An Efficient Threshold Access-Structure for RLWE-Based Multiparty Homomorphic Encryption

Christian Mouchet[1], Elliott Bertrand[2], and Jean-Pierre Hubaux[1]

[1] École polytechnique fédérale de Lausanne
[2] Effixis SA

Abstract. We propose and implement a multiparty homomorphic encryption (MHE) scheme with a t-out-of-N-threshold access-structure that is efficient and does not require a trusted dealer in the common reference-string model. We construct this scheme from the ring-learning-with-error (RLWE) assumptions, and as an extension of the MHE scheme of Mouchet et al. (PETS 21). By means of a specially adapted *share-resharing* procedure, this extension can be used to relax the N-out-of-N-threshold access structure of the original scheme into a t-out-of-N-threshold one. This procedure introduces only a single round of communication during the setup phase to instantiate the t-out-of-N-threshold access structure. Then, the procedure requires only local operations for any set of t parties to compute a t-out-of-t additive sharing of the secret key; this sharing can be used directly in the scheme of Mouchet et al.
We show that, by performing the re-sharing over the MHE ciphertext-space with a carefully chosen exceptional set, this reconstruction procedure can be made secure and has negligible memory and CPU-time overhead. Hence, in addition to fault tolerance, lowering the corruption threshold also yields considerable efficiency benefits, by enabling the distribution of batched secret-key operations among the online parties. We implemented and open-sourced our scheme in the Lattigo library.

Keywords: Homomorphic encryption · Threshold access-structures

1 Introduction

Multiparty Homomorphic Encryption (MHE) enables computations on encrypted data provided by multiple users, without requiring decryption. By generalizing traditional single-party homomorphic encryption (HE) to multiple users, MHE techniques constitute a promising family of solutions for the secure multiparty computation setting (MPC), where N parties aim to compute a function value over their joint inputs while keeping these inputs private. Notably, these MHE-based solutions have a low number of interaction rounds and low communication complexity, and they are compatible with the paradigms of cloud-computing [MTPBH21,AJLA+12] such as light-client/powerful-server types of architecture.

Several generations of MHE schemes were proposed over the years, generally following the advances of single-party HE constructions. As the most recent generation of HE schemes, based on ring-learning-with-errors (RLWE), has now reached several application domains and is being implemented and standardized, recent works have also brought multiparty variants of these schemes [MTPBH21]. Among these multiparty schemes, *threshold* schemes [MTPBH21] have been demonstrated as particularly efficient and are already included in several open-source implementations [MBTPH20,pal].

MHE-based MPC. Multiparty homomorphic encryption techniques can be used to construct efficient secure multiparty computation protocols, commonly referred to as *two-round MPC*. These MHE-based MPC protocols consist in a one-time *Setup* phase, after which any number of function evaluations can be performed in a two-round online phase [AJLA+12].

In the *Setup* phase, the parties make use of a special-purpose multiparty protocol in order to generate a collective public key that supports encryption and homomorphic evaluation, and for which the corresponding secret-key is securely distributed among the parties. The online, input-dependent phase consists in three steps: *Input*, *Evaluation*, and *Output*. During the *Input* phase, the parties use the collective public encryption key to encrypt their inputs and disclose the resulting ciphertexts to the other parties. Then, the desired computation is carried out (non-interactively), by using the homomorphic operations of the HE scheme. Finally, the parties take part in a multiparty protocol to decrypt the result ciphertext(s) in the *Output* phase. Contrary to their counterparts based on linear secret-sharing schemes (LSSS) or garbled circuits, the offline *Setup* phase of MHE-based MPC solutions produces public keys that can be reused for an unlimited number of function evaluations.

Mouchet et al. propose a RLWE-based MHE scheme in which the secret key is additively shared between the parties and for which the threshold-decryption protocol requires a single round of interaction [MTPBH21]. They show that, as for its precursor based on learning-with-errors (LWE) [AJLA+12], this scheme has a fully public transcript and can support MPC tasks over any public authenticated channel. As a result, this scheme can support computation among a large number of resource-limited parties by using a third-party honest-but-curious cloud provider that acts as a share aggregator (for the *setup* and *output* protocols) and homomorphic evaluator (for the *input* and *evaluation* phases).

Access Structures. For a secret-sharing scheme over a set of parties $\mathcal{P}$, we refer to a subset $S \subset \mathcal{P}$ of parties that can reconstruct the secret as a *qualifying set* and to the set $A \subset \mathsf{Powerset}(\mathcal{P})$ of all qualifying sets as the *access structure* of the scheme. The access structure to the secret key of an MHE scheme determines the access structure to the encrypted inputs which, in turn, determines the security properties of the corresponding MPC protocol instance. The scheme of Mouchet et al. uses an additive structure for its secret key, which instantiates an N-out-of-N-threshold access-structure: all parties have to collaborate for the decryption protocol to succeed. Although this enforces the strictest access-structure (only

one qualifying set) hence provides strong security guarantees, this also requires more stringent availability requirements on the protocol participants. In practical systems involving many parties, we would typically want to extend the semi-honest model with the case of parties going offline for an undetermined amount of time (e.g., due to technical issues). For scenarios in which a fraction $\frac{t}{N}$ of honest participants can be guaranteed, t-out-of-N-*threshold* access-structures can relax this requirement by enabling decryption (and other types of secret-key operations) to be performed among subgroups of at least t parties.

1.1 Our Results

In this work, we introduce a t-out-of-N-*threshold* homomorphic encryption scheme based on RLWE. We contribute our scheme as a simple and efficient extension to the N-out-of-N-threshold scheme of Mouchet et al. [MTPBH21] that relaxes its access structure to a t-out-of-N-*threshold* one. We also contribute its implementation in the Lattigo library [lat22,MBTPH20] and evaluate its performance.

The t-out-of-N-Threshold Scheme. We propose a set of procedures that extend, in a natural and efficient way, the scheme of Mouchet et al. to a t-out-of-N-threshold access structure. We follow the known approach of *re-sharing* the secret-key shares with the Shamir secret-sharing scheme [Sha79], but with a specially adapted instance of the Shamir secret-sharing that we define over the ciphertext-space ring. This enables us to take advantage of the linearity of the MHE scheme's secret-key operations to pre-aggregate the shares, which makes the re-sharing scheme compact and efficient. The resulting procedure adds a single round of interaction during the offline setup phase and a simple non-interactive pre-computation step in the MHE operations that depend on the shares of the secret-key. Our construction is generic and can be used to instantiate multiparty variants of the BGV, BFV and CKKS schemes.

Implementation and Benchmarks. We implemented our construction using Lattigo [MBTPH20], an open-source library for multiparty homomorphic encryption. We report on the benchmark performance for our implementation and analyze the results in the context of MHE-based MPC. Furthermore, we show how to harness the t-out-of-N-threshold access-structure to accelerate the execution of *batches* of secret-key operations in both the offline-setup and online phases. We exemplify this through the task of generating a public bootstrapping key for the multiparty CKKS scheme, which requires generating a batch of more than one hundred rotation keys to support the necessary automorphisms.

The remainder of this paper is organized as follows: We review the existing works on threshold encryption for lattice-based construction in Section 1.2, and provide the necessary background in RLWE-based MHE and secret-sharing techniques in Section 2. Then, we develop the main technique, in Section 3, and its implementation and evaluation, in Section 4.

1.2 Related Work

Bendlin and Damgård considered the case where the parties obtain Shamir secret-shares of a secret-key by means of pseudo-random secret sharing (PRSS) techniques [BD10]. This results in a non-interactive secret-key-generation procedure, but it is non-compact as it requires one key per possible subset of adversarial parties. Due to this factorial expansion, this scheme would not be practical for large number of parties.

Asharov et al. noticed that share-re-sharing could be used to achieve a t-out-of-N-threshold access structure in (the extended version of) their seminal work on LWE-based multiparty homomorphic encryption [AJLA$^+$12]. However, they did not specify the concrete secret-sharing scheme and assumed an extra round of interaction, prior to the decryption round, to reconstruct a failing party's share. Additionally, directly reconstructing the shares is undesirable in practice, as it would reveal the failing party's share to the parties. We show that this is not needed in practice, as reconstruction can be performed within the secure decryption protocol directly.

Boneh et al. propose a t-out-of-N-threshold homomorphic encryption scheme based on learning-with-errors that also relies on re-sharing the secret-key shares yet in a *stronger* setting where parties are unable, at the time of generating their decryption shares, to determine which other parties are online [BGG$^+$18]. This additional constraint is necessary for the composability of their scheme, that they use as a building-block for higher-level cryptographic primitives in their work. However, it comes with a significant complexity and performance overhead, and their setup phase requires a trusted dealer to perform the sharing. We elaborate on these differences in Section 4.1, where we provide a comparison between their construction and our scheme.

In their work, Boneh et al. observe that enabling the parties to determine, before the decryption phase, which parties are online would lead to a simpler scheme. We confirm this observation by showing that, in the semi-honest model with failures, there indeed exists a much simpler and more efficient scheme that does not require a trusted dealer.

2 Preliminaries

We first present our system and adversary model, as well as the main system goals. Then, we present the main building blocks of our solution.

2.1 Adversary Model and System Goals

We consider a set $\mathcal{P}$ of N parties $\{P_1, ..., P_N\}$ (*the system*) in a secure-multiparty-computation setting, where an adversary $\mathcal{A}$ is able to corrupt up to $t-1$ parties. We assume that the adversary is static and passive, yet we further enable the adversary to take the corrupted parties offline for an arbitrary amount of time. The parties can communicate through private authenticated channels and through a public, synchronous, authenticated channel. Finally, we assume that the parties have access to a public Common Reference String (CRS).

System Goals. Let $x_i \in \mathcal{M}$ be the private input of party P_i in some message space $\mathcal{M}$, and $f : \mathcal{M}^N \to \mathcal{M}$ be a public arithmetic function over the message space. We formulate the following system goals:

Functionality. The system must compute $y = f(x_1, ..., x_N)$ through a multiparty protocol.

Privacy. There must exist a simulator program SIM_f that can simulate all the interactions between the parties, when provided only with the output y and the inputs from the adversary. For an attacker to distinguish between the real and simulated interaction, the success probability must be lower than 2^{-128}.

Fault Tolerance. After the inputs are received for all parties, the output y should be delivered to the honest parties as long as at least t parties are online and active.

Efficiency. The space, time, and communication complexity must be at most quadratic in the number of parties for the setup phase; and the parties should be able to perform any number of function evaluations after it is complete. For a single evaluation of the function in the online phase, we require that the time and communication complexity do not exceed those of an equivalent plaintext multiparty circuit by more than a linear term in the number of parties.

Informally, the protocol execution should not reveal anything more about the inputs than that which can be deduced from the output y alone. We also observe that the fault-tolerance requirement, *guaranteed output-delivery*, is limited to the case where faulty parties provided their inputs before going offline. This is because not all functions can be successfully computed under partial inputs.

We now briefly introduce the building blocks of our construction: the scheme of Mouchet et al. [MTPBH21], its instantiation as an MPC protocol, and the secret-sharing scheme of Shamir [Sha79].

2.2 *N*-out-of-*N*-Threshold Encryption for RLWE

We recall the notation and core procedures of the RLWE N-out-of-N-threshold Encryption scheme (MHE Scheme) [MTPBH21] that we extend in Section 3. Its ciphertext space is a polynomial quotient ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ where the polynomial degree n is a power of two and where the polynomial-coefficient modulus q is a product of L different primes $q_1, ..., q_L$. Hence, we can use the isomorphism $R_q \cong R_{q_1} \times ... \times R_{q_L}$ provided by the Chinese remainder theorem (CRT) to perform the operations in the residue rings, without resorting to arbitrary-precision integer arithmetic. Moreover, we chose each q_i such that $q_i \equiv 1 \mod 2n$, which enables the existence of a number-theoretic transform (NTT), under which both ring operations can be performed coefficient-wise. We denote $a \leftarrow \mathsf{D}$ the sampling of a according to a distribution D. We simplify this notation for the case of uniform sampling of a ring element that we denote $a \leftarrow R_q$. Let $\mathsf{Key}(R_q)$ be a secret-key distribution over R_q for which the coefficients are sampled uniformly in $\{-1, 0, 1\}$, let $\mathsf{Err}(R_q)$ be an error distribution where the coefficients are sampled from a discrete Gaussian distribution of small variance σ^2, and let $\mathsf{Smudge}(R_q)$ be a suitable *smudging* distribution for the noise flooding technique

[AJLA+12,MTPBH21] (typically, a discrete Gaussian distribution of large variance). Finally, let $\mathsf{CRS}(R_q)$ be the uniform distribution according to the common reference string (i.e., elements sampled from this distribution are the same for all parties).

Scheme: MHE

- $\mathsf{MHE.Setup}$: The parties agree on the public parameters $(n, q, \sigma, \mathsf{Key}, \mathsf{Err})$.
- $\mathsf{MHE.SecKeyGen}$: Each party P_i samples $s_i \leftarrow \mathsf{Key}(R_q)$.
- $\mathsf{MHE.PubKeyGen}(s_1, ..., s_N)$:
  1. Each party P_i samples $p_1 \leftarrow CRS(R_q)$, $e \leftarrow \mathsf{Err}(R_q)$ and discloses $p_{0,i} = -s_i p_1 + e$.
  2. Each party computes $p_0 = \sum p_{0,i}$ and sets $\mathsf{pk} = (p_0, p_1)$.
- $\mathsf{MHE.Encrypt}(\mathsf{pk}, m)$: Sample $u \leftarrow \mathsf{Key}(R_q)$, $e_0, e_1 \leftarrow \mathsf{Err}(R_q)$ and output $\mathsf{ct} = (c_0, c_1) = (m + up_0 + e_0, up_1 + e_1)$.
- $\mathsf{MHE.Decrypt}(\mathsf{ct}, s_1, ..., s_N)$:
  1. Each party P_i samples $e_i \leftarrow \mathsf{Smudge}(R_q)$ and discloses $h_i = c_1 s_i + e_i$.
  2. Each party can then compute $m \approx c_0 + \sum h_i$.

We refer to $s = \sum_{i=1}^{N} s_i$ as the *ideal secret-key* for the scheme. As the full collective knowledge of s is required to decrypt ciphertexts, the MHE scheme implements an N-out-of-N-threshold access-structure over its ciphertexts. More generally, we refer to the secret-key-dependent operations of the scheme as *secret-key operations*. Note that we omitted the $\mathsf{MHE.Eval}$ procedure as it depends on the specific plaintext-encoding strategy of the RLWE scheme in use and as it is independent of the access structure (we briefly discuss the encoding strategy below).

Plaintext Encoding and Homomorphic Evaluation. Note that, in RLWE HE schemes, the decryption procedure yields an approximate message, due to the inherent ciphertext error. The way to encode a plaintext into a message m and to decode it back after decryption is specific to the scheme in use. Common strategies include scaling the plaintext up by a factor Δ and rely on quantization and roundings for the decoding [CKKS17,FV12]. Furthermore, it is common to apply FFT-like transforms to the plaintext polynomials in order to enable coefficient-wise encrypted arithmetic. Such techniques, often referred to as *packed* encoding, enable users to encode $\mathbb{Z}_q$ messages into up to n independent *slots*, where n is the polynomial degree. The chosen encoding strategy defines how the homomorphic operations are performed (i.e., the specific Eval algorithm). Yet, these considerations are independent of the secret key and the core MHE scheme can be used to instantiate multiparty variants of the BFV [Bra12,FV12], CKKS [CKKS17] or BGV schemes [BGV14]. Our t-out-of-N-threshold access-structure will preserve this property.

Evaluation Keys Some homomorphic operations require the evaluator to be provided with operation-specific public-keys, often referred to as *evaluation keys*. For example, compact multiplication involves a relinearization operation [FV12]

which requires a so-called *relinearization key*. Likewise, plaintext slots rotation can be operated as an homomorphic automorphism which requires a *rotation-key* per rotation amount. Although generating a single key for a one-slot rotation would suffice to operate any rotation in theory, it is more efficient to generate keys for all (or most) of the rotations required by the circuit, in order to operate all (or most) rotations in constant-time. We refer the reader to the original scheme [MTPBH21] for details about the generation of evaluation keys (they are straightforward adaptation of the MHE.PubKeyGen procedure). In the scope of this work, suffice to observe that these procedures are secret-key operations and that generating many rotation-keys (e.g., as required by the bootstrapping operation) represents a significant cost. In Section 3.5, we observe that this cost can be efficiently distributed among the parties by taking advantage of the t-out-of-N-threshold access-structure.

Secure Multiparty Computation. The MHE scheme directly yields a generic secure multiparty computation protocol in the *two-rounds MPC* model. This model often comprises two phases, the first being input-independent and optional in the PKI setting (hence is usually not counted as one of the two rounds).

In the offline *Setup* phase, the parties run the MHE.Setup, MHE.SecKeyGen and MHE.PubKeyGen procedures. The output of this phase are the parties' individual secret-keys and a set of collective public encryption- and evaluation-keys that can be used for an unlimited number of iterations of the second phase.

In the *Online* phase, the parties use the MHE.Encrypt to encrypt their private inputs to the computation and send the resulting ciphertexts to the other parties. Then, the function evaluation is performed under encryption by using the Eval algorithm of the scheme in use. Finally, the parties use the MHE.Decrypt procedure to output the final result.

Within our system model, the MHE-based MPC protocol satisfies the *functionality*, *privacy*, and *efficiency* system goals of Section 2.1, but not the *fault tolerance* one.

Fault Tolerance. The MHE-MPC protocol naturally provides *some* fault tolerance against parties going offline for a finite amount of time. As opposed to its LSSS-based counterparts, a party that goes offline after providing its inputs does not prevent the computation from making progress, as the homomorphic evaluation is performed non-interactively. The same is true for a party that crashes after the *Offline* phase, except that, similarly to the plaintext case, the party's input will not be available to the computation. In both cases, the main drawback is that all parties need to connect *eventually* (to participate in the decryption protocol of the output phase) for the output to be delivered. This might be problematic in settings where a group of parties seek to tolerate a fraction of them going offline for an undetermined amount of time. In our construction, we use the Shamir secret-sharing scheme to solve this problem.

2.3 Shamir Secret-Sharing

We recall the secret-sharing scheme of Shamir that implements a t-out-of-N-threshold access-structure on its secrets, based on polynomial interpolation in a finite field. For the sake of notation, we consider the reconstruction from the first t shares. Indeed, the procedure generalizes to any set of at least t shares.

- $\mathsf{Shamir.Setup}$: The parties agree on a field K and each party $P_i \in \mathcal{P}$ is associated with a non-zero element $\alpha_i \in K$ such that for $i \neq j$ then $\alpha_i \neq \alpha_j$.
- $\mathsf{Shamir.Share}(s,\ t,\ \alpha_1, ..., \alpha_N)$: To share a message $s \in K$ among N parties such that t shares are needed to reconstruct s, sample $c_1, ..., c_{t-1} \leftarrow K$ and sends $s_i = s + \sum_{k=1}^{t-1} c_k \alpha_i^k$ to party P_i.
- $\mathsf{Shamir.Combine}(s_1, ..., s_t,\ \alpha_1, ..., \alpha_t)$: To reconstruct a message s from shares $s_1, ..., s_t$, compute

$$s = \sum_{i=1}^{t} s_i \prod_{j=1, j\neq i}^{t} \frac{\alpha_j}{\alpha_j - \alpha_i}. \tag{1}$$

We observe that the $\mathsf{Shamir.Share}$ procedure samples a degree-$(t-1)$ polynomial $S(X) \in K[X]$ such that $S(0) = s$ and distributes $S(\alpha_i)$ to party P_i, and the $\mathsf{Shamir.Combine}$ procedure computes the Lagrange interpolation at point $X = 0$ to reconstruct the secret. We refer to the sequence of public points $(\alpha_1, ..., \alpha_N)$ as the *Shamir public-points*.

3 t-out-of-N-Threshold Encryption for RLWE

We now present our main contribution. We provide an overview of the main ideas behind the scheme in Section 3.1. Then, we present the secret-sharing scheme that we use for the share re-sharing in Section 3.2. Finally, we present our t-out-of-N-Threshold Encryption for RLWE in Section 3.3.

3.1 Overview

The idea is to apply the Shamir secret-sharing scheme to the additive shares of the *ideal secret-key* s of the MHE scheme. Intuitively, this technique, often referred to as *share re-sharing*, enables any set of at least t parties to reconstruct the shares of the missing parties and to take their place in the decryption procedure. However, a naive instantiation of this idea that would use an arbitrary secret-sharing space would be inefficient: It would require the non-failing parties to either reconstruct the shares of the failing parties (which would forever remove them from the access structure and add a communication round) or to compute their shares by running a secure computation over the secret-sharing space (which would require to emulate R_q arithmetic over this space). Also, it would require each party to store all N re-shares throughout the entire protocol.

Instead, we perform Shamir re-sharing directly over the ring R_q. In this way, we can exploit the linearity of both the ideal secret-key and the re-sharing

scheme to obtain a much more compact and communication-efficient scheme. More specifically, assuming R_q is our Shamir secret-sharing space, we denote $S_i \in R_q[X]$ the secret degree-$(t-1)$ polynomial sampled by party P_i during the Shamir.Share procedure, and $\lambda_i = \prod_{j=1, j\neq i}^{t} \frac{\alpha_j}{\alpha_j - \alpha_i}$ be the i-th Lagrange coefficient in the reconstruction using the Shamir public-points $\alpha_1, ..., \alpha_t$. Then, the Shamir.Combine operation commutes with the ideal-secret-key reconstruction:

$$s = \sum_{i=1}^{N} s_i = \sum_{i=1}^{N} \sum_{j=1}^{t} S_i(\alpha_j)\lambda_j = \sum_{j=1}^{t} \lambda_j \sum_{i=1}^{N} S_i(\alpha_j) = \sum_{j=1}^{t} s'_j. \tag{2}$$

Remark 1. The Shamir secret-sharing scheme is usually defined over an arbitrary field, which guarantees the correctness and security of the Lagrange interpolation for enforcing the access structure. However, there are no such guarantees over arbitrary rings. For Eq. (2) to be correct and the resulting scheme to be secure, we need to show that these properties hold in the ring R_q.

Remark 2. By rearranging the terms in Eq. (1), we observe that the new sharing over t parties has an additive structure for which the j-th term can be locally (pre-)computed by each $P_j \in \mathcal{P}_t$ if the set of participating parties is known before performing the secret-key operation.

Remark 3. The newly computed t-out-of-N share s'_i can be seen as a new additive sharing of s and can simply be used by the parties instead of s_i, their N-out-of-N counterpart, in the usual MHE decryption protocol.

We present the concrete Shamir secret-sharing scheme in Section 3.2 and show that it satisfies the requirements of a secret-sharing scheme (as per Remark 1). Then, we present our t-out-of-N-threshold scheme; we can formulate it as a direct extension of the N-out-of-N-threshold MHE scheme for RLWE (due to Remarks 2 and 3).

3.2 Shamir Secret-Sharing in R_q

We begin by detailing how Shamir secret-sharing can be instantiated over R_q. Then, we define the concrete scheme that we use for share re-sharing.

Shamir Secret-Sharing in a Ring. The usual Shamir secret-sharing scheme is instantiated over a field. This guarantees that all non-zero elements are units hence that Lagrange coefficients exist. Indeed, computing a Lagrange coefficient requires inverting elements of the form $\alpha_i - \alpha_j$ where α_i and α_j are the Shamir public-points.

However, working in a field is not a requirement. In fact, it is a known result that using a ring is possible, as long as the set of Shamir public-points form an *exceptional sequence* [ACD+19,CDN15]. We now briefly present this result that is a direct translation of the result of [ACD+19] in our notation and terminology.

Definition 1. *(From [ACD+19]) For a ring R, the sequence $\alpha_1, ..., \alpha_N$ of elements of R is an exceptional sequence if $\alpha_i - \alpha_j$ is a unit in R for all $i \neq j$.*

Theorem 1. *(From [ACD+19]) Let R be a commutative ring and $\alpha_1, ..., \alpha_N$ be an exceptional sequence in R. Then, a Shamir secret-sharing scheme instantiated in R with Shamir public-points, $\alpha_1, ..., \alpha_N$, is correct and secure.*

Let us assume that $\alpha_1, ..., \alpha_N$ is an exceptional sequence for R_q. Then, by instantiating a t-out-of-N Shamir secret-sharing scheme that uses the elements of this exceptional sequence as the Shamir public-points, we obtain from Theorem 1 that our secret-sharing scheme for R_q is correct and secure for a threshold access-structure. Intuitively, the inversion of the elements $\alpha_i - \alpha_j$ was the only operation of the Shamir secret-sharing scheme that was not guaranteed to work in a ring. By restricting our choice of Shamir public-points, we obtain a correct and secure variation of the Shamir secret-sharing scheme in a ring.

It remains to define a way of sampling Shamir public-points from R_q that would guarantee that the obtained elements form an exceptional sequence.

Choice of Shamir public-points. We first observe that checking whether an arbitrary sequence of R_q elements form an exceptional sequence is easy: For each non-zero pairwise differences, it suffices to check that all coefficients of the difference polynomial under the CRT and NTT representation is non-zero. This holds because the inverse of each non-zero coefficient can be computed individually by the little Fermat theorem. However, computing these inverses for arbitrary elements of R_q would represent a costly operation that would result in an inefficient Combine operation.

Instead, we propose to restrict the choice of Shamir public-points to constant polynomials in $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ (i.e., polynomials of the form αX^0 for $\alpha \in \mathbb{Z}_q^*$). On the one hand, it yields a huge performance boost as the multiplications are reduced to a single scalar computation in $\mathbb{Z}_q$. On the other hand, this provides us with a simple procedure for choosing Shamir public-points that guarantee an exceptional sequence. Let $q_{min} = \mathsf{min}(q_1, ..., q_L)$ with $q_1, ..., q_L$ the prime factors of q. We observe that for $N < q_{min}$, choosing N distinct values in $\mathbb{Z}_{q_{\mathsf{min}}}$ as the Shamir public-points will guarantee an exceptional sequence. Indeed, for any $i \neq j$, $-q_{min} < \alpha_i - \alpha_j < q_{min}$, $\alpha_i - \alpha_j \neq 0$ and the residue $\mathrm{mod} q_k$ is non-zero for any prime factor q_k of q. Then, a simple application of the CRT on R_q is enough to prove that $\alpha_i - \alpha_j$ is a unit in R_q.

Therefore, any mapping from $\mathcal{P}$ onto $\mathbb{Z}_{q_{\mathsf{min}}}$ can be used, including the *textbook* Shamir secret-sharing one that commonly uses i for party P_i, when $i > 0$. We observe that it is critical for implementations to check that Shamir public-points are non-zero.

3.3 Scheme Extension

We present our t-out-of-N-threshold scheme for RLWE, which we formulate as an extension of the N-out-of-N-threshold scheme of Mouchet et al. [MTPBH21].

We first present the re-sharing scheme, which uses the Shamir sharing over R_q that we introduced in Section 3.2. Then, we show how to combine the re-sharing scheme with the MHE scheme of Section 2.2.

Share Re-sharing Scheme. For a set of parties $\mathcal{P}$ in the MHE scheme where $P_i \in \mathcal{P}$ holds secret-key share s_i, we define our re-sharing scheme as the three-tuple of procedures (Setup, Thresholdize, Combine) defined in Scheme T. Informally, Scheme T applies our Shamir secret-sharing scheme over R_q to the parties' key, which *relaxes* the N-out-of-N access-structure of the MHE scheme of Section 2.2 to a t-out-of-N-threshold one.

Scheme: T

- T.Setup: Each party $P_i \in \mathcal{P}$ is associated with a public point $\alpha_i \in R_q$ such that $\alpha_i - \alpha_j$ is a unit for all i, j, $i \neq j$.
- T.Thresholdize(t, $s_1, ..., s_N$, $\alpha_1, ..., \alpha_N$):
  1. Each party P_i samples $c_{i,1}, ..., c_{i,t-1} \leftarrow R_q$.
  2. Each party P_i sends $\tilde{s}_{i,j} = s_i + \sum_{k=1}^{t-1} c_{i,k}\alpha_j^k$ to each party P_j.
  3. Each party P_i receives $\tilde{s}_{j,i}$ from each party P_j and computes $\tilde{s}_i = \sum_{j=1}^{N} \tilde{s}_{j,i}$.
- T.Combine($\tilde{s}_1, ..., \tilde{s}_t$, $\alpha_1, ..., \alpha_t$): Each party $P_i \in \mathcal{P}_{online}$ where $\mathcal{P}_{online} \subseteq \mathcal{P}$ and $|\mathcal{P}_{online}| \geq t$ computes $s'_i = \tilde{s}_i \prod_{j=1, i\neq j}^{t} \frac{\alpha_j}{\alpha_j - \alpha_i}$.

We observe that the output of the T.Thresholdize is only one ring element per party, due to the re-share being aggregatable. This is formalized as Remark 4.

Remark 4. The summation in N on the right-hand side of Eq. (2) does not depend on which t of the N parties participate in the reconstruction and can be pre-computed by each party P_i, after it receives all the $S_j(\alpha_i)$ from its peers.

We also observe that only the T.Thresholdize procedure is interactive and that it requires a single round of pairwise interactions between the parties over confidential channels. Once performed, the parties have access to Shamir shares $(\tilde{s}_1, ..., \tilde{s}_N)$, from which each party P_i can locally compute its share s'_i in an additive sharing $(s'_1, ...s'_t)$ of s among any subgroup of at least t parties in $\mathcal{P}$ (as per remark 2). Consequently, each party P_i can simply use its new share s'_i directly in the MHE procedures. This is the main idea for our next construction.

t-out-of-N-Threshold MHE scheme. As per Remark 2, the T.Combine procedure requires each party to obtain the set of participating parties from the environment. We formalize this requirement by providing the parties with an oracle access to the set of online parties. We denote $\mathcal{P}_{online} \leftarrow$ Env such an oracle query where $\mathcal{P}_{online} \subseteq \mathcal{P}$ is the set of online parties. In our model, this oracle can be realized with a simple broadcast round of communication to gather the identities of online parties, yet with the small caveat that, after this broadcast round, the parties might fail. In Section 3.4, we discuss how to deal with faulty oracles that return an incorrect set of online parties.

The resulting t-out-of-N-threshold encryption scheme can be expressed as the union tuple $\mathsf{MHE} \cup \mathsf{T}$, which we now detail as TMHE.

Scheme: TMHE

- TMHE.Setup: run the MHE.Setup and T.Setup procedures.
- TMHE.SecKeyGen:
  1. run $(s_1, ..., s_N) \leftarrow$ MHE.SecKeyGen.
  2. run T.Thresholdize$(t, s_1, ..., s_N, \alpha_1, ..., \alpha_N)$.
- TMHE.PubKeyGen$(\tilde{s}_1, ..., \tilde{s}_t)$:
  1. obtain $\mathcal{P}_{online} \leftarrow$ Env
  2. if $|\mathcal{P}_{online}| < t$, return $\perp$
  3. choose t parties $\mathcal{P}_{online}$ and run $(s'_1, ..., s'_t) \leftarrow$ T.Combine
  4. execute the MHE.PubKeyGen$(s'_1, ..., s'_t)$ protocol.
- TMHE.Encrypt(pk, m): run the MHE.Encrypt procedure.
- TMHE.Decrypt(ct, $\tilde{s}_1, ..., \tilde{s}_t$):
  1. obtain $\mathcal{P}_{online} \leftarrow$ Env
  2. if $|\mathcal{P}_{online}| < t$, return $\perp$
  3. choose t parties $\mathcal{P}_{online}$ and run $(s'_1, ..., s'_t) \leftarrow$ T.Combine
  4. execute the MHE.Decrypt$(\mathsf{ct}, s'_1, ..., s'_t)$ protocol.

TMHE-*based MPC protocol.* The instantiation of an MPC protocol from our scheme is the same as for the MHE scheme of Mouchet et al., yet it satisfies the *fault tolerance* requirement of Section 2.1. This is, it tolerates up to $N-t$ parties going offline for an undetermined amount of time, as long as the failing parties completed the TMHE.SecKeyGen procedure and provided their encrypted inputs to the computation. We elaborate on the differences between the TMHE and MHE instantiations in Section 4.1.

3.4 Dealing with Faulty Oracles

Our model does not exclude the possibility of a party crashing after the oracle response. In such a case, step 4 of the TMHE.Decrypt cannot be completed due to missing share(s) in the disclose phase of the MHE.Decrypt protocol. In practice, such a failure is generally detected and resolved by setting a time limit (timeout) for the parties to provide their decryption shares, and by defining the parties' behaviour in the case of such timeouts. Whereas the exact values for the timeout are indeed application dependant, we now discuss how parties can react to such timeouts to guarantee the eventual decryption of a ciphertext in a secure way.

Let $\mathcal{P}_{timeout}$ be the set of parties for which a timeout occurred; a partial yet insecure solution is to repeat the steps 3 and 4 with $\mathcal{P}'_{online} \leftarrow \mathcal{P}_{online} \setminus \mathcal{P}_{timeout}$ where $\setminus$ denotes the set difference. As such, this solution is insecure because the underlying MHE.Decrypt procedure is not secure under the composition of several decryptions of the same ciphertext $\mathsf{ct} = (c_0, c_1)$ (informally, $(sc_1 + e_1, sc_1 + e_2)$ leaks information about sc_1 when e_1 and e_2 are sampled independently). However, the key observation is that obtaining a new ciphertext ct' such that

$\mathsf{Dec}(\mathsf{ct}) = \mathsf{Dec}(\mathsf{ct}')$ is easy with any additive homomorphic scheme. Hence, we propose to add a re-randomization step by adding a fresh encryption of zero to the target ciphertext before repeating the $\mathsf{MHE.Decrypt}$ step.

We observe that, if $\mathcal{P}_{online} \cap \mathcal{P}'_{online} \neq \emptyset$, the parties in the intersection can efficiently update their local share s'_i from the previous iteration by computing

$$s'_i \leftarrow s'_i \prod_{P_j \in \mathcal{P}_{timeout}} \frac{\alpha_j - \alpha_i}{\alpha_j} \prod_{P_j \in \mathcal{P}'_{online} \setminus \mathcal{P}_{online}} \frac{\alpha_j}{\alpha_j - \alpha_i},$$

which is more efficient than running the full $\mathsf{T.Combine}$ procedure when the size of the intersection is larger than one. Indeed, as computing the Lagrange coefficient is a scalar function, the acceleration would be noticeable for only large values of the $\frac{N}{|\mathcal{P}_{timeout}|}$ ratio.

3.5 Accelerating Batched Multiparty Secret-Key Operations

The t-out-of-N-Threshold access-structure also enables the group of key-share holders to efficiently parallelize batches of secret-key operations, when more than t participants are online. Performing batches of secret-key operations is common in MHE-based MPC protocols:

- At the Setup phase - when the parties have to generate a number of key-switching keys (often referred to as *evaluation key*) to support non-linear operations such as ciphertext-ciphertext multiplication and ciphertext-slot rotations.
- At the Evaluation phase - if the parties rely on interactive protocols to reduce the noise or to raise the level of ciphertexts as a part of the circuit in order to avoid the overhead of using bootstrapping [MTPBH21,SPTP$^+$20]. These protocols usually perform masked decryption and re-encryption on each ciphertext that is required to continue the evaluation.
- At the Output phase - when the function's output consists in multiple ciphertexts. This could be by design or because the encryption parameters do not enable packing enough values in one ciphertext.

Let k be the number of secret-key operations to be performed (e.g., the number of rotation keys to be generated), and let $\mathcal{P}_{on}$ be the set of online parties. The parties in $\mathcal{P}_{on}$ can be organized into k subgroups of t distinct parties, and the work can be distributed among the subgroups. Mouchet et al. show that the overhead of running one MHE secret-key operation protocols within each subgroup of size t can be made constant in t for each party, by relying on tree-based share aggregation patterns [MTPBH21]. Hence, the total overhead for each party in performing the k secret-key operations can be reduced to $(kt)/\mathcal{P}_{online}$, which is $t/\mathcal{P}_{online}$ times the overhead of performing these same k operations in the N-out-of-N-threshold scheme. We evaluate the effect of using this technique on the task of generating a public bootstrapping key for the CKKS scheme, in Section 4.3.

4 Evaluation

We now discuss our proposed construction from the theoretical and practical standpoints.

4.1 Theoretical Evaluation

We provide a theoretical comparison between our proposed TMHE scheme and the existing constructions. We first study the overhead and additional assumptions of the threshold scheme, with respect to the original MHE scheme. Then, we discuss the main differences between the threshold scheme of Boneh et al. and our proposed construction.

Comparison with the Base MHE *Scheme.* From the system-model standpoint, the main difference between the TMHE scheme, and the base MHE scheme of Mouchet et al. [MTPBH21] is indeed that our construction enables t-out-of-N access structures. Hence, instantiating the MHE-based MPC protocol with our scheme satisfies the *fault tolerance* requirement of Section 2.1. Moreover, the TMHE-based instantiation retains most of the features from the MHE-based one: (a) Its *offline* phase is re-usable and has to be performed only once for a given set of parties and encryption parameters. (b) Its *online* phase has a fully public transcript and consists in only two rounds of interaction among the parties. However, the TMHE.SecKeyGen relies on confidential communication channels between the parties (to execute the T.Thresholdize re-sharing procedure), which is not the case for the original MHE.SecKeyGen procedure. In other words, the TMHE-based MHE-MPC protocol does not have a fully public transcript in its *offline* phase, whereas the MHE-based one does. However, private communication is required for only a single round of communication and is not a major obstacle in many peer-to-peer and cloud-assisted models.

From the computational-cost standpoint, the threshold extension requires additional state to be stored and exchanged by each party. We summarize the related costs in Table 1. The TMHE.SecKeyGen is the only operation where this overhead is not negligible: It requires each party to store a degree-$(t-1)$ polynomial in $R_q[X]$, to evaluate this polynomial N times (for X a degree-0 polynomial), and to send and receive $N-1$ Shamir secret shares. Whereas, the base scheme does not require any interaction to generate the secret-key. The fact that the key-generation phase is only a one-time offline phase that is re-usable for any number of circuit-evaluation enables the amortization of this step in many applications. Regarding secret-key operations (PubKeyGen and Decrypt), the only overhead is the local computation of the Combine procedure that is $\mathcal{O}(t)$. This overhead, however, is close to negligible in practice. This is because the computation of the Lagrange coefficient, which is done over $\mathbb{Z}_q$ thanks to the compact Shamir public-points, is the only part of this computation that depends on t. We demonstrate this by benchmarking our implementation, in Section 4.2.

Table 1. Threshold extension costs, measured in number of R_q elements per-party for the internal state and network communication, and in asymptotic function of N and t for the per-party computational cost. We distinguish between the costs associated with the generation (SecKeyGen) operation and the usage (SecKeyOp $\in$ {PubKeyGen, Decrypt}) of the secret-key.

	Party's state		Network cost per party		Comp. cost	
	SecKeyGen	SecKeyOp	SecKeyGen	SecKeyOp	SecKeyGen	SecKeyOp
MHE	1	1	0	1	$\mathcal{O}(1)$	$\mathcal{O}(1)$
TMHE	t	1	$2(N-1)$	1	$\mathcal{O}(t+Nt+N)$	$\mathcal{O}(t)$

Comparison with the Related Work. The threshold MHE scheme of Boneh et al. [BGG+18] assumes a more general setting, where parties do not have to know the set of online parties before generating their shares during secret-key operations. This is necessary to achieve their goal of building a universal thresholdizer for cryptographic primitives, for which the security properties of the thresholdized primitive depend on the composability of the MPC protocol. Their solution is essentially to homomorphically perform the Lagrange interpolation, when aggregating the shares. But such an aggregation can only be performed when the Lagrange coefficients are small with respect to q. Therefore, their first solution consists in using a $\{0,1\}$-LSSS to share the secret key of the scheme. For t-out-of-N-threshold access-structure, this implies a per-party state in $\mathcal{O}\big(N^{4.2}\big)$ to store the secret-key shares. Their second solution consists in using Shamir secret-sharing, which requires only a $\mathcal{O}(1)$ storage for the secret-key shares (assuming a trusted setup). However, this requires increasing the size of the modulus q by a $\mathcal{O}\big(N!^3\big)$ multiplicative factor, thus rendering the encryption scheme non-compact and more difficult to parametrize (increasing the coefficient modulus while keeping the other parameters fixed reduces the security of RLWE).

Our scheme requires a $\mathcal{O}(1)$ storage for the secret-key shares without requiring a trusted dealer. It is also conceptually simple, which enabled its implementation in an existing library. Hence, it can be seen as trading-off composability for efficiency and simplicity. We now elaborate on this trade-off and on how to pick the right scheme for a given MPC application.

Applications. To map an MPC application to the correct side of the composability-versus-efficiency trade-off, we propose to distinguish between two categories of applications based on their ideal functionality. In the first category, *non-cryptographic functionalities*, the protocol is instantiated in a standalone way to compute non-cryptographic functions such as image processing or machine-learning model training and inference. In this category of applications, the primary requirement is for the MHE scheme to be cost efficient in terms of computation-time and network-load. In the second category, *cryptographic functionalities*, the protocol is instantiated as a building block to construct a higher-level cryptographic primitive. The composability of the MHE scheme is crucial for this category of applications, in order to prove the security of the resulting primitive, which is often at the cost of performance. Under this classification, we can easily

Table 2. Benchmarked HE Parameters. The polynomial degree n and coefficient modulus q size in bits are taken from the standardization document [ACC+18]. L is the number of prime factors of q.

Set	Pol. deg. (n)	Coeff. size (L)	Coeff. size ($\log_2 q$)
I	2^{13}	4	218
II	2^{14}	8	438
III	2^{15}	15	881

Table 3. Threshold extension benchmarks in milliseconds for the TMHE.SecKeyGen and TMHE.Decrypt (i.e., Thresholdize) procedures for $N = 20$ parties. These values represent the per-party CPU time.

	Param.	I			II			III		
	t	7	14	19	7	14	19	7	14	19
SecKeyGen	**Step 1**	6.1	13.0	18.1	26.4	57.5	79.4	92.9	201.3	278.0
	Step 2	4.4	8.9	12.0	17.6	35.4	48.6	69.8	148.1	201.8
	Step 3	0.2	0.2	0.2	0.9	0.9	0.9	3.4	3.4	3.4
	Total	10.7	22.1	30.4	44.9	93.8	128.9	166.2	352.8	483.2
Decrypt	**T.Combine**	<0.1	<0.1	<0.1	0.1	0.1	0.1	0.3	0.4	0.4
	MHE.Decrypt		0.8			2.8			11.6	
	Total	0.8	0.8	0.9	2.9	2.9	2.9	11.9	12.0	12.0

characterize the scheme of Boneh et al., that targets the construction of a general thresholdizer for cryptographic primitive, as being particularly adapted to cryptographic functionalities, due to its strong composability guarantees. Whereas our scheme lacks such composability, it satisfies the security and efficiency requirements of non-cryptographic functionalities that represent the *end-user* side of any cryptographic construction.

4.2 Basic Operations Benchmarks

We implemented the scheme extension T in the Lattigo library [MBTPH20], that implements the basic RLWE-based MHE scheme. We report on the performance figures for the T.Thresholdize and T.Combine for several common choices of encryption parameters (summarized in Table 2) and several values of the threshold t in Table 3. The benchmarks were conducted on an AMD Ryzen 9 5900X CPU (3.7GHz clock, 6M of L2-cache). We observe that the Thresholdize algorithm is the most expensive operation, with a consistently higher network cost. We also observe that the cost of the procedure grows in $\mathcal{O}(Nt)$ as expected. Hence, for adversarial models admitting a fixed fraction $(t-1)/N$ of dishonest parties, the per-party CPU-cost of the setup will be quadratic in the number of participants. Due to the compact Shamir public-point technique described in Section 3, the Combine step is very efficient and its cost is significantly lower than the operations of the MHE scheme to which it is a pre-processing (in the TMHE scheme). For example, the cost of generating a party's decryption share in the TMHE scheme for parameter set III with $N = 20$, $t = 7$ is 12.0 ms, only

Table 4. Threshold MHE Setup cost for $N = 8$ parties, $t = 8, 6, 4$, $k = 130$ switching keys and $4 \leq |\mathcal{P}_{online}| \leq 8$. The per-party costs are the maximum values measured among all parties. The values represent the cumulative CPU-time and network volume for the execution of the bootstrapping-key setup.

t		8 (MHE)	6 (TMHE)			4 (TMHE)				
	$\lvert\mathcal{P}_{online}\rvert$	8	6	7	8	4	5	6	7	8
Party	**CPU time** [s]	7.4	7.4	6.4	5.6	7.4	5.9	4.9	4.3	3.7
	Net. Sent [GB]	3.8	3.8	3.3	2.9	3.8	3.1	2.6	2.2	1.9
Cloud	**CPU time** [s]	2.2	1.8			1.3				
	Net. Recvd. [GB]	30.5	22.9			15.3				

0.4 ms of which are spent on the Combine operation. We conclude that, from a CPU-time perspective, the threshold access-structure comes at an almost negligible cost with respect to the non-threshold scheme. Consequently, the main overhead of the scheme remains the pairwise exchange of Shamir secret-shares during the key-generation phase.

4.3 TMHE Setup for Bootstrapping Key Generation

We used our implementation to simulate a scenario where $N = 8$ parties seek to generate $k = 130$ switching-keys, for $t = 4, 6, 8$, for several values of $|P_{online}|$. The scheme was parametrized with a ring-degree $d = 2^{15}$ and a modulus q of 768 bits ($L = 16$ primes), which supports the bootstrapping operation [CCS19,HK20,BMTPH21]. We chose $k = 130$ as the number of rotation-keys required to support the bootstrapping operation of Bossuat et al. when using a sparse secret-key and with full packing (2^{14} plaintext slots) [BMTPH21]; this represents a bootstrapping-key size of 7.6 GB. The case of $t = 8$ uses the N-out-of-N-Threshold scheme (MHE) that we use as a baseline. For simplicity, we consider the cloud-assisted model where parties use a central server to centralize and aggregate their shares. Our simulation measures the per-party CPU time needed to compute all its shares on an AMD Ryzen 9 5900X CPU and the network load of sending these shares to the cloud server. We also report on the CPU time and network load associated with the reception and aggregation of the shares on the cloud server side. Our simulation does not account for network introduced delays or other implementation-related variables such as serialization of the shares to and from the network. The results are summarized in Table 4.

The simulation confirms that, as the number of online parties grows, the cost of generating the keys can be distributed among the parties. For example, with $N = |\mathcal{P}_{online}| = 8$ and $t = 4$, the per-party CPU time and network cost is effectively divided by 2, reducing from 7.4s to 3.7s and 3.8 GB to 1.9 GB. Moreover, the cost associated with share aggregation now depends on the threshold t and no longer on N. Hence, the simulation confirms that, when failures are rare events, the costs associated with the TMHE setup is linear in t.

5 Conclusion

In this work, we have extended the multiparty-homomorphic encryption scheme over ring learning-with-errors with a t-out-of-N-threshold access-structure. We have demonstrated that the approach of re-sharing the secret-key shares composes well with the most recent approach limited to N-out-of-N-threshold access-structures, and that this yields an elegant and efficient solution. Notably, the extension introduces additional interaction at the key-generation phase only and, due to our technique for compact Shamir public-points, has only a negligible memory and CPU-time overhead with respect to the base scheme. Finally, we demonstrated how t-out-of-N-threshold access-structure brings many opportunities to capture an application's trust model and opens several new possibilities in reducing the cost of the offline phase of MHE-based MPC protocols. We implemented our scheme and open-sourced it in the Lattigo open-source multiparty homomorphic encryption library.

References

ACC+18. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

ACD+19. Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan. Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In *Theory of Cryptography Conference*, pages 471–501. Springer, 2019.

AJLA+12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 483–501. Springer, 2012.

BD10. Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In *Theory of Cryptography Conference*, pages 201–218. Springer, 2010.

BGG+18. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Annual International Cryptology Conference*, pages 565–596. Springer, 2018.

BGV14. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

BMTPH21. Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 587–617. Springer, 2021.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

CCS19. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 34–54. Springer, 2019.

CDN15. Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*, page 236–298. Cambridge University Press, 2015.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.

HK20. Kyoohyung Han and Dohyeong Ki. Better bootstrapping for approximate homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 364–390. Springer, 2020.

lat22. Lattigo v2.4.0. Online: `https://github.com/ldsec/lattigo`, January 2022. EPFL-LDS.

MBTPH20. Christian Mouchet, Jean-Philippe Bossuat, Juan Troncoso-Pastoriza, and J Hubaux. Lattigo: A multiparty homomorphic encryption library in Go. In *WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, volume 15, 2020.

MTPBH21. Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *Proceedings on Privacy Enhancing Technologies*, 2021(4):291–311, 2021.

pal. Palisade homomorphic encryption software library. Online: `https://palisade-crypto.org/`.

Sha79. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

SPTP+20. Sinem Sav, Apostolos Pyrgelis, Juan R Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. Poseidon: Privacy-preserving federated neural network learning. *arXiv preprint arXiv:2009.00349*, 2020.