk}_{\mathsf{cl}}$.

The second half of the simulation is very similar to the first. The only difference is that the parties now need to generate a ciphertext $\mathsf{ct}_P = (\mathsf{ct}_1, \mathsf{ct}_2)$ consisting of two components. Again the $\mathcal{S}$ lets all $P_i \in \mathcal{H} \setminus \{P_h\}$ follow the protocol and chooses P_h's values such that the honest parties' contributions match what $\mathcal{F}_{\mathsf{TE}}$ outputs as preliminary ciphertext ct^*_P. Then it uses the Feldman shares to extract the randomness contributions β_i from all corrupted parties that were not caught misbehaving. For all other parties β_i is reset to 0 according to the protocol, so that they become irrelevant.

Hence, we conclude that overall the environment's view in the KeyGen simulation is indistinguishable from its view of a real protocol execution.

Indistinguishability of the Simulation of Decrypt First, recall that, in the simulation, $(\gamma_1, \ldots, \gamma_N)$ is a secret sharing of $\alpha := \sum_{P_i \in \mathcal{Q}} \alpha_i$, *not* of $\mathsf{sk}_{\mathsf{cl}}$, which is unknown to $\mathcal{S}$.

If $\mathcal{S}$ would let the honest parties execute the protocol using their shares γ_h, then the decrypted value will most likely be invalid and not match the output of $\mathcal{F}_{\mathsf{TE}}$, since the ciphertext would essentially be decrypted with a random secret key α, which is unrelated to $\mathsf{sk}_{\mathsf{cl}}$. Instead it uses the partially decrypted message M received from $\mathcal{F}_{\mathsf{TE}}$ to compute messages w_i for $P_i \in \mathcal{H}$ that are consistent with M and the adversary's view. Note that the t shares γ_i of the corrupted parties $P_i \in \mathcal{C}$ and the actual secret key $\mathsf{sk}_{\mathsf{cl}}$ uniquely define what the honest parties need to send, since all points are supposed to lie on some degree-t polynomial $\bar{f}$.

Hence, we use Lagrange interpolation in the exponent of ct_1 to find values consistent with $\bar{f}$ such that $\bar{f}(0) = \mathsf{sk}_{\mathsf{cl}}$ and $\bar{f}(i) = \gamma_i$ for $P_i \in \mathcal{C}$. By definition of M, we have $M = \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}_{\mathsf{cl}}} \iff \mathsf{ct}_1^{\mathsf{sk}_{\mathsf{cl}}} = \mathsf{ct}_2 \cdot M^{-1}$, and set $\overline{w}_0 := \mathsf{ct}_2 \cdot M^{-1}$. If the corrupted parties would behave correctly, they would send $w_i := \mathsf{ct}_1^{\gamma_i \cdot \Delta}$. Since Lagrange interpolation in the exponent already adds a factor Δ, we work with $\overline{w}_i := \mathsf{ct}_1^{\gamma_i}$. For each $P_i \in \mathcal{H}$, we set

$$\begin{aligned} w_i := \overline{w}_0^{\ell_0^A(i) \cdot \Delta} \cdot \prod_{P_j \in \mathcal{C}} \overline{w}_j^{\ell_j^A(i) \cdot \Delta} &= \mathsf{ct}_1^{\mathsf{sk}_{\mathsf{cl}} \cdot \ell_0^A(i) \cdot \Delta} \cdot \prod_{P_j \in \mathcal{C}} \mathsf{ct}_1^{\gamma_j \cdot \ell_j^A(i) \cdot \Delta} \\ &= \mathsf{ct}_1^{\mathsf{sk}_{\mathsf{cl}} \cdot \ell_0^A(i) \cdot \Delta + \sum_{P_j \in \mathcal{C}} \gamma_j \cdot \ell_j^A(i) \cdot \Delta} = \mathsf{ct}_1^{\bar{f}(j) \cdot \Delta}. \end{aligned}$$

Hence, w_i is exactly, what the adversary expects the honest parties to send given the view of the corrupted parties and the result of the decryption.

Now $\mathcal{S}$ does not know the discrete logarithms of the w_i to base ct_1, it uses the SHVZK simulator for Π_{EqDLog} to simulate proof transcripts in Step 8 for a randomly chosen challenge $\mathsf{chl} \in [C]$, and then programs the simulated $\mathcal{F}_{\mathsf{Rand}}$ to output chl as challenge. By SHVZK, the simulated messages $(\mathsf{com}_i)_{P_i \in \mathcal{H}}, (\mathsf{res}_i)_{P_i \in \mathcal{H}}$ are statistically indistinguishable from the messages appearing in an honestly generated proof. □

Guaranteed Output Delivery Assuming public parameters $\mathsf{pp}_{\mathsf{cl}}$ available, our protocol achieves *guaranteed output delivery (GOD)*: The adversary cannot prevent the honest parties from successfully completing the protocol to generate a shared key or decrypt a ciphertext. This holds because the simulator $\mathcal{S}$ defined in the proof of Theorem 13 never lets $\mathcal{F}_{\mathsf{TE}}$ abort, but always instructs it to deliver the output to the honest parties after a constant number of rounds.

7 MPC

We follow the approach of Damgård et al. [DN03] and define an arithmetic black box functionality (ABB) $\mathcal{F}_{\mathsf{ABB}}^q$ for reactive secure computation over the field $\mathbb{F}_q$ (see Figure 8). It allows parties to privately Input field elements to the ABB and to let it publicly Output stored values to all parties. Moreover, $\mathcal{F}_{\mathsf{ABB}}^q$ allows to compute Linear combinations on stored values with public coefficients, and to $\mathsf{Multiply}$ stored values. For all methods, $\mathcal{F}_{\mathsf{ABB}}^q$ conducts the operation if it receives corresponding messages from at least $t+1$ parties. This ensures that the (at most) t corrupted parties cannot prevent the honest parties from proceeding with the computation.

We realize this functionality with the protocol Π_{ABB}^q(see Figures 9 and 10), and state its security in Theorem 14. The construction of Π_{ABB}^q essentially follows the CDN [DN03] paradigm: All values in the computation are encrypted with $\Pi_{\mathsf{hsm\text{-}cl}}^*$ and the parties use $\mathcal{F}_{\mathsf{TE}}$ to generate a public key and decrypt ciphertexts. Whenever a party publishes a ciphertext or performs a multiplication it uses Π_{PoPK} and Π_{Mult} to prove correctness. We use that $\Pi_{\mathsf{hsm\text{-}cl}}^*$ has lossy public keys to prove indistinguishability of the simulation in the SPDZ-style [Dam+12].

Arithmetic Black Box Functionality $\mathcal{F}_{\mathsf{ABB}}^q$

Init $\mathcal{F}_{\mathsf{ABB}}^q$ starts with an internal state initialized as $\mathsf{st} := \emptyset$.

Input On input $(\mathsf{Input},\ P_i,\ \mathsf{vid},\ x)$ from party P_i and $(\mathsf{Input},\ P_i,\ \mathsf{vid},\ ?)$ from at least t other parties $P_j \neq P_i$, $\mathcal{F}_{\mathsf{ABB}}^q$ stores $\mathsf{st} := \mathsf{st} \cup \{(\mathsf{vid}, x)\}$.

Output On input $(\mathsf{Output},\ \mathsf{all},\ \mathsf{vid})$ from at least $t+1$ parties where $(\mathsf{vid}, x) \in \mathsf{st}$, $\mathcal{F}_{\mathsf{ABB}}^q$ sends x to $\mathcal{S}$. Once $\mathcal{S}$ sends $\mathsf{continue}$, send x to all parties.

Linear On input $(\mathsf{Linear},\ \mathsf{vid}_o,\ (\mathsf{vid}_1, \cdots, \mathsf{vid}_n),\ (a_0, \ldots, a_n))$ from at least $t+1$ parties, where $(\mathsf{vid}_i, x_i) \in \mathsf{st}$ for $i \in [n]$, all $a_i \in \mathbb{F}_q$, and $(\mathsf{vid}_o, \cdot) \notin \mathsf{st}$, $\mathcal{F}_{\mathsf{ABB}}^q$ stores $\mathsf{st} := \mathsf{st} \cup \{(\mathsf{vid}_o, a_0 + \sum_{i\in[n]} a_i \cdot x_i)\}$.

Multiply On input $(\mathsf{Multiply},\ \mathsf{vid}_x,\ \mathsf{vid}_y,\ \mathsf{vid}_z)$ from at least $t+1$ parties, where $(\mathsf{vid}_x, x), (\mathsf{vid}_y, y) \in \mathsf{st}$ and $(\mathsf{vid}_z, \cdot) \notin \mathsf{st}$, $\mathcal{F}_{\mathsf{ABB}}^q$ stores $\mathsf{st} := \mathsf{st} \cup \{(\mathsf{vid}_z, x \cdot y)\}$.

For all methods, $\mathcal{S}$ can decide, when the honest parties should receive their output.

Fig. 8. Arithmetic black box functionality for reactive secure computation over $\mathbb{F}_q$

MPC Protocol Π_{ABB}^q (Part I)

Init 1. Send $(\mathsf{Init},\ 1^\lambda,\ q)$ to $\mathcal{F}_{\mathsf{TE}}$ to obtain $\mathsf{pp}_{\mathsf{cl}}$.

2. Send (KeyGen) to $\mathcal{F}_{\mathsf{TE}}$ and receive special public key $\widetilde{\mathsf{pk}}_{\mathsf{cl}}$.

Let Π_{PoPK} and Π_{Mult} be proofs of plaintext knowledge/correct multiplication for the relations $\mathsf{R}_{\mathsf{Enc}}$ and $\mathsf{R}_{\mathsf{Mult}}$.

Input For $(\mathsf{Input},\ P_i,\ \mathsf{vid},\ x)$:

1. P_i samples $r \leftarrow \mathcal{D}_q$, computes $\mathsf{ct} \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, x; r)$ and $(\mathsf{com}, \mathsf{stt}) \leftarrow \Pi_{\mathsf{PoPK}}.\mathsf{Commit}(\mathsf{ct}, (x, r))$, and broadcasts ct and com.
2. Each P_j send $(\mathsf{Rand},\ [C])$ to $\mathcal{F}_{\mathsf{Rand}}$ which replies with chl.
3. P_i broadcasts $\mathsf{res} \leftarrow \Pi_{\mathsf{PoPK}}.\mathsf{Respond}(\mathsf{ct}, (x, r), \mathsf{stt}, \mathsf{chl})$.
4. If $\Pi_{\mathsf{PoPK}}.\mathsf{Verify}(\mathsf{ct}, \mathsf{com}, \mathsf{chl}, \mathsf{res}) = \bot$, the other parties set $x := 0$ and deterministically recompute $\mathsf{ct} \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, 0; 0)$.
5. Each P_j stores $(\mathsf{vid}, \mathsf{ct})$.

Output For $(\mathsf{Output},\ \mathsf{all},\ \mathsf{vid})$: Let $(\mathsf{vid}, \mathsf{ct})$ be the stored ciphertext. All parties send $(\mathsf{Decrypt},\ \mathsf{ct})$ to $\mathcal{F}_{\mathsf{TE}}$ and receive $m \in \mathbb{F}_q$ and output m.

Linear For $(\mathsf{Linear},\ \mathsf{vid}_o,\ (\mathsf{vid}_1, \cdots, \mathsf{vid}_n),\ (a_0, \ldots, a_n))$: Let $(\mathsf{vid}_i, \mathsf{ct}_i)_{i\in[n]}$ be the stored ciphertexts. Each P_j locally computes $\mathsf{ct}_o \leftarrow a_0 + \sum_{i\in[n]} a_i \cdot \mathsf{ct}_i$ without rerandomization. Each party stores the resulting ciphertext $(\mathsf{vid}_0, \mathsf{ct}_o)$.

Fig. 9. MPC protocol in the $(\mathcal{F}_{\mathsf{TE}},\ \mathcal{F}_{\mathsf{Rand}})$-hybrid model

Theorem 14 (Security of Π_{ABB}^q). *The protocol Π_{ABB}^q securely realizes the functionality $\mathcal{F}_{\mathsf{ABB}}^q$ in the $(\mathcal{F}_{\mathsf{Rand}},\ \mathcal{F}_{\mathsf{TE}})$-hybrid model with broadcast with static and computational security tolerating up to $t < N/2$ corruptions given that Π_{PoPK} and Π_{Mult} are proofs of plaintext knowledge and correct multiplication.*

Proof. We construct a simulator $\mathcal{S}$ in Figure 11. It setups simulated instances of $\mathcal{F}_{\mathsf{Rand}}$ and $\mathcal{F}_{\mathsf{TE}}$ as well as simulated parties, and gives control of the corrupted parties to the environment $\mathcal{Z}$. Since $\mathcal{S}$ controls the $\mathcal{F}_{\mathsf{TE}}$ functionality, it know the secret key of the encryption scheme, and can, thus, decrypt all valid ciphertexts sent by the corrupted parties. This way it can extract the

MPC Protocol Π^q_{ABB} (Part II)

Multiply For $(\mathsf{Multiply}, \mathsf{vid}_z, \mathsf{vid}_x, \mathsf{vid}_y)$: Let $(\mathsf{vid}_x, \mathsf{ct}_x)$ and $(\mathsf{vid}_y, \mathsf{ct}_y)$ be the stored ciphertexts. All P_i proceed in parallel as follows:

1. Sample mask d
   (a) Sample $d_i \in_R \mathbb{F}_q$, $r_i \leftarrow \mathcal{D}_q$, compute $\mathsf{ct}_{d_i} \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, d_i; r_i)$ and $(\mathsf{com}^1_i, \mathsf{stt}^1_i) \leftarrow \Pi_{\mathsf{PoPK}}.\mathsf{Commit}(\mathsf{ct}_{d_i}, (d_i, r_i))$. Broadcast ct_{d_i} and com^1_i.
   (b) Send $(\mathsf{Rand}, [C]^N)$ to $\mathcal{F}_{\mathsf{Rand}}$ so that all parties receive $(\mathsf{chl}^1_j)_{j\in[N]} \in [C]^N$.
   (c) Broadcast $\mathsf{res}^1_i \leftarrow \Pi_{\mathsf{PoPK}}.\mathsf{Respond}(\mathsf{ct}_{d_i}, (d_i, r_i), \mathsf{stt}^1_i, \mathsf{chl}^1_i)$.
   (d) If $\Pi_{\mathsf{PoPK}}.\mathsf{Verify}(\mathsf{ct}_{d_j}, \mathsf{com}^1_j, \mathsf{chl}^1_j, \mathsf{res}^1_j) = \perp$ for some $P_j \neq P_i$, set $d_j := 0$ and deterministically recompute $\mathsf{ct}_{d_j} \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, 0; 0)$.
   (e) Let $d = \sum_{i=1}^N d_i$.
2. Decrypt masked x
   (a) Locally compute $\mathsf{ct}_{x+d} \leftarrow \mathsf{ct}_x + \sum_{i=1}^N \mathsf{ct}_{d_i}$.
   (b) Send $(\mathsf{Decrypt}, \mathsf{ct}_{x+d})$ to $\mathcal{F}_{\mathsf{TE}}$ so that all parties receive $x + d$.
3. Compute additive shares $(x_1, \ldots, x_N)$ of x
   (a) Let $x_1 := (x + d) - d_1$ and $x_j := -d_j$ for $j \in [2, N]$ such that $x = \sum_{i\in[N]} x_i$ and P_i knows x_i.
   (b) Locally compute $\mathsf{ct}_{x_1} \leftarrow (x + d) - \mathsf{ct}_{d_1}$ and $\mathsf{ct}_{x_j} \leftarrow -\mathsf{ct}_{d_j}$ for $j \in [2, N]$.
4. Multiply the x_i with ct_y
   (a) Sample $s_i \leftarrow \mathcal{D}_q$, and compute $\mathsf{ct}_{x_i\cdot y} \leftarrow x_i \cdot^{s_i}_R \mathsf{ct}_y$ and $(\mathsf{com}^2_i, \mathsf{stt}^2_i) \leftarrow \Pi_{\mathsf{Mult}}.\mathsf{Commit}((\mathsf{ct}_y, \mathsf{ct}_{x_i}, \mathsf{ct}_{x_i\cdot y}), (x_i, -r_i, s_i))$. Broadcast $\mathsf{ct}_{x_i\cdot y}$ and com^2_i.
   (b) Send $(\mathsf{Rand}, [C]^N)$ to $\mathcal{F}_{\mathsf{Rand}}$ so that all parties receive $(\mathsf{chl}^2_j)_{j\in[N]} \in [C]^N$.
   (c) Broadcast $\mathsf{res}^2_i \leftarrow \Pi_{\mathsf{Mult}}.\mathsf{Respond}((\mathsf{ct}_y, \mathsf{ct}_{x_i}, \mathsf{ct}_{x_i\cdot y}), (x_i, -r_i, s_i), \mathsf{stt}^2_i, \mathsf{chl}^2_i)$.
   (d) If $\Pi_{\mathsf{Mult}}.\mathsf{Verify}((\mathsf{ct}_y, \mathsf{ct}_{x_j}, \mathsf{ct}_{x_j\cdot y}), \mathsf{com}^2_j, \mathsf{chl}^2_j, \mathsf{res}^2_j) = \perp$ for some $P_j \neq P_i$, send $(\mathsf{Decrypt}, \mathsf{ct}_{x_j})$ to $\mathcal{F}_{\mathsf{TE}}$ so that all parties receive x_j. Then deterministically compute $\mathsf{ct}_{x_j\cdot y} \leftarrow x_j \cdot \mathsf{ct}_y$.
   (e) Locally compute $\mathsf{ct}_z \leftarrow \sum_{i\in[N]} \mathsf{ct}_{x_i\cdot y}$ and store $(\mathsf{vid}_z, \mathsf{ct}_z)$.

Fig. 10. MPC protocol in the $(\mathcal{F}_{\mathsf{TE}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model

corrupted parties' inputs. For the honest parties' inputs, $\mathcal{S}$ encrypts lossy values. By IND-CPA security of the encryption scheme, these are indistinguishable from the ciphertexts in the real execution. When an output is to be produced, $\mathcal{S}$ simulates the decryption by $\mathcal{F}_{\mathsf{TE}}$ and inserts the value that it obtains from $\mathcal{F}^q_{\mathsf{ABB}}$. For all values occurring during ABB operations, $\mathcal{S}$ stores the corresponding ciphertexts.

Init: $\mathcal{S}$ simulates the calls to $\mathcal{F}_{\mathsf{TE}}$ to generate the CL public parameters $\mathsf{pp}_{\mathsf{cl}}$ and a key pair $(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}})$. For the latter, $\mathcal{F}_{\mathsf{TE}}$ allows some adversarial influence. Hence, $\mathcal{S}$ allows $\mathcal{Z}$ to specify the bias as defined for $\mathsf{BiasedSpecialKeyGen}^{\mathcal{Z}}_1$.

Input: If the value providing party P_i is honest, $\mathcal{S}$ does not know its input value. In this case, it simulates the protocol execution as if P_i had the input $x = 0$. If P_i is corrupted, $\mathcal{S}$ needs to extract its inputs to provide it to $\mathcal{F}^q_{\mathsf{ABB}}$. If P_i misbehaves by sending an invalid ciphertext ct or if its proof does not verify, $\mathcal{S}$ simulates the honest parties setting $x := 0$ as in the real protocol. Otherwise $\mathcal{S}$ uses $\mathsf{sk}_{\mathsf{cl}}$ to decrypt $x \leftarrow \mathsf{Dec}(\mathsf{sk}_{\mathsf{cl}}, \mathsf{ct})$, and then forwards x to $\mathcal{F}^q_{\mathsf{ABB}}$.

Output: $\mathcal{S}$ obtains the output x as leakage from $\mathcal{F}^q_{\mathsf{ABB}}$. It then simulates the call to $(\mathsf{Decrypt}, \mathsf{ct})$ of $\mathcal{F}_{\mathsf{TE}}$ and simulates the leakage $M := f^x$ to $\mathcal{Z}$.

Linear: Since linear operations are computes locally, $\mathcal{S}$ only needs to compute the ciphertext for the resulting value.

Multiply: Recall that $\mathcal{S}$ has ciphertexts stored for both inputs. Note that the decrypted values in the Multiply protocol are independent of any secrets: In Step 2 the factor x is decrypted only after masking it with a uniform value $d \in_R \mathbb{F}_q$, and in Step 4 only the shares x_j of corrupted $P_j \in \mathcal{C}$ may be revealed. Therefore, $\mathcal{S}$ can just let the simulation proceed according to $\mathit{\Pi}^q_{\mathsf{ABB}}$.

```
Simulator for $\mathit{\Pi}^q_{\mathsf{ABB}}$

Init  For ($\mathsf{Init}$), simulate the interaction with $\mathcal{F}_{\mathsf{TE}}$:
  1. Simulate the call to ($\mathsf{Init}$, $1^\lambda$, $q$) by computing $\mathsf{pp}_{\mathsf{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q)$.
  2. Simulate the call to ($\mathsf{KeyGen}$), by computing $\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}} \leftarrow \mathsf{BiasedSpecialKeyGen}^{\mathcal{Z}}_1(\mathsf{pp}_{\mathsf{cl}})$.
Store $\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}}$.
Input  For ($\mathsf{Input}$, $P_i$, $\mathsf{vid}$, $x$):
  1. if $P_i \in \mathcal{H}$
     (a) Simulate the protocol with input $x = 0$ and $\mathsf{ct} \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, x)$.
  2. if $P_i \in \mathcal{C}$
     (a) Receive ciphertext $\mathsf{ct}$ from $P_i$.
     (b) Simulate the verification of $P_i$'s proof.
     (c) If the proof verifies, set $x := \mathsf{Dec}(\mathsf{sk}_{\mathsf{cl}}, \mathsf{ct})$.
     (d) Otherwise, set $x := 0$ and recompute $\mathsf{ct} \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, 0; 0)$.
     (e) Send $x$ as input to $\mathcal{F}^q_{\mathsf{ABB}}$ on behalf of $P_i$.
  3. Store ($\mathsf{vid}, \mathsf{ct}$).
Output  For ($\mathsf{Output}$, $\mathsf{all}$, $\mathsf{vid}$):
  1. Let ($\mathsf{vid}, \mathsf{ct}$) be the stored ciphertext.
  2. Receive $x$ from $\mathcal{F}^q_{\mathsf{ABB}}$.
  3. Simulate the call to ($\mathsf{Decrypt}$, $\mathsf{ct}$) of $\mathcal{F}_{\mathsf{TE}}$, and simulate the leakage of the partial decryption
     $M := f^x$.
Linear  For ($\mathsf{Linear}$, $\mathsf{vid}_o$, ($\mathsf{vid}_1, \cdots, \mathsf{vid}_n$), ($a_0, \ldots, a_n$)):
  1. Compute $\mathsf{ct}_o \leftarrow a_0 + \sum_{i \in [n]} a_i \cdot \mathsf{ct}_i$ as in $\mathit{\Pi}^q_{\mathsf{ABB}}$ and store ($\mathsf{vid}_o, \mathsf{ct}_o$).
Multiply  For ($\mathsf{Multiply}$, $\mathsf{vid}_z$, $\mathsf{vid}_x$, $\mathsf{vid}_y$):
  1. Use the stored values ($\mathsf{vid}_x, \mathsf{ct}_x$) and ($\mathsf{vid}_y, \mathsf{ct}_y$) to simulate the multiplication as in $\mathit{\Pi}^q_{\mathsf{ABB}}$.
  2. Store the resulting ($\mathsf{vid}_z, \mathsf{ct}_z$).
```

Fig. 11. Simulator for the MPC protocol in $\mathit{\Pi}^q_{\mathsf{ABB}}$

Claim. The simulation is computationally indistinguishable to the real execution.

Proof of Claim. To prove indistinguishability, we follow the approach from SPDZ [Dam+12], and exploit that our encryption scheme has lossy public keys which are indistinguishable to normal public keys. Hence, we reduce indistinguishability to Lemma 3.

Assume towards contradiction there is an environment $\mathcal{Z}$ that can distinguish the simulation of $\mathcal{S}$ from a real protocol execution with non-negligible advantage $\mathsf{Adv}_{\mathcal{Z}}(\lambda)$. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{Z}$ as a subroutine to distinguish working public keys from lossy public keys.

Initially, the challenger in this indistinguishability game samples a bit $b \in_R \{0, 1\}$, which determines whether a lossy ($b = 0$) or a working ($b = 1$) public key will be produced.

Then $\mathcal{B}$ also tosses a coin $b^*_{\mathcal{B}} \leftarrow_R \{\mathsf{real}, \mathsf{simulation}\}$. Depending on the outcome $\mathcal{B}$ either produces the environment a view of a real protocol execution or a simulation. *This is not a UC simulation.* So in both cases, $\mathcal{B}$ controls all UC entities (ideal functionalities, honest parties), and uses $\mathcal{Z}$ as the environment. Hence, $\mathcal{B}$ learns in particular what inputs $\mathcal{Z}$ selects for the honest parties, and it is also allowed to rewind $\mathcal{Z}$.

In both cases, $\mathcal{B}$ makes some adjustments:

1. $\mathcal{B}$ simulates the $\mathcal{F}_{\mathsf{TE}}$ functionality, but it changes the behavior of the calls to Init and KeyGen: Instead of generating the key pair itself during the simulation or execution of the Init phase of $\mathit{\Pi}^q_{\mathsf{ABB}}$, $\mathcal{B}$ forwards the messages between the challenger and $\mathcal{Z}$ such that they compute a key pair $\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}} \leftarrow \mathsf{BiasedSpecialKeyGen}^{\mathcal{Z}}_b(\mathsf{pp}_{\mathsf{cl}})$. Hence, $\mathcal{Z}$ can specify the bias and the resulting $\widetilde{\mathsf{pk}}_{\mathsf{cl}}$ is either a lossy ($b = 0$) or a working ($b = 1$) public key. This change does not change $\mathcal{Z}$'s view.
2. Moreover, since $\mathcal{B}$ does not know $\mathsf{sk}_{\mathsf{cl}}$, it cannot decrypt the ciphertexts of the corrupted parties to extract their inputs, but needs to use rewinding and the extractor of the $\mathit{\Pi}_{\mathsf{PoPK}}$ and $\mathit{\Pi}_{\mathsf{Mult}}$ protocols instead: When a corrupted party $P_i \in \mathcal{C}$ has an input and proves knowledge of the plaintext $x \in \mathbb{F}_q$ corresponding to the broadcasted ciphertext ct, $\mathcal{B}$ proceeds as follows: Let $(\mathsf{com}, \mathsf{chl}, \mathsf{res})$ be the transcript of P_i's proof.
   - Case I – $\mathit{\Pi}_{\mathsf{PoPK}}.\mathsf{Verify}(\mathsf{ct}, \mathsf{com}, \mathsf{chl}, \mathsf{res}) = \bot$: $\mathcal{B}$ lets the protocol continue as in $\mathit{\Pi}^q_{\mathsf{ABB}}$, i.e., the honest parties set P_i's input to 0 and continue.
   - Case II – $\mathit{\Pi}_{\mathsf{PoPK}}.\mathsf{Verify}(\mathsf{ct}, \mathsf{com}, \mathsf{chl}, \mathsf{res}) = \top$: $\mathcal{B}$ tries to extract the message. It starts running two loops in parallel until one of them terminates:
     (a) $\mathcal{B}$ rewinds $\mathcal{Z}$ and the whole system. It programs $\mathcal{F}_{\mathsf{Rand}}$ with a fresh random challenge $\mathsf{chl}' \in_R [C] \setminus \{\mathsf{chl}\}$ to obtain another transcript $(\mathsf{com}, \mathsf{chl}', \mathsf{res}')$. If the proof verifies, $\mathit{\Pi}_{\mathsf{PoPK}}.\mathsf{Verify}(\mathsf{ct}, \mathsf{com}, \mathsf{chl}', \mathsf{res}') = \top$, it terminates the loops.
     (b) $\mathcal{B}$ does the same as in the first loop, but instead of sampling random challenges, it iterates over $\mathsf{chl}' = 1, 2, \ldots, C$ until it finds $\mathsf{chl}' \neq \mathsf{chl}$ that leads to a valid proof. If no such chl' is found, $\mathcal{B}$ aborts.

     If $\mathcal{B}$ has not aborted, then it has now two accepting transcripts and can compute $x \leftarrow \mathit{\Pi}_{\mathsf{PoPK}}.\mathsf{Extract}(\mathsf{ct}, (\mathsf{com}, \mathsf{chl}, \mathsf{res}), (\mathsf{com}, \mathsf{chl}', \mathsf{res}'))$.

   If extraction was successful, $\mathcal{B}$ continues the protocol execution/simulation with the original challenge chl.

   Let p be the probability of Case II. If $p > 1/C$, then the first loop finds a second transcript after expected $1/p$ iterations. If $p = 1/C$, then P_i only responds to the challenge chl, and the second loop terminates after C iterations. Hence, over both cases the expected number of loop iteration is constant, and the overall running time of $\mathcal{B}$ is expected polynomial time. Moreover, since $1/C = \mathsf{negl}(\lambda)$, $\mathcal{B}$ aborts only with negligible probability.

   In the $\mathsf{Multiply}$ case, all parties run the proofs in parallel, and each P_i receives a different challenge. Therefore, during the rewinding process, $\mathcal{B}$ only needs to resample the challenges for the corrupted parties, and can keep the same challenges for the honest parties.

   By keeping track of all values during the computation, $\mathcal{B}$ is able to simulate the calls to $\mathsf{Decrypt}$ of $\mathcal{F}_{\mathsf{TE}}$ without having to know the secret key.
3. Finally, instead of computing the proofs of the honest parties honestly as defined by $\mathit{\Pi}^q_{\mathsf{ABB}}$ and $\mathcal{S}$, $\mathcal{B}$ uses the SHVZK simulators of $\mathit{\Pi}_{\mathsf{PoPK}}$ and $\mathit{\Pi}_{\mathsf{Mult}}$ and programming of $\mathcal{F}_{\mathsf{Rand}}$ to generate convincing proofs. By statistical SHVZK of the protocols, this change is not detectable by $\mathcal{Z}$.

If $b^*_{\mathcal{B}} = \mathsf{real}$, then $\mathcal{B}$ simulates a real protocol execution of $\mathit{\Pi}^q_{\mathsf{ABB}}$ with the adjustments defined above, where it lets the honest parties run with the inputs obtained from $\mathcal{Z}$, and the corrupted parties act as instructed by $\mathcal{Z}$. If $b^*_{\mathcal{B}} = \mathsf{simulation}$, then $\mathcal{B}$ creates a simulation setup, where $\mathcal{S}$ talks to a $\mathcal{F}^q_{\mathsf{ABB}}$, and runs the simulation. Finally, $\mathcal{Z}$ outputs a bit $b^*_{\mathcal{Z}} \in \{\mathsf{real}, \mathsf{simulation}\}$ depending on whether it thinks it experienced the real or the simulated execution. If $b^*_{\mathcal{Z}} = b^*_{\mathcal{B}}$, i.e., $\mathcal{Z}$ guessed right, $\mathcal{B}$ outputs $b' := 1$, otherwise it outputs $b' := 0$.

In case $b = 1$, the $\mathcal{Z}$'s view statistically indistinguishable to either the real execution of $\mathit{\Pi}^q_{\mathsf{ABB}}$ or the simulation with $\mathcal{S}$, and $\mathcal{B}$ wins iff $\mathcal{Z}$ guesses correctly. Hence, we have $\Pr[\mathcal{B} \text{ wins} \mid b = 1] = 1/2 + \mathsf{Adv}_{\mathcal{Z}}(\lambda) - \mathsf{negl}(\sigma)$.

In the other case $b = 0$, encryptions made with the (lossy key) $\widetilde{\mathsf{pk}_{\mathsf{cl}}}$ contain statistically no information about the encrypted value – they are always just encryptions of 0.

Moreover, since $\mathcal{B}$ simulates the honest parties' proofs, $\mathcal{Z}$'s view in the real execution is statistically independent of the honest parties inputs. In fact the real execution statistically coincides with the $\mathsf{simulation}$. Hence, $\mathcal{Z}$'s advantage is negligible in σ. Since $\mathcal{B}$ wins in this case if $\mathcal{Z}$ guesses wrongly, we have $\Pr[\mathcal{B} \text{ wins} \mid b = 0] = 1/2 - \mathsf{negl}(\sigma)$.

Overall, we combine the two cases to obtain

$$\Pr[\mathcal{B} \text{ wins}] = 1/2 + \mathsf{Adv}_{\mathcal{Z}}(\lambda)/2 - \mathsf{negl}(\sigma).$$

Therefore, $\mathcal{B}$ has advantage $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{hsm\text{-}cl},*,\mathsf{pk}}(\lambda) \geq \mathsf{Adv}_{\mathcal{Z}}(\lambda)/2 - \mathsf{negl}(\sigma)$ in the key distinguishing game. If σ is chosen such that $\mathsf{negl}(\sigma)$ is also negligible in λ, then $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{hsm\text{-}cl},*,\mathsf{pk}}(\lambda)$ is non-negligible in λ, which contradicts the premise. Hence, such an algorithm $\mathcal{B}$ cannot exist, and we can conclude that the simulation is indistinguishable from the real execution of $\mathit{\Pi}_{\mathsf{ABB}}^{q}$. ■

□

8 YOSO MPC

The YOSO framework (as defined in [Gen+21]) is defined for a universe of M players (servers), and we use their model for computationally secure YOSO protocols. This concretely means that the parties in the protocol we specify are actually roles that in an actual execution will be assigned to physical servers by a *Role Assignment Mechanism.* It further means that a public encryption key is assigned to each party (role). In an actual execution, the physical server assigned to play a role will learn the corresponding secret key, but the adversary will not learn the identity of the server until it speaks. This mechanism is abstracted away in the model. We can therefore think of a large set of parties, they each own a key pair and can only speak once. Furthermore, if we divide the parties into disjoint committees with N members each, we can assume that each committee has honest majority if we choose N large enough.

A construction of a role assignment mechanism was presented in [Ben+20]. Their implementation allows us to choose the encryption scheme freely, so in this paper we will use the CL encryption scheme. However we will use separate class-group set-up, where the plaintext space is numbers mod q', where q' is chosen large enough compared to q. We will specify below what this means more precisely.

In the YOSO framework, we assume that public information can be reliably posted for everyone to see: the motivation for this is that the framework is intended for doing MPC on a blockchain. When we say in the following that something is "publicly known" or "on public display", this means it has been posted or can be efficiently computed from what is posted. Further, a committee is only allowed to speak once, that is, committee members decrypt whatever ciphertext is intended for them, read the public information, post a single message and do nothing further. The idea is that this allows us to assume that even an adaptive adversary who can corrupt a large fraction of the physical servers, will not be able to corrupt a majority of any committee. See [Gen+21] for details.

Committees will be denoted by COM_i, for $i = 0, 1, \ldots,$ and their public keys will be denoted by $\mathsf{pk}_1^i, \ldots, \mathsf{pk}_N^i$. In the protocol, committees will do their work in numeric order, starting with COM_0. We assume that the role assignment mechanism is called such that when COM_i is about to start, the public keys of (at least) $\mathsf{COM}_{i+1}, \mathsf{COM}_{i+2}, \mathsf{COM}_{i+3}, \mathsf{COM}_{i+4}$ are publicly known.

In the following, we show how to adapt our honest majority MPC protocol to the YOSO setting. For this, we will need non-interactive zero-knowledge arguments of knowledge (NIAoK) for various statements. Any instantiation will do, as long as it is simulation extractable[12], meaning that you can extract a witness from a proof that verifies, even if the prover has access to simulated proofs. We also require that only transparent set-up is needed, as our overall goal is MPC with transparent set-up. In this paper, we describe Σ-protocols for all the statements needed, so one approach is to make these non-interactive using the random oracle model and the Fischlin-transform [Fis05] which will have the required properties. By careful protocol redesign and analysis, we believe it would be possible to use the more efficient Fiat-Shamir transform, but leave this for future work. Another option is to use a recent construction of SNARKS with transparent set-up based on class groups [Aru+23]. We will denote a non-interactive argument on public instance x and witness w by $\mathsf{NIAoK}(x, w; eq(x, w))$, where $eq(x, w)$ is the equation that x and w are proven to satisfy.

8.1 Threshold Encryption

Key Generation We assume the specification of the class group and the generators is given (recall that this set-up is transparent). We first need a protocol that allows a committee to obtain shares of a random value η, while leaking only g_q^η to the adversary.

We will need this for key generation, but also for other purposes, and in fact it is a simple adaptation of Steps 1 and 2 of the distributed key generation protocol Π_{TE} (Figure 4). It involves two committees COM_u and COM_v, where COM_u will generate the randomness required, and COM_v will receive the shares.

The idea is that members of the first committee COM_u will do a Feldman secret-sharing of their contribution to η, see Protocol 2. The shares will be intended for COM_v, so members of COM_u will encrypt the shares under $\mathsf{pk}_1^v, \dots, \mathsf{pk}_N^v$, do a non-interactive ZK proof that these are correct, and post the ciphertexts and proofs. See Section 8.3 for more details. The plaintext modulus q' for the pk_v^i's is chosen such that q' is larger than any honestly generated share. Then members of COM_v will add the valid contributions together to get g_q^η and shares of η. The protocol is shown in Figure 12. Towards understanding the protocol, note that P_i in the first step does an integer secret sharing of η_i as a part of the Feldman procedure, resulting in shares $y_{i,j}$. Correctness of a share can be verified by the equation $g_q^{\Delta y_{i,j}} = C_{i,0}^{\Delta^2} \cdot \prod_{k=1}^t C_{i,k}^{j^k}$, as the right hand side effectively evaluates the underlying polynomial in the exponent. After the protocol, every party who looks at the public information will be able to compute g_q^η and the Γ_j's, using the equations in the last step of the protocol.

Definition 7. *We say that* a committee COM_ℓ holds $\mathsf{VSS}(\eta; \mathsf{f})$ *if it is the case that* f *is an integer polynomial of degree at most t, where $\mathsf{f}(0) = \Delta\eta$, each committee member P_i has a share $\mathsf{f}(i)$, and the value $g_q^{\Delta \mathsf{f}(i)}$ is on public display. Furthermore, also g_q^η is publicly known.*

With this notation, the CreateVSS protocol can be described as a protocol that is executed by two committees COM_u and COM_v, and which results in COM_v holding $\mathsf{VSS}(\eta; \mathsf{f})$ for some random value η, where only g_q^η is leaked to the adversary. Note that the underlying structure is linearly homomorphic: if a committee holds $\mathsf{VSS}(\eta; \mathsf{f})$ and $\mathsf{VSS}(\eta'; \mathsf{f}')$, it effectively also holds $\mathsf{VSS}(\eta + \eta'; \mathsf{f} + \mathsf{f}')$, by adding the underlying shares and multiplying the public values. We use CreateVSS for distributed key generation (Figure 13). It needs to be executed by committees 0–3, as this step must happen before anything else.

[12] Actually a weak form of extraction (see Section 5) will suffice, where we only extract a part of the witness. This is because the MPC protocol we describe here is a simple adaptation of Π_{ABB}^q for the non-YOSO model.

Protocol CreateVSS

Create and send shares Each member P_i of COM_u samples the contribution to the secret $\eta_i \leftarrow \mathcal{D}_q$, and samples $s_{i,j} \leftarrow \mathcal{D}_q{}'$. Then it shares η_i as $(y_{i,1}, \ldots, y_{i,N}; C_{i,0}, \ldots, C_{i,t}) \leftarrow \mathsf{F\text{-}Share}(\eta_i; \mathbf{r}_i; g_q)$, computes $\mathsf{ct}_{i,j} = \mathsf{Enc}(\mathsf{pk}_j^v, y_{i,j}; s_{i,j})$ for $j \in [N]$. Finally, P_i posts:

- $(C_{i,0}, \ldots, C_{i,t})$
- $\mathsf{NIAoK}(C_{i,0}, \eta_i; C_{i,0} = g_q^{\eta_i})$
- $\mathsf{NIAoK}(C_{i,k}, r_{i,k}; C_{i,k} = g_q^{\Delta r_{i,k}})$ for $k \in [t]$, where $\Delta = N!$
- $\mathsf{NIAoK}((\mathsf{pk}_j^v, \{C_{i,k}\}, \mathsf{ct}_{i,j}), (y_{i,j}, s_{i,j}); \mathsf{ct}_{i,j} = \mathsf{Enc}(\mathsf{pk}_j^v, y_{i,j}; s_{i,j}),\ g_q^{\Delta y_{i,j}} = C_{i,0}^{\Delta^2} \cdot \prod_{k=1}^t C_{i,k}^{j^k})$, for $j \in [M]$

Receive Shares Each member P_j of COM_v does the following:

1. Check all the proofs posted by COM_u and form a set $\mathcal{Q}$ of members (of COM_u) for which the proofs validate (NB: all honest P_j will agree on $\mathcal{Q}$).
2. Decrypt $\mathsf{ct}_{i,j}$ to get $y_{i,j}$.
3. Compute a share of η as $\gamma_j = \sum_{i \in \mathcal{Q}} y_{i,j}$, where $\eta := \sum_{i \in \mathcal{Q}} \eta_i$.
4. Compute $g_q^\eta = \prod_{i \in \mathcal{Q}} C_{i,0}$.
5. Compute a public verification value corresponding to γ_j, namely

$$\Gamma_j = g_q^{\Delta \gamma_j} = \prod_{i \in \mathcal{Q}} g_q^{\Delta y_{i,j}} = \prod_{i \in \mathcal{Q}} C_{i,0}^{\Delta^2} \cdot \prod_{k=1}^{t} C_{i,k}^{j^k}.$$

Fig. 12. Create verifiable secret-sharing for a committee

Protocol DistKeyGen

Create and share the secret key Committees COM_0 and COM_1 run CreateVSS so that COM_1 holds $\mathsf{VSS}(\eta; \mathsf{f})$, and we set $\mathsf{pk}_{\mathsf{cl}} := g_q^\eta$ and $\mathsf{sk}_{\mathsf{cl}} := \eta$.

Create a special ciphertext The goal is to generate a ciphertext $\mathsf{ct}_P = (\mathsf{ct}_1, \mathsf{ct}_2) = (g_q^\beta, f \cdot \mathsf{pk}_{\mathsf{cl}}^\beta)$, where β is a large random value. Our construction in Step 3 of Π_{TE} does this in the standard non-YOSO setting. We adapt it for YOSO in exactly the same way as we adapted Steps 1 and 2 to get CreateVSS. The resulting protocol is executed by next two committees $\mathsf{COM}_2, \mathsf{COM}_3$. They can clearly do this based only on the public key $\mathsf{pk}_{\mathsf{cl}}$. As a result, ct_P is now on public display, so now $\widetilde{\mathsf{pk}}_{\mathsf{cl}} = (\mathsf{pk}_{\mathsf{cl}}, \mathsf{ct}_P)$ is publicly known.

Fig. 13. Distributed key generation in the YOSO setting

Resharing the Secret Key We need a protocol for passing the shares of the secret key from one committee COM_v to another committee $\mathsf{COM}_{v'}$. The idea is for an earlier committee COM_u to share the same random value for both COM_v and $\mathsf{COM}_{v'}$, and this will help to do the transfer. The Reshare protocol (Figure 14) will be called several times, so the indices u, v, v' should be thought of as parameters, that will be determined by the global MPC protocol. Execution of Reshare adds a factor of Δ to the secret key held by the receiving committee, but this is not a problem, as we shall see.

Threshold Decryption Note that our threshold decryption protocol for the standard setting (Figure 5) is already a one round protocol, once we make the proofs of equality of discrete logs non-interactive. We need the protocol to work for a committee who holds $\mathsf{VSS}(\Delta^\ell \mathsf{sk}_{\mathsf{cl}}, \mathsf{f})$ for some

Protocol Reshare

Create mask for the secret key We execute CreateVSS twice, one instance is done by $\mathsf{COM}_u, \mathsf{COM}_v$ and one by $\mathsf{COM}_u, \mathsf{COM}_{v'}$, with two adjustments:
First, the two instances are correlated such that the shared secret value η is the same in both cases: as part of CreateVSS, each member P_i must make public $g_q^{\eta_i}$, where η_i is their additive contribution to the secret η. The receiving committees check that this value is the same in both instances and discard the contribution if not. Second, each η_i is chosen as a random number of bitlength σ larger than the maximal length of $\mathsf{sk}_{\mathsf{cl}}$. Also, the coefficient in the polynomials used are chosen to be σ bits longer than those used in the DistKeyGen protocol.
Open masked key We can now assume that COM_v holds $\mathsf{VSS}(\mathsf{sk}_{\mathsf{cl}}, \mathsf{f})$ as well as $\mathsf{VSS}(\eta, \mathsf{g})$. It therefore also holds $\mathsf{VSS}(\mathsf{sk}_{\mathsf{cl}} - \eta, \mathsf{f} - \mathsf{g})$. Each member P_j of COM_v posts its share $(\mathsf{f} - \mathsf{g})(j)$. Note that this share can be publicly verified, as $g_q^{\Delta \cdot (\mathsf{f}-\mathsf{g})(j)}$ is publicly known.
Adjust shares Members of $\mathsf{COM}_{v'}$ can now identify (at least) $t+1$ posted correct shares and reconstruct (by "integer interpolation") $\Delta(\mathsf{sk}_{\mathsf{cl}} - \eta)$. They then form $\mathsf{VSS}(\Delta(\mathsf{sk}_{\mathsf{cl}} - \eta), \mathsf{h}_0)$ where h_0 is a default polynomial, say the degree-0 polynomial that always takes the value $\Delta(\mathsf{sk}_{\mathsf{cl}} - \eta)$. This can then be added to $\Delta \cdot \mathsf{VSS}(\eta, \mathsf{h})$ received by the committee in the first step to form $\mathsf{VSS}(\Delta \mathsf{sk}_{\mathsf{cl}}, \mathsf{h} + \mathsf{h}_0)$.

Fig. 14. Transfer secret-sharing to a new committee

ℓ, because each execution of Reshare adds a factor of Δ. This is not a problem, we will obtain the plaintext times a factor of $\Delta^\ell \bmod q$, and this factor can be divided out locally.

8.2 MPC

To realize the arithmetic black-box functionality in the YOSO setting, in particular secure multiplication, we follow the standard approach of generating so-called multiplication triples, i.e., triples $(\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3)$ of ciphertexts containing plaintext values a, b, c where a, b are random and $c = a \cdot b$. We first construct a protocol for generating such triples while leaking nothing but the ciphertexts.

The basic steps of the final MPC protocol are found in Figures 15 and 16, and are a straightforward adaptation of Π^q_{ABB} which we described for the standard non-YOSO model. We assume here a global scheduling of which committees do the various types of work, where the first 4 committees do the key generation. After this, committees only need to work to do multiplications and output. Assume we are about to do a number of multiplications on encrypted values and that COM_i is the committee with the largest index who holds shares of the secret key but has not spoken yet. Now, $\mathsf{COM}_{i+1}, \mathsf{COM}_{i+2}$ will generate an appropriate number of multiplication triples, and COM_{i+3} will do the first part of the Reshare protocol, in the role of the auxiliary committee COM_u, and using COM_i and COM_{i+4} as the sending and receiving committees. Now, COM_i can do its part of the multiplication and post the resulting data, as well as data from the Reshare protocol, allowing COM_{i+4} to get shares of the secret key. The system is now ready to do the next layer of multiplications, with COM_{i+1} assuming the role COM_i played before.

Proof Sketch for Security The CreateVSS protocol is the same as the first part of the key generation of the non-YOSO protocol, except that shares are sent in encrypted form instead of being sent on point to point channels. We can therefore do the same simulation except that shares meant for corrupt parties are encrypted under their keys, and for honest parties we encrypt

Protocol $\mathsf{CreateTriple}$

Create encryption of a Each member P_i of COM_u samples $a_i \in_R \mathbb{F}_q$, $r_i \leftarrow \mathcal{D}_q$, and computes $\mathsf{ct}_{a_i} \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, a_i; r_i)$ It then posts

$$\mathsf{ct}_{a_i},\ \mathsf{NIAoK}((\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{ct}_{a_i}), (a_i, r_i)); \mathsf{ct}_{a_i} = \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, a_i; r_i))$$

using the non-interactive version of Π_{PoPK}. All servers compute $\mathsf{ct}_a = \sum_i \mathsf{ct}_{a_i}$.
Create encryption of b and multiply by a Next, COM_v creates an encryption of a random b and an encryption of $c = ab$: Each member P_j samples $b_j \in_R \mathbb{F}_q$, $r_j \leftarrow \mathcal{D}_q$, and computes $\mathsf{ct}_{b_j} \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, b_j; r_j)$.
Next, P_j samples $s_j \leftarrow \mathcal{D}_q$, computes $\mathsf{ct}_{b_j \cdot a} \leftarrow b_j \cdot_R^{s_j} \mathsf{ct}_a$, and posts $\mathsf{ct}_{b_j}, \mathsf{ct}_{b_j \cdot a}$, and $\mathsf{NIAoK}((\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{ct}_a, \mathsf{ct}_{b_j}, \mathsf{ct}_{b_j \cdot a}), (r_j, s_j); \mathsf{ct}_{b_j} = \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, b_j; r_j), \mathsf{ct}_{b_j \cdot a} = b_j \cdot_R^{s_j} \mathsf{ct}_a)$. All parties can now compute $\mathsf{ct}_b = \sum_j \mathsf{ct}_{b_j}$ and $\mathsf{ct}_c = \sum_j \mathsf{ct}_{b_j \cdot a}$.

Fig. 15. Create multiplication triple

Protocol $\mathsf{YOSO} - \mathsf{ABB}$

Input A server supplying input value $x \in \mathbb{F}_q$ samples $r \leftarrow \mathcal{D}_q$, computes $\mathsf{ct}_x \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, x; r)$. Using the non-interactive version of Π_{PoPK}, it then posts ct_x, $\mathsf{NIAoK}((\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{ct}_x), (x, r)); \mathsf{ct}_x = \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, x; r))$.
Linear Operations To compute $x + \alpha y$, where α is public and encryptions $\mathsf{ct}_x, \mathsf{ct}_y$ of x, y are public, all servers compute $\mathsf{ct}_{x+\alpha y} = \mathsf{ct}_x + \alpha \cdot \mathsf{ct}_y$.
Multiplication To compute xy from x, y, where encryptions $\mathsf{ct}_x, \mathsf{ct}_y$ of x, y are public, the next committee that holds $\mathsf{VSS}(\mathsf{sk}_{\mathsf{cl}}, \mathsf{f})$ uses the next available triple $\mathsf{ct}_a, \mathsf{ct}_b, \mathsf{ct}_c$. It executes the decryption protocol on inputs $\mathsf{ct}_x - \mathsf{ct}_a$ and $\mathsf{ct}_y - \mathsf{ct}_b$, allowing all servers to compute plaintext values ϵ, δ. From this all servers compute $\mathsf{ct}_{xy} = \delta \mathsf{ct}_a + \epsilon \mathsf{ct}_b + \mathsf{ct}_c + \epsilon\delta$.
Output To output x, where ct_x is public, the next committee that holds $\mathsf{VSS}(\mathsf{sk}_{\mathsf{cl}}, \mathsf{f})$ executes the decryption protocol on input ct_x.

Fig. 16. The YOSO MPC protocol

dummy values and simulate the zero-knowledge proofs of correctness. This works by IND-CPA security of the encryption scheme. The same idea works for simulation of the rest of the key generation.

Simulation for the Reshare uses the same ideas, plus the observation that the data made public (values of the polynomial $(\mathsf{f} - \mathsf{g})(j)$) can be simulated because $\mathsf{f} - \mathsf{g}$ is statistically indistinguishable from a random polynomial by construction. Thus, these protocols leak nothing but the (biased) public key, exactly as the non-YOSO version. The decryption protocol can use the same simulation as the non-YOSO version, and this will prove that it only leaks the plaintext of what is decrypted. Therefore, simulation of the CreateTriple and $\mathsf{YOSO} - \mathsf{ABB}$ protocols can be done following the same ideas as the non-YOSO version.

8.3 Zero-Knowledge Proof for Correctness of Encryptions

We sketch here how to construct a Σ-protocol as needed in the CreateVSS protocol from Section 8.1. It works with a public key pk_j^v, public values $C_{i,0}, \dots, C_{i,k}$ created by the prover P_i in a Feldman

VSS, and ciphertexts $\mathsf{ct}_{i,j}$. The goal is to prove that the ciphertext was correctly formed from $y_{i,j}, r_{i,j}$ such that $\mathsf{ct}_{i,j} = \mathsf{Enc}(\mathsf{pk}_j^v, y_{i,j}, s_{i,j})$), and where $y_{i,j}$ is determined by the public values $C_{i,0}, \ldots, C_{i,k}$ that P_i posts. More concretely, it follows from the definition of F-Share that

$$g_q^{\Delta y_{i,j}} = C_{i,0}^{\Delta^2} \cdot \prod_{k=1}^{t} C_{i,k}^{j^k},$$

which can be publicly computed. Furthermore, $\mathsf{ct}_{i,j}$ is supposed to be of form

$$\mathsf{ct}_{i,j} = (g_q^{s_{i,j}}, f^{y_{i,j}} \cdot (\mathsf{pk}_j^v)^{s_{i,j}})$$

Recall that the public keys pk_j^v are from a different class group set-up where the order of f is q', chosen large enough that $q' > \Delta y_{i,j}$ for honestly generated $y_{i,j}$.

We now observe that $\Delta y_{i,j}$ appears in the exponent in both $g_q^{\Delta y_{i,j}}$ and $\mathsf{ct}_{i,j}^{\Delta} = (g_q^{\Delta r_{i,j}}, f^{\Delta y_{i,j}} \cdot (\mathsf{pk}_j^v)^{\Delta r_{i,j}})$. Therefore correctness of $\mathsf{ct}_{i,j}$ can be proved using a standard Schnorr-style Sigma-protocol. The only caveat is that if $y_{i,j}$ is too large, we may get overflow modulo q' and decryption would return an incorrect value. We therefore also need a range proof to show that $q' > \Delta y_{i,j}$. There are a number of ways to do this, a straightforward approach is to set up an integer (Pedersen) commitment scheme based on the class group we already have and using Boudot's [Bou00] well known 4-squares approach to doing the range proof.

Acknowledgements This research was supported by the Concordium Blockhain Research Center, Aarhus University, Denmark, the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM), and the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC). We thank Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker for clarifications regarding the CL framework.

References

[Abr+22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. *An Algebraic Framework for Silent Preprocessing with Trustless Setup and Active Security*. In: *Advances in Cryptology – CRYPTO 2022, Part IV*. Aug. 2022. DOI: 10.1007/978-3-031-15985-5_15.

[AF04] Masayuki Abe and Serge Fehr. *Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography*. In: *Advances in Cryptology – CRYPTO 2004*. Aug. 2004. DOI: 10.1007/978-3-540-28628-8_20.

[Aru+23] Arasu Arun, Chaya Ganesh, Satya V. Lokam, Tushar Mopuri, and Sriram Sridhar. *Dew: A Transparent Constant-Sized Polynomial Commitment Scheme*. In: *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II*. May 2023. DOI: 10.1007/978-3-031-31371-4_19.

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. *A Survey of Two Verifiable Delay Functions*. Cryptology ePrint Archive, Report 2018/712. https://eprint.iacr.org/2018/712. 2018.

[BCS19] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. *Using TopGear in Overdrive: A More Efficient ZKPoK for SPDZ*. In: *SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography*. Aug. 2019. DOI: 10.1007/978-3-030-38471-5_12.

[BDO23] Lennart Braun, Ivan Damgård, and Claudio Orlandi. *Secure Multiparty Computation from Threshold Encryption Based on Class Groups*. In: *Advances in Cryptology – CRYPTO 2023, Part I*. Aug. 2023. DOI: 10.1007/978-3-031-38557-5_20.

[Bea92] Donald Beaver. *Efficient Multiparty Protocols Using Circuit Randomization*. In: *Advances in Cryptology – CRYPTO'91*. Aug. 1992. DOI: 10.1007/3-540-46766-1_34.

[Ben+11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. *Semi-homomorphic Encryption and Multiparty Computation*. In: *Advances in Cryptology – EUROCRYPT 2011*. May 2011. DOI: 10.1007/978-3-642-20465-4_11.

[Ben+20] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. *Can a Public Blockchain Keep a Secret?* In: *TCC 2020: 18th Theory of Cryptography Conference, Part I*. Nov. 2020. DOI: 10.1007/978-3-030-64375-1_10.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. *Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)*. In: *20th Annual ACM Symposium on Theory of Computing*. May 1988. DOI: 10.1145/62212.62213.

[Bou00] Fabrice Boudot. *Efficient Proofs that a Committed Number Lies in an Interval*. In: *Advances in Cryptology – EUROCRYPT 2000*. May 2000. DOI: 10.1007/3-540-45539-6_31.

[Cas+19] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. *Two-Party ECDSA from Hash Proof Systems and Efficient Instantiations*. In: *Advances in Cryptology – CRYPTO 2019, Part III*. Aug. 2019. DOI: 10.1007/978-3-030-26954-8_7.

[Cas+20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. *Bandwidth-Efficient Threshold EC-DSA*. In: *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*. May 2020. DOI: 10.1007/978-3-030-45388-6_10.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. *Multiparty Unconditionally Secure Protocols (Extended Abstract)*. In: *20th Annual ACM Symposium on Theory of Computing*. May 1988. DOI: 10.1145/62212.62214.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Multiparty Computation from Threshold Homomorphic Encryption*. In: *Advances in Cryptology – EUROCRYPT 2001*. May 2001. DOI: 10.1007/3-540-44987-6_18.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. *Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols*. In: *Advances in Cryptology – CRYPTO'94*. Aug. 1994. DOI: 10.1007/3-540-48658-5_19.

[CF02] Ronald Cramer and Serge Fehr. *Optimal Black-Box Secret Sharing over Arbitrary Abelian Groups*. In: *Advances in Cryptology – CRYPTO 2002*. Aug. 2002. DOI: 10.1007/3-540-45708-9_18.

[CL15] Guilhem Castagnos and Fabien Laguillaumie. *Linearly Homomorphic Encryption from* DDH. In: *Topics in Cryptology – CT-RSA 2015*. Apr. 2015. DOI: 10.1007/978-3-319-16715-2_26.

[CL84] Henri Cohen and Hendrik W. Lenstra. *Heuristics on class groups of number fields*. In: *Number Theory Noordwijkerhout 1983*. 1984.

[CLT18] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. *Practical Fully Secure Unrestricted Inner Product Functional Encryption Modulo p*. In: *Advances in Cryptology – ASIACRYPT 2018, Part II*. Dec. 2018. DOI: 10.1007/978-3-030-03329-3_25.

[CLT22] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. *Threshold Linearly Homomorphic Encryption on* $\mathbf{Z}/2^k\mathbf{Z}$. In: *Advances in Cryptology – ASIACRYPT 2022, Part II*. Dec. 2022. DOI: 10.1007/978-3-031-22966-4_4.

[Cor+13] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. *Distributed ElGamal à la Pedersen: Application to Helios*. In: *Workshop on Privacy in the Electronic Society – WPES 2013*. Nov. 2013. DOI: 10.1145/2517840.2517852.

[Cou+21] Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle. *Efficient Range Proofs with Transparent Setup from Bounded Integer Commitments*. In: *Advances in Cryptology – EUROCRYPT 2021, Part III*. Oct. 2021. DOI: 10.1007/978-3-030-77883-5_9.

[Cra97] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis. Universiteit van Amsterdam, Jan. 31, 1997.

[Dam+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. *Multiparty Computation from Somewhat Homomorphic Encryption*. In: *Advances in Cryptology – CRYPTO 2012*. Aug. 2012. DOI: 10.1007/978-3-642-32009-5_38.

[DF92] Yvo Desmedt and Yair Frankel. *Shared Generation of Authenticators and Signatures (Extended Abstract)*. In: *Advances in Cryptology – CRYPTO'91*. Aug. 1992. DOI: 10.1007/3-540-46766-1_37.

[DN03] Ivan Damgård and Jesper Buus Nielsen. *Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption*. In: *Advances in Cryptology – CRYPTO 2003*. Aug. 2003. DOI: 10.1007/978-3-540-45146-4_15.

[DT06] Ivan Damgård and Rune Thorbek. *Linear Integer Secret Sharing and Distributed Exponentiation*. In: *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*. Apr. 2006. DOI: 10.1007/11745853_6.

[EFR21] Andreas Erwig, Sebastian Faust, and Siavash Riahi. *Large-Scale Non-Interactive Threshold Cryptosystems Through Anonymity*. Cryptology ePrint Archive, Report 2021/1290. https://eprint.iacr.org/2021/1290. 2021.

[ElG84] Taher ElGamal. *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. In: *Advances in Cryptology – CRYPTO'84*. Aug. 1984.

[Fel87] Paul Feldman. *A Practical Scheme for Non-interactive Verifiable Secret Sharing*. In: *28th Annual Symposium on Foundations of Computer Science*. Oct. 1987. DOI: 10.1109/SFCS.1987.4.

[Fis05] Marc Fischlin. *Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors*. In: *Advances in Cryptology – CRYPTO 2005*. Aug. 2005. DOI: 10.1007/11535218_10.

[Fra+97] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. *Optimal Resilience Proactive Public-Key Cryptosystems*. In: *38th Annual Symposium on Foundations of Computer Science*. Oct. 1997. DOI: 10.1109/SFCS.1997.646127.

[Gen+07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. *Secure Distributed Key Generation for Discrete-Log Based Cryptosystems*. In: *Journal of Cryptology* 1 (Jan. 2007). DOI: 10.1007/s00145-006-0347-3.

[Gen+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. *YOSO: You Only Speak Once - Secure MPC with Stateless Ephemeral Roles*. In: *Advances in Cryptology – CRYPTO 2021, Part II*. Aug. 2021. DOI: 10.1007/978-3-030-84245-1_3.

[GLM22] S. Dov Gordon, Phi Hung Le, and Daniel McVicker. *Linear Communication in Malicious Majority MPC*. Cryptology ePrint Archive, Report 2022/781. https://eprint.iacr.org/2022/781. 2022.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. *Overdrive: Making SPDZ Great Again*. In: *Advances in Cryptology – EUROCRYPT 2018, Part III*. Apr. 2018. DOI: 10.1007/978-3-319-78372-7_6.

[KRY22] Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. *Towards Efficient YOSO MPC Without Setup*. Cryptology ePrint Archive, Report 2022/187. https://eprint.iacr.org/2022/187. 2022.

[OP01] Tatsuaki Okamoto and David Pointcheval. *The Gap-Problems: A New Class of Problems for the Security of Cryptographic Schemes*. In: *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*. Feb. 2001. DOI: 10.1007/3-540-44586-2_8.

[Pai99] Pascal Paillier. *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. In: *Advances in Cryptology – EUROCRYPT'99*. May 1999. DOI: 10.1007/3-540-48910-X_16.

[Ped91] Torben P. Pedersen. *A Threshold Cryptosystem without a Trusted Party (Extended Abstract) (Rump Session)*. In: *Advances in Cryptology – EUROCRYPT'91*. Apr. 1991. DOI: 10.1007/3-540-46416-6_47.

[Pie19] Krzysztof Pietrzak. *Simple Verifiable Delay Functions*. In: *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*. Jan. 2019. DOI: 10.4230/LIPIcs.ITCS.2019.60.

[Rab98] Tal Rabin. *A Simplified Approach to Threshold and Proactive RSA*. In: *Advances in Cryptology – CRYPTO'98*. Aug. 1998. DOI: 10.1007/BFb0055722.

[Sch90] Claus-Peter Schnorr. *Efficient Identification and Signatures for Smart Cards*. In: *Advances in Cryptology – CRYPTO'89*. Aug. 1990. DOI: 10.1007/0-387-34805-0_22.

[Sha79] Adi Shamir. *How to Share a Secret*. In: *Communications of the Association for Computing Machinery* 11 (Nov. 1979).

[Ste+21] Oliver Stengele, Markus Raiber, Jörn Müller-Quade, and Hannes Hartenstein. *ETHTID: Deployable Threshold Information Disclosure on Ethereum*. In: *Conference on Blockchain Computing and Applications – BCCA 2021*. Nov. 2021. DOI: 10.1109/BCCA53669.2021.9657019.

[Tuc20] Ida Tucker. *Chiffrement fonctionnel et signatures distribuées fondés sur des fonctions de hachage à projection, l'apport des groupes de classes*. PhD thesis. École normale supérieure de Lyon, Oct. 19, 2020.

[Wes19] Benjamin Wesolowski. *Efficient Verifiable Delay Functions*. In: *Advances in Cryptology – EUROCRYPT 2019, Part III*. May 2019. DOI: 10.1007/978-3-030-17659-4_13.

A Supplementary Material on the Encryption Scheme

A.1 The Original HCM-CL Encryption Scheme

Theorem 15 (IND-CPA Security of $\Pi_{\mathsf{hsm\text{-}cl}}$ [CLT18]). *The $\Pi_{\mathsf{hsm\text{-}cl}}$ encryption scheme (Figure 17) provides indistinguishability under chosen-plaintext attacks (IND-CPA) under the HSM assumption (Definition 2).*

```
Setup(1^λ, q)
  1. Output $\mathsf{pp_{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q)$.

KeyGen(pp_cl)
  1. Sample $\mathsf{sk_{cl}} \leftarrow \mathcal{D}_q$, set $\mathsf{pk_{cl}} := g_q^{\mathsf{sk_{cl}}}$.
  2. Output $(\mathsf{pk_{cl}}, \mathsf{sk_{cl}})$.

Enc(pk_cl, m ∈ F_q; r)
  1. Sample $r \leftarrow \mathcal{D}_q$.
  2. Output $\mathsf{ct} := (g_q^r, f^m \cdot \mathsf{pk_{cl}^r})$.

Dec(sk_cl, ct)
  1. Compute $M := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk_{cl}}}$.
  2. Output $\mathsf{CLSolve}(q, \mathsf{pp_{cl}}, M)$.

EvalAdd(pk_cl, ct, m; r)
  1. Sample $r \leftarrow \mathcal{D}_q$.
  2. Output $\mathsf{ct}' = (g_q^r \cdot \mathsf{ct}_1, \mathsf{pk_{cl}^r} \cdot \mathsf{ct}_2 \cdot f^m)$.

EvalSum(pk_cl, ct, ct'; r)
  1. Sample $r \leftarrow \mathcal{D}_q$.
  2. Output $\mathsf{ct}'' = (g_q^r \cdot \mathsf{ct}_1 \cdot \mathsf{ct}'_1, \mathsf{pk_{cl}^r} \cdot \mathsf{ct}_2 \cdot \mathsf{ct}'_2)$.

EvalScal(pk_cl, ct, a; r)
  1. Sample $r \leftarrow \mathcal{D}_q$.
  2. Output $\mathsf{ct}' = (g_q^r \cdot \mathsf{ct}_1^a, \mathsf{pk_{cl}^r} \cdot \mathsf{ct}_2^a)$.
```

Fig. 17. The $\Pi_{\mathsf{hsm\text{-}cl}}$ linearly homomorphic encryption scheme by Castagnos et al. [CLT18; Cas+20]. We added the EvalAdd procedure to add a constant to a ciphertext, and use the additional argument r when we want to make the used randomness explicit.

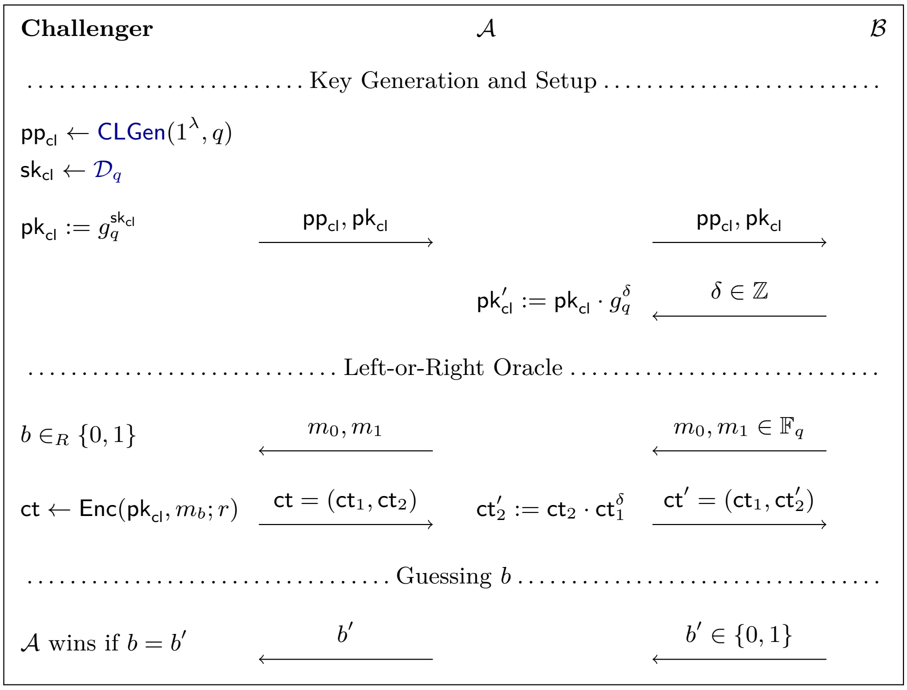
A.2 Reduction in the Proof of Lemma 2

See Figure 18.

A.3 Proof of Lemma 3

Proof of Lemma 3. In the reduction (Figure 19), we assume towards contradiction that an adversary $\mathcal{B}$ can distinguish between real and lossy keys with non-negligible advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},*,\mathsf{pk}}$. We use $\mathcal{B}$ to construct an adversary $\mathcal{A}$ that breaks IND-CPA security of $\Pi^1_{\mathsf{hsm\text{-}cl}}$ with non-negligible advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},1}$.

First $\mathcal{A}$ forwards the public parameters $\mathsf{pp_{cl}}$ generated by the challenger to $\mathcal{B}$. Then $\mathcal{A}$ relays the messages between the IND-CPA challenger and $\mathcal{B}$ such that they create a biased key pair $(\mathsf{pk_{cl}}, \mathsf{sk_{cl}})$ and everybody receives $\mathsf{pk_{cl}}$. In the next phase $\mathcal{A}$ uses the left-or-right query of the IND-CPA game to emulate the execution of $\mathsf{BiasedSpecialKeyGen}_b^{\mathcal{B}}$ towards $\mathcal{B}$: $\mathcal{A}$ sends two messages $m_0 = 0$ and $m_1 = 1$ to the challenger, who samples a bit $b \in_R \{0, 1\}$ and sends a ciphertext ct of m_b back to $\mathcal{A}$. $\mathcal{A}$ forwards ct to $\mathcal{B}$, who responds with a value ε. Given ε, everyone can compute the special public key ct_P. Finally, $\mathcal{B}$ outputs a bit b' whether it believes the generated key is a real or a lossy key, and $\mathcal{A}$ outputs the same bit.



Challenger
$\mathcal{A}$
$\mathcal{B}$
Key Generation and Setup
$\mathsf{pp}_{\mathsf{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q)$
$\mathsf{sk}_{\mathsf{cl}} \leftarrow \mathcal{D}_q$
$\mathsf{pk}_{\mathsf{cl}} := g_q^{\mathsf{sk}_{\mathsf{cl}}}$
$\mathsf{pp}_{\mathsf{cl}}, \mathsf{pk}_{\mathsf{cl}}$
$\mathsf{pp}_{\mathsf{cl}}, \mathsf{pk}_{\mathsf{cl}}$
$\mathsf{pk}'_{\mathsf{cl}} := \mathsf{pk}_{\mathsf{cl}} \cdot g_q^\delta$
$\delta \in \mathbb{Z}$
Left-or-Right Oracle
$b \in_R \{0,1\}$
m_0, m_1
$m_0, m_1 \in \mathbb{F}_q$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{cl}}, m_b; r)$
$\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$
$\mathsf{ct}'_2 := \mathsf{ct}_2 \cdot \mathsf{ct}_1^\delta$
$\mathsf{ct}' = (\mathsf{ct}_1, \mathsf{ct}'_2)$
Guessing b
$\mathcal{A}$ wins if $b = b'$
b'
$b' \in \{0,1\}$

Fig. 18. Reduction of IND-CPA security of $\mathit{\Pi}^1_{\mathsf{hsm\text{-}cl}}$ with biased keys to the IND-CPA security of $\mathit{\Pi}_{\mathsf{hsm\text{-}cl}}$ (Proof of Lemma 2)

If the challenger decides to encrypt $m_b = b$, then $\mathcal{B}$'s view is exactly the same as in Step 4 of $\mathsf{BiasedSpecialKeyGen}_b^{\mathcal{B}}$. Hence, its view is the same as specified in the Lemma, and if $\mathcal{B}$ guesses correctly, then so does $\mathcal{A}$. Therefore $\mathcal{A}$ wins the IND-CPA game with advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},1} = \mathsf{Adv}_{\mathcal{B}}^{\mathsf{hsm\text{-}cl},*,\mathsf{pk}}$. Since this contradicts the premise that $\mathit{\Pi}^1_{\mathsf{hsm\text{-}cl}}$ is IND-CPA secure, such a $\mathcal{B}$ cannot exist. □

A.4 Proof of Lemma 4

Proof of Lemma 4. In the reduction (Figure 20), we assume towards contradiction that an adversary $\mathcal{B}$ can break the IND-CPA security of $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ with non-negligible advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},*}$. We use $\mathcal{B}$ to construct an adversary $\mathcal{A}$ that can distinguish between real and lossy keys of $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$ with non-negligible advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},*,\mathsf{pk}}$.

First $\mathcal{A}$ forwards the public parameters $\mathsf{pp}_{\mathsf{cl}}$ generated by the challenger to $\mathcal{B}$. Then $\mathcal{A}$ relays the messages between the IND-CPA challenger and $\mathcal{B}$ such that they create a biased special key pair $(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}}) \leftarrow \mathsf{BiasedSpecialKeyGen}_b^{\mathcal{B}}(\mathsf{pp}_{\mathsf{cl}})$, where $b \in_R \{0,1\}$ by the challenger, is generated. Then, $\mathcal{A}$ and $\mathcal{B}$ run the IND-CPA game with the generated $\widetilde{\mathsf{pk}}_{\mathsf{cl}}$. If $\mathcal{B}$ wins this game, then $\mathcal{A}$ guesses that a real key was generated ($b' = 1$). Otherwise, it guesses that a lossy key was generated ($b' = 0$).

In case $b = 1$, the game is in $\mathcal{B}$'s view identical to the IND-CPA game for $\mathit{\Pi}^*_{\mathsf{hsm\text{-}cl}}$, and $\mathcal{A}$ wins iff $\mathcal{B}$ wins. Hence, we have $\Pr[\mathcal{A} \text{ wins} \mid b = 1] = 1/2 + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},*}(\lambda)$.

In the other case $b = 0$, $\mathcal{B}$ needs to distinguish between the distributions $\mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, m_0; r)$ and $\mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, m_1; r)$. However, since $r \leftarrow \mathcal{D}_q$ induces an almost-uniform distribution over G^q, they

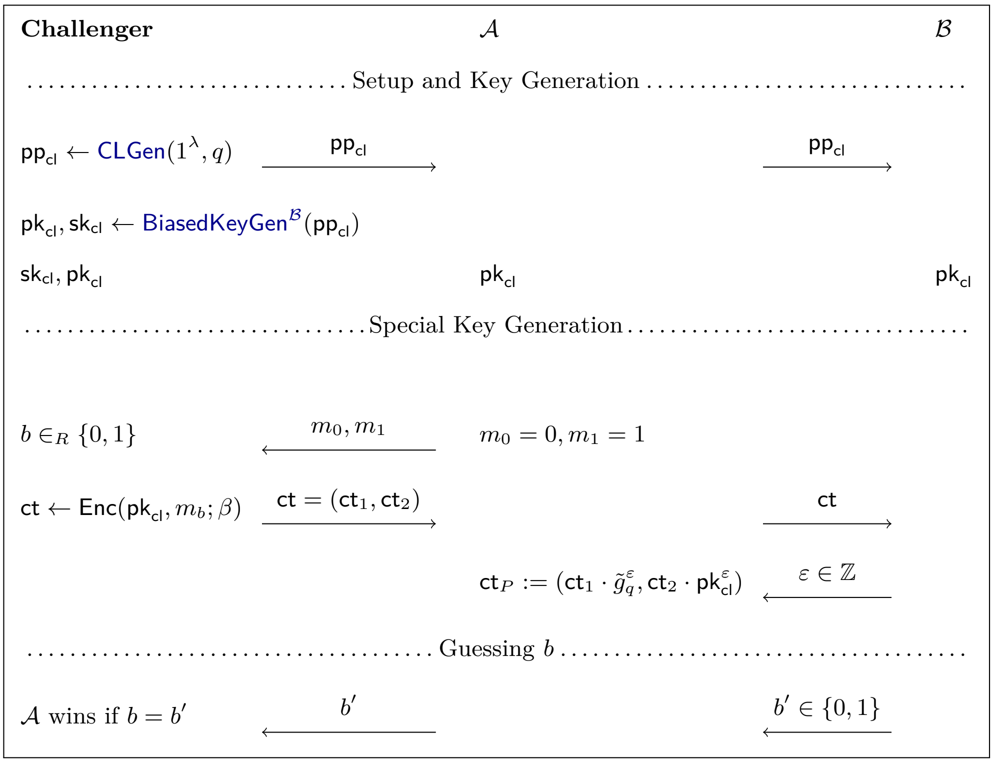

Challenger
$\mathcal{A}$
$\mathcal{B}$
Setup and Key Generation
$\mathsf{pp}_{\mathsf{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q)$
$\mathsf{pp}_{\mathsf{cl}}$
$\mathsf{pp}_{\mathsf{cl}}$
$\mathsf{pk}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}} \leftarrow \mathsf{BiasedKeyGen}^{\mathcal{B}}(\mathsf{pp}_{\mathsf{cl}})$
$\mathsf{sk}_{\mathsf{cl}}, \mathsf{pk}_{\mathsf{cl}}$
$\mathsf{pk}_{\mathsf{cl}}$
$\mathsf{pk}_{\mathsf{cl}}$
Special Key Generation
$b \in_R \{0,1\}$
m_0, m_1
$m_0 = 0, m_1 = 1$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathsf{cl}}, m_b; \beta)$
$\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$
ct
$\mathsf{ct}_P := (\mathsf{ct}_1 \cdot \tilde{g}_q^{\varepsilon}, \mathsf{ct}_2 \cdot \mathsf{pk}_{\mathsf{cl}}^{\varepsilon})$
$\varepsilon \in \mathbb{Z}$
Guessing b
$\mathcal{A}$ wins if $b = b'$
b'
$b' \in \{0,1\}$

Fig. 19. Reduction of the indistinguishability of special keys of $\Pi^{*}_{\mathsf{hsm\text{-}cl}}$ to the IND-CPA security of $\Pi^{1}_{\mathsf{hsm\text{-}cl}}$ (Proof of Lemma 3)

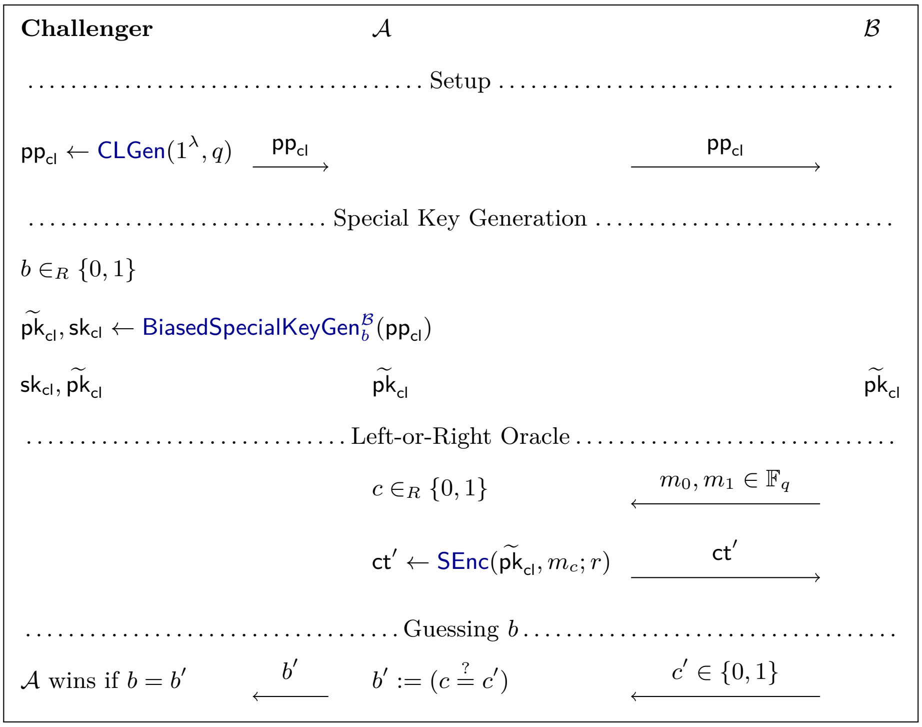

Challenger
$\mathcal{A}$
$\mathcal{B}$
Setup
$\mathsf{pp}_{\mathsf{cl}} \leftarrow \mathsf{CLGen}(1^\lambda, q)$
$\xrightarrow{\mathsf{pp}_{\mathsf{cl}}}$
$\xrightarrow{\mathsf{pp}_{\mathsf{cl}}}$
Special Key Generation
$b \in_R \{0,1\}$
$\widetilde{\mathsf{pk}}_{\mathsf{cl}}, \mathsf{sk}_{\mathsf{cl}} \leftarrow \mathsf{BiasedSpecialKeyGen}_b^{\mathcal{B}}(\mathsf{pp}_{\mathsf{cl}})$
$\mathsf{sk}_{\mathsf{cl}}, \widetilde{\mathsf{pk}}_{\mathsf{cl}}$
$\widetilde{\mathsf{pk}}_{\mathsf{cl}}$
$\widetilde{\mathsf{pk}}_{\mathsf{cl}}$
Left-or-Right Oracle
$c \in_R \{0,1\}$
$\xleftarrow{m_0, m_1 \in \mathbb{F}_q}$
$\mathsf{ct}' \leftarrow \mathsf{SEnc}(\widetilde{\mathsf{pk}}_{\mathsf{cl}}, m_c; r)$
$\xrightarrow{\mathsf{ct}'}$
Guessing b
$\mathcal{A}$ wins if $b = b'$
$\xleftarrow{b'}$
$b' := (c \overset{?}{=} c')$
$\xleftarrow{c' \in \{0,1\}}$

Fig. 20. Reduction of IND-CPA security of $\Pi^*_{\mathsf{hsm\text{-}cl}}$ to indistinguishability of working/lossy special keys of $\Pi^*_{\mathsf{hsm\text{-}cl}}$ (Proof of Lemma 4)

are statistically close. Hence, $\mathcal{B}$ can only guess b' with probability $1/2 + \mathsf{negl}(\sigma)$. As $\mathcal{A}$ wins iff $\mathcal{B}$ guesses wrongly, we have $\Pr[\mathcal{A} \text{ wins} \mid b = 0] = 1/2 - \mathsf{negl}(\sigma)$.

Overall, we combine the two cases to obtain

$$\Pr[\mathcal{A} \text{ wins}] = 1/2 + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},*}(\lambda)/2 - \mathsf{negl}(\sigma).$$

Therefore, $\mathcal{A}$ has advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},*,\mathsf{pk}}(\lambda) \geq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},*}(\lambda)/2 - \mathsf{negl}(\sigma)$ in the key distinguishing game. If σ is chosen such that $\mathsf{negl}(\sigma)$ is also negligible in λ, then $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{hsm\text{-}cl},*,\mathsf{pk}}(\lambda)$ is non-negligible in λ, which contradicts the premise. □

B Supplementary Material on Secret Sharing

B.1 Privacy of Shamir's Secret Sharing over $\mathbb{Z}$

Definition 8 (Privacy of a Secret Sharing Scheme [DT06]). *A secret sharing scheme is statistically private if for any set of corrupted parties $\mathcal{C} \subset \{P_1, \ldots, P_N\}$ with $|\mathcal{C}| \leq t$, any two secrets $\alpha, \alpha' \in [0, 2^\ell)$ and independent random coins $\mathbf{r}, \mathbf{r}'$, we have that the statistical distance between $\{\mathsf{Share}_i(\alpha, \mathbf{r}) \mid i \in \mathcal{C}\}$ and $\{\mathsf{Share}_i(\alpha', \mathbf{r}') \mid i \in \mathcal{C}\}$ is negligible in σ.*

Protocol 1 achieves this definition of privacy. The proof is along the lines of [Rab98; DT06] and uses the existence of so called sweeping polynomials $h_\mathcal{C}(X)$ for any set $\mathcal{C}$ of at most t corrupted parties such that $h(0) = \Delta$ and $h(i) = 0$ for $P_i \in \mathcal{C}$. Let h_{max} be a bound on the coefficients of the $h_\mathcal{C}$.

Theorem 16. *Protocol 1 provides statistical privacy according to Definition 8, when $\ell_0 \geq \ell + \lceil \log_2(h_{\mathsf{max}} \cdot t) \rceil + 1$.*

In the proof, we use the so-called Sweeping polynomial:

Definition 9 (Sweeping Polynomial). *Let $\mathcal{C} \subseteq [N]$ such that $|\mathcal{C}| = t$. Then we define the* sweeping polynomial *$h_\mathcal{C} := \sum_{i=0} h_{\mathcal{C},i} \cdot X^i \in \mathbb{Z}[X]_{\leq t}$ as the unique polynomial of degree at most t such that $h(0) = \Delta$ and $h(i) = 0$ for all $i \in \mathcal{C}$. Moreover, let h_{max} be an upper bound on the coefficients of the sweeping polynomials, e.g, $h_{\mathsf{max}} \geq \max\{h_{\mathcal{C},i} \mid i \in [0,t], \mathcal{C} \subseteq [N], |\mathcal{C}| = t\}$*

Lemma 17 (Existence of Sweeping Polynomial). *For any $\mathcal{C} \subseteq \{1, \ldots, N\}$ with $|\mathcal{C}| = t$, there exists $h_\mathcal{C} \in \mathbb{Z}[X]_{\leq t}$ satisfying Definition 9.*

Proof. From the conditions $h(0) = \Delta$ and $h(j) = 0$ for $j \in \mathcal{C}$, we get by Lagrange interpolation (Equation 1)

$$h(X) = \Delta \cdot \prod_{j \in \mathcal{C}} \frac{j - X}{j}. = \Delta_\mathcal{C} \cdot \prod_{j \in \mathcal{C}} (j - X)$$

with $\Delta_\mathcal{C} := \Delta \cdot (\prod_{j \in \mathcal{C}} j)^{-1} = \prod\{n \mid 1 \leq n \leq N \wedge n \notin \mathcal{C}\} \in \mathbb{Z}$. Hence, we also have $h \in \mathbb{Z}[X]_{\leq t}$. □

Proof of Theorem 16. This proof is adapted from the proof of Lemma 1 in [DT06].

Let $\alpha, \alpha' \in [0, 2^\ell)$ be arbitrary with $\tilde{\alpha} = \alpha \cdot \Delta$ and $\tilde{\alpha}' = \alpha' \cdot \Delta$, and let $\mathcal{C} \subseteq [N]$ be an arbitrary set of corrupted parties of size $|\mathcal{C}| = t$.

Suppose α is shared as $\mathsf{Share}(\alpha, \mathbf{r})$ using a polynomial $f = \tilde{\alpha} + \sum_{i=1}^t r_i \cdot X^i \in \mathbb{Z}[X]_{\leq t}$ derived from the randomness $\mathbf{r} = (r_1, \ldots, r_t) \in [0, 2^{\ell_0 + \sigma})^t$. Then $f(0) = \tilde{\alpha}$, and $\mathsf{Share}_j(\alpha, \mathbf{r}) = f(j)$ for $j \in [N]$.

The adversary sees the $t = |\mathcal{C}|$ shares $\{f(j) \mid j \in \mathcal{C}\}$. By Lagrange interpolation, this induces a one-to-one map from possible secrets to sharing polynomials. We use the sweeping polynomial $h_\mathcal{C}$ from Definition 9 to make this map explicit as $\alpha^* \mapsto f(X) + (\alpha^* - \alpha) \cdot h_\mathcal{C}(X)$.

We say a sharing polynomial is good if all coefficients are good, i.e., lie in the specified range $[0, 2^{\ell_0 + \sigma})$. Otherwise, we call them bad.

Since the sharing algorithm generates a polynomial with coefficients sampled uniformly from this range, each possible secret that maps to a good sharing polynomial is consistent with the adversaries view, and – given no further information – equally likely due to the one-to-one correspondence.

By applying the map to α', we get $f'(X) := f(X) + (\alpha' - \alpha) \cdot h_\mathcal{C}(X)$, which we can write as $f' = \tilde{\alpha}' + \sum_{i=1}^t r'_i \cdot X^i$ with $\mathsf{r}' = (r'_1, \ldots, r'_t)$. Then $f'(0) = \tilde{\alpha}'$ and $\mathsf{Share}_j(\alpha, \mathbf{r}) = f'(j) = f(j)$ for all $j \in \mathcal{C}$. By the above, if f' is good, the adversary cannot distinguish whether α or α' was originally shared. Hence, only if f' is bad it can distinguish the two cases. The probability that f' is bad is – by the union bound – at most t times the probability that one of the coefficients r'_i is bad.

These are computed as $r'_i = r_i + (\alpha' - \alpha) \cdot h_{\mathcal{C},i}$. We know that $|\alpha' - \alpha| \in [0, 2^\ell)$, and we also know that $h_{\mathcal{C},i} \leq h_{\mathsf{max}}$. Hence, if $r_i \in [2^\ell \cdot h_{\mathsf{max}}, 2^{\ell_0+\sigma} - 2^\ell \cdot h_{\mathsf{max}}]$ then r'_i will be good. So we can bound the probability that r'_i is bad by $(2 \cdot 2^\ell \cdot h_{\mathsf{max}})/(2^{\ell_0+\sigma})$. Since we have $\ell_0 \geq \ell + \lceil \log_2(h_{\mathsf{max}} \cdot t) \rceil + 1$, it follows that $(t \cdot 2 \cdot 2^\ell \cdot h_{\mathsf{max}})/(2^{\ell_0+\sigma}) \leq 2^{-\sigma}$. So the views of the adversary are statistically indistinguishable. □

B.2 Integer Reconstruction

Let $\mathcal{X} \subseteq [0, N]$ with $|\mathcal{X}| = t + 1$. We write $\mathcal{X} = \{x_0, \dots, x_t\}$ with $x_0 < x_1 < \dots < x_t$, and define the Vandermonde matrix $V_{\mathcal{X}}$ as

$$V_{\mathcal{X}} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^t \\ 1 & x_1 & x_1^2 & \dots & x_1^t \\ 1 & x_2 & x_2^2 & \dots & x_2^t \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & x_t^2 & \dots & x_t^t \end{pmatrix} \in \mathbb{Z}^{(t+1)\times(t+1)}.$$

By the same reasoning as with the Lagrange interpolation formula we get that the inverse of a Vandermonde matrix multiplied by Δ has integer entries:

Lemma 18 (Inverse of the Vandermonde Matrix). *Let $V_{\mathcal{X}}$ be a Vandermonde matrix as defined above. Then, $\Delta \cdot V_{\mathcal{X}}^{-1}$ is a $(t+1) \times (t+1)$ matrix with integer entries, i.e., $\Delta \cdot V_{\mathcal{X}}^{-1} \in \mathbb{Z}^{(t+1)\times(t+1)}$.*

Proof of Lemma 5. Let $\mathcal{X} := \{x_0, \dots, x_t\}$ and $V_{\mathcal{X}}$ the corresponding Vandermonde matrix (see above), and $\mathbf{y} := (y_0 \ \cdots \ y_t)^T$. By Lagrange interpolation, there exists a polynomial $g[X] \in \mathbb{Q}[X]$ of degree at most t such that $g(x_i) = y_i$ for $i = 0, \dots, t$. Let $\mathbf{b} := (b_0 \ \cdots \ b_t)^T \in \mathbb{Q}^{t+1}$ be the coefficient vector of g. Then $V_{\mathcal{X}} \cdot \mathbf{b} = \mathbf{y}$. Since the x_i are pairwise distinct, $V_{\mathcal{X}}$ is invertible. So we have $\mathbf{b} = V_{\mathcal{X}}^{-1} \cdot \mathbf{y}$. Multiplying both sides with Δ yields $\Delta \cdot \mathbf{b} = \Delta \cdot V_{\mathcal{X}}^{-1} \cdot \mathbf{y}$.

Set $f := \Delta \cdot g$, so that $\mathbf{a} := \Delta \cdot \mathbf{b}$ is the corresponding coefficient vector, and $f(x_i) = \Delta \cdot y_i$ for $i \in [0, t]$. By Lemma 18, the entries of $\Delta \cdot V_{\mathcal{X}}^{-1}$ are integers. Hence, the coefficients of f can be written as linear combinations of the y_i with integer coefficients. □

C The Issue With Using Rabin's VSS Protocol

In [Rab98], Rabin describes a VSS protocol that is very similar to our Feldman-style VSS: the protocol works over the group $\mathbb{Z}_n^*$ where n is an RSA modulus, and uses a large order element $g \in \mathbb{Z}_n^*$. The dealer does an Shamir-style integer secret sharing of Δs with threshold t, where s is the secret, $\Delta = N!$ and N is the number of players. She also publishes g^{r_j} where the r_j's are the coefficients of the underlying polynomial, and sends a share privately to each player. Shares are now verified to satisfy an equation that essentially evaluates the underlying polynomial in the exponent using g as the base. This is the same idea we also use and it comes from Feldman's work.

The reconstruction protocol starts from at least $t+1$ shares. It selects a (sub)set of $t+1$ shares that satisfy the verification equation, and then interpolates to get the secret. Concretely this is done by doing the computation modulo a large prime P. If P is large enough that overflow does not occur, this mimics an interpolation over the rationals. The last step is to multiply by $\Delta^{-1} \bmod P$, which will return s, assuming the interpolation returned Δs.

It is not proved in [Rab98] that reconstruction works, it is just stated that it follows directly from Feldman's work. However, we were not able to reconstruct the argument. The problem is the following: Feldman's suggestion was to use a group of known public order, such $\mathbb{Z}_p^*$ for a prime p, or a prime order subgroup. In such a setting, it is the case that from any $t+1$ shares that verify, one can reconstruct the correct secret, and this can be concluded without relying on any computational assumptions. But unfortunately, this is not true for the case where integer secret sharing is used over an unknown order group. Intuitively, the reason is that we use integer secret sharing, but the verification of shares only guarantees that they are correct modulo the group order, not that they are the correct integers. More concretely, we found that there is an attack: a corrupt dealer who knows the group order can break reconstruction, and we sketch the attack below.

This shows that one cannot prove reconstruction is secure without relying on the assumption that the group order is hard to compute. However, we were not able to find a way to use this assumption in a proof of security of the protocol from [Rab98]. Instead, we designed our own version where the dealer must prove in zero-knowledge that she knows the discrete log of g^s where s is the secret and g is the fixed base element used in the share verification.

For completeness, here is a sketch of the attack mentioned above: assume the corrupt dealer knows the order D of the group used. We assume all other parties are not corrupted. She will then execute the sharing phase honestly with secret s, except that she will send $s_{t+1} + (t+1)D$ to P_{t+1} where s_{t+1} is the correct share. The share verification computes $g^{s_{t+1}+(t+1)D}$ and tests if this is some expected value, but since the factor $g^{(t+1)D}$ vanishes, the verification will work. Further, the share received is 0 modulo $t+1$ as it would be in a normal sharing, and in fact if the secret is large enough, $s_{t+1} + (t+1)D$ will be statistically indistinguishable from s_{t+1}. So the attack cannot be detected. Assume that parties $P_1, \ldots, P_{t+1}$ try to reconstruct. In this setting, the Lagrange interpolation coefficient for the share of P_{t+1} happens to be -1. This immediately implies that interpolation will return $\Delta s - (t+1)D$. Under the reasonable assumption that D is not divisible by $t!$, this result is not divisible by Δ, and so the reconstruction cannot be completed correctly: recall that in the honest case, the value of the polynomial in 0 is Δs. On the other hand, when a set of honest players not including P_{t+1} reconstruct, they will get the correct s.

D ElGamal Encryption with Biased Keys

Here we show the analogue of our result from Section 3.1 for the ElGamal encryption scheme, i.e., the latter stays IND-CPA secure, even if the adversary $\mathcal{A}$ can bias the key. We recall the ElGamal encryption scheme [ElG84] over a prime order group $\mathbb{G}$, which we denote as $\mathit{\Pi}_{\mathsf{EG}} := (\mathsf{Setup}_{\mathsf{EG}}, \mathsf{KeyGen}_{\mathsf{EG}}, \mathsf{Enc}_{\mathsf{EG}}, \mathsf{Dec}_{\mathsf{EG}})$, in Figure 21. Again, we additionally specify a modified version $\mathit{\Pi}^*_{\mathsf{EG}} := (\mathsf{Setup}_{\mathsf{EG}}, \mathsf{BiasedKeyGen}^{\mathcal{A}}_{\mathsf{EG}}, \mathsf{Enc}_{\mathsf{EG}}, \mathsf{Dec}_{\mathsf{EG}})$, where the adversary $\mathcal{A}$ is allowed to bias the generated key.

```
$\mathsf{Setup}_{\mathsf{EG}}(1^\lambda)$
  1. Choose an appropriate group $\mathbb{G}$ of prime order $q$ with generator $g$.
  2. Output $\mathsf{pp}_{\mathsf{EG}} := (\mathbb{G}, q, g)$.

$\mathsf{KeyGen}_{\mathsf{EG}}(\mathsf{pp}_{\mathsf{EG}})$
  1. Sample $\mathsf{sk}_{\mathsf{EG}} \in_R \mathbb{F}_q$, set $\mathsf{pk}_{\mathsf{EG}} := g^{\mathsf{sk}_{\mathsf{EG}}}$.
  2. Output $(\mathsf{pk}_{\mathsf{EG}}, \mathsf{sk}_{\mathsf{EG}})$.

$\mathsf{Enc}_{\mathsf{EG}}(\mathsf{pk}_{\mathsf{EG}}, m \in \mathbb{G}; r)$
  1. Sample $r \in_R \mathbb{F}_q$.
  2. Output $\mathsf{ct} := (g^r, \mathsf{pk}^r_{\mathsf{EG}} \cdot m)$.
```

```
$\mathsf{Dec}_{\mathsf{EG}}(\mathsf{sk}_{\mathsf{EG}}, \mathsf{ct})$
  1. Output $m := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}_{\mathsf{EG}}}$.

$\mathsf{BiasedKeyGen}^{\mathcal{A}}_{\mathsf{EG}}(\mathsf{pp}_{\mathsf{EG}})$
  1. Sample $\alpha \in_R \mathbb{F}_q$.
  2. Set $\mathsf{pk}^*_{\mathsf{EG}} := g^\alpha$.
  3. $\delta \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{EG}}, \mathsf{pk}^*_{\mathsf{EG}})$.
  4. $\mathsf{sk}_{\mathsf{EG}} := \alpha + \delta$ and
     $\mathsf{pk}_{\mathsf{EG}} := g^{\mathsf{sk}_{\mathsf{EG}}} = g^{\alpha+\delta}$.
  5. Output $(\mathsf{pk}_{\mathsf{EG}}, \mathsf{sk}_{\mathsf{EG}})$.
```

Fig. 21. Standard ElGamal encryption scheme and modified key generation $\mathsf{BiasedKeyGen}^{\mathcal{A}}_{\mathsf{EG}}$ that allow an adversary $\mathcal{A}$ to influence the distribution of public keys in a limited way.

Theorem 19 (IND-CPA Security of $\mathit{\Pi}^*_{\mathsf{EG}}$). *If $\mathit{\Pi}_{\mathsf{EG}}$ is indistinguishabile under chosen-plaintext attacks (IND-CPA) then so is $\mathit{\Pi}^*_{\mathsf{EG}}$.*

Proof. We prove the claim by showing that any adversary $\mathcal{B}$ that wins the IND-CPA game for $\mathit{\Pi}^*_{\mathsf{EG}}$ with advantage $\mathsf{Adv}^{\mathsf{EG},*}_{\mathcal{B}}$ can be transformed into an adversary $\mathcal{A}$ for $\mathit{\Pi}_{\mathsf{EG}}$ with the same advantage $\mathsf{Adv}^{\mathsf{EG}}_{\mathcal{A}} = \mathsf{Adv}^{\mathsf{EG},*}_{\mathcal{B}}$. The reduction is depicted in Figure 22. Given an adversary $\mathcal{B}$, we create $\mathcal{A}$ as follows: Initially $\mathcal{A}$ receives public parameters $\mathsf{pp}_{\mathsf{EG}}$ and a public key $\mathsf{pk}_{\mathsf{EG}}$ from the challenger. It gives $\mathsf{pk}_{\mathsf{EG}}$ to $\mathcal{B}$, which responds with δ. Define the biased public key $\mathsf{pk}'_{\mathsf{EG}} := \mathsf{pk}_{\mathsf{EG}} \cdot g^\delta$. This phase corresponds to the $\mathsf{BiasedKeyGen}^{\mathcal{B}}_{\mathsf{EG}}(\mathsf{pp}_{\mathsf{EG}})$ procedure. Then $\mathcal{B}$ selects $m_0, m_1 \in \mathbb{G}$, which $\mathcal{A}$ forwards to the challenger. The challenger samples $b \in_R \{0,1\}$, encrypts m_b and sends the resulting $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ to $\mathcal{A}$. $\mathcal{A}$ sends $\mathsf{ct}' = (\mathsf{ct}'_1, \mathsf{ct}'_2) = (\mathsf{ct}_1, \mathsf{ct}_2 \cdot \mathsf{ct}_1^\delta)$ to $\mathcal{B}$. Finally, $\mathcal{B}$ outputs a bit $b' \in \{0,1\}$, and $\mathcal{A}$ forwards this output to the challenger.

The challenger creates the ciphertext $\mathsf{ct} = (g^r, \mathsf{pk}^r_{\mathsf{EG}} \cdot m_b)$, where r denotes the randomness used during the encryption. We have $\mathsf{pk}'_{\mathsf{EG}} = \mathsf{pk}_{\mathsf{EG}} \cdot g^\delta_q$ and

$$\mathsf{ct}' = \big(g^r, (\mathsf{pk}^r_{\mathsf{EG}} \cdot m_b) \cdot (g^r)^\delta\big) = \big(g^r, (\mathsf{pk}_{\mathsf{EG}} \cdot g^\delta)^r \cdot m_b\big) = \big(g^r, (\mathsf{pk}'_{\mathsf{EG}})^r \cdot m_b\big).$$

Therefore, ct' is a valid encryption of m_b under the biased public key $\mathsf{pk}'_{\mathsf{EG}}$ with the same distribution as it would normally have.[13] Overall, $\mathcal{A}$ wins the game iff $\mathcal{B}$ answers correctly, which happens with probability $1/2 + \mathsf{Adv}^{\mathsf{EG},*}_{\mathcal{B}}$. Hence, the advantage in the case of biased keys is the same as in the standard scheme. □

[13] Independently of us, Stengele et al. [Ste+21] have made the same observation.

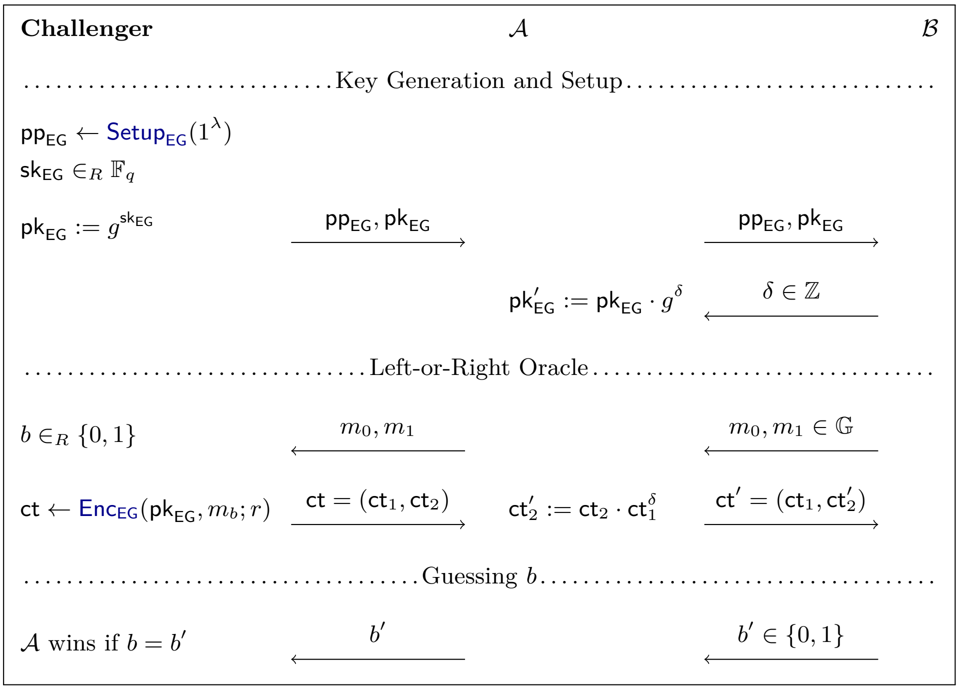

Challenger
$\mathcal{A}$
$\mathcal{B}$
Key Generation and Setup
$\mathsf{pp}_{\mathsf{EG}} \leftarrow \mathsf{Setup}_{\mathsf{EG}}(1^\lambda)$
$\mathsf{sk}_{\mathsf{EG}} \in_R \mathbb{F}_q$
$\mathsf{pk}_{\mathsf{EG}} := g^{\mathsf{sk}_{\mathsf{EG}}}$
$\xrightarrow{\mathsf{pp}_{\mathsf{EG}}, \mathsf{pk}_{\mathsf{EG}}}$
$\xrightarrow{\mathsf{pp}_{\mathsf{EG}}, \mathsf{pk}_{\mathsf{EG}}}$
$\mathsf{pk}'_{\mathsf{EG}} := \mathsf{pk}_{\mathsf{EG}} \cdot g^\delta$
$\xleftarrow{\delta \in \mathbb{Z}}$
Left-or-Right Oracle
$b \in_R \{0,1\}$
$\xleftarrow{m_0, m_1}$
$\xleftarrow{m_0, m_1 \in \mathbb{G}}$
$\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{EG}}(\mathsf{pk}_{\mathsf{EG}}, m_b; r)$
$\xrightarrow{\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)}$
$\mathsf{ct}'_2 := \mathsf{ct}_2 \cdot \mathsf{ct}_1^\delta$
$\xrightarrow{\mathsf{ct}' = (\mathsf{ct}_1, \mathsf{ct}'_2)}$
Guessing b
$\mathcal{A}$ wins if $b = b'$
$\xleftarrow{b'}$
$\xleftarrow{b' \in \{0,1\}}$

Fig. 22. Reduction of IND-CPA security of Π^*_{EG} with biased keys to the IND-CPA security of Π_{EG} (Proof of Theorem 19)

This result is relevant in a distributed setting where the key is generated by each party doing a Feldman VSS of a random value x_i and then combining those VSS's that were successful. More specifically, party P_i will publish $y_i = g^{x_i}$ as well as information related to the shares, and at the end the public key becomes $\prod_{i \in Q} g^{x_i}$ where Q is the set of parties that completed the VSS successfully.[14]

We can now make a reduction showing that this distributed key generation is equivalent to choosing a biased key as described above. Informally, when the challenger sends $\mathsf{pk}_{\mathsf{EG}}$, the reduction will choose random y_i for the honest players subject to the condition that $\mathsf{pk}_{\mathsf{EG}} = \prod_{i \in H} y_i$, where H is the set of honest parties. It will then simulate each of the Feldman VSS's given the y_i which can be done perfectly. It then let the adversary run VSS's for the corrupt parties, and can extract the secret x_i from each successful VSS. It now sets $\delta = \sum_{i \in Q \setminus H} x_i$, the sum of contributions from corrupt players. It sends δ to the challenger, and it is then clear that the biased public key seen by the challenger is exactly the public key computed by the distributed protocol.

[14] Cortier et al. [Cor+13] reached the same result by directly proving IND-CPA security of ElGamal with such a key generation based on the DDH assumption.