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
{christian.mouchet,jean-pierre.hubaux}@epfl.ch
[2] Effixis SA
elliott.bertrand@gmail.com

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 random-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 re-sharing* 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, after which any set of at least t parties can compute a t-out-of-t additive sharing of the secret key with no interaction; this new sharing can be used directly in the scheme of Mouchet et al. We show that, by performing Shamir re-sharing over the MHE ciphertext-space ring with a carefully chosen exceptional set, this reconstruction procedure can be made secure and has negligible overhead. Moreover, it only requires the parties to store a constant-size state after its setup phase. 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.

1 Introduction

Multiparty Homomorphic Encryption (MHE) enables computations to be carried out 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 are characterized by their low communication

This is the author version of the article published in the Journal of Cryptology, Volume 36, accessible at https://doi.org/10.1007/s00145-023-09452-8

complexity [3] as well as their amenability to the paradigms of cloud-computing such as light-client/powerful-server types of architecture [14].

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) is now being implemented, used and standardized, recent works have also introduced multiparty variants of these schemes [14]. Among these multiparty schemes, *threshold* schemes [14] have been demonstrated as particularly efficient due to their compactness, and are already included in some open-source libraries [13], [15].

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 [3].

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 (round one), 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, 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 (round two). 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 [14]. They show that, as for its precursor based on learning-with-errors (LWE) [3], 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.

Boneh et al. proposed an MHE scheme with t-out-of-N-*threshold* access-structure [5], but their construction requires to choose between non-compact party-states or non-compact ciphertexts, with both options resulting in a significant overhead in practice. However, their scheme targets the strong *asynchronous* setting, where parties do not need to synchronize with each other during the protocol execution. While this is necessary for their end result (of building a universal thresholdizer for non-threshold cryptographic primitive), it might be an overkill for many *end-user* MPC scenarios (e.g., encrypted federated learning) which are primarily concerned with computation and communication efficiency. Hence, this raises the question of whether there exist a scheme, in the synchronous setting, that is simpler and more efficient.

1.1 Our Results

In this work, we introduce an efficient t-out-of-N-*threshold* MHE scheme for the synchronous setting. We contribute our scheme as a simple extension to the N-out-of-N-threshold scheme of Mouchet et al. [14] that relaxes its access structure to a t-out-of-N-*threshold* one. We also contribute its implementation in the Lattigo library [12], [13] and evaluate its performance through microbenchmarks and an application case-study.

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 additive secret-key shares with the Shamir secret-sharing scheme [17], but with a specially adapted instance of the Shamir secret-sharing that we define over the ciphertext-space ring. Then, thanks to the linearity of the N-out-of-N-threshold scheme's secret-key operations (e.g., threshold decryption), we obtain a compact and efficient scheme. Notably, we show that the re-shares can be pre-aggregated, resulting in a constant-size state party state, and that the t-out-of-N secret-key-reconstruction can be performed efficiently within the secret-key operation itself. We show that, in the synchronous setting, this requires a simple non-interactive pre-computation to the corresponding operation in the N-out-of-N scheme, yet performed among $N = t$ parties. 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 constructions in Lattigo [13], 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 an application case-study: the end-to-end-encrypted federated neural network training of Sav et al. [16].

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 [4]. 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 [3]. 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. proposed a t-out-of-N-threshold HE scheme based on learning-with-errors that also relies on re-sharing the secret-key shares, yet in a stronger *asynchronous* setting where parties are unable to determine which other parties are online at the time of generating their decryption shares [5]. 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. Yet, Boneh et al. observe that enabling the parties to determine which other parties are online before the decryption phase would lead to a simpler scheme. We confirm this observation by showing that, in the semi-honest model with failures, there indeed exists a simpler, more compact and more efficient scheme that does not require a trusted dealer. We elaborate on these differences in Section 4.1, where we provide a comparison between their construction and our scheme.

Concurrently to our work, Urban and Rambaud proposed an alternative MHE-based MPC approach that provides guaranteed output delivery while minimizing the number of synchronous rounds needed in the setup phase and requiring no synchronous communication during the evaluation phase. Their approach is also based on a linear secret-sharing scheme over RLWE rings [18], but their construction targets generality rather than efficiency as it allows the FHE coefficient modulus to be a composite with factors that are smaller than the number of parties. Our construction targets efficiency for the parameterization supported by the current FHE implementation, for which the structure of the coefficient modulus is already constrained for efficiency reasons.

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 random string (CRS).

System Goals. Let x_i be the private input of party P_i in some message space $\mathcal{M}$, let $f : \mathcal{M}^N \to \mathcal{M}$ be a public arithmetic function over the message space, and let λ be a security parameter. 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 (the *transcript*), 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 a negligible function in λ.
- *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.

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. [14], its instantiation as an MPC protocol, and the secret-sharing scheme of Shamir [17].

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) [14] 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 representation of R_{q_i} elements under the number-theoretic transform domain (NTT), under which both ring operations are 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 $\{-1, 0, 1\} \mod q$, 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 [3], [14] (typically, a discrete Gaussian distribution of large variance). Finally, let $\mathsf{CRS}(R_q)$ be the uniform distribution in R_q according to the common reference string (i.e., elements sampled from this distribution are the same for all parties).

Scheme: MHE

- MHE.Setup: The parties agree on the public parameters $(n, q, \sigma, \mathsf{Key}, \mathsf{Err})$.
- MHE.SecKeyGen: Each party P_i samples $s_i \leftarrow \mathsf{Key}(R_q)$.
- 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 $\mathsf{pk} = (p_0, p_1)$.
- MHE.Encrypt(pk, m): Sample $u \leftarrow \mathsf{Key}(R_q)$, $e_0, e_1 \leftarrow \mathsf{Err}(R_q)$ and output: $$\mathsf{ct} = (c_0, c_1) = (m + up_0 + e_0, up_1 + e_1).$$
- MHE.Decrypt(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 MHE.Eval procedure as it depends on

the specific plaintext-encoding strategy of the RLWE scheme in use but does not depend on the access structure for threshold schemes (we briefly discuss the encoding strategy below).

Plaintext Encoding and Homomorphic Evaluation. Due to the error that is inherent to the encryption scheme, the MHE.Decrypt procedure outputs an approximate message, and users need to rely on plaintext encoding techniques 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 rounding for the decoding [9], [11]. 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 up to n $\mathbb{Z}_q$ messages in n independent ciphertext *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 [7], [11], CKKS [9] or BGV schemes [8]. 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 [11] which requires a so-called *relinearization key*. Likewise, plaintext slots rotation by k slots can be operated as an homomorphic automorphism which requires rotation-specific *rotation-keys* (i.e., a key for each needed rotation parameter k). Although generating a single key for a one-slot rotation ($k = 1$) 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 [14] 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 [6]) 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; we refer to this protocol as the MHE-MPC protocol. This model comprises two phases, the first being input-independent and optional in the PKI setting (hence is 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' indi-

vidual 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 procedure 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. 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* and *privacy* system goals of Section 2.1, but it does not satisfy 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 can be performed non-interactively. The same is true for a party that momentarily goes offline after the *Setup* 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 [17]. 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.

- 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 $\alpha_i \neq \alpha_j$.
- 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 send $s_i = s + \sum_{k=1}^{t-1} c_k \alpha_i^k$ to party P_i.
- 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 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 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

We start from the well-known *share re-sharing* approach, which 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 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 over 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 costly R_q arithmetic emulation over this space). Also, it would require each party to store all N re-shares throughout the entire protocol, whereas we require a constant-size state.

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 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, we observe that the Shamir.Combine operation commutes with the ideal-secret-key reconstruction:

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

This presents several opportunities for our construction, which we outline below as Remarks 1 to 3.

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. From Eq. (2), 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 parties that are participating to the secret-key operation is known.

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

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* [1], [10]. We now briefly present this result in our notation and terminology.

Definition 1. *(From [1]) 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 [1]) 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. Hence, we now define how to choose our Shamir public-points from R_q in such a way that guarantees an exceptional sequence and enables an highly efficient implementation.

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 significant performance boost as computing

the Lagrange coefficient now only require scalar multiplications 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 $\mathbb{Z}_{q_{min}}$ 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 $\mod q_k$ 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 $\mathbb{Z}_{q_{min}}$ can be used, including the *textbook* one that commonly uses i for party P_i, if i is a positive integer. We observe that it is critical for implementations to check that Shamir public-points are non-zero.

Choosing the Shamir public points from the restricted set has the side effect of limiting the number of parties to $q_{min} - 1$. But this is not an issue in most cases, because the factors of q are already constrained by the encryption scheme's requirement. More precisely, they have to be primes congruent to $1 \mod 2n$ where n is the degree of the ring (which is typically larger than 2^{11} in the FHE setting). However, this could be a limitation in a setting where parties independently and randomly sample their own public points, as the probability of a collision would be too high. For such use-cases it might be preferable to sample points in $\mathbb{Z}_q$ where the probability of collision is negligible, then check that the sequence forms an exceptional sequence, which occurs with high probability.

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. [14].

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 $\mathsf{T} = (\mathsf{Setup}, \mathsf{Thresholdize}, \mathsf{Combine})$. Intuitively, Scheme T applies the Shamir secret-sharing scheme over R_q introduced in Section 3.2 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

- $\mathsf{T.Setup}$: 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, $i \neq j$.
- $\mathsf{T.Thresholdize}(t,\ s_1, ..., s_N,\ \alpha_1, ..., \alpha_N)$:
  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: $\tilde{s}_i = \sum_{j=1}^{N} \tilde{s}_{j,i}$.
- $\mathsf{T.Combine}(\tilde{s}_1, ..., \tilde{s}_t,\ \alpha_1, ..., \alpha_t)$: For $\mathcal{P}' \subseteq \mathcal{P}$, $|\mathcal{P}'| \geq t$, each party $P_i \in \mathcal{P}'$ 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 because 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.** We propose our construction as the union tuple MHE$\cup$T, which provides a t-out-of-N-threshold encryption scheme. We detail this construction as Scheme TMHE. As per Remark 2, the T.Combine procedure requires the parties 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 at the time the environment is queried. We assume that the oracle returns the same set to all parties for a given secret-key operation. However, we do not assume this across different secret-key operations, and the set of parties performing the setup might differ from the set performing decryption. Indeed, as per Equation 2, any set of at least t parties can reconstruct s. In our (synchronous) 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.

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(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 which did not provide their share in time during a secret-key operation; a partial yet insecure solution is to repeat steps 3 and 4 of the operation, 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 asymmetric additive HE scheme. Hence, our solution consists in adding a re-randomization step, by adding a fresh encryption of zero to the target ciphertext before repeating the MHE.Decrypt step.

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 [14], [16]. These protocols can be abstracted as performing a masked decryption and a re-encryption, hence are secret-key operations.

– At the Output phase - when the function's output consists in multiple ciphertexts. This could be by design (of the ideal functionality), 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 $\mathcal{P}_{online}$ be the set of online parties. The parties in $\mathcal{P}_{online}$ 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 for each party, by relying on tree-based share aggregation patterns [14]. 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. In Section 4.3, we evaluate the effect of using this technique in the setup and in the evaluation phase of a concrete instance of the MHE-MPC protocol: the federated neural network training algorithm of Sav et al. [16].

4 Evaluation

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

4.1 Theoretical Evaluation

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. [14] 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.

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	$\mathcal{O}(t+Nt+N)$	$\mathcal{O}(t)$

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 selection of Section 3.3, is the only part of this computation that depends on t. We demonstrate this by benchmarking our implementation, in Section 4.2.

Comparison with the Scheme of Boneh et al. Boneh et al. proposed a t-out-of-N-threshold scheme as an essential building block to their universal thresholdizer for cryptographic primitives [5]. However, they consider a stronger asynchronous setting, where parties are unable to determine (or optimistically guess) the set of online other parties when performing secret-key operations. Essentially, their solution is to perform the Lagrange interpolation homomorphically, 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). In contrast, our scheme targets the synchronous setting, yet is much

Table 2. Benchmarked HE Parameters. The polynomial degree n and coefficient modulus q size in bits are taken from the standardization document [2]. 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 T benchmarks (with per-step breakdown for Thresholdize, see Section 3.3) for $N = 20$ parties and threshold $t = 7, 14, 19$. These values represent the per-party CPU time in milliseconds.

	Param.	I			II			III		
	t	7	14	19	7	14	19	7	14	19
Thresholdize	**Step 1**	6.0	13.0	17.9	26.2	56.8	78.7	91.7	198.2	275.6
	Step 2	4.2	8.8	12.3	16.6	35.6	50.0	67.3	146.9	202.1
	Step 3		0.2			0.9			3.4	
	Total	10.4	22.0	30.4	43.7	93.2	129.5	162.4	348.5	481.2
Combine		<0.1	<0.1	<0.1	0.1	0.1	0.1	0.3	0.4	0.4

Table 4. Threshold scheme TMHE benchmarks in milliseconds for $N = 20$ parties and threshold $t = 7, 14, 19$. These values represent the per-party CPU time in milliseconds.

	Param.	I			II			III		
	t	7	14	19	7	14	19	7	14	19
SecKeyGen	**MHE.SecKeyGen**		0.5			2.1			7.4	
	T.Thresholdize	10.4	22.0	30.4	43.7	93.2	129.5	162.4	348.5	481.2
	Total	10.9	22.5	30.9	45.8	95.3	131.6	169.8	355.9	488.6
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

simpler and more efficient, which enabled its implementation and its integration in an existing library. Notably, it can be seen as an extension of an existing scheme, requires a $\mathcal{O}(1)$ storage for the secret-key shares, and has negligible online overhead. Moreover, it does not require a trusted dealer.

4.2 Basic Operations Benchmarks

We implemented the scheme extension T in the Lattigo library [13] (that already implements the MHE scheme) and benchmarked its performance on an AMD Ryzen 9 5900X CPU (3.7GHz clock, 6M of L2-cache) for several common choices of encryption parameters (summarized in Table 2) and several values of the threshold t. Note that our implementation itself uses no parallelization, but its interface allows a party to generate the shares for each other party separately in the step 2 of the Thresholdize operation. Hence, this step can be parallelized and the actual latency divided by $\min(N, C)$ where C is the number of cores

available. In the scope of this micro-benchmark, we report the total CPU time to abstract this setting-dependant variable and to show the actual cost of the computation (the latency being relatively low in the context of a networked system). We report the results for the threshold extension T in Table 3 and for the relevant operations of the TMHE scheme in Table 4.

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 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 one-time key-generation phase.

4.3 Case-study: Encrypted Federated Neural Network Training

The main application of our TMHE scheme is the MHE-MPC protocol, which is a generic MPC protocol. To further demonstrate the effects of using our construction in a concrete application of this protocol, we now consider a federated learning scenario in which multiple parties seek to train a neural network model on their joint datasets, under encryption.

Sav et al. used the MHE-MPC protocol to perform federated neural network training and inference under N-out-of-N-threshold encryption [16]. Their approach relies on the CKKS variant of the MHE scheme and faces two important challenges: First, it relies heavily on ciphertext-slots rotations for many different rotation values (mostly for the matrix operations), hence requires many rotation-keys to be generated in the offline setup phase (see Section 2.2). Second, the high multiplicative depth of the training algorithm requires the parties to refresh the ciphertexts during the computation, by means of an interactive refresh protocol (to circumvent the high cost of a local bootstrapping), which can be seen as a masked decryption and a re-encryption of the ciphertext. The use of secret-key operations in the training phase has two consequences: it limits the system to *synchronous learning* scenarios (where all parties have to be online for the whole training phase) and it introduces a significant communication overhead which constitutes the system's main bottleneck.

We now describe the effect of using the CKKS variant of our TMHE scheme in Sav et al.'s system, assuming a t-out-of-N-threshold setting. In the scope of this case-study, we focus on their MNIST instantiation where $N = 10$ parties train a 3-layer neural network to perform handwritten digit recognition. This scenario uses a polynomial degree $n = 2^{14}$, a coefficient modulus of $\log_2 q = 438$

bits with $L = 9$ primes, and requires 623 rotation keys to be generated[3] along with the public encryption- and relinearization-keys.

Setup phase. To generate the public encryption-, relinearization- and rotation-keys, we propose to equally distribute the set of keys to be generated among the online parties (up to a difference of 1 key per party). Each party then picks a random set of $t-1$ other parties per key it is responsible for, queries these other parties for their shares and aggregates the them (as defined in the TMHE scheme). Finally, each party retrieves the aggregated share of the keys it is not responsible for (from the designated party for that key).

We implemented a proof of concept Go application for this setup procedure based on our open-source TMHE scheme implementation in Lattigo. The protocol interactions were implemented as a client-server application enabling the parties to query each other for their respective shares as well as for the aggregated shares they are responsible for. The application performs all queries to the other parties in parallel, to estimate the minimum wall-time latency of the setup phase. We benchmarked this implementation on a network of 10 machines equipped with an Intel Xeon E5-2680 v3 CPU (2.5 GHz, 30 MB cache) and 256 GB of RAM. To simulate a realistic WAN-like network, we limited the network's bitrate to 1 Gbits/sec and introduced a 10 ms latency. We instrumented our code to report the total amount of data sent and received for each party, as well as the total wall time for the execution of the setup phase, and we extracted the CPU time from the operating system's metric. Our experiment assumes that all parties are online to perform the setup.

The results for the MNIST setup are summarized in Table 5. Our experimental result confirm that the use of our TMHE scheme reduces the per-party cost when more than t parties are participating to the setup. We do not observe a factor $\frac{t}{N}$ reduction with respect to the $t = N$ case (which uses the MHE scheme directly). This is because the final phase (query of the aggregated shares) still depends on N when all parties are online. But the cost reduction remains significant, hence motivating the t-out-of-N-threshold scheme when the threat models allows it. We also observe a larger gap between the CPU and wall times for $t = 3$, as the parallelization of batched secret-key operation described in Section 3.5 starts being effective. We note that this effect should be observed also for $t = 5$, but is not. This suggests that more engineering would be needed. For example by partitioning the set of parties into two groups operating individually for the share generation and aggregation phase.

Online phase. The training algorithm used by Sav et al. is an iterative distributed gradient descent with two phases per iteration. The first phase is a local gradient descent, where each party computes its local gradients through forward-backward propagation. The second phase is a global model update where a designated party aggregates all the gradients and updates the model weights. The

[3] The work of Sav et al. actually abstracts the setup phase and their code is closed-source. This value was obtained through communication with the authors.

Table 5. Threshold MHE Setup cost for $N = 10$ parties, $t = 8, 6, 4$, 623 rotation keys. The values are the largest measured per-party costs among all parties along with their ratio with respect to the $t = N$ case.

	Scheme	MHE	TMHE		
	t	10	7	5	3
Time [s]	**CPU time**	149.9 (100%)	120.1 (80.1%)	108.6 (72.4%)	88.7 (59.1%)
	Wall time	67.1 (100%)	53.7 (80.0%)	48.6 (72.4%)	35.1 (52.3%)
Network [GB]	**Sent**	5.3 (100%)	4.5 (84.9%)	3.9 (73.6%)	3.3 (62.2%)
	Received	5.3 (100%)	4.4 (83.01%)	3.8 (71.7%)	3.2 (60.4%)

model weights and the gradients are encrypted throughout the whole process and the number of iterations is a parameter of the system. The source code of their system being closed-source, we study its online phase from a theoretical perspective. More precisely, we focus on its communication complexity because it constitutes the main bottleneck of the algorithm.

This bottleneck is caused by the use of the interactive refresh protocol for ciphertext that have reached a certain level L_{ref} (the smallest level at which the refresh protocol is correct and secure, see Section 5.F of [16]). In phase 1, each party requires β refresh where β is a function of model size and encryption parameters (also see Section 5.F of [16]). In phase 2, the designated party requires l refreshes (one per non-input layer). A single instance of the refresh protocol requires the initiator to broadcast the level-L_{ref} ciphertext to be refreshed and to collect one share per party. The ciphertext consists of two ring elements at level L_{ref} and each share consists of one ring element at level L_{ref} and one ring element at the largest level L. Assuming 8-bytes encoding for the coefficients, the transcript of a single refresh protocol is of size $E = 8n(3L_{ref} + L)$ bytes per party assisting the initiator in the protocol. In the N-out-of-N-threshold model, this represents a total communication of $N\beta(N-1)E$ bytes for the first phase and of $l(N-1)E$ in the second. For the MNIST instance, this represents a communication of 644.1 MB per iteration ($\beta = 4$, $l = 2$, $L = 7$ and L_{ref}).

We propose the following modification to the framework of Sav et al., which is again a straightforward application of the TMHE scheme: In the local gradient descent phase (1), each party picks a random subgroup of $t-1$ other parties in the set of online parties and performs all refresh protocols among this group. In the global-model update phase (2), the aggregator randomly partitions the set of online parties into $\lfloor \frac{|\mathcal{P}_{online}|}{t} \rfloor$ groups, and distributes the batch of l refresh protocols among the groups. The proposed changes extend the framework to the asynchronous learning scenario (with a tolerance of $N-t$ offline parties). In the case where all parties are online, it reduces the communication complexity for phase 1 and 2 to respectively $N\beta(t-1)E$ and $l(t-1)E$, which corresponds to a total of 286.3 MB per iteration for the MNIST instance. Additionally, it divides the latency of step 2 by $\lfloor \frac{|\mathcal{P}_{online}|}{t} \rfloor$. Hence, as for the setup phase, the use of our fault-tolerant scheme also comes with a general reduction of the online phase costs, especially when it relies on the refresh protocol.

5 Conclusion

In this work, we have extended the multiparty-homomorphic encryption scheme of Mouchet et al. [14] 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 their approach, 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. As a result, not only does our scheme provide fault-tolerance to the MHE-based MPC protocol, but it also reduces the per-party costs and overall latency when the number of online parties is above the threshold. We implemented our scheme and open-sourced it in the Lattigo library.

References

[1] M. Abspoel, R. Cramer, I. Damgård, D. Escudero, and C. Yuan, "Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings," in *Theory of Cryptography Conference*, Springer, 2019, pp. 471–501.

[2] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., Nov. 2018.

[3] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold FHE," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2012, pp. 483–501.

[4] R. Bendlin and I. Damgård, "Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems," in *Theory of Cryptography Conference*, Springer, 2010, pp. 201–218.

[5] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in *Annual International Cryptology Conference*, Springer, 2018, pp. 565–596.

[6] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux, "Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2021, pp. 587–617.

[7] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Annual Cryptology Conference*, Springer, 2012, pp. 868–886.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2017, pp. 409–437.

[10] R. Cramer, I. B. Damgård, and J. B. Nielsen, "Secure multiparty computation and secret sharing," in *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015, 236–298. DOI: 10.1017/CBO9781107337756.012.

[11] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption.," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[12] *Lattigo v3*, Online: https://github.com/tuneinsight/lattigo, EPFL-LDS, Tune Insight SA, Aug. 2022.

[13] C. Mouchet, J.-P. Bossuat, J. Troncoso-Pastoriza, and J Hubaux, "Lattigo: A multiparty homomorphic encryption library in Go," in *WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, vol. 15, 2020.

[14] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 4, pp. 291–311, 2021.

[15] *Palisade homomorphic encryption software library*, Online: https://palisade-crypto.org/.

[16] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, "Poseidon: Privacy-preserving federated neural network learning," *28th Annual Network and Distributed System Security Symposium*, 2021.

[17] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[18] A. Urban and M. Rambaud, *Share and shrink: Ad-hoc threshold fhe with short ciphertexts and its application to almost-asynchronous mpc*, Cryptology ePrint Archive, Paper 2022/378, https://eprint.iacr.org/2022/378, 2022. [Online]. Available: https://eprint.iacr.org/2022/378.