Threshold (T)FHE without smudging by means of correct threshold additive HE*

Antonina Bondarchuk[1], Renaud Sirdey[1], Aymen Boudguiga[1] and Olive Chakraborty[1]

Université Paris-Saclay, CEA, List, Palaiseau, France,
name.surname@cea.fr

Abstract. Designing threshold lattice-based FHE schemes remains challenging due to the noise leakage that may occur during distributed decryption. The mainstream approach to avoid this consists of relying on noise flooding or smudging techniques. However, using these techniques comes at the cost of the much larger parameters required to ensure reliable decryption, and they are not easily applicable to schemes with smaller ciphertext moduli such as TFHE. In this paper, we demonstrate that smudging can be avoided for LWE-based FHE schemes, such as TFHE, by using a *correct* Linear Homomorphic Encryption (LHE) scheme, like Paillier, to encrypt the b-term of an LWE pair obtained *after* completing a homomorphic computation and sanitizing the resulting ciphertext. The key intuition is that, because the b-term of the LWE pair is *not* revealed to the decrypting parties, noise leakage is no longer an issue (with sanitization ensuring noise/message independence). We then instantiate this approach using TFHE as the baseline LWE scheme as well as the Tiresias threshold variant of Paillier. We prove the resulting construction, denoted thPLWE, is IND-CPA secure under static corruption and adaptive queries (Scor-Adp-IND-CPA) under the assumption that Tiresias achieves the same. We then provide experimental results to assess the practicality of the approach and compare it to other recent works.

1 Introduction

Threshold Fully Homomorphic Encryption (ThFHE) extends the capabilities of standard FHE by enabling function evaluation and threshold decryption among multiple parties. In this model, a subset of participants collaboratively generates a common public key, which they can then use to locally encrypt messages.

* This work was supported by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute and ANR-23-PECL-0009 TRUSTINCloudS as well as by the European Union under Grant Agreement No. 101167964. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Competence Centre. Neither the European Union nor the European Cybersecurity Competence Centre can be held responsible for them.

These ciphertexts serve as inputs to a target homomorphic computation, whose resulting output is a ciphertext shared among all participants. Finally, the parties must cooperate to decrypt this final result, ensuring that no single party can access the underlying data in isolation. Developing secure ThFHE schemes introduces several significant challenges, particularly regarding coordination and security against colluding parties.

To better understand challenges of constructing threshold encryption based on Learning With Errors (LWE), we first review how standard (non-threshold) LWE operates. We consider encrypting a plaintext $m \in \mathbb{Z}_t$ using a standard LWE ciphertext of the form $(\mathbf{a}, b)$ such that $\mathbf{a} \in \mathbb{Z}_q^n$, $b = \langle \mathbf{a}, \mathsf{sk} \rangle + \Delta m + e \in \mathbb{Z}_q$, where t and q are plaintext and ciphertext moduli respectively, $\mathsf{sk} \in \{0,1\}^n$ a secret key, $\Delta = \frac{q}{t}$ a scaling factor and e a Gaussian noise (i.e., the error term). To enable threshold LWE, the secret key sk is distributed among K parties P_i for $i \in [\![0, K-1]\!]$ using a linear secret sharing scheme with respect to a threshold $T < K$. We denote these shares as $(\mathsf{sk})_i$. From the ciphertext $(\mathbf{a}, b)$, P_i can produce a secret share of the *phase* $\varphi = \Delta m + e$ by computing $b - \langle \mathbf{a}, (\mathsf{sk})_i \rangle = (\varphi)_i$. Then by collaborating in opening the latter value, all the parties can perform rounding and obtain the value of m. However, this partial decryption discloses the value of e. When combined with the ciphertext $(\mathbf{a}, b)$ and the message m, the exposure of e reveals information about the secret key sk. Thus, ensuring security requires hiding the error term e during the decryption. This issue is addressed in two ways. A first approach, called *noise flooding* or *smudging*, consists in adding a significantly large noise to the shared values $(\varphi)_i$ before the opening to mask the original error. The second approach recovers the message m from the shared value $(\varphi)_i$ by running a generic MPC protocol to perform the rounding obliviously.

To perform smudging, the decrypting parties derive an additional secret shared noise E [1] that statistically masks the error e. The parties then open the value $\Delta m + e + E$. Choosing E too small leaks information about e, while choosing it excessively large prevents correct recovery of m after rounding [2]. Consequently, smudging naturally leads to significant overhead on the cryptosystem parameters. Indeed, to guarantee correct decryption after the addition of smudging noise, the ciphertext modulus q must be substantially increased, from $q > 2tB_e$ in the standard case to $q > 2^{\lambda+1}tB_e$ in the threshold setting, where B_e is the bound on the error for the evaluated function and λ is the security parameter. In RLWE-based homomorphic encryption schemes such as BGV [3], BFV [4, 5] and CKKS [6], this problem is generally manageable because the ciphertext moduli are already very large, reaching several hundred bits, and provide enough margin to tolerate the additional inserted noise, while preserving decryption correctness. By contrast, schemes like FHEW [7] and TFHE [8, 9] employ much smaller parameters (typically with 32 or 64-bit ciphertext modulus), making smudging integration more difficult. An alternative to smudging consists of running an MPC-protocol between decrypting parties to directly extract m [10, 11]. That is, the MPC-protocol eliminates the noise entirely before any values are disclosed.

Summary of Contributions. In this paper, we propose a threshold LWE encryption scheme that eliminates the need for noise smudging. We introduce a novel threshold decryption technique inspired by [12], which combines threshold LWE-based encryption with a correct threshold Linear Homomorphic Encryption (LHE) scheme. Our core approach involves encrypting the LWE b-term and executing the linear part of the LWE decryption function under the LHE. This ensures that b remains inaccessible to the decrypting parties, thereby preventing secret key recovery attacks. We emphasize that our approach is simpler and more efficient than noise smudging, as it avoids the substantial parameter expansion typically required to accommodate smudging noise. Furthermore, for a large number of parties (20 or beyond), our construction offers a more efficient alternative compared to the threshold decryption approaches that use MPC-based message recovery, as it bypasses heavy MPC machinery and eliminates the need for a costly preprocessing phase. In essence, after homomorphic computation and sanitization (a technique which ensures circuit privacy and, perhaps more importantly, noise/message independence), a server encrypts the resulting b-term under the threshold LHE scheme thLHE, yielding $(\mathbf{a}, \mathsf{thLHE.Enc}(b))$. The parties P_i compute $\mathsf{thLHE.Enc}(b - \langle \mathbf{a}, (\mathsf{sk})_i \rangle) = \mathsf{thLHE.Enc}((\varphi)_i)$ and reconstruct the value $\mathsf{thLHE.Enc}(\varphi)$. Then, the parties jointly decrypt the LHE layer to retrieve the value $\varphi = \Delta m + e$. Note that the noise term e leaks no private information, as the LWE sample is sanitized before threshold decryption. Finally, the parties complete the LWE decryption by computing $\left\lfloor \frac{\varphi}{\Delta} \right\rceil$ to retrieve the message m. By following this algorithm, we avoid noise smudging while preventing an adversary from exploiting knowledge of e thanks to sanitization and the encryption of b under the LHE. We emphasize that our hybrid construction is generic and may be instantiated with any suitable threshold LWE and LHE schemes. However, for clarity, we first instantiate it using Tiresias, a threshold variant of Paillier. We adopt threshold Paillier as the primary building block because it is a standard, widely studied, and frequently deployed threshold LHE scheme. We then prove the security of our construction under the Static-corruption-Selective-query-IND-CPA model [13]. Subsequently, we propose concrete instantiations for both the LWE sanitization and the threshold LWE layer with TFHE. Our choice of threshold TFHE as the underlying scheme is motivated by its LWE-based structure and the fact that standard TFHE parameters provide an insufficient noise margin to accommodate smudging noise. Finally, we provide execution timings for our threshold decryption and discuss its overall efficiency.

Paper organization. This paper is organized as follows: Sect. 2 briefly surveys existing literature on MPC-based message recovery, as well as threshold FHE and Paillier schemes. Sect. 3 provides the necessary background on threshold LWE and LHE, establishing notation used throughout our work. In Sect. 4, we introduce security notions for threshold FHE schemes. In Sect. 5, we detail our core construction; this Sect. also includes a formal security analysis of both the Tiresias scheme [14] and our proposed protocol. Sect. 6.1 suggests concrete instantiations for the LWE sanitization step and the threshold LWE

layer, focusing on threshold TFHE, while Sect. 6.2 reports execution timings and performance benchmarks, and concludes the paper.

2 Related works

Threshold LWE-based FHE. In [15] Boneh et al. developed a general approach for adding threshold functionality to a large class of non-threshold schemes. They constructed a ThFHE scheme based on the LWE problem and introduced the concept of a *universal thresholdizer*. Since then, numerous (R)LWE-based constructions have been introduced [10, 16–20]. In most of these constructions, threshold decryption follows the noise flooding principle, which, as already highlighted, results in larger parameters for RLWE-based schemes such as BFV, BGV or CKKS and is difficult to adapt to TFHE due to the smaller moduli [21]. In a recent instantiation for TFHE [10], Bootland et al. presented mechanisms for robust threshold key generation and decryption for FHE schemes such as BGV, BFV and TFHE in the presence of malicious adversaries.

MPC message recovery. In [10], Bootland et al. perform threshold TFHE decryption via a generic MPC by applying *bit-extraction* techniques to extract the message m from the shares $(\varphi)_i$. In [11], Zyskind et al. propose a threshold FHE decryption protocol, where instead of masking the underlying ciphertext noise, the authors remove it via an MPC-based *rounding*.

Server-aided approach. Recently, Passelègue and Stehlé [22] introduced a server-aided approach for threshold FHE. In a nutshell, the syntax of ThFHE is extended by introducing an uncorrupted party, generally assumed to be a server that performs the homomorphic computation, that post-processes evaluated ciphertexts *before* the decrypting parties run the distributed decryption. This post-processing does not have to maintain any homomorphic properties. In exchange, it becomes possible in RLWE-based ThFHE to leverage this post-processing in order to optimize the amount of noise added during partial decryption [22]. Then, [23] adapts this approach to TFHE (by combining sanitization and small noise "flooding"). In some sense, the approach in this paper is a refinement of [23], where we completely remove the need for noise flooding.

Threshold Paillier. Motivated by applications in electronic voting, several threshold variants of the Paillier scheme were proposed [24–27], with all of them relying on a trusted dealer for the key setup. To address the need for a threshold Paillier with trustless distributed key generation and a robust proof system under standard assumptions, Friedman et al. introduced Tiresias [14]. Additionally, they establish that it is secure under the DCR assumption alone, the same assumption underlying standard Paillier CPA security.

3 Preliminaries

Let λ and τ respectively represent the computational and statistical security parameters. In the upcoming sections, we denote vectors by bold letters, so a vector $\mathbf{x}$ of n elements is $\mathbf{x} = (x_0, \ldots, x_{n-1})$. The inner product of two vectors

$\mathbf{x}$ and $\mathbf{y}$ is $\langle \mathbf{x}, \mathbf{y} \rangle$. We write $[\![0, K-1]\!]$ for $\{0, 1, \ldots, K-1\}$ and let $\boxplus$ and $\boxtimes$ be homomorphic addition and multiplication.

3.1 Zero-knowledge protocol

In a zero-knowledge protocol, a prover convinces a verifier of some fact regarding a public statement without revealing any underlying information beyond the truth of this fact. It must meet three criteria: completeness ensures that valid proofs are always accepted by honest verifier; soundness guarantees that valid proofs cannot be produced for false claims to deceive the verifier; and zero-knowledge means that the verifier learns nothing about the instance except the truth of the fact. Additionally, Special Honest Verifier Zero-Knowledge (SHVZK) ensures that no private information is leaked if the verifier acts honestly. This property implies that a simulator can recreate a convincing transcript without knowing the witness. For the formal definitions, we refer the reader to [14].

3.2 Sanitization

The goal of sanitization is to remove all information conveyed by the ciphertext ct except for the message m. Following [28], a sanitization algorithm $\mathsf{Sanitize}$ is sanitizing if there exists a polynomial time simulator Sim such that for any $m \in \mathcal{M}$, any ciphertext ct that encrypts m, it holds that $\big(\mathsf{Sim}(1^\lambda, m, \mathsf{pk}), \mathsf{sk}\big) \approx_s (\mathsf{Sanitize}(\mathsf{pk}, \mathsf{ct}), \mathsf{sk})$ for an honestly sampled pk.

In [29], Ducas and Stehlé proposed a sanitization solution based on repeatedly applying the addition of moderately sized noise and performing bootstrapping. The idea is that through the successive injection of noise into two ciphertexts, their statistical distance decreases, and after a sufficient number of iterations, the ciphertexts become statistically indistinguishable. Then in [28], Bourse and Izabachène, and in [30], Kluczniak proposed a solution that addresses the overhead of repeated bootstrapping in [29] by randomizing the entire TFHE bootstrapping procedure to achieve sanitization with only a single execution of the bootstrapping. However, the added randomization overhead results in poorer practical performance than the strategy from [29], mainly due to the costly use of discrete Gaussian sampling. To address the aforementioned limitations, in [31] Hwang et al. construct a TFHE sanitization algorithm that requires only a single invocation of bootstrapping and introduces randomization only at the beginning and end of the bootstrapping, rather than randomizing the entire process.

3.3 Threshold schemes

Let $\mathbb{A}_{T,K}$ denote a (T, K)-threshold access structure [13]. Let $\mathcal{M}$ and $\mathcal{C}$ denote plaintext and ciphertext spaces of a homomorphic scheme. In this paper, we partition $\mathcal{C}$ into two disjoint sets, $\mathcal{C} = \mathcal{C}_F \cup \mathcal{C}_E$, where $\mathcal{C}_F$ and $\mathcal{C}_E$ represent fresh and evaluated ciphertexts, respectively (for reasons that will become obvious, we will further always work under the assumption that $\mathcal{C}_F \cap \mathcal{C}_E = \emptyset$). Let SS

be a linear secret sharing scheme, for example, Shamir Secret Sharing (SSS) introduced by [32]. In the following, we formally define the two core primitives used in this paper: a general threshold LWE-based FHE scheme (thLWE) and a general threshold LHE scheme (thLHE).

Definition 1 (Threshold LWE-based FHE). *A threshold T-out-of-K public-key LWE-based encryption scheme for an access structure $\mathbb{A}_{T,K}$ over K parties, with plaintext space $\mathcal{M}$ and ciphertext space $\mathcal{C}$, consists of the following PPT algorithms* $\mathsf{thLWE} = (\mathsf{DistKeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Sanitize}, \mathsf{DistDec})$*:*

- $\mathsf{DistKeyGen}(1^\lambda, \mathbb{A}_{T,K}; \mathcal{P}) \rightarrow (\mathsf{pk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{K}})$*: the parties* $\mathcal{P} = \{P_i\}_{i\in\mathcal{K}} \in \mathbb{A}_{T,K}$ *execute a distributed key generation that returns a public key* pk *(containing* $\{\rho_i\}_{i\in\mathcal{K}}$ *and an evaluation key* evk*) and a set of K secret shares of the LWE secret key* sk*, denoted* $\{(\mathsf{sk})_i\}_{i\in\mathcal{K}}$*, where each party P_i receives only its corresponding share* $(\mathsf{sk})_i$ *generated via* $\mathsf{SS.Share}_{T,K}(\mathsf{sk}; r) \rightarrow \{\rho_i, (\mathsf{sk})_i\}_{i\in\mathcal{K}}$ *for randomness r.*
- $\mathsf{Enc}(\mathsf{pk}, m) \rightarrow \mathsf{ct} \in \mathcal{C}_F$*: using the public key, the encryption algorithm returns a ciphertext of the form* $\mathsf{ct} = (\mathbf{a}, b = \langle \mathbf{a}, \mathsf{sk}\rangle + \Delta m + e)$*, where e depends on the structure of the public key*[1]*, as a fresh LWE encryption of $m \in \mathcal{M}$, where $\mathbf{a} \in \mathbb{Z}_q^n$ and $b \in \mathbb{Z}_q$. Here, $\Delta = q/t$ is the scaling factor, q and t are the ciphertext and plaintext moduli, and n denotes the LWE dimension.*
- $\mathsf{Sanitize}(\mathsf{pk}, \mathsf{ct}) \rightarrow \mathsf{ct}' \in \mathcal{C}_E$*: the sanitization algorithm processes an LWE ciphertext* ct *to produce a sanitized ciphertext* ct'*.*
- $\mathsf{Eval}\left(\mathsf{pk}, f, \{\mathsf{ct}_i\}_{i\in[\![0,l]\!]}\right) \rightarrow \mathsf{ct}' \in \mathcal{C}_E$*: the evaluation algorithm computes a function f on ciphertexts* $\{\mathsf{ct}_i\}_{i=0}^{l}$ *producing* $\mathsf{ct} = f(\mathsf{ct}_0, \ldots, \mathsf{ct}_l)$*, and subsequently invokes the sanitization algorithm to return* $\mathsf{ct}' \leftarrow \mathsf{Sanitize}(\mathsf{pk}, \mathsf{ct})$*.*
- $\mathsf{DistDec}\left(\mathsf{pk}, \mathcal{T}, \{(\mathsf{sk})_i\}, \mathsf{ct}; \{P_i\}_{i\in\mathcal{T}}\right) \rightarrow \left\lfloor \frac{\varphi}{\Delta} \right\rceil \in \mathcal{M}$*: the parties* $\{P_i\}_{i\in\mathcal{T}}$ *where* $\mathcal{T} \subseteq \mathcal{K}$ *execute the distributed decryption on* $\mathsf{ct} \in \mathcal{C}$ *using three sub-algorithms to recover the plaintext:*
  - $\mathsf{ShareDec}\left((\mathsf{sk})_i, \mathsf{ct}; P_i\right) \rightarrow (\varphi)_i$*: each party P_i returns a partial decryption share of* ct *by computing* $(\varphi)_i = b - \langle \mathbf{a}, (\mathsf{sk})_i\rangle = (\Delta m + e)_i$*.*
  - $\mathsf{Comb}\left(\mathsf{pk}, \{(\varphi)_i\}_{i\in\mathcal{T}}; \{P_i\}_{i\in\mathcal{T}}\right) \rightarrow \varphi$*: the parties* $\{P_i\}_{i\in\mathcal{T}}$ *jointly reconstruct* $\varphi = \Delta m + e$ *by invoking the* $\mathsf{SS.Recon}(\mathsf{pk}, \{(\varphi)_i\}_{i\in\mathcal{T}})$*.*
  - $\mathsf{FinDec}(\mathsf{pk}, \varphi) \rightarrow m$*: this step finalizes the LWE decryption, returning some $m \in \mathcal{M}$ such that* $m = \left\lfloor \frac{\varphi}{\Delta} \right\rceil$*.*

Definition 2 (Threshold LHE). *A threshold T-out-of-K LHE scheme for an access structure $\mathbb{A}_{T,K}$ on K parties, with plaintext space $\mathcal{M}$ and ciphertext space $\mathcal{C}$, consists of the following PPT algorithms* $\mathsf{thLHE} = (\mathsf{DistKeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{DistDec})$*:*

- $\mathsf{DistKeyGen}(1^\lambda, \mathbb{A}_{T,K}; \mathcal{P}) \rightarrow (\mathsf{pk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{K}})$*: the parties* $\mathcal{P} = \{P_i\}_{i\in\mathcal{K}} \in \mathbb{A}_{T,K}$ *execute a distributed key generation that returns an LHE public key* pk

[1] For example, when using the usual Regev-style public key made of $n \log_2 q$ encryptions of 0 (generated by means of the secret key), the noise e is the sum of the noises of the public-key element selected in the random subset used for encryption.

together with evaluation points $\{\rho_i\}_{i\in\mathcal{K}}$ *and a set of* K *secret shares of the LHE secret key* sk*, denoted* $\{(\mathsf{sk})_i\}_{i\in\mathcal{K}}$*, where each party* P_i *receives only its corresponding share* $(\mathsf{sk})_i$ *generated via* $\mathsf{SS.Share}_{T,K}(\mathsf{sk}; r) \to \{\rho_i, (\mathsf{sk})_i\}_{i\in\mathcal{K}}$ *for randomness* r*.*

- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct} \in \mathcal{C}_F$*: the encryption algorithm samples* $r \in \mathcal{M}^*$ *and returns* $\mathsf{ct} \in \mathcal{C}_F$ *as a fresh LHE encryption of* $m \in \mathcal{M}$ *using randomness* r*.*
- $\mathsf{Eval}\left(\mathsf{pk}, f, \{\mathsf{ct}_i\}_{i\in[\![0,l]\!]}\right) \to \mathsf{ct} \in \mathcal{C}_E$*: the evaluation algorithm computes an affine function* f *on ciphertexts* $\{\mathsf{ct}_i\}_{i=0}^{l}$ *producing* $\mathsf{ct} = f(\mathsf{ct}_0, \ldots, \mathsf{ct}_l)$ *by consequently applying*
  - $\mathsf{Add}(\mathsf{ct}_0, \mathsf{ct}_1) \to \mathsf{ct}' \in \mathcal{C}_E$*: it adds* $\mathsf{ct}_0, \mathsf{ct}_1 \in \mathcal{C}$ *returning* $\mathsf{ct}' = \mathsf{ct}_0 \boxplus \mathsf{ct}_1$*.*
  - $\mathsf{cAdd}(\mathsf{ct}, x) \to \mathsf{ct}' \in \mathcal{C}_E$*: it adds* $\mathsf{ct} \in \mathcal{C}$ *and* $x \in \mathcal{M}$ *returning* $\mathsf{ct}' = \mathsf{ct} \boxplus x$*.*
  - $\mathsf{cMult}(\mathsf{ct}, x) \to \mathsf{ct}' \in \mathcal{C}_E$*: it multiplies* $\mathsf{ct} \in \mathcal{C}$ *and* $x \in \mathcal{M}$ *returning* $\mathsf{ct}' = \mathsf{ct} \boxtimes x$*.*
- $\mathsf{DistDec}\left(\mathsf{pk}, \mathcal{T}, \{(\mathsf{sk})_i\}, \mathsf{ct}; \{P_i\}_{i\in\mathcal{T}}\right) \to m \in \mathcal{M}$*: the parties* $\{P_i\}_{i\in\mathcal{T}}$ *where* $\mathcal{T} \subseteq \mathcal{K}$ *execute the distributed decryption on* $\mathsf{ct} \in \mathcal{C}$ *using three sub-algorithms to recover the plaintext:*
  - $\mathsf{ShareDec}((\mathsf{sk})_i, \mathsf{ct}; P_i) \to (\mathsf{pd})_i$*: each party* P_i *returns a partial decryption share* $(\mathsf{pd})_i$ *of* ct*.*
  - $\mathsf{Comb}(\mathsf{pk}, \{(\mathsf{pd})_i\}; \{P_i\}_{i\in\mathcal{T}}) \to \mathsf{c}$*: the parties* $\{P_i\}_{i\in\mathcal{T}}$ *jointly reconstruct* c *by invoking the* $\mathsf{SS.Recon}(\mathsf{pk}, \{(\mathsf{pd})_i\})$*.*
  - $\mathsf{FinDec}(\mathsf{pk}, \mathsf{c}) \to m$*: this step finalizes the decryption by returning some* $m \in \mathcal{M}$ *from the reconstructed value* c*.*

In this paper, we require thLHE *schemes to be perfectly correct, i.e. for any* $\mathbb{A}_{T,K}$*, sets* $\mathcal{P} = \{P_i\}_{i\in\mathcal{K}} \in \mathbb{A}_{T,K}$ *and* $\{P_i\}_{i\in\mathcal{T}} \in \mathbb{A}_{T,K}$ *where* $\mathcal{T} \subseteq \mathcal{K}$*,* $m_0, \ldots, m_l \in \mathcal{M}$ *and* $(\mathsf{pk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{K}}) \in \mathsf{DistKeyGen}(1^\lambda, T, K; \mathcal{P})$*, it satisfies*

$$\Pr\left[m' = f(m_0, \ldots, m_l) \;\middle|\; \begin{array}{l} \mathsf{ct}_j \leftarrow \mathsf{Enc}(\mathsf{pk}, m_j) \text{ for } j \in [\![0, l]\!] \\ \mathsf{ct} \leftarrow \mathsf{Eval}\left(\mathsf{pk}, f, \{\mathsf{ct}_j\}_{j\in[\![0,l]\!]}\right) \\ m' \leftarrow \mathsf{DistDec}(\mathsf{pk}, \mathcal{T}, \{(\mathsf{sk})_i\}, \mathsf{ct}; \{P_i\}_{i\in\mathcal{T}}) \end{array}\right] = 1,$$

with the probability taken encryption randomness.

4 Security notions

In this paper, we assume the asynchronous setting, where any authorized subset of T parties can decrypt without knowing in advance which T decryptors will contribute. This is in contrast with the synchronous setting, where the subset of T parties is known at the outset. ThFHE security in the synchronous setting as recently been addressed in [33].

In the standard CPA game, the adversary A is only granted access to encryption and challenge oracles, while in the threshold setting, A additionally obtains the secret key shares of corrupted parties and naturally sees the associated partial decryptions. This is modeled by providing A with a partial decryption oracle, which answers two types of partial decryption queries: (I) queries on non-challenge ciphertexts (i.e. those not depending on the challenge bit); (II) queries

on challenge ciphertexts (i.e. whose underlying plaintexts depend on the challenge bit). Still, even in the CPA setting, corruptions and query timing must be considered. Corruption timing determines whether an adversary is restricted to static corruption (Scor), where the corrupted set is fixed upfront, or can adaptively corrupt (AdpCor) parties during execution up to a maximal threshold $T + 1$ (MaxCor). Similarly, query timing defines whether the adversary must selectively declare (Sel) its game requests before the protocol begins or can issue them adaptively (Adp) based on the ongoing interaction. Also, we highlight that *LWE-based FHE presently offers no practical alternative for achieving CCA1 (or beyond) in the single-key setting, even based on heuristics*. Although some theoretical works indicate that this may be possible, to the best of our knowledge, all present LWE-based CCA-secure FHE constructions (e.g. [34]) are only of theoretical interest.

In the following, we adapt the recent definitions for IND-CPA and SIM-CPA security notions for Threshold Public Key Encryption (TPKE) schemes proposed in [13, Sect. 2.1] to cope with FHE schemes. Our adaptation essentially consists in providing the adversary with an evaluation oracle (over well-formed ciphertexts) such that it can also ask for partial decryption of well-formed evaluated ciphertexts and not only fresh ones. In spirit, the security games below are generalizations of the CPA^D games (for both the IND and SIM notions) introduced by Li&Micciancio [35] to the threshold setting. In particular, as in [35], we make explicit the challenger game state and how it is handled. Note that, as in CPA^D [35, Def. 2], the encryption oracle is necessary in both the private and public-key setting in order to fill the challenger state with well-formed ciphertexts.

Our game model assumes a multi-challenge setting with adaptive (Adp) queries (including Type II partial decryption queries) and considers the adversary A that statically corrupts (Scor) a subset of parties. We permit adaptive queries because selective security is considered too weak to be a reasonable target notion, as it requires the adversary to declare its queries upfront. In contrast, allowing adaptive queries ensures the model captures more realistic adversarial behaviours, where A's strategy can evolve based on previously observed oracles' responses. Furthermore, we allow Type II queries; security models that exclude these are inherently weaker for TPKE applications, where the adversary may observe partial decryptions related to the challenge ciphertext. Eventually, we adopt a static corruption model because, as demonstrated in [13, Thm. 3.6], simulation security with adaptive corruption is not achievable, assuming adaptive partial decryption queries. This choice is further motivated by Sect. 5.2, where we establish the security of our construction through a reduction to the simulation-based security of the threshold Paillier scheme.

Definition 3 (Scor-Adp-IND-CPA for ThFHE). *Given a threshold FHE scheme* $\mathsf{thFHE} = (\mathsf{DistKeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{DistDec})$ *with plaintext space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$*, we define a security game for an adversary* A *and a security parameter* λ*. The game is parameterized by a bit* $\gamma \leftarrow\!\!\$\, \{0, 1\}$*, unknown to* A*, and an initially empty state* Ω *consisting of message-message-ciphertext-index*

$(m_0, m_1, \mathsf{ct}, \mathsf{ind})$ *quadruples. At the beginning of the game,* A *statically corrupts a set of parties* $\mathcal{S} = \{P_i\}_{i\in\mathcal{T}}$ *for some* $\mathcal{T} \subset \mathcal{K}$*:*

- *Key generation: given* $\mathcal{P} = \{P_i\}_{i\in\mathcal{K}}$*, check* $(\mathcal{S} \subset \mathcal{P}) \wedge (\mathcal{S} \notin \mathbb{A}_{T,K})$*. If* True*, run* $(\mathsf{pk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{K}}) \leftarrow \mathsf{thFHE.DistKeyGen}(1^\lambda, \mathbb{A}_{T,K})$ *and return to* A *a public key* pk *and a set* $\{(\mathsf{sk})_i\}_{i\in\mathcal{T}}$*.*
- *Encryption request: when* A *queries* $(\mathsf{plaintext}, m)$*, where* $m \in \mathcal{M}$*, compute* $\mathsf{ct} \leftarrow \mathsf{thFHE.Enc}(\mathsf{pk}, m)$*, return* ct *to* A *and do* $\Omega := [\Omega; (m, m, \mathsf{ct}, \{\emptyset\})]$*.*
- *Challenge request: when* A *queries* $(\mathsf{test}, m_0, m_1)$*, where* $m_0, m_1 \in \mathcal{M}$*,* $m_0 \neq m_1$*, get* $\mathsf{ct} \leftarrow \mathsf{thFHE.Enc}(\mathsf{pk}, m_\gamma)$ *and return* ct *to* A *after updating* $\Omega := [\Omega; (m_0, m_1, \mathsf{ct}, \{\emptyset\})]$*.*
- *Evaluation request: when* A *queries* $(\mathsf{eval}, f, \ell_0, \ldots, \ell_v)$*,* $\forall i$*:* $\ell_i \in [\![0, |\Omega| - 1]\!]$*, set* $m'_0 = f(\Omega[\ell_0].m_0, \ldots, \Omega[\ell_v].m_0)$*,* $m'_1 = f(\Omega[\ell_0].m_1, \ldots, \Omega[\ell_v].m_1)$ *and* $\mathsf{ct}' \leftarrow \mathsf{thFHE.Eval}(\mathsf{pk}, f, \Omega[\ell_0].\mathsf{ct}, \ldots, \Omega[\ell_v].\mathsf{ct})$*, then return* ct' *to* A *after updating* $\Omega := [\Omega; (m'_0, m'_1, \mathsf{ct}', \{\emptyset\})]$*.*
- *Partial decryption request: when* A *queries* $(\mathsf{ciphertext}, \ell, j)$*,* $\ell \in [\![0, |\Omega| - 1]\!]$*,* $j \in \mathcal{K}$*, if* $\Omega[\ell].m_0 = \Omega[\ell].m_1$*,* $(\mathsf{pd})_j \leftarrow \mathsf{thFHE.ShareDec}((\mathsf{sk})_j, \Omega[\ell].\mathsf{ct})$*, and return* $(\mathsf{pd})_j$ *to* A*. If* $\Omega[\ell].m_0 \neq \Omega[\ell].m_1$*, do* $\Omega[\ell].\mathsf{ind} := [\Omega[\ell].\mathsf{ind}; \{j\}]$ *and check* $(\{P_i\}_{i\in\Omega[\ell].\mathsf{ind}} \cup \mathcal{S}) \notin \mathbb{A}_{T,K}$*; if* True*, compute* $(\mathsf{pd})_j$ *and return it to* A*.*
- *Guessing stage (after polynomially many interleaved encryption, challenge, evaluation and partial decryption requests): when* A *outputs* $(\mathsf{guess}, \gamma')$*, if* $\gamma' = \gamma$ *then* A *wins the game. Otherwise,* A *loses the game.*

In addition, we give a definition of the associated (strictly stronger [13, Prop. 3.2]) simulation-based notion that we also use later in the paper.

Definition 4 (Scor-Adp-SIM-CPA for ThFHE). *Given a threshold FHE scheme* $\mathsf{thFHE} = (\mathsf{DistKeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{DistDec})$ *with plaintext space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$*, we define a security game for an adversary* A *and a security parameter* λ*. Let* Sim *be a PPT (stateful) simulator, which maintains its own state* ω*. The game is parameterized by a bit* $\gamma \leftarrow\!\!\$ \{0, 1\}$*, unknown to* A*, and an initially empty state* Ω *consisting of message-message-ciphertext-index* $(m_0, m_1, \mathsf{ct}, \mathsf{ind})$ *quadruples. At the beginning of the game,* A *statically corrupts a set of parties* $\mathcal{S} = \{P_i\}_{i\in\mathcal{T}}$ *for some* $\mathcal{T} \subset \mathcal{K}$*:*

- *Key generation: given* $\mathcal{P} = \{P_i\}_{i\in\mathcal{K}}$*, check* $(\mathcal{S} \subset \mathcal{P}) \wedge (\mathcal{S} \notin \mathbb{A}_{T,K})$*. If* True*, then proceed as*
  - *if* $\gamma = 0$*, run* $(\mathsf{pk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{K}}) \leftarrow \mathsf{thFHE.DistKeyGen}(1^\lambda, \mathbb{A}_{T,K})$*,*
  - *if* $\gamma = 1$*, run* $(\mathsf{pk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{T}}, \omega) \leftarrow \mathsf{Sim}(1^\lambda, \mathbb{A}_{T,K}, \mathcal{S})$*,*

  return to A *a public key* pk *and a set* $\{(\mathsf{sk})_i\}_{i\in\mathcal{T}}$*.*
- *Encryption request: when* A *queries* $(\mathsf{plaintext}, m)$*, where* $m \in \mathcal{M}$*,*
  - *if* $\gamma = 0$*, compute* $\mathsf{ct} \leftarrow \mathsf{thFHE.Enc}(\mathsf{pk}, m)$*,*
  - *if* $\gamma = 1$*, compute* $\mathsf{ct} \leftarrow \mathsf{Sim}(m, \omega)$*,*

  return ct *to* A *and update* $\Omega := [\Omega; (m, m, \mathsf{ct}, \{\emptyset\})]$*.*
- *Challenge request: when* A *queries* (test, m)*, where* $m \in \mathcal{M}$*,*
  - *if* $\gamma = 0$*, compute* $\mathsf{ct} \leftarrow \mathsf{thFHE.Enc}(\mathsf{pk}, m)$*,*

· *if* $\gamma = 1$, *compute* $\mathsf{ct} \leftarrow \mathsf{Sim}(\omega)$,

pick a random $m' \in \mathcal{M}$, *return* ct *to* A *after updating* $\Omega := [\Omega; (m, m', \mathsf{ct}, \{\emptyset\})]$.

- *Evaluation request: when* A *queries* $(\mathsf{eval}, f, \ell_0, \ldots, \ell_v)$, $\forall i$*:* $\ell_i \in [\![0, |\Omega| - 1]\!]$, *set* $m'_0 = f(\Omega[\ell_0].m_0, \ldots, \Omega[\ell_v].m_0)$, $m'_1 = f(\Omega[\ell_0].m_1, \ldots, \Omega[\ell_v].m_1)$ *and compute* $\mathsf{ct}' \leftarrow \mathsf{thFHE.Eval}(\mathsf{pk}, f, \Omega[\ell_0].\mathsf{ct}, \ldots, \Omega[\ell_v].\mathsf{ct})$, *then return* ct' *to* A *after updating* $\Omega := [\Omega; (m'_0, m'_1, \mathsf{ct}', \{\emptyset\})]$.
- *Partial decryption request: when* A *queries* $(\mathsf{ciphertext}, \ell, j)$, $\ell \in [\![0, |\Omega| - 1]\!]$, $j \in \mathcal{K}$,
  - *if* $\gamma = 0$, *compute* $(\mathsf{pd})_j \leftarrow \mathsf{thFHE.ShareDec}((\mathsf{sk})_j, \Omega[\ell].\mathsf{ct})$,
  - *if* $\gamma = 1$, *update* $\Omega[\ell].\mathsf{ind} := [\Omega[\ell].\mathsf{ind}; \{j\}]$ *and check* $(\Omega[\ell].m_0 \neq \Omega[\ell].m_1) \wedge \big((\{P_i\}_{i \in \Omega[\ell].\mathsf{ind}} \cup \mathcal{S}) \in \mathbb{A}_{T,K}\big)$*; if* True, $(\mathsf{pd})_j \leftarrow \mathsf{Sim}(\Omega[\ell].\mathsf{ct}, \Omega[\ell].m_0, j, \omega)$*; else,* $(\mathsf{pd})_j \leftarrow \mathsf{Sim}(\Omega[\ell].\mathsf{ct}, j, \omega)$,

  then return $(\mathsf{pd})_j$ *to* A.
- *Guessing stage (after polynomially many interleaved encryption, challenge, evaluation and partial decryption requests): when* A *outputs* $(\mathsf{guess}, \gamma')$, *if* $\gamma' = \gamma$, *then* A *wins the game. Otherwise,* A *loses the game.*

5 Threshold LWE without smudging

In this paper, we leverage the *server-aided* framework of Passelègue and Stehlé [22] (Sect. 2) that extends the definition of ThFHE by introducing an uncorrupted party (modelled as a server) that post-processes evaluated LWE ciphertexts before the decryption parties participate. Consequently, the parties execute the threshold decryption algorithm on the post-processed LWE ciphertext provided by the server, following homomorphic evaluation.

We now present our LWE without smudging construction, employing instantiations of threshold LWE and LHE layers. Let $\mathsf{thLWE} = (\mathsf{DistKeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Sanitize}, \mathsf{DistDec})$ and $\mathsf{thLHE} = (\mathsf{DistKeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{DistDec})$ denote generic T_{L}-out-of-K threshold LWE-based FHE (Def. 1) and T_{P}-out-of-K threshold LHE[2] (taking threshold Paillier as a concrete instance, we follow the construction in Def. 2). We define the plaintext/ciphertext moduli as (t, q) for thLWE and (μ, μ^2) for thLHE, where μ is an RSA-style composite modulus. Our core strategy is to shield the LWE b-term from the decryption parties by keeping it encrypted under thLHE throughout the distributed decryption. After sanitizing an LWE sample $(\mathbf{a}, b)$, the server S computes $(\mathbf{a}, \mathsf{thLHE.Enc}(b))$ and distributes it to the parties. Each party P_i homomorphically computes a partial decryption share $\mathsf{thLHE.Enc}(b - \langle \mathbf{a}, (\mathbf{s})_i \rangle) = \mathsf{thLHE.Enc}((\varphi)_i)$. Then the parties jointly reconstruct φ under thLHE, getting $\mathsf{thLHE.Enc}(\varphi)$, and decrypt the thLHE layer to recover the value φ. Finally, they retrieve the message by computing $\lfloor \varphi / \Delta \rceil$. Although φ may allow the parties to obtain the ciphertext noise, the prior sanitization of the LWE sample ensures circuit privacy.

Let $\mathcal{P} = \{P_i\}_{i \in \mathcal{K}}$ be the set of participating parties and S the associated server. We define the access structures as $\mathbb{A}_{T,K} = (\mathbb{A}_{T_{\mathsf{L}},K}, \mathbb{A}_{T_{\mathsf{P}},K})$, where $\mathcal{T}_{\mathsf{L}}, \mathcal{T}_{\mathsf{P}} \subseteq$

[2] We note that any other LHE scheme, which satisfies Def. 2, can be used as well.

$\mathcal{K}$ denote the respective index sets of parties participating in the partial decryptions of thLWE and thLHE. Our construction thPLWE = (DistKeyGen, Enc, Eval, PartDec, DistDec) is formally defined as follows:

- $\mathsf{DistKeyGen}(1^\lambda, \mathbb{A}_{T,K}; \mathcal{P}) \to (\mathsf{thLWE.(pk}, \{(\mathsf{sk})_i\}), \mathsf{thLHE.(pk}, \{(\mathsf{sk})_i\}))$: The algorithm, run by $\mathcal{P}$ executes

$$(\mathsf{thLWE.pk}, \{(\mathsf{thLWE.sk})_i\}) \leftarrow \mathsf{thLWE.DistKeyGen}(1^\lambda, \mathbb{A}_{T_\mathsf{L},K}; \mathcal{P}),$$
$$(\mathsf{thLHE.pk}, \{(\mathsf{thLHE.sk})_i\}) \leftarrow \mathsf{thLHE.DistKeyGen}(1^\lambda, \mathbb{A}_{T_\mathsf{P},K}; \mathcal{P}),$$

  returning thLWE, thLHE public keys thLWE.pk, thLHE.pk, and two tuples of K secret shares of thLWE.sk, thLHE.sk, i.e. $\{(\mathsf{thLWE.sk})_i\}$, $\{(\mathsf{thLHE.sk})_i\}$.
- $\mathsf{Enc}(\mathsf{thLWE.pk}, m) \to \mathsf{ct}$: encrypts $m \in \mathbb{Z}_t$ as $\mathsf{ct} = (\mathbf{a}, b = \langle \mathbf{a}, \mathsf{thLWE.sk} \rangle + \Delta m + e) \leftarrow \mathsf{thLWE.Enc}(\mathsf{thLWE.pk}, m)$, where $\mathbf{a} \in \mathbb{Z}_q^n$, $b \in \mathbb{Z}_q$ and $\Delta = \frac{q}{t}$.
- $\mathsf{Eval}(\mathsf{thLWE.pk}, f, \mathsf{ct}_0, \ldots, \mathsf{ct}_l; \mathsf{S})$: evaluates a function f on $\{\mathsf{ct}_i\}_{i \in [\![0,l]\!]}$. It computes $\mathsf{ct} \leftarrow \mathsf{thLWE.Eval}(\mathsf{thLWE.pk}, f, \mathsf{ct}_0, \ldots, \mathsf{ct}_l)$ and sanitizes ct as $\mathsf{ct}' = (\mathbf{a}', b') \leftarrow \mathsf{thLWE.Sanitize}(\mathsf{thLWE.pk}, \mathsf{ct})$. It finally does $p \leftarrow \mathsf{thLHE.Enc}(\mathsf{thLHE.pk}, b')$ and outputs $\mathsf{ct}'' = (\mathbf{a}', p)$.
- $\mathsf{PartDec}(\mathsf{thLWE.pk}, \{(\mathsf{thLWE.sk})_i\}, \mathsf{ct}''; \{P_i\}_{i \in \mathcal{T}_\mathsf{L}}) \to \mathsf{thLHE.Enc}(\varphi)$: each party P_i for $i \in \mathcal{T}_\mathsf{L}$ homomorphically computes $\varphi = b - \langle \mathbf{a}, \mathsf{sk} \rangle = \Delta m + e$ on ct'' under thLHE returning $\mathsf{thLHE.Enc}((\varphi)_i) = \mathsf{thLHE.cAdd}(p, [\langle -\mathbf{a}', (\mathsf{thLWE.sk})_i \rangle]_q)$. Then the parties $\{P_i\}_{i \in \mathcal{T}_\mathsf{L}}$ execute thLWE.Comb on $\mathsf{thLHE.Enc}((\varphi)_i)$ by homomorphically running $\mathsf{SS.Recon}(\mathsf{thLWE.pk}, \{(\varphi)_i\}_{i \in \mathcal{T}_\mathsf{L}})$ under thLHE. They compute the Lagrange coefficients $\lambda_i \in \mathbb{Z}_q$ and return[3]

$$\mathsf{thLHE.Enc}(\varphi) = \bigoplus_{i=0}^{|\mathcal{T}_\mathsf{L}|-1} \left(\mathsf{thLHE.cMult}\left(\mathsf{thLHE.Enc}((\varphi)_i), \lambda_i\right)\right),$$

  where $\oplus$ defines thLHE.Add.
- $\mathsf{DistDec}(\mathsf{thLHE.pk}, \{(\mathsf{thLHE.sk})_i\}, \mathsf{thLHE.Enc}(\varphi); \{P_i\}_{i \in \mathcal{T}_\mathsf{P}}) \to m$: the parties $\{P_i\}_{i \in \mathcal{T}_\mathsf{P}}$ compute $(\mathsf{pd})_i \leftarrow \mathsf{thLHE.ShareDec}((\mathsf{thLHE.sk})_i, \mathsf{thLHE.Enc}(\varphi))$. By combining $T_\mathsf{P} + 1$ valid partial decryption shares $(\mathsf{pd})_i$, a designated party obtains $\mathsf{c} \leftarrow \mathsf{thLHE.Comb}(\mathsf{thLHE.pk}, \{(\mathsf{pd})_i\})$. The decryption is finalized by $\varphi = \Delta m + e \leftarrow \mathsf{thLHE.FinDec}(\mathsf{thLHE.pk}, \mathsf{c})$, $m \leftarrow \mathsf{thLWE.FinDec}(\mathsf{thLWE.pk}, \varphi)$.

Remark that fresh ciphertexts can also be decrypted, as long as they are also post-processed by the server. In this case, there is no need for sanitization, as the fresh ciphertext noise is inherently independent of the message. Also, the above construction uses the schemes in Defs. 1 and 2 as building blocks and is not an instantiation of any of them. Thus, it can adopt a modified syntax: during thPLWE.PartDec, the decrypting parties homomorphically

[3] Note that the correctness of φ reconstruction under thLHE is preserved. The operation is linear and $q \ll \mu$ (with $\log_2 q = 32$ and $\log_2 \mu = 2048$ as typical values), meaning no modulo μ reduction occurs under thLHE. By ensuring that the evaluation points are chosen from an Exceptional Set in $\mathbb{Z}_q$ [10], it is guaranteed that $\lambda_i \in \mathbb{Z}_q$. Thus, in the worst case, it requires $\lceil \log_2(2q-1) \rceil$ bits to store $(\varphi)_i$ and $\lceil \log_2(|\mathcal{T}_\mathsf{L}|) + 2\log_2(q) + 1 \rceil$ bits to store $\mathsf{thLHE.Enc}(\varphi)$.

compute $\mathsf{thLHE.Enc}((\varphi)_i)$ and jointly reconstruct φ under thLHE. Then, by running $\mathsf{thPLWE.DistDec}$, they remove the thLHE layer to recover φ and retrieve the message.

5.1 Tiresias threshold simulation-based security (Scor-Adp-SIM-CPA)

In this section, we prove the Scor-Adp-SIM-CPA security of Tiresias (App. A). This will later allow us to assume that Tiresias is Scor-Adp-IND-CPA secure (since the former implies the latter from [13, Prop. 3.2]); however, the proof is more easily done in the simulation-based setting. We recall that if the zero-knowledge property (Sect. 3.1) does not hold, the verifier learns information about the witness beyond what is available from the statement and proof. This implies the verifier can distinguish real proofs from simulated ones with a non-negligible advantage. Moving forward, we assume the existence of two PPT simulators for the Tiresias scheme: $\mathsf{Sim}_{\text{DKG}}$ for distributed key generation [14, Sect. F.5] and $\mathsf{Sim}_{\text{ZKP}}$ for the zero-knowledge proofs [14, Sect. 4.3].

Proposition 5. *Tiresias is Scor-Adp-SIM-CPA secure (static corruption, Scor, adaptive queries, Adp, and simulation chosen plaintext) under the zero-knowledge property of Tiresias ZKP.*

Proof. The simulation game is parameterized by a secret bit $\gamma \leftarrow\!\!\$\, \{0,1\}$, unknown to A, and an initially empty state Ω of message-message-ciphertext-index $(m_0, m_1, \mathsf{ct}, \mathsf{ind})$ quadruples managed by the challenger C.

Game 0. This is the original Scor-Adp-SIM-CPA game against Tiresias where encryption, challenge, evaluation and partial decryption requests are processed as in Def. 4. As a reminder, the key generation is handled as follows:

- The adversary A selects a set of corrupted parties $\mathcal{S}$ and communicates it to the challenger C on the Scor-Adp-SIM-CPA of Tiresias. After verifying that $\mathcal{S} \subset \mathcal{P}$ and $\mathcal{S} \notin \mathbb{A}_{T,K}$, C chooses a numbering for the corrupted parties by $\mathcal{T} \subset \mathcal{K}$ and performs key generation
  - if $\gamma = 0$, it runs $(\mathsf{pk}, \mathsf{vk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{K}}) \leftarrow \mathsf{Tir.DistKeyGen}(1^\lambda, \mathbb{A}_{T,K})$;
  - if $\gamma = 1$, it runs $(\mathsf{pk}, \mathsf{vk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{T}}) \leftarrow \mathsf{Sim}_{\text{DKG}}(1^\lambda, \mathbb{A}_{T,K}, \mathcal{S})$.

  Finally, C returns $(\mathsf{pk}, \mathsf{vk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{T}})$ to A.

Game 1. This game is the same as Game 0, except for the way C handles key generation. The adversary A selects a set of corrupted parties $\mathcal{S}$ and communicates it to C. After verifying that $\mathcal{S} \subset \mathcal{P}$ and $\mathcal{S} \notin \mathbb{A}_{T,K}$, C runs $(\mathsf{pk}, \mathsf{vk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{K}}) \leftarrow \mathsf{Tir.DistKeyGen}(1^\lambda, \mathbb{A}_{T,K})$ and returns $(\mathsf{pk}, \mathsf{vk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{T}})$ to A.
Game 0 and Game 1 are indistinguishable, which follows directly from the simulation indistinguishability of Tir keys. Hence, $|\mathsf{Adv}^{G_1}_{\mathsf{Tir}}(\mathsf{A}) - \mathsf{Adv}^{G_0}_{\mathsf{Tir}}(\mathsf{A})| \leq \mathsf{neg}(\lambda)$.

Game 2. This game is the same as Game 1 except for the way C returns simulated encryptions:

– Encryption request: when A queries $(\mathsf{plaintext}, m)$, where $m \in \mathcal{M}$, return $\mathsf{ct} \leftarrow \mathsf{Tir.Enc}(\mathsf{pk}, m)$ to A after updating $\Omega := [\Omega; (m, m, \mathsf{ct}, \{\emptyset\})]$.
– Challenge request: when A queries (test, m), where $m \in \mathcal{M}$, return $\mathsf{ct} \leftarrow \mathsf{Tir.Enc}(\mathsf{pk}, m)$ to A, pick a random $m' \in \mathcal{M}$ and do $\Omega := [\Omega; (m, m', \mathsf{ct}, \{\emptyset\})]$.

Game 1 and Game 2 are computationally indistinguishable, which follows directly from the DCR assumption. Hence, $|\mathsf{Adv}^{G_2}_{\mathsf{Tir}}(\mathsf{A}) - \mathsf{Adv}^{G_1}_{\mathsf{Tir}}(\mathsf{A})| \leq \mathsf{neg}(\lambda)$. By transitivity, Game 0 is computationally indistinguishable from Game 2: $|\mathsf{Adv}^{G_2}_{\mathsf{Tir}}(\mathsf{A}) - \mathsf{Adv}^{G_0}_{\mathsf{Tir}}(\mathsf{A})| \leq \mathsf{neg}(\lambda)$.

Final reduction. We now prove that, from an adversary A against Game 2 we can build an adversary B against the zero-knowledge property of Tiresias ZKP, which uses A as a subroutine while achieving the same advantage as A.

– Key generation: the adversary A selects a set of corrupted parties $\mathcal{S}$ and communicates it to B, which forwards it to the challenger C against the zero-knowledge property of Tiresias ZKP. After verifying that $\mathcal{S} \subset \mathcal{P}$ and $\mathcal{S} \notin \mathbb{A}_{T,K}$, C runs $(\mathsf{pk}, \mathsf{vk}, \{(\mathsf{sk})_i\}_{i \in \mathcal{K}}) \leftarrow \mathsf{Tir.DistKeyGen}(1^\lambda, \mathbb{A}_{T,K})$ and returns $(\mathsf{pk}, \mathsf{vk}, \{(\mathsf{sk})_i\}_{i \in \mathcal{T}})$ to B, which provides them to A.
– When A issues an encryption request over $m \in \mathcal{M}$, B sends $\mathsf{ct} \leftarrow \mathsf{Tir.Enc}(\mathsf{pk}, m)$ to A after updating $\Omega := [\Omega; (m, m, \mathsf{ct}, \{\emptyset\})]$.
– When A issues a challenge request over $m \in \mathcal{M}$, B sends $\mathsf{ct} \leftarrow \mathsf{Tir.Enc}(\mathsf{pk}, m)$ to A. Then B picks a random $m' \in \mathcal{M}$ and updates $\Omega := [\Omega; (m, m', \mathsf{ct}, \{\emptyset\})]$.
– When A issues an evaluation request over $(f, \ell_0, \ldots, \ell_v), \forall i\colon \ell_i \in [\![0, |\Omega|-1]\!]$, B sets $m'_0 = [f(\Omega[\ell_0].m_0, \ldots, \Omega[\ell_v].m_0)]_\mu$, $m'_1 = [f(\Omega[\ell_0].m_1, \ldots, \Omega[\ell_v].m_1)]_\mu$ and computes $\mathsf{ct}' \leftarrow \mathsf{Tir.Eval}(\mathsf{pk}, f, \Omega[\ell_0].\mathsf{ct}, \ldots, \Omega[\ell_v].\mathsf{ct})$. B returns ct' to A after updating $\Omega := [\Omega; (m'_0, m'_1, \mathsf{ct}', \{\emptyset\})]$.
– When A issues a partial decryption request over $\ell \in [\![0, |\Omega| - 1]\!]$, $i \in \mathcal{K}$, B sends $(\Omega[\ell].\mathsf{ct}, i)$ to C. Then
  · if $\gamma = 0$, C computes $((\mathsf{pd})_i, \nu_i) \leftarrow \mathsf{Tir.ShareDec}((\mathsf{sk})_i, \mathsf{ct})$,
  · if $\gamma = 1$, C computes $(\mathsf{pd})_i = \mathsf{ct}^{(\mathsf{sk})_i}$ and $\nu_i \leftarrow \mathsf{Sim}_{\mathrm{ZKP}}(\mathsf{ct}, i)$.
  C returns $((\mathsf{pd})_i, \nu_i)$ to B, which sends it to A.

Hence, if A breaks the Scor-Adp-SIM-CPA security of Tir with a non-negligible advantage, then B breaks the zero-knowledge property of Tir ZKP with the same advantage. □

5.2 Security of our construction

We now construct the LHE layer of thPLWE with the Tiresias scheme (Sect. A). We prove the security of thPLWE via a reduction to the CPA security of Tiresias from Sect. 5.1. We note that as Tiresias is secure under Scor-Adp-SIM-CPA, then it implies that it is also secure under Scor-Adp-IND-CPA [13, Prop. 3.2]. Below we consider a sequence of hybrid games starting from the original Scor-Adp-IND-CPA game for thPLWE. In a nutshell, we end up with a new challenger that also stores left and right values for the b-terms in its internal state. This allows it to duplicate homomorphic calculations in order to delay the Tir part of post-evaluation challenge ciphertexts generation after the effective

homomorphic evaluation is done. This mechanic will then allow a reduction towards a challenger on the Scor-Adp-IND-CPA security of Tiresias to simulate evaluation requests on ciphertexts depending on the challenge bit.

Proposition 6. thPLWE *is Scor-Adp-IND-CPA secure under the assumption that* Tir *achieves the same.*

Proof. The proof first performs a number of game hops in which we gradually modify the Scor-Adp-IND-CPA challenger for thPLWE in order to allow for a final reduction towards the Scor-Adp-IND-CPA security of Tir.

Game 0. This is the original Scor-Adp-IND-CPA game against thPLWE, where, as a reminder, encryption, challenge and evaluation requests are processed as follows by the challenger:

- When A issues an encryption request over $m \in \mathbb{Z}_t$, B computes $(\mathbf{a}, b) = \mathsf{thLWE.Enc}(\mathsf{thLWE.pk}, m)$ and returns $\mathsf{ct} = (\mathbf{a}, b)$ to A after updating its internal state $\Omega := [\Omega; (m, m, (\mathbf{a}, b), \{\emptyset, \emptyset\})]$.
- When A issues a challenge request over $m_0, m_1 \in \mathbb{Z}_t^2$, $m_0 \neq m_1$ the challenger C against Scor-Adp-IND-CPA security of thPLWE returns $\mathsf{ct} = (\mathbf{a}, b) \leftarrow \mathsf{thLWE.Enc}(\mathsf{thLWE.pk}, m_\gamma)$ to A after doing $\Omega := [\Omega; (m_0, m_1, (\mathbf{a}, b), \{\emptyset, \emptyset\})]$.
- When A issues an evaluation request over $(f, \ell_0, \ldots, \ell_v)$, $\forall i\colon \ell_i \in [\![0, |\Omega|-1]\!]$, C sets $m'_0 = [f(\Omega[\ell_0].m_0, \ldots, \Omega[\ell_v].m_0)]_t$, $m'_1 = [f(\Omega[\ell_0].m_1, \ldots, \Omega[\ell_v].m_1)]_t$, $(\mathbf{a}', b') \leftarrow \mathsf{thLWE.Eval}(\mathsf{thLWE.pk}, f, \Omega[\ell_0].(\mathbf{a}, b), \ldots, \Omega[\ell_v].(\mathbf{a}, b))$, $(\mathbf{a}'', b'') \leftarrow \mathsf{thLWE.Sanitize}(\mathsf{thLWE.pk}, (\mathbf{a}', b'))$, $p \leftarrow \mathsf{Tir.Enc}(\mathsf{Tir.pk}, b'')$ and return $\mathsf{ct} = (\mathbf{a}'', p)$ to A after updating $\Omega := [\Omega; (m'_0, m'_1, (\mathbf{a}'', p), \{\emptyset, \emptyset\})]$.

For conciseness, we do not remind how partial decryptions are handled, as we do not modify it in our sequence of game hops (please refer to Def. 3).

Game 1. This game is the same as Game 0 except that we modify the way the challenger handles challenge requests:

- When A issues a challenge request over $m_0, m_1 \in \mathbb{Z}_t^2$, $m_0 \neq m_1$, C computes $(\mathbf{a}, b) \leftarrow \mathsf{thLWE.Enc}(\mathsf{thLWE.pk}, 0)$, sets $b_0 = [b + \Delta m_0]_q$, $b_1 = [b + \Delta m_1]_q$ and returns $\mathsf{ct} = (\mathbf{a}, b_\gamma)$ to A after updating $\Omega := [\Omega; (m_0, m_1, (\mathbf{a}, b_\gamma), \{\emptyset, \emptyset\})]$.

Game 0 and Game 1 are identical from A's view. In Game 1, C just modifies the way it computes the challenge ciphertext: it first computes $(\mathbf{a}, b_0)$ and $(\mathbf{a}, b_1)$, embeds the messages via an encryption of zero, and then selects b_γ. Either way the same challenge ciphertext is returned to A. Hence, $|\mathsf{Adv}^{G_1}_{\mathsf{thPLWE}}(\mathsf{A}) - \mathsf{Adv}^{G_0}_{\mathsf{thPLWE}}(\mathsf{A})| = 0$.

Game 2. This game is the same as Game 1 except the following changes in the handling of encryption, challenge and evaluation requests:

- When A issues an encryption request over $m \in \mathbb{Z}_t$, B computes $(\mathbf{a}, b) = \mathsf{thLWE.Enc}(\mathsf{thLWE.pk}, m)$ and returns $\mathsf{ct} = (\mathbf{a}, b)$ to A after updating its internal state $\Omega := [\Omega; (m, m, (\mathbf{a}, b, b, \perp), \{\emptyset, \emptyset\})]$.
- When A issues a challenge request over $m_0, m_1 \in \mathbb{Z}_t^2$, $m_0 \neq m_1$, C computes $(\mathbf{a}, b) \leftarrow \mathsf{thLWE.Enc}(\mathsf{thLWE.pk}, 0)$, sets $b_0 = [b + \Delta m_0]_q$, $b_1 = [b + \Delta m_1]_q$ and returns $\mathsf{ct} = (\mathbf{a}, b_\gamma)$ to A after doing $\Omega := [\Omega; (m_0, m_1, (\mathbf{a}, b_0, b_1, \perp), \{\emptyset, \emptyset\})]$.

– When A issues an evaluation request over $(f, \ell_0, \ldots, \ell_v)$, $\forall i$: $\ell_i \in [\![0, |\Omega| - 1]\!]$, C sets $m'_0 = [f(\Omega[\ell_0].m_0, \ldots, \Omega[\ell_v].m_0)]_t$, $m'_1 = [f(\Omega[\ell_0].m_1, \ldots, \Omega[\ell_v].m_1)]_t$ and computes

$$
\begin{aligned}
(\mathbf{a}', b'_0) &\leftarrow \mathsf{thLWE.Eval}(\mathsf{thLWE.pk}, f, \Omega[\ell_0].(\mathbf{a}, b_0), \ldots, \Omega[\ell_v].(\mathbf{a}, b_0)),\\
(\mathbf{a}', b'_1) &\leftarrow \mathsf{thLWE.Eval}(\mathsf{thLWE.pk}, f, \Omega[\ell_0].(\mathbf{a}, b_1), \ldots, \Omega[\ell_v].(\mathbf{a}, b_1)),\\
(\mathbf{a}'', b''_0) &\leftarrow \mathsf{thLWE.Sanitize}(\mathsf{thLWE.pk}, (\mathbf{a}', b'_0)),\\
(\mathbf{a}'', b''_1) &\leftarrow \begin{cases} (\mathbf{a}'', b''_0 + \Delta(m'_1 - m'_0)), & \text{if } b'_0 \neq b'_1,\\ (\mathbf{a}'', b''_0), & \text{otherwise,} \end{cases}\\
p &\leftarrow \mathsf{Tir.Enc}(\mathsf{Tir.pk}, b''_\gamma).
\end{aligned}
$$

C sends $\mathsf{ct} = (\mathbf{a}'', p)$ to A after setting $\Omega := [\Omega; (m'_0, m'_1, (\mathbf{a}'', \perp, \perp, p), \{\emptyset, \emptyset\})]$.

In a nutshell, C now also stores left and right values for the b-terms and duplicate homomorphic calculations to allow delaying the Tir part of post-evaluation challenge ciphertexts generation after the effective homomorphic evaluation is done. Game 1 and Game 2 are identical. In Game 2, C only changes its internal bookkeeping for the state Ω and the evaluation output:

– For an evaluation request that does not depend on the challenge bit γ (i.e. which does not involve any ciphertext obtained by means of a challenge request), we have $m'_0 = m'_1$, $b'_0 = b'_1$ as well as, $b''_0 \neq b''_1$ so both game proceed identically.
– For an evaluation request that depends on the challenge bit γ, although m'_0 may be equal to m'_1, we have $b'_0 \neq b'_1$ (with overwhelming probability), and hence $b''_0 \neq b''_1$. In Game 1, we proceed only with the evaluation yielding b'_γ (and b''_γ). In Game 2, the challenger proceeds with both candidate evaluations with either $b'_\gamma = b'_0$ (and $b''_\gamma = b''_0$) or $b'_\gamma = b'_1$ (and $b''_\gamma = b''_1$), so the same ciphertext is eventually returned to A, and both games also proceed identically in this case.

Hence, these changes do not alter the view of A and the advantage remains the same: $|\mathsf{Adv}^{G_2}_{\mathsf{thPLWE}}(\mathsf{A}) - \mathsf{Adv}^{G_1}_{\mathsf{thPLWE}}(\mathsf{A})| = 0$.

Game 3. This game is the same as Game 2 except that we further modify challenge request handling as follows:

– When A issues a challenge request over $m_0, m_1 \in \mathbb{Z}_t^2$, $m_0 \neq m_1$, C computes $(\mathbf{a}, b) \leftarrow \mathsf{thLWE.Enc}(\mathsf{thLWE.pk}, 0)$, sets $b_0 = [b + \Delta m_0]_q$, $b_1 = [b + \Delta m_1]_q$ and returns $\mathsf{ct} = (\mathbf{a}, b_0)$ to A after doing $\Omega := [\Omega; (m_0, m_1, (\mathbf{a}, b_0, b_1, \perp), \{\emptyset, \emptyset\})]$.

Game 2 and Game 3 are computationally indistinguishable, which follows directly from the LWE assumption, i.e. the computational indistinguishability of $(\mathbf{a}, b_0), (\mathbf{a}, b_1)$ LWE encryptions. Hence, $|\mathsf{Adv}^{G_3}_{\mathsf{thPLWE}}(\mathsf{A}) - \mathsf{Adv}^{G_2}_{\mathsf{thPLWE}}(\mathsf{A})| \leq \mathsf{neg}(\lambda)$. By transitivity, Game 0 is computationally indistinguishable from Game 3: $|\mathsf{Adv}^{G_3}_{\mathsf{thPLWE}}(\mathsf{A}) - \mathsf{Adv}^{G_0}_{\mathsf{thPLWE}}(\mathsf{A})| \leq \mathsf{neg}(\lambda)$.

Final reduction. We now prove that from an adversary A against Game 3, we can build an adversary B against the Scor-Adp-IND-CPA security of Tir,

which uses A as a subroutine while achieving the same advantage as A. First, A chooses two sets of corrupted parties $\mathcal{S}_\mathsf{L}$ and $\mathcal{S}_\mathsf{P}$ for thLWE and Tir, respectively, indexed by sets $\mathcal{T}'_\mathsf{L} \subset \mathcal{T}_\mathsf{L}$ and $\mathcal{T}'_\mathsf{P} \subset \mathcal{T}_\mathsf{P}$, and sends them to the reduction B, which then proceeds as follows. First, B checks $(\mathcal{S}_\mathsf{L} \subset \mathcal{P}) \wedge (\mathcal{S}_\mathsf{L} \notin \mathbb{A}_{T_\mathsf{L},K})$; if True, it runs $(\mathsf{thLWE.pk}, \{(\mathsf{thLWE.sk})_i\}_{i\in\mathcal{K}}) \leftarrow \mathsf{thLWE.DistKeyGen}(1^\lambda, \mathbb{A}_{T_\mathsf{L},K})$. B sends $\mathcal{S}_\mathsf{P}$ to C, who checks $(\mathcal{S}_\mathsf{P} \subset \mathcal{P}) \wedge (\mathcal{S}_\mathsf{P} \notin \mathbb{A}_{T_\mathsf{P},K})$; if True, runs $(\mathsf{Tir.pk}, \mathsf{Tir.vk}, \{(\mathsf{Tir.sk})_i\}_{i\in\mathcal{K}}) \leftarrow \mathsf{Tir.DistKeyGen}(1^\lambda, \mathbb{A}_{T_\mathsf{P},K})$ and returns to B a set $(\mathsf{Tir.pk}, \mathsf{Tir.vk}, \{(\mathsf{Tir.sk})_i\}_{i\in\mathcal{T}'_\mathsf{P}})$, which forwards it to A together with $(\mathsf{thLWE.pk}, \{(\mathsf{thLWE.sk})_i\}_{i\in\mathcal{T}'_\mathsf{L}})$. Lastly, B maintains an initially empty state Ω containing message-message-ciphertext-index $(m_0, m_1, \mathsf{ct}, \mathsf{ind})$ quadruples and handles A's request by doing:

- When A issues an encryption request over $m \in \mathbb{Z}_t$, B computes $(\mathbf{a}, b) = \mathsf{thLWE.Enc}(\mathsf{thLWE.pk}, m)$ and returns $\mathsf{ct} = (\mathbf{a}, b)$ to A after updating its internal state $\Omega := [\Omega; (m, m, (\mathbf{a}, b, b, \perp), \{\emptyset, \emptyset\})]$.
- When A issues a challenge request over $m_0, m_1 \in \mathbb{Z}_t^2$, $m_0 \neq m_1$, B does $(\mathbf{a}, b) \leftarrow \mathsf{thLWE.Enc}(\mathsf{thLWE.pk}, 0)$, sets $b_0 = [b + \Delta m_0]_q$, $b_1 = [b + \Delta m_1]_q$ and returns $\mathsf{ct} = (\mathbf{a}, b_0)$ to A after setting $\Omega := [\Omega; (m_0, m_1, (\mathbf{a}, b_0, b_1, \perp), \{\emptyset, \emptyset\})]$.
- When A issues an evaluation request over $(f, \ell_0, \ldots, \ell_v)$, $\forall i\colon \ell_i \in [\![0, |\Omega|-1]\!]$, B sets $m'_0 = [f(\Omega[\ell_0].m_0, \ldots, \Omega[\ell_v].m_0)]_t$, $m'_1 = [f(\Omega[\ell_0].m_1, \ldots, \Omega[\ell_v].m_1)]_t$ and computes

$$
\begin{aligned}
(\mathbf{a}', b'_0) &\leftarrow \mathsf{thLWE.Eval}(\mathsf{thLWE.pk}, f, \Omega[\ell_0].(\mathbf{a}, b_0), \ldots, \Omega[\ell_v].(\mathbf{a}, b_0)),\\
(\mathbf{a}', b'_1) &\leftarrow \mathsf{thLWE.Eval}(\mathsf{thLWE.pk}, f, \Omega[\ell_0].(\mathbf{a}, b_1), \ldots, \Omega[\ell_v].(\mathbf{a}, b_1)),\\
(\mathbf{a}'', b''_0) &\leftarrow \mathsf{thLWE.Sanitize}(\mathsf{thLWE.pk}, (\mathbf{a}', b'_0)),\\
(\mathbf{a}'', b''_1) &\leftarrow \begin{cases} (\mathbf{a}'', b''_0 + \Delta(m'_1 - m'_0)) & \text{if } b'_0 \neq b'_1,\\ (\mathbf{a}'', b''_0), & \text{otherwise,} \end{cases}
\end{aligned}
$$

  obtaining two samples $(\mathbf{a}'', b''_0)$ and $(\mathbf{a}'', b''_1)$. B then forwards (b''_0, b''_1) to C as the challenge request to get $p \leftarrow \mathsf{Tir.Enc}(\mathsf{Tir.pk}, b''_\gamma)$ and returns $\mathsf{ct} = (\mathbf{a}'', p)$ to A after updating $\Omega := [\Omega; (m'_0, m'_1, (\mathbf{a}'', \perp, \perp, p), \{\emptyset, \emptyset\})]$.
- When A issues a decryption request over $\ell \in [\![0, |\Omega| - 1]\!]$, $i, j \in \mathcal{K}$, if $\Omega[\ell].p \neq \perp$, B proceeds as
  - if $\Omega[\ell].m_0 = \Omega[\ell].m_1$, B computes

$$p_i \leftarrow \mathsf{Tir.cAdd}(\Omega[\ell].p, [-\langle \Omega[\ell].\mathbf{a}, (\mathsf{thLWE.sk})_i \rangle]_q)$$

    and sends (p_i, j) to C to get $(\mathsf{pd}_{i,j}, \nu_{i,j}) \leftarrow \mathsf{Tir.ShareDec}((\mathsf{Tir.sk})_j, p_i)$ and return it to A.
  - if $\Omega[\ell].m_0 \neq \Omega[\ell].m_1$, B updates the index state $\Omega[\ell].\mathsf{ind} := [\Omega[\ell].\mathsf{ind}; \{i, j\}]$ and checks $(\{P_i\}_{i\in\Omega[\ell].\mathsf{ind}[0]} \cup \mathcal{S}_\mathsf{L}) \notin \mathbb{A}_{T_\mathsf{L},K}$; if True, B computes p_i. Then B checks $(\{P_i\}_{i\in\Omega[\ell].\mathsf{ind}[1]} \cup \mathcal{S}_\mathsf{P}) \notin \mathbb{A}_{T_\mathsf{P},K}$; if True, B sends (p_i, j) to C to get $(\mathsf{pd}_{i,j}, \nu_{i,j})$ and returns it to A; if False, B returns p_i to A.

It follows that if A breaks the Scor-Adp-IND-CPA security of thPLWE with a non-negligible advantage, then B breaks the Scor-Adp-IND-CPA security of Tir with the same advantage. □

6 Concrete construction and experimental results

The thPLWE, introduced in Sect. 5, combines threshold LWE and threshold LHE layers, i.e. it is generic in nature. However, as previously noted, we construct thPLWE threshold LHE layer with Tiresias ([14] and App. A). Further, as an example, we consider in this section a concrete construction that reuses the distributed key generation from the threshold TFHE variant in [10]. In our concrete instantiation, we also use the LWE sanitization from [31]. Hence, to illustrate the practicality of thPLWE, we only provide timings for these underlying algorithms *based on the experimental results reported in* [10], [31] *and* [14]. Finally, we also assess the efficiency of our threshold decryption by providing benchmarks in Fig. 2 and compare its performance against threshold decryption with smudging as well as MPC-based threshold decryption. *Since we compare timings that have not been consistently obtained on a single machine, we provide orders of magnitude.*

6.1 Construction

Threshold TFHE key generation. More specifically, we propose constructing thLWE.DistKeyGen using the threshold key generation described in [10, Sect. 7.5.4.1, Fig. 81]. This algorithm is a thresholdized version of the classical TFHE key generation. It maintains correctness and security as long as it is executed over a robust MPC protocol. The online phase of the algorithm requires 3 rounds of MPC communication, while the full DKG is completed in 5 rounds. We provide the expected computational and communication complexities in Fig. 1 for three (t, q, n) TFHE parameter sets, with a fixed threshold $T = \frac{K}{4}$ and varying numbers of parties $K \in \{10, 100, 1000\}$. For a detailed security analysis and an evaluation of the analytic complexity, we refer the reader to [10, Sect. 8.5.4.1, Sect. 9.6.5].

$(t,q,n) = (2, 2^{32}, 550)$				$(t,q,n) = (16, 2^{32}, 750)$				$(t,q,n) = (2, 2^{64}, 1024)$			
K	$T+1$	Comm.	Comp.	K	$T+1$	Comm.	Comp.	K	$T+1$	Comm.	Comp.
10	3	2^{44}	2^{46}	10	3	2^{46}	2^{48}	10	3	2^{49}	2^{51}
100	26	2^{51}	2^{54}	100	26	2^{54}	2^{56}	100	26	2^{56}	2^{59}
1000	251	2^{58}	2^{62}	1000	251	2^{61}	2^{64}	1000	251	2^{63}	2^{67}

Table 1: Expected communication (bits) and computational (operations) complexities computed following [10, Sect. 9.6.13] for the TFHE DKG for three TFHE parameter sets. We obtain the first two TFHE parameter sets (detailed in [12]) following the methodology in [36]. The third parameter set is taken from [11] and is a standard choice in TFHE implementations like [9]. The parameter sets achieve 128-bit security according to the lattice-estimator [37].

LWE sanitization. The sanitization from [31] limits randomization to the steps before key-switching and after blind rotation, which reduces computational overhead. We recall that blind rotation homomorphically computes $\mathrm{tv} \cdot X^{b-\sum_{i=0}^{n-1} a_i \mathsf{sk}_i}$, where tv is a public polynomial and $\mathsf{ct} = (\mathbf{a}, b)$ is a scaled LWE ciphertext. The process begins by multiplying tv by X^b, followed by a sequence of multiplications by $X^{a_i \mathsf{sk}_i}$ for each $i \in [\![0, n-1]\!]$ using the blind rotation keys. While prior works [28, 30] randomize the entire blind rotation to preserve privacy, [31] reduces this overhead by reordering the evaluation. First, $\mathbf{a}$ is randomized by adding an encryption of zero. The evaluation order is then modified to defer the private term b until the end: the algorithm computes $\mathrm{tv} \cdot X^{\sum a_i \mathsf{sk}_i}$ first. The final multiplication by X^b is performed using the randomized linear evaluation from [38], which enables circuit-private multiplication of ring elements with ciphertexts. Therefore, sanitization requires randomization only at the beginning and at the end of the blind rotation, keeping the core algorithm intact. Experimental results demonstrate that [31] reduces sanitization latency to just 42.17ms, significantly lower than the 7500ms in [28] or 1330ms in [30]. Their sanitization algorithm adds only 0.81ms of overhead to standard TFHE bootstrapping. For detailed benchmark results, see [31, Sect. 6.2].

6.2 Threshold decryption efficiency evaluation

thPLWE decryption efficiency. Recall that to eliminate the need for smudging or the MPC noise removal, we sanitize an LWE ciphertext and hide the resulting b-term by encrypting it under Tir, $p = \mathsf{Tir.Enc}(\mathsf{Tir.pk}, b)$, before proceeding to the threshold decryption. Note that thPLWE threshold decryption consists of two steps, $\mathsf{PartDec}$ and $\mathsf{DistDec}$, which may involve different sets of participating parties and different thresholds. The core idea of $\mathsf{thPLWE.PartDec}$ is to let LWE decryption parties $\{P_i\}_{i \in \mathcal{T}_\mathsf{L}}$ compute the partial decryption of φ under Tir as

$$\mathsf{Tir.Enc}((\varphi)_i) = \mathsf{Tir.cAdd}(p, [\langle -\mathbf{a}, (\mathsf{thLWE.sk})_i \rangle]_q), \tag{1}$$

and jointly reconstruct $\mathsf{Tir.Enc}(\varphi) = \mathsf{Tir.Enc}(\Delta m + e)$ such that

$$\mathsf{Tir.Enc}(\varphi) = \bigoplus_{i=0}^{|\mathcal{T}_\mathsf{L}|-1} \left(\mathsf{Tir.cMult}\left(\mathsf{Tir.Enc}((\varphi)_i), \lambda_i\right)\right), \tag{2}$$

where $\oplus$ defines $\mathsf{Tir.Add}$. Note that, by getting rid of smudging, we naturally solve the problem of error growth during secret share reconstruction, as we avoid the scaling of smudging noise by Lagrange coefficients. This allows us to resolve the same issue addressed in [33] without restricting ourselves to the synchronous model. Finally, the decryption is completed by $\mathsf{thPLWE.DistDec}$, which consists of the $\mathsf{Tir.DistDec}$ algorithm (run by $\{P_i\}_{i \in \mathcal{T}_\mathsf{P}}$ to output φ) followed by the message recovery $\left\lceil \frac{\varphi}{\Delta} \right\rfloor$.

Below, we analyze the computational complexity of our decryption approach and present experimental results comparing it with state-of-the-art threshold

decryption techniques to demonstrate its efficiency. In the Paillier cryptosystem, ciphertext-ciphertext addition involves a single modular multiplication in $\mathbb{Z}^*_{\mu^2}$. This requires $\mathcal{O}(M(\kappa))$ bit operations, where $M(\kappa)$ denotes the cost of multiplying two κ-bit integers. Plaintext-ciphertext addition, $[\mathsf{ct} \cdot g^m]_{\mu^2}$, where $g^m = [1 + m\mu]_{\mu^2}$ for $g = 1 + \mu$, requires one modular multiplication.

Finally, plaintext-ciphertext multiplication requires a modular exponentiation, ct^m. This is typically implemented via the square-and-multiply algorithm, requiring $\mathcal{O}(\ell)$ modular multiplications where $\ell = \lceil \log_2 m \rceil$. Consequently, plaintext-ciphertext multiplication incurs $\mathcal{O}(\ell \cdot M(\kappa))$ bit operations.

Computing (1) requires a dot product for vectors of size n, along with one Paillier plaintext-ciphertext addition. This results in a computational complexity of $\mathcal{O}(n+M(\kappa))$. Let $T+1 = |\mathcal{T}_\mathsf{L}|$. Computing (2) involves $T+1$ Paillier plaintext-ciphertext multiplications and T ciphertext-ciphertext additions[4]. Therefore, the computational complexity of (2) is given by $\mathcal{O}(T \cdot \log q \cdot M(\kappa))$.

To highlight the efficiency of our threshold LWE decryption, we implemented (1) and (2) in C using GMP and libpaillier [40] libraries. Benchmarks were performed on a 12th Gen Intel Core i7-12700H processor. Fig. 2 reports timings for a 2048-bit modulus μ, the three (t, q, n) TFHE parameter sets, and varying numbers of parties $K \in \{10, 100, 1000\}$. We evaluate threshold values $T \in \left\{\frac{K}{4}, \frac{2K}{3}\right\}$, assuming that $T + 1$ parties participate in thPLWE.PartDec.

$(t,q,n) = (2, 2^{32}, 550)$				$(t,q,n) = (16, 2^{32}, 750)$				$(t,q,n) = (2, 2^{64}, 1024)$			
K	$T+1$	(1)	(2)	K	$T+1$	(1)	(2)	K	$T+1$	(1)	(2)
10	3	0.158	0.237	10	3	0.168	0.235	10	3	0.544	0.970
	7	0.156	0.678		7	0.169	0.708		7	0.501	2.931
100	26	0.142	2.644	100	26	0.158	2.618	100	26	0.540	11.505
	67	0.161	7.296		67	0.187	7.392		67	0.561	32.210
1000	251	0.173	27.396	1000	251	0.173	27.472	1000	251	0.553	125.998
	667	0.172	73.358		667	0.165	74.671		667	0.557	324.616

Table 2: Performance comparison (ms) for PartDec ((1) and (2)) for a 2048-bit modulus μ. All timings are reported as the mean of 10 single-core executions.

It is important to note that the overall efficiency of thPLWE threshold decryption is highly dependent on the performance of the underlying LHE distributed decryption (Tir.DistDec in this case). We highlight that Tiresias adapts batching techniques by [41], which allows a party to generate a ZKP for the correctness of multiple decryption shares instead of one proof per ciphertext. Batching is highly effective for scalability, as it significantly reduces the amortized cost per decryption. Fig. 3 illustrates the efficiency of Tir.DistDec using benchmarks from

[4] This sequence constitutes a multi-exponentiation, where the cost of modular squarings is amortized across all shares. Employing [39] yields a smaller complexity.

[14]. These timings consider a 2048-bit μ, $K \in \{10, 100, 1000\}$ parties, $B \in \{1, 100, 1000\}$ ciphertexts and a threshold $T = \frac{2K}{3}$ for the participating $\{P_i\}_{i \in \mathcal{T}_\mathsf{P}}$.

K	B	Comp.	Comb.
10	1	151.93	89.517
	100	873.98	763.5
	1000	7424.6	6746.1
100	1	193.28	262.53
	100	959.41	1559.8
	1000	7909.8	12362.0
1000	1	790.67	9722.5
	100	2138.8	35058.0
	1000	14558.0	266640.0

Table 3: Performance comparison (ms) for Tir.DistDec. Third columns report timings for a single party to produce B decryption shares (where B is a batch size) and ZKPs of equality for K parties. Fourth columns report timings for ZKP verification and decryption share combination.

For example, for decryption share computation (3rd columns, Fig. 3) with $K \in \{10, 100, 1000\}$, a prover requires $\{151, 193, 790\}$ms to generate a share for a single ciphertext. However, when batching $B = 1000$ ciphertexts, the cost drops to only $\{7, 7, 14\}$ms per share, representing an improvement of approximately $\{21\times, 27\times, 56\times\}$, respectively. This efficiency gain extends to the decryption share combination and the proof verification (4th columns, Fig. 3).

To assess the practical performance of our threshold decryption, we combine the costs of the PartDec with the underlying Tir.DistDec across various batch sizes. Since each of the B ciphertexts requires an individual PartDec execution to produce the corresponding Tir ciphertexts, the PartDec-related costs scale linearly with the batch size. Fig. 4 summarizes the total estimated execution time for both thPLWE decryption steps (excluding communication overhead) using the parameters described above with thresholds $T = \frac{2K}{3}$, where $T + 1 = |\mathcal{T}_\mathsf{L}| = |\mathcal{T}_\mathsf{P}|$.

Finally, Fig. 5 confirms that thPLWE.DistDec remains quite scalable. By batching $B = 1000$ ciphertexts, the average latency per decryption for $K = 1000$ parties is reduced from over 10 seconds to between 354ms and 606ms, depending on the TFHE parameter set.

Comparison with the state-of-the-art. As a starting point, we analyze the performance of [21] threshold TFHE with smudging. In the Noah's Ark, decryption employs the offline phase when $\binom{K}{T} > 100$.[5] It utilizes a generic MPC protocol to generate random bit shares, which are then used to produce two uniformly random 128-bit smudging terms. Hence, the parties must collectively generate 256 shared random bits per decryption to obtain the necessary masks. To accommodate the smudging noise, Noah's Ark requires a significant parameter size increase: the LWE parameters $(q, n) = (2^{64}, 777)$ are increased to $(2^{128}, 4096)$, which significantly expands the ciphertext size. The authors report that the non-interactive part of the decryption takes between 241 and 316ms. In contrast, the interactive distributed decryption requires approximately 2ms when

[5] It holds for $K = 10$, $T \in [\![3, 7]\!]$; $K = 100$, $T \in [\![2, 98]\!]$; $K = 1000$, $T \in [\![1, 999]\!]$.

$(t,q,n)=(2,2^{32},550)$				$(t,q,n)=(16,2^{32},750)$				$(t,q,n)=(2,2^{64},1024)$			
K	$T+1$	B	Total (ms)	K	$T+1$	B	Total (ms)	K	$T+1$	B	Total (ms)
10	7	1	242.281	10	7	1	242.324	10	7	1	244.879
		100	1720.880			100	1725.180			100	1980.680
		1000	15004.700			1000	15047.700			1000	17602.700
100	67	1	463.267	100	67	1	463.389	100	67	1	488.581
		100	3264.910			100	3277.110			100	5796.310
		1000	27728.800			1000	27850.800			1000	53042.800
1000	667	1	10586.700	1000	667	1	10588.006	1000	667	1	10838.343
		100	44549.800			100	44680.400			100	69714.100
		1000	354728.000			1000	356034.000			1000	606371.000

Table 4: Total estimated timings (ms) for the thPLWE threshold decryption. The timings reflect the total latency to process a batch of size B, running B individual PartDec and the subsequent Tir.DistDec.

$(t,q,n)=(2,2^{32},550)$				$(t,q,n)=(16,2^{32},750)$				$(t,q,n)=(2,2^{64},1024)$			
K	$T+1$	B	Amort.	K	$T+1$	B	Amort.	K	$T+1$	B	Amort.
10	7	1	242.281	10	7	1	242.324	10	7	1	244.879
		100	17.209			100	17.252			100	19.807
		1000	15.005			1000	15.048			1000	17.603
100	67	1	463.267	100	67	1	463.389	100	67	1	488.581
		100	32.649			100	32.771			100	57.963
		1000	27.729			1000	27.851			1000	53.043
1000	667	1	10586.700	1000	667	1	10588.006	1000	667	1	10838.343
		100	445.498			100	446.804			100	697.141
		1000	354.728			1000	356.034			1000	606.371

Table 5: Total amortized estimated timings (ms) for the thPLWE threshold decryption. Each entry represents the average latency to process a single ciphertext within a batch of size B. The values are obtained by summing the time for one PartDec with the amortized timing of the Tir.DistDec.

$(K,T)=(4,1)$, 3ms when $(K,T)=(10,3)$, and 19ms when $(K,T)=(40,13)$. *With respect to Noah's Ark, the key takeaway is that our approach offers comparable unitary costs and latencies for decrypting one ciphertext but* without *increasing the size of the TFHE parameters (i.e. without incurring a significant decrease in the performances of the homomorphic computations) and with an additional potential for amortization that is not offered by the former.*

By employing MPC-based rounding, Zyskind et al. [11] avoid smudging and the consequent modulus expansion seen in [21], so, *as we do*, they keep standard parameters, however, at the cost of a heavier offline phase. While online decryption [11] is efficient, it relies on a computationally and memory-intensive (offline) preprocessing to generate Beaver triples and shared random bits. Each threshold decryption call consumes approximately 64 shared random bits and requires 2478 multiplications. Since each multiplication utilizes one Beaver triple, the offline communication cost (using [42] triple generation at 14.9kbit per triple) amounts to roughly 4.5MB per decryption and approximately 36ms of transfer time. For [11] online phase, the online preprocessing for 1000 runs of the decryption requires 1.35sec and 79MB of communication per party (excluding offline triple and random-bit generation costs). Finally, Zyskind et al. report $\{8.48, 10.08, 24.23\}$ms needed for $K \in \{4, 8, 16\}$ parties to run a single online decryption, i.e. 8.48ms for $K = 4$ compared to approximately 315.62ms of [21]. However, as the number of parties increases, the online latency demonstrates non-linear growth, rising to 24ms at $K = 16$. Extrapolating it to 100 or 1000 parties suggests that it would likely push the latter latencies from milliseconds into the range of several seconds or even minutes. *In contrast, our approach has no offline phase beyond key generation. Although it is less efficient for small K, it offers much better scaling when the number of parties increases.*

In [10], a threshold TFHE decryption is implemented via MPC bit extraction. Because the authors do not provide concrete execution timings, we cannot perform a direct empirical comparison. Instead, [10, Sect. 9.6.13] evaluates the protocol via round, communication and computational complexities. For concrete parameters with $q = 2^{64}$ and $T = \frac{K}{4}$, the protocol requires 16 rounds of communication. While the number of rounds remains constant, the overhead scales with K: for $K = 10$, it requires 2^{25} bits of communication/storage and 2^{27} computational operations; for $K = 100$, these rise to 2^{36} and 2^{104}; and for $K = 1000$, they reach 2^{46} and 2^{837}. While the $K = 10$ case is practical, the $K = 100$ and $K = 1000$ cases are strictly theoretical. *In contrast, our solution requires only* 4 *rounds and scales linearly with the number of parties.*

Concluding remarks. Our approach has the desirable property of not impacting TFHE parameters, thus leaving unchanged TFHE ciphertext size as well as the performances of the homomorphic operators. We emphasize that thPLWE decryption requires only a single additional execution of the LHE distributed key generation (performed via MPC) as an offline preprocessing step. Our decryption algorithm requires just 4 communication rounds, compared to 16 in [10]: 2 to compute PartDec (Eqs. 1 and 2) and 2 to run DistDec (Tir.ShareDec and Tir.Comb). While [10, 11] MPC message recovery is more efficient for a small number of parties, $K < 20$, our approach becomes advantageous as the number of parties grows. Furthermore, by avoiding the need for MPC machinery during threshold decryption, our approach scales efficiently up to $K = 1000$ participating parties. By keeping $(q, n) = (2^{32}, 550)$, we require just 350ms (amortized) per decryption for $K = 1000$. We also highlight that the cost of our solution

	[21]	[11]	[10]	Our Work
(q, n, K)	$(2^{128}, 4096, 10)$	$(2^{64}, 1024, 16)$	$(2^{64}, 1024, 10)$	$(2^{32}, 750, 10)$
Preprocessing	≈ 5 ms	≈ 36 ms	≈ 32 ms	–
Preproc. bandwidth	≈ 1 KB	≈ 4.5 MB	≈ 4 MB	–
Local computation	$\approx$ 241-316 ms	≈ 36 ms	2^{27} ops	≈ 152.10 ms
Online latency	≈ 3 ms	≈ 24.23 ms	not reported	≈ 92.23 ms
Comm. rounds	1	multi-round	16	4
Bandwidth ($B = 1$)	≈ 1.31 MB	≈ 4.5 MB	≈ 4.2 MB	≈ 70 KB

Table 6: Comparison of threshold decryption (Sect. 6.2) for a single LWE. Note that we excluded key generation since [10, 11, 21] do not report any timings for this (offline) step and, as already stated, our intent is to reuse the one from [10] in a black-box fashion.

is dominated by Tiresias, which we have used as a black-box. Hence, any other (possibly more efficient) correct linearly homomorphic threshold PKE can be plugged into our construction. Lastly, we emphasize that, at the conceptual level, our construction is quite simple.

References

[1] Gilad Asharov et al. "Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE". In: *EUROCRYPT 2012*. 2012, pp. 483–501.

[2] Marina Checri et al. "On the practical CPAD security of "exact" and threshold FHE schemes and libraries". In: *CRYPTO 2024*. 2024, pp. 3–33.

[3] Zvika Brakerski et al. "Fully Homomorphic Encryption without Bootstrapping". In: *ITCS'12*, pp. 309–325.

[4] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *CRYPTO 2012*. 2012, pp. 868–886.

[5] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2012/144. 2012.

[6] Jung Hee Cheon et al. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: *ASIACRYPT 2017*. 2017, pp. 409–437.

[7] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second". In: *EUROCRYPT 2015*, pp. 617–640.

[8] Ilaria Chillotti et al. "TFHE: Fast Fully Homomorphic Encryption Over the Torus". In: *Journal of Cryptology*. 2020, pp. 34–91.

[9] Zama. *TFHE-rs*. https://github.com/zama-ai/tfhe-rs. 2022.

[10] Carl Bootland et al. *Threshold (Fully) Homomorphic Encryption*. Cryptology ePrint Archive, Paper 2025/699. 2025.

[11] Guy Zyskind et al. "High-Throughput Universally Composable Threshold FHE Decryption". In: *ACM SIGSAC, CCS 2025*. 2025, pp. 2339–2353.

[12] Antonina Bondarchuk et al. “Downlink (T)FHE Ciphertexts Compression”. In: *SAC 2025*. 2025, pp. 647–678.
[13] Chris Brzuska et al. “Threshold Public-Key Encryption: Definitions, Relations, and CPA-to-CCA Transforms”. In: *PKC 2026*. 2026.
[14] Offir Friedman et al. “Tiresias: Large Scale, UC-Secure Threshold Paillier”. In: *ASIACRYPT 2024*. 2024, pp. 141–173.
[15] Dan Boneh et al. “Threshold Cryptosystems from Threshold Fully Homomorphic Encryption”. In: *CRYPTO 2018*. 2018, pp. 565–596.
[16] Zvika Brakerski et al. *Threshold FHE with Efficient Asynchronous Decryption*. Cryptology ePrint Archive, Paper 2025/712. 2025.
[17] Christian Mouchet et al. “Multiparty Homomorphic Encryption from Ring-Learning-with-Errors”. In: *Proc. Priv. Enh. Techn.* 4 (2021), pp. 291–311.
[18] Christian Mouchet et al. “An Efficient Threshold Access-Structure for RLWE-Based Multiparty Homomorphic Encryption”. In: *J. Cr.* 36.2 (2023).
[19] Antoine Urban and Matthieu Rambaud. “Robust Multiparty Computation from Threshold Encryption Based on RLWE”. In: *(ISC) 2024*, pp. 294–314.
[20] Yijia Chang and Songze Li. “Arbitrary-threshold fully homomorphic encryption with lower complexity”. In: *SEC'25*. 2025.
[21] Morten Dahl et al. “Noah's Ark: Efficient Threshold-FHE Using Noise Flooding”. In: *WAHC CCS 2023*. ACM, 2023, pp. 35–46.
[22] Alain Passelègue and Damien Stehlé. “Low Communication Threshold Fully Homomorphic Encryption”. In: *ASIACRYPT 2024*, pp. 297–329.
[23] Nigel P. Smart and Michael Walter. *Error-Simulatable Sanitization for TFHE and Applications*. Cryptology ePrint Archive, Paper 2025/275. 2025.
[24] Pierre-Alain Fouque et al. “Sharing Decryption in the Context of Voting or Lotteries”. In: *FC 2000*, pp. 90–104.
[25] Olivier Baudron et al. “Practical multi-candidate election system”. In: *ACM, PODC 2001*. 2001, pp. 274–283.
[26] Ivan Damgård and Mads Jurik. “A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System”. In: *PKC 2001*. 2001, pp. 119–136.
[27] Ivan Dåmgard et al. “A generalization of Paillier's public-key system with applications to electronic voting”. In: *Int. J. of Inf. Sec.* 2010, pp. 371–385.
[28] Florian Bourse and Malika Izabachène. *Plug-and-play sanitization for TFHE*. Cryptology ePrint Archive, Paper 2022/1438. 2022.
[29] Léo Ducas and Damien Stehlé. “Sanitization of FHE Ciphertexts”. In: *EUROCRYPT 2016*. 2016, pp. 294–310.
[30] Kamil Kluczniak. “Circuit Privacy for FHEW/TFHE-Style Fully Homomorphic Encryption in Practice”. In: *IACR Commun. Cryptol.* 2024.
[31] Intak Hwang et al. “Practical TFHE Ciphertext Sanitization for Oblivious Circuit Evaluation”. In: *SIGSAC, CCS 2025*. 2025, pp. 4064–4078.
[32] Adi Shamir. “How to Share a Secret”. In: *Com. of ACM*. 1979, pp. 612–613.
[33] François Colin de Verdière et al. *On Threshold Fully Homomorphic Encryption with Synchronized Decryptors*. Cr. ePrint Arch., Paper 2026/031.

[34] R. Yang et al. "Fully Homomorphic Encryption with Chosen-Ciphertext Security from LWE". In: *CRYPTO*. 2025, pp. 371–405.
[35] Baiyu Li and Daniele Micciancio. "On the Security of Homomorphic Encryption on Approximate Numbers". In: *EUROCRYPT 2021*, pp. 648–677.
[36] P-E. Clet et al. "ComBo: A Novel Functional Bootstrapping Method for Efficient Evaluation of Nonlinear Functions in the Encrypted Domain". In: *AFRICACRYPT 2023*. 2023, pp. 317–343.
[37] Martin R. Albrecht et al. "On the concrete hardness of Learning with Errors". In: *J.Math.Cryptol.* 2015, pp. 169–203.
[38] Leo de Castro et al. *More Efficient Lattice-based OLE from Circuit-private Linear HE with Polynomial Overhead*. ePrint Archive, Paper 2024/1534.
[39] Nicholas Pippenger. "On the evaluation of powers and related problems". In: *SFCS'76*. 1976, pp. 258–263.
[40] John Bethencourt. *libpaillier*. https://github.com/mortendahl/libpaillier.
[41] Riza Aditya et al. "Batch Verification for Equality of Discrete Logarithms and Threshold Decryptions". In: *ACNS 2004*. 2004, pp. 494–508.
[42] Sebastian Hasler et al. "Multipars: Reduced-Communication MPC over Z2k". In: *Proc. Priv. Enhancing Technol.* 2024, pp. 5–28.
[43] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *EUROCRYPT'99*. 1999, pp. 223–238.

A Tiresias scheme

Let $\mathsf{Tir} = (\mathsf{DistKeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{DistDec})$ be a T-out-of-K Tiresias scheme:

- $\mathsf{DistKeyGen}(1^\lambda, \mathbb{A}_{T,K}; \mathcal{P}) \to (\mathsf{pk}, \mathsf{vk}, \{(\mathsf{sk})_i\}_{i\in\mathcal{K}})$: $\mathcal{P} = \{P_i\}_{i\in\mathcal{K}}$ run Setup_1 to get μ and shares of its factors p', q': $(\mu; \{(p')_i, (q')_i\}_{i\in\mathcal{K}}) \leftarrow \mathsf{Setup}_1(K, 1^\lambda, 1^\tau)$. Then, they run Setup_2 to generate shares of $\mathsf{sk} = d$, using SSS over the Integers ($\mathsf{SS}^{\mathbb{Z}}$) [14, Sect. 2.1], and verification keys $(\{(\mathsf{sk})_i, (\mathsf{vk})_i\}_{i\in\mathcal{K}}) \leftarrow \mathsf{Setup}_2(K, \mu, 1^\lambda, 1^\tau; \{(p')_i, (q')_i\}_{i\in\mathcal{K}})$. The algorithm outputs $\mathsf{pk} = \mu$ and $\mathsf{vk} = \{(\mathsf{vk})_i\}_{i\in\mathcal{K}}$, where each $(\mathsf{vk})_i = \mathbf{g}^{(\mathsf{sk})_i}$ for a base $\mathbf{g} = \prod g_i$ sampled by the parties, and the secret $(\mathsf{sk})_i = (d)_i$ owned by each P_i.
- $\mathsf{Enc}(\mathsf{pk}, m)$, $\mathsf{Eval}\left(\mathsf{pk}, f, \{\mathsf{ct}_i\}_{i\in[\![0,l]\!]}\right)$ are identical to classical Paillier [43].
- $\mathsf{DistDec}\left(\mathsf{vk}, \mathcal{T}, \{(\mathsf{sk})_i\}, \mathsf{ct}; \{P_i\}_{i\in\mathcal{T}}\right) \to (m \in \mathbb{Z}_\mu) \cup \perp$: the parties $\{P_i\}_{i\in\mathcal{T}}$ execute a distributed decryption on $\mathsf{ct} \in \mathbb{Z}_{\mu^2}$, consisting of three sub-algorithms:
  - $\mathsf{ShareDec}((\mathsf{sk})_i, \mathsf{ct}; P_i) \to ((\mathsf{pd})_i, \nu_i)$: each party P_i outputs $(\mathsf{pd})_i = \mathsf{ct}^{(\mathsf{sk})_i}$ and a ZKP ν_i, which proofs that $\log_{\mathsf{ct}}(\mathsf{pd})_i = \log_{\mathbf{g}}(\mathsf{vk})_i = (\mathsf{sk})_i$. It ensures that P_i used its correct secret share $(\mathsf{sk})_i$ as the witness.
  - $\mathsf{Comb}(\mathsf{pk}, \{(\mathsf{pd})_i, \nu_i\}_{i\in\mathcal{T}}; \{P_i\}_{i\in\mathcal{T}}) \to \mathsf{c} \cup \perp$: if $\geq T+1$ shares have valid proofs, return $\mathsf{c} = \mathsf{ct}^{4\Delta_K^3 d} \leftarrow \mathsf{SS}^{\mathbb{Z}}.\mathsf{Recon}(\{(\mathsf{pd})_i, (\mathsf{sk})_i\}_{i\in\mathcal{T}})$, where $\Delta_K = K!$;[6] otherwise, return $\perp$.
  - $\mathsf{FinDec}(\mathsf{pk}, \mathsf{c}) \to m$: the final step retrieves $m = \left[\left(\frac{\mathsf{c}-1}{\mu}\right)(4\Delta_K^3)^{-1}\right]_\mu$.

[6] Multiplication by Δ_K clears denominators in the Lagrange coefficients, ensuring that every exponent used in reconstruction is in $\mathbb{Z}$. The factor $4\Delta_K^3$ arises from combining several scaling sources. For further details, see [14, Sect. 2.1, D].