Multiparty Homomorphic Encryption: From Theory to Practice

Christian Mouchet Juan Troncoso-Pastoriza Jean-Pierre Hubaux
École polytechnique fédérale de Lausanne
first.last@epfl.ch

***Abstract*—We propose and evaluate a secure-multiparty-computation (MPC) solution, in the semi-honest model with dishonest majority, based on multiparty homomorphic encryption (MHE). To support this solution, we introduce a multiparty version of the Brakerski-Fan-Vercauteren lattice-based homomorphic cryptosystem, implement it in an open-source library, and evaluate its performance. We show that such MHE-based MPC solutions have several advantages over current approaches: Their public transcripts and non-interactive circuit-evaluation capabilities enable a broad variety of computing paradigms, ranging from the traditional peer-to-peer setting to cloud-outsourcing and smart-contract technologies. Exploiting these properties, the communication complexity of MPC tasks can be reduced from quadratic to linear in the number of parties, thus enabling secure computation among thousands of parties. Additionally, MHE-based approaches can outperform the state-of-the-art even for a small number of parties. We demonstrate this for three circuits: *component-wise vector multiplication* with application to private-set intersection, *private input selection* with application to private-information retrieval, and *multiplication triples generation*. For the first circuit evaluated among eight parties, our approach is 8.6 times faster and requires 39.3 times less communication than the state-of-the-art approach. The input selection circuit over eight thousand parties completed in 61.7 seconds and required 1.31 MB of communication per party.**

***Index Terms*—secure multiparty computation, homomorphic encryption, lattice-based cryptography**

I. Introduction

Secure Multiparty Computation (MPC) protocols enable a group of parties to *securely* compute joint functions over their private inputs while enforcing specific security guarantees throughout the computation. Although the exact definition of security depends on how the adversary is modeled, the most common requirement, *input privacy*, informally states that parties should not obtain more information about other parties' inputs than what can be deduced from the output of the computation. Combining this strong security guarantee with such a general functionality makes the study of MPC techniques highly relevant, from both a research and a practical perspective. The last decade has seen this established theoretical field evolve into an applied one, notably due to its potential for securing *data-sharing* scenarios in the financial [1]–[3], biomedical [4]–[6] and law enforcement [7], [8] sectors, as well as for protecting digital assets [9], [10].

The transition of MPC techniques to their application domains however still faces significant obstacles. In the settings where no honest majority of parties can be guaranteed, current MPC protocols are typically based on secret-sharing [11] of the input data and interactive circuit evaluation. These approaches have two practical limitations: First, standard protocols require many rounds of communication over private channels between the parties, which makes them inadequate for low-end devices and unreliable networks. Second, current approaches require a per-party communication that increases linearly in the circuit size (that itself increases at least linearly in the number of parties). Hence, this quadratic factor quickly presents an efficiency bottleneck for large circuits and/or large number of parties. As a result, current MPC applications are mostly focused on two- or three-party settings where the MPC task is delegated to a set of *computing parties* that are assumed not to collude in revealing the inputs [5], [12]–[16].

We propose, implement, and evaluate a passive-adversary dishonest-majority MPC protocol that can support several thousands of parties without making such assumption. Based on multiparty homomorphic encryption (MHE), this protocol reduces the per-party communication complexity to being linear in the circuit's inputs and outputs. We achieve this by exploiting recent progress in lattice-based homomorphic encryption that, despite being used in current MPC solutions, was mostly confined to their *offline* pre-computations but not used during the circuit evaluation. We argue that homomorphic encryption has reached the required level of usability to play a larger role in the online phase of MPC protocols and to enable new applications. Notably, we discuss how MHE-based approaches can be integrated within existing secret-sharing-based ones to reduce their communication overhead. Furthermore, we show that MHE-based solution can remove the need for private party-to-party communication channels. Hence, in addition to the traditional peer-to-peer setting, this solution can operate in a broad range of computing platforms, ranging from cloud-outsourcing to distributed-ledger smart-contract technologies, without requiring non-collusion assumptions.

Our solution is based on a multiparty version of the BFV homomorphic encryption-scheme [17] and it can be extended to other mainstream schemes of the same family such as the CKKS cryptosystem [18]. It uses secret-sharing techniques [11] to distribute the secret encryption key among the parties, as proposed by Asharov et al. [19], [20] and Boneh et al. [21]. In addition to bringing these works to more recent cryptosystems and to practice, we make the following contributions:

- We propose a novel multiparty version of the BFV lattice-based somewhat-homomorphic encryption scheme (Section IV). This includes a novel protocol for bootstrapping a BFV ciphertext in a multiparty setting.

- We discuss the features of the MHE-based approach and how it can both be integrated into existing solutions, and be used directly as a standalone one, enabling secure computation among thousands of parties (Section V).
- We provide security arguments for the protocols that define the multiparty cryptosystem (Section VI).
- We implement our scheme in an open-source library and evaluate it for three example circuits (Section VII).

We believe this work to be a significant step in bridging the gap between currently available MPC solutions and well-established works on multiparty lattice-based cryptosystems.

II. Related Work

We classify N-party dishonest-majority MPC approaches in two categories: The category (a), *data-level secret-sharing*, which is predominantly implemented in generic MPC solutions [9], [22], consists in applying *secret-sharing* [11] to input data. The category (b), *multiparty encryption schemes* , uses an homomorphic scheme to encrypt and exchange the input data, that can then be operated on in a non-interactive way with encrypted arithmetic.

(a) Data-Level Secret-Sharing (for short: secret-sharing-based): Most of the available generic MPC solutions, such as Sharemind [23] and SPDZ [24]–[26], rely on applying secret-sharing to the input data. The evaluation of these arithmetic circuits is generally enabled by the homomorphism of the secret-sharing scheme, or by interactive protocols [27], [28] (when no such homomorphism is available), the most widely implemented protocol being Beaver's triple-based protocol [27]. The strength of approach (a) is to rely, at the evaluation phase, on only simple and efficient primitives in terms of which the circuit can be decomposed by code-to-protocol compilers, thus providing great usability. However, this approach imposes two constraints on the system: First, using an interactive protocol at each multiplication gate requires all parties to be online and active during the whole computation, and to exchange private messages with their peers, often at a high frequency that is determined by the round complexity of the circuit. Second, the triple-based multiplication protocol requires a prior distribution of one-time triples; this can be performed in a pre-computing phase either by a trusted third-party or by the parties themselves. Interestingly, the latter peer-to-peer case can also be formulated as an independent, yet equivalent, MPC task (generating the triples requires multiparty multiplication). Hence, these approaches are often hybrids that generate the multiplication triples [22] by using other techniques such as oblivious transfer [29] or plain homomorphic encryption [24], [26] in an *offline phase*.

(b) Multiparty Encryption Schemes: In this approach, each party holds a secret-share of an *ideal* secret-key to a homomorphic encryption-scheme. The parties can provide their inputs encrypted under this secret key and the computation can be performed using the homomorphic operations of the HE scheme. Finally, the decryption requires the collaboration between the parties according to the secret-sharing scheme. Such constructions are commonly referred to as *threshold* [30] and *distributed* cryptosystem, depending on the decryption structure they enforce. We use the term *multiparty encryption scheme* to designate these constructions in a general way. We define this term as a primitive and the MPC protocol it enables in Section III-B. The idea of reducing the volume of interaction in MPC by using homomorphic encryption can be traced back to a work by Franklin and Haber [31], later improved by Cramer et al. [32]. But, at the time, the lack of homomorphic schemes that preserve two distinct algebraic operations ruled out complete non-interactivity at the evaluation phase, thus rendering these approaches comparatively less attractive than approach (a). Recently, task-specific instances that use multiparty additive-homomorphic encryption have nevertheless been successful in supporting use-cases in distributed machine learning [33], [34], highlighting the potential a generic and usable fully-homomorphic solution would have. This is possible since 2009, when Gentry proposed the first *fully homomorphic* encryption (FHE) scheme [35].

This is the idea behind the joint line of work by Asharov et al. [19], [20] and López-Alt et al. [36], later generalized by Boneh et al. [21] by the idea of a *universal thresholdizer*. These contributions propose various multiparty schemes in which the *ideal* secret-key is additively shared among the parties, and they analyse the theoretical MPC solution these schemes enable. Although of great interest, these lines of work did not find as much echo in applications as approach (a) has. One possible reason was the lack, at the time, of available and efficient implementations of *Learning with Errors* [37] (LWE) -based homomorphic schemes, in terms of which these schemes were presented. Today, however, multiple ongoing efforts aim at standardizing homomorphic encryption [38] and at making it available to a broader public [39]–[43]. This new generation of schemes is mostly based on the *Ring Learning with Errors* (RLWE) problem [44] and have brought HE from being practical to being efficient. Therefore, we argue that MHE-based approaches deserve more than theoretical interest, as they are now mature and usable enough to support more than the offline phase of secret-sharing-based approaches.

It is worth mentioning that the *multi-key* class of multiparty encryption schemes, as introduced by López-Alt et al. [45], can also be considered as multiparty cryptosystems. These schemes enable the parties to provide their input data encrypted under their own secret key, and to operate on them directly by using encrypted arithmetic. As the encryption scheme is both message-homomorphic and key-homomorphic, the computation result is then encrypted under an *on-the-fly* key that is a joint function of the input secret keys. Hence, only the involved parties are required to participate in the decryption sub-protocol. Unfortunately, this feature comes at a prohibitive performance cost, limiting its applicability for use-cases with large number of parties. Notably, current multi-key schemes lack the compactness property: the size of their ciphertexts and the timing of arithmetic operations are, respectively, linear and quadratic in the number of keys used for the inputs. They also require significant interactive pre-computation of potentially large keys [46]–[48] that usually increase quadratically or cubically in the number of parties Hence, we chose to stick with the secret-key-level secret-sharing approach in our novel MHE scheme.

We will show that, in addition to be naturally interoperable with approach (a), MHE-based solutions also enable scenarios where an interactive circuit evaluation does not fit the system model (e.g., parties going offline during evaluation, outsourced scenarios) or the network model (e.g., low-end devices, public communication channels). Hence, we propose and build an open-source MPC solution that brings the theoretical work on multiparty cryptosystems [19]–[21] to recent state-of-the-art RLWE cryptography and to practice.

III. Background

We first introduce our problem statement and the required building block of our MHE-based MPC solution.

A. Problem Statement and System Model

We model a secure-multiparty-computation setting in terms of a problem that needs to be solved under a set of security constraints (by an MPC *protocol*). Indeed, different threat models result in different sets of constraints. Definition 1 formulates the *secure-multiparty-computation problem* we consider for the scope of this work.

Definition 1. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$ be a set of N parties respectively holding inputs $(x_1, x_2, \ldots, x_N)$ (*input parties*) and let $\mathcal{R}$ be a party that can be either inside or outside of $\mathcal{P}$ (*receiver party*). Let $f(x_1, x_2, \ldots, x_N) = y$ be a function (*ideal functionality*) over the *input parties*' inputs. Let $\mathcal{A}$ be a static semi-honest adversary that can corrupt $\mathcal{R}$ and/or up to $N-1$ parties in $\mathcal{P}$.
The *secure-multiparty-computation problem* consists in providing the receiver $\mathcal{R}$ with $y = f(x_1, x_2, \ldots, x_N)$, yet $\mathcal{A}$ must learn *nothing* more about $\{x_i\}_{P_i \notin \mathcal{A}}$ than what can be deduced from the inputs $\{x_i\}_{P_i \in \mathcal{A}}$ and output y it controls (*input-privacy*).

The solution to a secure-multiparty-computation problem is a protocol denoted π_f that realizes the problem's ideal functionality f and preserves input privacy. Equivalently, this property requires that the execution of π_f emulates an ideal setting where parties are provided with an incorruptible environment that, provided with their inputs, will carry out the computation of f and that will provide $\mathcal{R}$ with its output.

Our definition permits that the receiver party $\mathcal{R}$ is outside of the set of input parties $\mathcal{P}$. This reflects the fact that, in such case, there is no need for $\mathcal{R}$ to have an active role in the input's access control mechanism. Indeed, only the parties having input in the ideal functionality should have such role.

We assume that the parties in $\mathcal{P}$ have access to a uniformly random *Common Reference String* (CRS) [49], and they are connected through authenticated channels. Note that these channels are not required to be confidential.

For consistency with our concrete solution, we model MPC tasks with multiple receivers as the composition of single-receiver sub-tasks: Let $\mathcal{O}$ be a set of $N_{\mathcal{O}}$ output parties, we define $F_{\mathcal{O}} = (f_1, f_2, \ldots, f_{N_{\mathcal{O}}})$ (*joint ideal functionality*), where $f_i(x_1, x_2, \ldots, x_N) = y_i$ for each output party $i \in \mathcal{O}$, as the composition of all *private ideal functionalities*.

B. Multiparty Encryption Schemes

A *multiparty* encryption-scheme securely splits the secret key of an encryption scheme among a group of N parties, according to a secret-sharing scheme. Hence, the operations that depends on the secret key (in the original scheme) are implemented as special-purpose secure-multiparty protocols (in the multiparty scheme). Notably, when considering a traditional asymmetric encryption scheme, this is the case for the public encryption-key generation and decryption procedures. Definition 2 formalizes this intuition. In Section IV, we construct a multiparty version of the Brakerski-Fan-Vercauteren somewhat-homomorphic encryption scheme [17], where the secret key is additively shared among the parties.

Definition 2. Let $E = (\mathsf{SecKeyGen}, \mathsf{PubKeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an asymmetric encryption scheme, whose security is parameterized by λ, and let $S = (\mathsf{Share}, \mathsf{Combine})$ be an N-party secret sharing scheme. The associated *multiparty encryption scheme* is obtained by applying the secret-sharing scheme S to E's secret key sk (*ideal secret key*) and is defined as the tuple $E^S = (\pi_{\mathsf{SecKeyGen}}, \pi_{\mathsf{PubKeyGen}}, \pi_{\mathsf{Dec}})$ of multiparty protocols having the following *private ideal functionalities* for each party P_i:

Ideal secret-key generation:

$$f_{i,\pi_{\mathsf{SecKeyGen}}}(\lambda) = \mathsf{sk}_i = S.\mathsf{Share}_i(E.\mathsf{SecKeyGen}(\lambda))$$

Collective public-key generation:

$$f_{i,\pi_{\mathsf{PubKeyGen}}}(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_N) = E.\mathsf{PubKeyGen}(S.\mathsf{Combine}(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_N)),$$

Collective decryption:

$$f_{i,\pi_{\mathsf{Dec}}}(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_N, \mathsf{ct}) = E.\mathsf{Dec}(S.\mathsf{Combine}(\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_N), \mathsf{ct}).$$

A direct consequence of Definition 2 is that E and E^S are *compatible*: secret-sharing a valid secret key for E results in a valid instantiation of E^S where key holders can collectively decrypt ciphertexts of E. Conversely, reconstructing the ideal secret key of E^S yields a valid secret key for E that can decrypt ciphertexts of E^S. The output of the $\pi_{\mathsf{PubKeyGen}}$, the *collective public key*, is also a valid public key for E.

As the $S.\mathsf{Combine}$ operation has to be embedded in $f_{\pi_{\mathsf{Dec}}}$, the secret-sharing scheme defines an access structure [11] that directly characterizes the access structure of E^S. For example, using additive secret-sharing of the secret key results in a scheme where all parties must collaborate to decrypt a ciphertext, whereas only a *threshold* number of them would be required when using Shamir secret-sharing [11]. As the strictest security guarantee can be too restrictive for some system models, our future work comprises threshold access structures for the RLWE encryption schemes.

C. Multiparty Homomorphic Encryption (MHE)

When considering multiparty *homomorphic* encryption, E (and, by extension, E^S) is augmented with the Eval procedure

Protocol 1: MHE–MPC

Public input: the ideal functionality f to be computed
Private input: x_i for each $P_i \in \mathcal{P}$
Output for $\mathcal{R}$: $y = f(x_1, x_2, \ldots, x_N)$

Setup: the parties in $\mathcal{P}$ instantiate the multiparty scheme E^*

$$\mathsf{sk}_i = E^*.\pi_{\mathsf{SecKeyGen}}(\lambda, \kappa),$$

$$\mathsf{cpk} = E^*.\pi_{\mathsf{PubKeyGen}}(\kappa, \mathsf{sk}_1, \ldots, \mathsf{sk}_N),$$

In: each P_i encrypts its input and provides it to $\mathcal{C}$

$$c_i = E^*.\mathsf{Encrypt}(\mathsf{cpk}, x_i),$$

Eval: $\mathcal{C}$ computes the encrypted output for the ideal functionality f

$$c' = E^*.\mathsf{Eval}(f, c_1, c_2, \ldots, c_N),$$

Out: the parties in $\mathcal{P}$ execute the decryption protocol

$$y = E^*.\pi_{\mathsf{Dec}}(\mathsf{sk}_1, \ldots, \mathsf{sk}_N, c').$$

that enables encrypted arithmetic on its ciphertexts. We denote E^* this augmented scheme:

$$E^* = (\pi_{\mathsf{SecKeyGen}}, \pi_{\mathsf{PubKeyGen}}, E.\mathsf{Enc}, \pi_{\mathsf{Dec}}, E.\mathsf{Eval}),$$

An homomorphic scheme requires proper parametrization in order to support the evaluation of the target ideal functionality. Hence, we model this dependency by introducing an abstract *homomorphic capacity* parameter, which we denote κ.

D. MHE-Based MPC Protocol

We provide an overview of the MHE-based MPC protocol, that we formulate for an abstract MHE scheme E^* in Protocol 1. The idea of using homomorphic cryptosystems as a standalone MPC solution has been discussed in cryptographic research [20], [32]. However, to the best of our knowledge, no concrete MPC framework has been built to exploit those ideas. In this work, we take this step and show that we can build efficient systems, not only in the traditional peer-to-peer setting but also in the outsourced one: Where parties are assisted by a semi-honest entity such as a cloud provider.

Hence, we consider an abstract *computing party* $\mathcal{C}$ that carries out the homomorphic evaluation of the ideal functionality. In purely peer-to-peer settings, the parties themselves assume the role of $\mathcal{C}$, either by distributing the computed circuit, or by delegating the computation to one designated party. In the cloud-assisted setting, a semi-honest cloud service provider can assume this role. Although it is frequent to define the role of *computing party* in current MPC applications [5], [9], [15], it is usually a part of the N-party to 2-party problem reduction that introduces non-collusion assumptions. The fundamental difference with the MHE-based protocol is that $\mathcal{R}$ owns no share of the ideal secret-key, and thus plays no role access-structure of the private inputs.

The *Setup* step instantiates the multiparty encryption scheme. It is independent from the rest of the protocol: it has to be run only once for a given set of parties and a given choice (λ, κ) of cryptographic parameters. Whereas it is tempting to compare this step to the *offline* phase of the secret-sharing-based approaches, it is fundamentally different in that it produces public-keys that can be used for an unlimited number of circuit evaluations. This implies that the Setup step does not depend directly on the number of multiplication gates in the circuit, but only on the maximum circuit depth the parties want to support. This is because the encryption scheme has to be parameterized, and the necessary public keys generated, to support a sufficient *homomorphic capacity* κ.

The *In* step corresponds to the input phase: The parties use the public encryption procedure of E^* (using the *collective public-key* cpk) to encrypt their inputs and to provide them to $\mathcal{C}$ for the evaluation to be carried out.

The *Eval* step consists in the evaluation of the *ideal functionality*, by exploiting the homomorphic property of the scheme to carry out the computation of the encrypted output. This step requires no secret input from the parties and can hence be performed by any semi-honest entity $\mathcal{C}$.

The *Out* step enables the receiver $\mathcal{R}$ to obtain its output. This requires collaboration among the parties in $\mathcal{P}$, according to the access structure defined by the sharing of the ideal secret key. As a public output might not be acceptable in all scenarios, we augment the distributed cryptosystem E^* with a *collective key switching* protocol $\pi_{\mathsf{ColKeySwitch}}$, which enables the parties to obliviously re-encrypt a ciphertext that originally decrypts under a shared secret key sk into a new ciphertext that decrypts under the receiver's secret key sk'. This protocol generalizes π_{Dec}; in fact, decryption (or the ability to decrypt) can be mapped to the particular case of switching to a secret key $\mathsf{sk}' = 0$ (or any publicly known value). Given that pk' is a public key for the secret key sk', the ideal functionality of key switching,

$$f_{\mathsf{ColKeySwitch}} = E.\mathsf{Encrypt}(\mathsf{pk}', E.\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})),$$

can be computed with no secret input from the receiver, hence fully decoupling this party from the secure multiparty computation problem.

E. Mathematical Notation

We denote $[\cdot]_q$ the reduction of an integer modulo q, and $\lceil \cdot \rfloor$, $\lfloor \cdot \rfloor$, $\lfloor \cdot \rceil$ the rounding to the next, previous, and nearest integer respectively. When applied to polynomials, these operations are performed coefficient-wise. We use regular letters for integers and polynomials, and boldface letters for vectors of integers and of polynomials. $\boldsymbol{a}^T$ denotes the transpose of a vector $\boldsymbol{a}$. Given a probability distribution α over a ring R, $a \leftarrow \alpha$ denotes the sampling of an element $a \in R$ according to α, and $a \leftarrow R$ implicitly denotes uniform sampling in R. For a polynomial a we denote its infinity norm by $\|a\|$.

F. The Brakerski-Fan-Vercauteren Encryption Scheme

The BFV cryptosystem is a ring-learning-with-errors [44] scheme that supports both additive and multiplicative homomorphic operations. Due to its practicality, it has been implemented in most of the current lattice-based cryptographic libraries [39], [41]–[43] and is included as part of the ongoing standardization effort [38].

We first recall the original and most common instantiation of the (centralized) BFV encryption scheme [17] that is detailed in Scheme 1. The ciphertext space is $R_q = \mathbb{Z}_q[X]/(X^n+1)$, the quotient ring of the polynomials with coefficients in $\mathbb{Z}_q$ modulo (X^n+1), where n is a power of 2. We use $[-\frac{q}{2}, \frac{q}{2})$ as the set of representatives for the congruence classes modulo q. Unless otherwise stated, we consider the arithmetic in R_q, so polynomial reductions are omitted in the notation. The relinearization operation relies on an intermediary base $w < q$, in which ciphertexts are temporarily decomposed to reduce the growth this operation incurs on the noise [17]. We write $l = \lceil \log_w(q) \rceil$ and $\mathbf{w} = (w^0, w^1, ..., w^l)^T$. The plaintext space is the ring $R_t = \mathbb{Z}_t[X]/(X^n+1)$ for $t < q$. We denote $\Delta = \lfloor q/t \rfloor$, the integer division of q by t.

The scheme is based on two kinds of secrets, commonly sampled from small-normed yet different distributions: The key distribution is denoted $R_3 = \mathbb{Z}_3[X]/(X^n+1)$, where coefficients are uniformly distributed in $\{-1, 0, 1\}$. The RLWE error distribution χ over R_q has coefficients distributed according to a centered discrete Gaussian with standard deviation σ and truncated support over $[-B, B]$. Tables V and VI in Appendix A summarize the cryptosystem parameters and symbols used in the encryption scheme formulation.

The security of BFV is based on the hardness of the *decisional-RLWE* problem [44], that is informally stated as follows: Given a uniformly random $a \leftarrow R_q$, a secret $s \leftarrow R_3$, and an error term $e \leftarrow \chi$, it is computationally hard for an adversary that does not know s and e to distinguish between the distribution of $(sa+e, a)$ and that of (b, a) where $b \leftarrow R_q$.

Encrypted arithmetic operations must preserve the plaintext arithmetic. The BFV.Multiply operation outputs a ciphertext consisting of three R_q elements; it can be seen as a *degree two* ciphertext. This higher degree ciphertext can be further operated on and decrypted. But it is often desirable to reduce the ciphertext degree back to one, by using the BFV.Relinearize operation. This operation is public but requires the generation of a special type of public key, the relinearization key rlk.

In the BFV scheme, decryption of a ciphertext (c_0, c_1) can be seen as a two-step process. The first step requires the secret key to compute a noisy plaintext in R_q as

$$[c_0 + sc_1]_q = \Delta m + e_{\mathsf{ct}}, \tag{1}$$

where e_{ct} is the ciphertext overall error, or *ciphertext noise*. In the second step, the message is decoded from the noisy term in R_q to a plaintext in R_t, by rescaling and rounding:

$$[\lfloor \frac{t}{q} (\Delta m + e_{\mathsf{ct}}) \rceil]_t = [\lfloor m + at + v \rceil]_t, \tag{2}$$

where $m \in R_t$, a has integer coefficients, and v has coefficients in $\mathbb{Q}$. Provided that $\|v\| < \frac{1}{2}$, Eq. (2) outputs m. Hence, the correctness of the scheme is conditioned on the noise magnitude $\|e_{\mathsf{ct}}\|$, that must be kept below $\frac{q}{2t}$ throughout the homomorphic computation, notably by choosing a sufficiently large q. This choice depends on the operations to be performed, hence on the application.

Scheme 1: BFV

BFV.SecKeyGen(1^λ): Sample $s \leftarrow R_3$. Output: $\mathsf{sk} = s$

BFV.PubKeyGen(sk):
Let $\mathsf{sk} = s$. Sample $p_1 \leftarrow R_q$, and $e \leftarrow \chi$. Output:
$$\mathsf{pk} = (p_0, p_1) = (-(sp_1 + e), p_1)$$

BFV.RelinKeyGen(sk, $\mathbf{w}$):
Let $\mathsf{sk} = s$. Sample $\mathbf{r}_1 \leftarrow R_q^l$, $\mathbf{e} \leftarrow \chi^l$. Output:
$$\mathsf{rlk} = (\mathbf{r}_0, \mathbf{r}_1) = (s^2\mathbf{w} - s\mathbf{r}_1 + \mathbf{e},\ \mathbf{r}_1)$$

BFV.Encrypt(pk, m):
Let $\mathsf{pk} = (p_0, p_1)$. Sample $u \leftarrow R_3$ and $e_0, e_1 \leftarrow \chi$. Output:
$$\mathsf{ct} = (\Delta m + up_0 + e_0\ ,\ up_1 + e_1)$$

BFV.Decrypt(sk, ct):
Let $\mathsf{sk} = s$, $\mathsf{ct} = (c_0, c_1)$. Output:
$$m' = [\lfloor \frac{t}{q}[c_0 + c_1 s]_q \rceil]_t$$

BFV.Add(ct, ct′):
Let $\mathsf{ct} = (c_0, c_1)$ and $\mathsf{ct}' = (c_0', c_1')$ Output:
$$\mathsf{ct}_{\text{add}} = (c_0 + c_0',\ c_1 + c_1')$$

BFV.Multiply(ct, ct′):
Let $\mathsf{ct} = (c_0, c_1)$ and $\mathsf{ct}' = (c_0', c_1')$ Output:
$$\mathsf{ct}_{\text{mul}} = [\lfloor \frac{t}{q}(c_0c_0'\ ,\ c_0c_1' + c_0'c_1\ ,\ c_1c_1') \rceil]_q$$

BFV.Relinearize(ct, rlk):
Let $\mathsf{ct} = (c_0, c_1, c_2)$, $\mathsf{rlk} = (\mathbf{r}_0, \mathbf{r}_1)$
Express c_2 in base w s.t. $c_2 = \sum_{b=0}^{l} c_2^{(b)} w^b$ and output:
$$\mathsf{ct}_{\text{relin}} = (c_0 + \sum_{b=0}^{l} \mathbf{r}_{0,b} c_2^{(b)}\ ,\ c_1 + \sum_{b=0}^{l} \mathbf{r}_{1,b} c_2^{(b)})$$

IV. The Multiparty BFV Scheme

We introduce a novel multiparty version of the Brakerski-Fan-Vercauteren (BFV) cryptosystem [17] that supports the MPC protocol of Section III-D. It is worth noting that, although formulated for the BFV scheme, the introduced protocols can be straightforwardly adapted to other RLWE-based cryptosystems, such as BGV [50] or the more recent CKKS [18] that enables homomorphic approximate arithmetic. In fact, we implemented both multiparty versions for the BFV and CKKS schemes in an open-source library [43].

We use additive secret-sharing to distribute the BFV secret key, denoted as s in the following, among the N parties in $\mathcal{P}$. We denote s_i the secret key share of party P_i, thus

$$s = \left[\sum_{P_i \in \mathcal{P}} s_i \right]_q. \tag{3}$$

As a result, this scheme tolerates up to $N-1$ colluding corrupted nodes in the passive adversary model and can be viewed as a *N-out-of-N threshold* encryption scheme. Hence,

when used as E^* in the MHE–MPC protocol, this scheme can solve secure-multiparty-computation problems in the strictest dishonest-majority formulation: where no set of colluding parties should be able to extract the inputs of an honest party.

We refer to the original *centralized* scheme as *the ideal scheme*: the ideal centralized functionality that needs to be emulated in a multiparty setting. By extension, we also refer to s as *the ideal secret key*, to emphasize that it exists as such only through interaction between the parties. In the next subsections, we reformulate all the private operations of the original BFV scheme (i.e., those that depend on the secret key) as secure N-party protocols. In Section VI, we detail the security arguments.

These protocols follow a round-based approach where each round decomposes in two phases: *share generation* (Gen) and *share aggregation* (Aggr). In the Gen phase, the parties generate their public shares for the current round (using their own private share of the ideal secret key) and they disclose them to the rest of the parties. In the Aggr phase, the shares of all the parties are aggregated into a single output for the round. For each protocol, the *output procedure* (Out), computes the final protocol output from the round outputs. In order to abstract the actual system model, we do not define how parties disclose and aggregate their shares yet. In Section V, we present concrete system models and discuss their features.

A. Ideal-Secret-Key Generation

We propose a simple ideal-secret-key generation procedure, in which each party independently samples its own share as $s_i = \mathsf{BFV.SecKeyGen}(1^\lambda)$. Thus, the ideal secret-key is generated in a non-interactive way. Although Eq. (3) applies, this does not result in a *usual* sharing of s, in the sense that the distribution of the shares is not uniform in R_q. This is not an issue because, as discussed in Section VI, the security of our scheme does not rely on this property. Moreover, the norm of the resulting ideal secret key grows with N. This has an effect on the noise growth that we analyze in Appendix C.

By using techniques such as those described in [51], it might be possible to generate ideal secret keys in R_3 as if they were produced in a trusted setup. However, this would introduce the need for private channels between the parties.

B. Collective Encryption-Key Generation (EncKeyGen)

The collective encryption-key generation, detailed in Protocol 2, emulates the BFV.PubKeyGen procedure. It is part of the setup phase of the MHE–MPC protocol. In addition to the public parameters of the cryptosystem (which we will omit in the following), the procedure requires a public polynomial p_1, uniformly sampled in R_q, to be agreed upon by all the parties. For this purpose, they sample its coefficients from the common reference string (CRS).

After the execution of the EncKeyGen protocol, the parties have access to a copy of the collective public key

$$\mathsf{cpk} = ([\sum_{P_i\in\mathcal{P}} p_{0,i}]_q,\ \ p_1) = ([-(p_1 \sum_{P_i\in\mathcal{P}} s_i + \sum_{P_i\in\mathcal{P}} e_i)]_q,\ \ p_1)\,; \tag{4}$$

Protocol 2: EncKeyGen

Public Input: p_1 (a common random polynomial)
Private Input for P_i: $s_i = \mathsf{sk}_i$ (the party's secret key share)
Public Output: $\mathsf{cpk} = (p_0, p_1)$ (the collective encryption key)

Each party P_i:
1) samples $e_i \leftarrow \chi$ and discloses $p_{0,i} = -(p_1 s_i + e_i)$

Out: from $p_0 = \sum_{P_j\in\mathcal{P}} p_{0,j}$, outputs $\mathsf{cpk} = (p_0\ ,\ p_1)$

it has the same form as the ideal public key pk in Scheme 1, with larger worst-case norms $\|s\|$ and $\|e\|$. As detailed in Appendix C, the norm grows linearly in N hence is not a concern, even for large number of nodes. Another notable feature of the EncKeyGen protocol is that it would apply to any kind of additive sharing of s, including uniformly random shares. In fact, in the multiparty scheme, the magnitude of the secret-key shares is irrelevant (as long as they have the required entropy); only their sum in R_q is.

C. Collective Relinearization-Key Generation (RelinKeyGen)

In order to enable the computing entity $\mathcal{C}$ to perform non-interactive relinearizations (hence, non-interactive multiplications), the parties need to generate a public *relinearization key* (rlk) associated with their ideal secret key s. Protocol 3 (RelinKeyGen) emulates the centralized BFV.RelinKeyGen for this purpose; it produces pseudo-encryptions of $s^2 w^b$ for each power $b = 0 \dots l$ of the decomposition basis parameter w. This protocol is also part of the setup phase of the MHE–MPC protocol. It requires a public input $\mathbf{a}$, uniformly sampled in R_q^l from the CRS. We use vector notation to express that these pseudo-encryptions are generated in parallel for every element of the decomposition base $\mathbf{w} = (w^0, w^1, ..., w^l)^T$.

Asharov et al. proposed a method to produce relinearization keys for multiparty schemes based on the LWE problem [20]. This method could be adapted to our scheme but results in significantly increased noise in the rlk (hence, higher noise in relinearized ciphertexts) with respect to the centralized scheme. One cause for this extra noise is the use of the public encryption algorithm to produce the mentioned pseudo-encryptions. From the observation that the collective encryption key is not needed for this purpose (when the secret key is collectively known), we propose Protocol 3 as an improvement over the method by Asharov et al., and we compare both approaches in Appendix D.

After completing Protocol 3, the parties have access to a relinearization key of the form:

$$\begin{aligned}\mathsf{rlk} =\ & (\mathbf{r}_0, \mathbf{r}_1)\\ =\ & (-s^2\mathbf{a} + s^2\mathbf{w} + s\mathbf{e}_0 + \mathbf{e}_1 + (u-s)\mathbf{e}_2\ ,\ s\mathbf{a} + \mathbf{e}_2)\\ =\ & (-s\mathbf{b} + s^2\mathbf{w} + s\mathbf{e}_0 + \mathbf{e}_1 + u\mathbf{e}_2 + \mathbf{e}_3\ \ ,\ \ \mathbf{b}\,)\\ =\ & (-s\mathbf{b} + s^2\mathbf{w} + \mathbf{e}_{rlk}\ \ ,\ \ \mathbf{b}\,),\end{aligned} \tag{5}$$

where $\mathbf{b} = s\mathbf{a} + \mathbf{e}_2$. Hence, as opposed to the technique by Asharov et al., the additional error with respect to the ideal generation using BFV.RelinKeyGen is only introduced in the $\mathbf{r}_0$ component. In Appendix C, we show that this results in a significant reduction in the relinearization noise.

Protocol 3: RelinKeyGen

Public Input: $\mathbf{a} \in R_q^l$, $\mathbf{w}$
Private Input of P_i: $\mathsf{sk} = s_i$
Output: $\mathsf{rlk} = (\mathbf{r}_0, \mathbf{r}_1)$

Each party P_i:

1) samples $\mathbf{e}_{0,i} \leftarrow \chi^l$, $u_i \leftarrow R_3$ and discloses $\mathbf{h}_i = -u_i\mathbf{a} + s_i\mathbf{w} + \mathbf{e}_{0,i}$
2) from $\mathbf{h} = \sum_{P_j \in \mathcal{P}} \mathbf{h}_j$, samples $\mathbf{e}_{1,i}, \mathbf{e}_{2,i} \leftarrow \chi^l$ and discloses
$$\mathbf{h}'_{0,i} = s_i\mathbf{h} + \mathbf{e}_{1,i} \quad \text{and} \quad \mathbf{h}'_{1,i} = s_i\mathbf{a} + \mathbf{e}_{2,i}$$
3) from $\mathbf{h}'_0 = \sum_{P_j \in \mathcal{P}} \mathbf{h}'_{0,j}$ and $\mathbf{h}'_1 = \sum_{P_j \in \mathcal{P}} \mathbf{h}'_{1,j}$, samples $\mathbf{e}_{3,i} \leftarrow \chi^l$ and discloses $\mathbf{h}''_i = (u_i - s_i)\mathbf{h}'_1 + \mathbf{e}_{3,i}$

Out: from $\mathbf{h}'' = \sum_j \mathbf{h}''_j$, outputs $\mathsf{rlk} = (\mathbf{r}_0, \mathbf{r}_1) = (\mathbf{h}'_0 + \mathbf{h}'' \ , \ \mathbf{h}'_1)$

A relevant feature of the proposed RelinKeyGen protocol is its independence from the actual decomposition basis $\mathbf{w}$: It is compatible with other decomposition techniques, such as the one used for type II relinearization [17], and those based on the Chinese Remainder Theorem, as proposed by Bajard et al. [52] and Cheon et al. [53]. In our implementation, we use an hybrid approach for efficiency, but the protocol is unchanged.

D. Collective Key-Switching Protocols

The key-switching functionality enables the oblivious re-encryption operation to support the output procedure of the MHE–MPC protocol. That is, given a ciphertext ct decrypting under some *input key* s along with an *output key* s', the key-switching procedure computes ct' such that $\mathsf{Dec}(s, \mathsf{ct}) = \mathsf{Dec}(s', \mathsf{ct}')$. Its instantiation as a protocol depends on whether the parties performing the re-encryption have access to a sharing of the output secret key (i.e., have a collective access to it), or only have its corresponding public-key. Therefore, we develop protocols that perform key-switching for these two settings: when s' is collectively known, the ColKeySwitch protocol is used and when only a public key is known, the PubColKeySwitch protocol is used.

Both protocols require some fresh noise terms to be sampled from a special noise distribution χ_{CKS} that depends on the ciphertext that is key-switched. This fresh noise implements the *smudging* technique (as introduced by Asharov et al. [20]), whose motivation and implementation we discuss in Section IV-G. Given a ciphertext ct, we denote $\mathsf{var}(\mathsf{ct})$ the variance of its noise term (see Eq. (1)).

*a) Collective Key-Switching (*ColKeySwitch*):* Protocol 4 details the steps for operating a key switching when the input parties collectively know the output secret key s'. In the context of the MHE–MPC protocol, this would be the case in multiple scenarios such as the decryption procedure (as discussed below) and in the case of an ideal secret key update (when a party leaves or joins the system). Also, assuming confidential party-to-party channels, a receiver could provide the input parties with secret-shares of a secret key it knows.

Protocol 4: ColKeySwitch

Public input: $\mathsf{ct} = (c_0, c_1)$ with $\mathsf{var}(\mathsf{ct}) = \sigma^2_{\mathsf{ct}}$
Private input for P_i: s_i, s'_i
Public output: $\mathsf{ct}' = (c'_0, c_1)$

Each party P_i:

1) samples $e_i \leftarrow \chi_{\mathsf{CKS}}(\sigma_{\mathsf{ct}})$ and discloses $h_i = (s_i - s'_i)c_1 + e_i$

Out: from $h = \sum_j h_j$, outputs $\mathsf{ct}' = (c'_0, c_1) = (c_0 + h, c_1)$

After the execution of the ColKeySwitch protocol on an input $\mathsf{ct} = (c_0, c_1)$, for which $c_0 + sc_1 = \Delta m + e_{\mathsf{ct}}$ where e_{ct} is the ciphertext's error, the parties have access to ct' satisfying

$$\begin{aligned}
\mathsf{BFV.Dec}(s', \mathsf{ct}') &= \lfloor \frac{t}{q}[c'_0 + s'c_1]_q \rceil \\
&= \lfloor \frac{t}{q}[c_0 + \sum_j \big((s_j - s'_j)c_1 + e_j\big) + s'c_1]_q \rceil \\
&= \lfloor \frac{t}{q}[c_0 + (s - s')c_1 + e_{\mathsf{CKS}} + s'c_1]_q \rceil \\
&= \lfloor \frac{t}{q}[\Delta m + e_{\mathsf{ct}} + e_{\mathsf{CKS}}]_q \rceil = m, \qquad (6)
\end{aligned}$$

where $e_{\mathsf{CKS}} = \sum_j e_j$, and where the last equality holds, provided that $\|e_{\mathsf{ct}} + e_{\mathsf{CKS}}\| < q/(2t)$; i.e., if the output ciphertext noise plus the protocol-induced noise remains within decryptable bounds. The ColKeySwitch protocol yields a decryption procedure, as the special case where $s'_j = 0 \ \ \forall P_j \in \mathcal{P}$, and is the basis for bridging MHE-based and secret-sharing-based approaches, as explained in Section IV-E.

*b) Collective Public-Key Switching (*PubColKeySwitch*):* When considering an external receiver for the output, the parties in $\mathcal{P}$ do not have access to the output secret key s'. In this case, the ColKeySwitch protocol can only be used when assuming individual confidential channels between the external receiver and each party in $\mathcal{P}$. This channel could be used by the external receiver (i) to collect decryption shares from all parties, or (ii) to upload an additive sharing of its secret key to the system. However, (i) would quickly become expensive for large number of parties, and (ii) would would require $\mathcal{R}$ to trust at least one party in $\mathcal{P}$. Additionally, confidential point-to-point channels might not fit the system model (e.g., on public smart-contract technologies). Hence, we introduce the PubColKeySwitch protocol to overcome this issue.

Protocol 5 details the steps for key switching when the input parties know only a public key for the output secret key s'. As it requires only public input from the external receiver and its output is encrypted under the receiver's key, the PubColKeySwitch turns the ColKeySwitch protocol into a public-transcript, public-output one and decouples the receiver from the SMC problem at hand.

After the execution of the PubColKeySwitch protocol on an input ciphertext $\mathsf{ct} = (c_0, c_1)$ for which $c_0 + sc_1 = \Delta m + e_{\mathsf{ct}}$, and a target public key $\mathsf{pk} = (p'_0, p'_1)$ such that $p'_0 = -(s'p'_1 +$

Protocol 5: PubColKeySwitch

Public input: $\mathsf{pk}' = (p'_0, p'_1)$,
$\mathsf{ct} = (c_0, c_1)$ with $\mathsf{var}(\mathsf{ct}) = \sigma^2_{\mathsf{ct}}$
Private input for P_i: s_i
Public output: $\mathsf{ct}' = (c'_0, c'_1)$

Each party P_i:

1) samples $u_i \leftarrow R_3$, $e_{0,i} \leftarrow \chi_{\mathsf{CKS}}(\sigma_{\mathsf{ct}})$, $e_{1,i} \leftarrow \chi$ and discloses
$$h_{0,i} = s_i c_1 + u_i p'_0 + e_{0,i} \quad \text{and} \quad h_{1,i} = u_i p'_1 + e_{1,i}$$

Out: from $h_0 = \sum_j h_{0,j}$ and $h_1 = \sum_j h_{1,j}$, outputs $\mathsf{ct}' = (c'_0, c'_1) = (c_0 + h_0, h_1)$

e_{pk}), the parties have access to ct' satisfying

$$\begin{aligned}
&\mathsf{BFV.Dec}(s', \mathsf{ct}') = \lfloor \frac{t}{q}[c'_0 + s'c'_1]_q \rceil \\
&= \lfloor \frac{t}{q}[c_0 + \sum_j \left(s_j c_1 + u_j p'_0 + e_{0,j}\right) + s' \sum_j \left(u_j p'_1 + e_{1,j}\right)]_q \rceil \\
&= \lfloor \frac{t}{q}[c_0 + sc_1 + up'_0 + s'up'_1 + e_0 + s'e_1]_q \rceil \\
&= \lfloor \frac{t}{q}[\Delta m + e_{\mathsf{ct}} + e_{\mathsf{PCKS}}]_q \rceil = m, \qquad (7)
\end{aligned}$$

where $e_d = \sum_j e_{d,j}$ for $d = 0, 1$, $u = \sum_j u_j$, and the total added noise $e_{\mathsf{PCKS}} = e_0 + s'e_1 + ue_{\mathsf{pk}}$ depends on both the protocol-induced and the public-key noises. Provided that $\|e_{\mathsf{ct}} + e_{\mathsf{PCKS}}\| < q/(2t)$, Equation (7) holds.

E. Bridging MPC approaches

The flexibility of the ColKeySwitch protocol can be harnessed to bridge MHE-based and secret-sharing-based MPC approaches. We provide two procedures enabling *encryption-to-shares* and *shares-to-encryption* functionalities:

*1) Encryption-to-Shares (*Enc2Share*):* Given an encryption (c_0, c_1) of a plaintext $m \in R_t$, the parties can produce an additive sharing of m over R_t by masking their share in the decryption (i.e., ColKeySwitch with $s' = 0$) protocol: Each party $P_i \in \{P_2, P_N\}$ samples its own additive share $M_i \leftarrow R_t$ and adds a $-\Delta M_i$ term to its decryption share h_i before disclosing it. Party P_1 does not disclose its decryption share h_1 and derives its own additive secret-share of m as

$$M_1 = \mathsf{BFV.Decrypt}(s_1, (c_0 + \sum_{i=2}^{N} h_i, c_1)) = m - \sum_{i=2}^{N} M_i.$$

*2) Shares-to-Encryption (*Share2Enc*):* Given a secret shared value $m \in R_t$ such that $m = \sum_{i=1}^{N} M_i$, the parties can produce an encryption $\mathsf{ct} = (c_0, c_1)$. To do so, each party P_i samples a from the CRS and and produces a ColKeySwitch share for the ciphertext $(\Delta M_i, a)$ with input key 0 and output key s. The ciphertext centralizing the secret-shared value m is then $\mathsf{ct} = (\sum_{i=1}^{N} c_{0,i}, a)$. This is equivalent to a *multiparty encryption* protocol.

*F. Collective Bootstrapping (*ColBootstrap*)*

The Share2Enc and Enc2Share protocols can be combined into a *multiparty bootstrapping* procedure (Protocol 6), enabling the parties to reduce the ciphertext noise back to a *fresh*-looking one, which further enable homomorphic computation to be performed even when reaching the homomorphic capacity limits. This is a crucial functionality for the BFV scheme, for which no practical bootstrapping procedure exists.

Protocol 6: ColBootstrap

Public input: a and $\mathsf{ct} = (c_0, c_1)$ with $\mathsf{var}(\mathsf{ct}) = \sigma^2_{\mathsf{ct}}$
Private input for P_i: s_i
Public output: $\mathsf{ct}' = (c'_0, c'_1)$ with $\mathsf{var}(\mathsf{ct}') = N\sigma^2$

Each party P_i:

1) samples $M_i \leftarrow R_t$, $e_{0,i} \leftarrow \chi_{\mathsf{CKS}}(\sigma_{\mathsf{ct}})$, $e_{1,i} \leftarrow \chi$ and discloses
$$h_{0,i} = s_i c_1 - \Delta M_i + e_{0,i} \text{ and } h_{1,i} = -s_i a + \Delta M_i + e_{1,i},$$

Out: from $h_0 = \sum_j h_{0,j}$ and $h_1 = \sum_j h_{1,j}$, outputs $\mathsf{ct}' = (c'_0, c'_1) =$
$$([\lfloor \frac{t}{q}([c_0 + h_0]_q)\rceil]_t \Delta + h_1 \ , \ a)$$

Intuitively, the ColBootstrap protocol consists in a conversion from an encryption to secret-shares and back, implemented as a parallel execution of the Enc2Share and Share2Enc protocols. It compensates the absence of a public bootstrapping with an efficient single-round protocol that the parties can use during the evaluation phase. Note, however, that making use of that functionality introduces interaction at the evaluation step of the MHE–MPC protocol. In practice, a broad range of applications would not (or seldom) need to rely on this primitive, as the circuit complexity enabled by the practical parameters of the BFV scheme would suffice (e.g., this is the case for all three examples in Section VII). But, MHE–MPC offers a trade-off between computation and communication, and more flexibility when the computation circuit is not known in advance.

G. Smudging

RLWE-based cryptosystems have the fundamental property of decrypting to noisy plaintext messages (Eq. (1)) that are then *decoded* by the decryption algorithm (Eq. (2)). As the noise depends on the evaluation circuit and its intermediate values (this dependency is characterized in Appendix C), this cryptosystem family does not ensure *circuit privacy*. This has an important implication for our multiparty scheme, where Equations (6) and (7) show that both ColKeySwitch and PubColKeySwitch permit the final error term to be obtained by the receiver. Two important facts must be considered: (a) The hardness assumption for the cryptosystem holds only if the adversary has sufficient uncertainty about the noise terms, and (b) the key-switched ciphertext noise distribution depends on the source secret key and can depend on the plaintext messages when the ciphertext is not fresh (see Appendix C). By exploiting the aforementioned dependencies, an attacker with knowledge of the output key could attempt to extract *some* information about the input key intermediate plaintext values in the computation.

Although characterizing this indirect leakage in a computational setting is currently an open question, we can already address this problem by means of *smudging* techniques, as introduced by Asharov et al. [20]. Conceptually, smudging with a noise distribution χ_{CKS} ensures that the decryption of a key-switched ciphertext, before quantization and rounding, is indistinguishable from the decryption of a *fresh* one that would be encrypted using χ_{CKS} as its error distribution. This is achieved by sampling part of the noise introduced during the ColKeySwitch and PubColKeySwitch protocols from a $\chi_{\mathsf{CKS}}(\sigma_{\mathsf{ct}})$ distribution that must have variance σ^2_{smg} significantly *larger* than that of the input ciphertext noise distribution (represented by σ^2_{ct}). Concretely, choosing

$$\sigma^2_{\mathrm{smg}} = 2^{\lambda}\sigma^2_{\mathsf{ct}} \tag{8}$$

guarantees that this is the case. Analogously to the *fresh* noise distribution, the smudging noise can also be drawn from a truncated Gaussian. This technique assumes that the system keeps track of the ciphertext noise-level throughout the MHE–MPC protocol, which must already be the case to ensure correctness of the computation. We introduce smudging noise exclusively during the ColKeySwitch and PubColKeySwitch protocols (and their derivatives Enc2Share and ColBootstrap), where it is a single additive term in the c'_0 element. This differs from the method of Asharov et al., where they introduce smudging noise also during evaluation. We detail the corresponding security argument in Section VI-B.

H. Vector Packed-Encoding and Rotation Keys

One of the most powerful features of RLWE-based schemes is the ability to embed vectors of plaintext values into a single ciphertext. Such techniques, commonly referred to as *packing*, enable arithmetic operations to be performed in a *single-instruction multiple-data* fashion, where encrypted arithmetic results in component-wise plaintext arithmetic. Provided with public so-called *rotation keys*, any semi-honest party can operate arbitrary rotations over the vector components, which opens up homomorphic function evaluation to a broad kind of non-linear functions. Producing these rotation keys requires the secret key and can be done in the multiparty scheme, by means of an RotKeyGen sub-protocol. We do not detail the RotKeyGen protocol, as it is a straightforward adaptation of EncKeyGen. The generation and usage of rotation keys is part of our implementation and is showcased in Section VII.

V. Features Analysis

The MHE-based MPC solution can be used in conjunction with secret-sharing-based protocols or as a standalone one. In both cases, the MHE-based solution covers several limitations of the traditional ones. In this section, we summarize the fundamental features of the MHE–MPC protocol and relate them to new design and system model they enable.

A. Public Non-interactive Circuit Evaluation

Although the homomorphic operations of HE schemes are computationally more expensive than local operations of secret-shared arithmetic, the former do not require private inputs from the parties. Hence, as long as no key switching or bootstrapping is needed, the procedure is non-interactive and can be performed by any semi-honest party. This not only enables the circuit evaluation to be efficiently distributed among the parties in the traditional peer-to-peer setting, but also enables new computation models for MPC:

1) Cloud-Outsourced Model: The homomorphic circuit evaluation can be outsourced to a cloud-like service, by providing it with the inputs and necessary public-keys (encryption and evaluation keys). The parties can arbitrarily go offline during the evaluation and reconnect for the final output phase. In this model, resource-constrained parties can take part in MPC tasks involving thousands of other parties.

2) Smart Contracts: A special case of an outsourced MPC task is the execution of a smart contract over private data, which becomes feasible under the MHE-based MPC solution.

B. Public-Transcript Protocols

All the protocols of Section IV have public transcripts, which effectively removes the need for direct party-to-party communication. Hence, the whole MHE–MPC protocol can be executed over any public authenticated channel. This also brings new possibilities in designing MPC systems:

1) Efficient Communication Pattern: The presented protocols rely solely on the ability for the parties to publicly *disclose* their shares and to aggregate them, thus leaves flexibility for using efficient communication patterns: The parties can be organized in a topological way, as nodes in a tree, where each node would interact solely with its parent and children nodes. We observe that for all the protocols, the shares are always combined by computing their sum. Hence, for a given party in our protocols, a round would consist in

1) Computing its own share in the protocol,
2) Collecting and aggregating the share of each of its children and its own,
3) Sending the result *up the tree* to its parent.

Such an execution enables the parties to compute their shares in parallel and results in a constant network traffic at each node. Inbound traffic can be kept low by ensuring that the branching factor of the tree (i.e., the number of children per node) is manageable for each node. Because the share aggregation can also be computed by any semi-honest third-party, the tree can contain nodes that are not part of $\mathcal{P}$ (i.e., nodes that would not have input in the MPC problem and have no share of the ideal secret key) and are simply aggregating and forwarding their children's shares. We demonstrate the efficiency of the tree topology in the multiplication triple generation example benchmark in Section VII-D.

2) Cloud-Assisted MPC Model: The special case of a single root node holding no share of the key corresponds to a cloud-assisted setting where parties run the protocols, while interacting solely with a central node. This model complements the circuit evaluation outsourcing feature by completely removing the need for synchronous and private party-to-party communication. It reduces the number of party-cloud interactions to two per circuit evaluation: one to provide

input and one to decrypt the output. We use the cloud-assisted model for the first two example circuits of Section VII.

C. Discussion

By re-balancing the cost of MPC tasks from interaction to computation, the MHE-based approach enables new multiparty settings that other approaches did not always cover efficiently. Notably, it supports MPC tasks that involve thousands of parties, without having to introduce non-collusion assumptions or impractical network loads. Its cloud-assisted instantiation strongly reduces the need for input parties to be online and active during the evaluation phase.

VI. Security Analysis

We analyze the security of the proposed multiparty BFV scheme in the passive adversary model. This section provides an intuition of the security argument for our specific protocols that are based on the *decision ring-learning-with-errors problem* [44]. For a more thorough analysis, we refer the reader to the works by Asharov et al. [20] and Boneh et al. [21]. Thanks to their generality, these works already cover most of our constructions (although they are formulated for the *learning-with-errors problem* [37]). We provide arguments in terms of the ideal/real simulation formalism [54] for the EncKeyGen (Section VI-A) and ColKeySwitch (Section VI-B) protocols, as the arguments for the remaining protocols are very similar to the latter. Because our RelinKeyGen protocol differs from the one by Asharov et al., we provide the security argument in Appendix E. We also extend their proofs by integrating smudging directly into the simulation-based proof for the ColKeySwitch protocol, instead of presenting it as an *ad-hoc* security measure.

Let $\mathcal{A}$ denote the adversary, defined as a subset of at most $N-1$ corrupted parties in $\mathcal{P}$. We prove by construction that, for every possible $\mathcal{A}$, there exists a *simulator program* S that, when provided only with $\mathcal{A}$'s input and output, can simulate $\mathcal{A}$'s view in the protocol. To achieve the privacy requirement, we require that $\mathcal{A}$ must not be able to distinguish the real view (generated from the honest parties' inputs) from the simulated one (generated with the adversary's inputs and outputs only). For a given value x, we denote $\tilde{x}$ its simulated equivalent. Unless otherwise stated, we consider computational indistinguishability between distributions, denoted $\tilde{x} \overset{c}{\equiv} x$.

Our threat model implies that there is at least one honest player that we denote P_h. The choice for P_h, among multiple honest parties, does not reduce generality. It does, however, help simplify the formulation of the security argument. We denote $\mathcal{H}$ the set $\mathcal{P} \setminus (\mathcal{A} \cup \{P_h\})$ of all other honest parties. Hence, the tuple $(\mathcal{A}, \mathcal{H})$ can represent any partition of $\mathcal{P} \setminus \{P_h\}$. In particular, both $\mathcal{A}$ and $\mathcal{H}$ can be empty in the following arguments.

A. Collective-Key Generation

We consider an adversary $\mathcal{A}$ that attacks the EncKeyGen protocol defined in Protocol 2. Along with s_i, we consider e_i as private inputs to the protocol for each party P_i (as if they were sampled before the protocol starts). Thus, we model the ideal functionality of the EncKeyGen protocol as $f_{\mathsf{CKG}}(\{s_i, e_i \mid P_i \in \mathcal{P}\}) = \mathsf{cpk}$, where $\mathsf{cpk} = (p_0, p_1)$ is the output for all parties, as in Eq. (4).

We observe that the view of each party in the execution of the EncKeyGen protocol comprises the tuple $(p_{0,1}, p_{0,2}, \ldots, p_{0,N})$ of all the players' shares, which corresponds to an additive sharing of p_0. S can simulate these shares by randomizing them under two constraints: (1) the simulated shares must sum up to p_0, and (2) the adversary shares must be equal to the real ones (otherwise, it could easily distinguish them). S can generate the share $\widetilde{p}_{0,i}$ of party P_i as

$$\widetilde{p}_{0,i} = \begin{cases} [-(s_i p_1 + e_i)]_q & \text{if } P_i \in \mathcal{A} \\ \leftarrow R_q & \text{if } P_i \in \mathcal{H} \\ [p_0 - \sum\limits_{P_j \in \mathcal{A} \cup \mathcal{H}} \widetilde{p}_{0,j}]_q & \text{if } P_i = P_h\,. \end{cases}$$

To show that $(\widetilde{p}_{0,1}, \widetilde{p}_{0,2}, \ldots, \widetilde{p}_{0,N}) \overset{c}{\equiv} (p_{0,1}, p_{0,2}, \ldots, p_{0,N})$, we observe that any probabilistic-polynomial-time adversary that distinguishes, with non-negligible advantage, between real and simulated shares of those players in $\mathcal{H}$ would directly yield a distinguisher for the decision-RLWE problem [44]. For the share of player P_h, we consider two cases: (1) When $\mathcal{H} \neq \emptyset$, the share $p_{0,h}$ is a uniformly random element in R_q because $[\sum_{P_j \in \mathcal{H}} p_{0,j}]_q$ is itself so, and the same indistinguishability argument as above applies. (2) In the presence of $N-1$ adversaries, $\mathcal{H} = \emptyset$, and S computes the real value for the honest party's share, hence outputting the real view.

B. Collective Key-Switching

The security argument for the ColKeySwitch protocol is inherently more complex than the previous ones, as the real protocol output only approximates the ideal one. As we show below, this enables us to formally express and characterize the need for *smudging* in multiparty lattice-based schemes. We show the analysis for the ColKeySwitch protocol only. However, the argument easily transfers to the other protocols that perform some form of decryption (either of messages or of the involved noise terms), such as the special case of collective decryption, the PubColKeySwitch and the ColBootstrap protocols.

Given a ciphertext $\mathsf{ct} = (c_0, c_1)$ decrypting under s, the ideal functionality of the ColKeySwitch protocol (Protocol 4) is to compute $\mathsf{ct}' = (c_0', c_1)$ decrypting under secret key s'. We first formulate this functionality implicitly: as the computation of c_0' satisfying

$$c_0 + s c_1 - e = c_0' + s' c_1 - e',$$

where e and e' are the noise terms resulting from the decryption of ct and ct', respectively. Hence, we consider its explicit form as an equivalent and minimal ideal multiparty functionality $\hat{f}_{\mathsf{CKS}}$, such that

$$\hat{f}_{\mathsf{CKS}}(\{s_i, s_i', e_i' | \forall P_i \in \mathcal{P}\}) = c_0' - c_0 = (s - s')c_1 - \hat{e} + e' = \hat{h},$$

where c_0, c_1 are considered public, $s = \sum_i s_i$, $s' = \sum_i s_i'$, and $\hat{e} = e$ is an ideal error term cancelling e. This is because, ideally, the output ciphertext must look fresh, even for an

adversary that knows all shares of s'; this is allowed in the ColKeySwitch protocol. As this term cannot be efficiently computed in practice, the real output differs from the ideal one. Simulation-based proofs permit this difference, as long as it can be proven that the ideal and real outputs are undistinguishable for the adversary (Property 1). This formalizes the need of smudging within the security argument. Then, we show that, even when the adversary has access to the real output, it cannot distinguish the simulated view from the real one (Property 2). Therefore, Properties (1) and (2) imply

$$(\widetilde{h}_1, \widetilde{h}_2, ..., \widetilde{h}_N,\ \hat{h}) \stackrel{c}{\equiv} (h_1, h_2, ..., h_N,\ h):$$

that the ColKeySwitch securely computes its functionality.

1) Output indistinguishability: We want to show that

$$(s+s')c_1 - \hat{e} + e' = \hat{h} \quad \stackrel{c}{\equiv} \quad h = (s+s')c_1 + e',$$

where h denotes the real protocol output. As the adversary is allowed to know s', we cannot rely on computational indistinguishability of the RLWE-like structure of h. Such an adversary can extract the noise from the decryption of the key-switched ciphertext, as $e+e' = c'_0+h+s'c_1-\Delta m$. Hence, we require this extracted noise to be *statistically indistinguishable* (denoted $\stackrel{s}{\equiv}$) from the fresh noise of the ideal output:

$$e' = e - \hat{e} + e' \stackrel{s}{\equiv} e + e'.$$

As e is the key-switched ciphertext error, it follows a centered Gaussian distribution whose variance we denote σ^2_{ct}. The second term e' is the sum of all the noise terms protecting the key-switching shares. It contains the smudging noise and are sampled according to the $\chi_{\mathsf{ColKeySwitch}}(\sigma_{\mathsf{ct}})$ distribution with variance $\sigma^2_{\mathsf{ColKeySwitch}}$. Thus, as long as the ratio $\sigma^2_{\mathsf{ct}}/\sigma^2_{\mathsf{ColKeySwitch}}$ is negligible, the two distributions are statistically indistinguishable, which implies that $\hat{h} \stackrel{c}{\equiv} h$.

2) View Indistinguishability: The view of any party in the ColKeySwitch protocol is an additive sharing $(h_1, h_2, ..., h_N)$ of h, which S can simulate as

$$\widetilde{h}_i = \begin{cases} [(-s_i + s'_i)c_1 + e'_i]_q & \text{if } P_i \in \mathcal{A} \\ a_i \leftarrow R_q & \text{if } P_i \in \mathcal{H} \\ [h - \sum\limits_{P_i \in \mathcal{A} \cup \mathcal{H}} \widetilde{h}_i]_q & \text{if } P_i = P_H. \end{cases}$$

When considering the distribution of the simulated and real views alone, the usual decision-RLWE assumption suffices: $(-s_ic_1 + e'_i, c_1)$ is undistinguishable from $(a \leftarrow R_q, c_1)$ for an adversary that does not know s_i and e'_i. However, we need to consider this distribution jointly with that of the real output. We recall that an adversary having access to s' can extract $e + e'$ from the output and might be able to estimate e'_i for $i \notin \mathcal{A}$. Hence, we need to make sure that the uncertainty the adversary has in estimating e'_i is sufficiently large to protect each share h_i in the ColKeySwitch protocol. As such requirement is application-related, we formalize it in the following condition.

Condition 1. *An input ciphertext (c_0, c_1) to the* ColKeySwitch *protocol is such that $c_0 + sc_1 = \Delta m + e_{\mathsf{ct}}$ where $e_{\mathsf{ct}} = e_{\mathcal{A}} + e_h$ includes a term e_h that is unknown to, and independent from, the adversary. Furthermore, e_h follows a distribution according to the RLWE hardness assumptions.*

If Condition 1 holds, we know that $\mathcal{A}$ can only approximate the term e_h up to an error $e_{\mathsf{ct},h}$, which is enough to make (h_h, c_1) indistinguishable from $(a \leftarrow R_q, c_1)$. In the scope of the MHE–MPC protocol, as long as all parties provide at least one input (for which the noise will be fresh), the requirement of Condition 1 is satisfied.

VII. Performance Analysis

We implemented the multiparty BFV scheme in an open-source library [43]. It provides Go implementations of the two most widespread RLWE homomorphic schemes: BFV and CKKS, along with their multiparty versions. The library uses state-of-the-art optimizations based on the chinese remainder theorem [52], number theoretic transforms [55], and generalized key switching procedures [56]. We evaluate the performance of the MHE–MPC protocol in example circuits, both in the cloud-assisted and peer-to-peer settings.

In the cloud-assisted setting, we consider two example circuits: (i) The component-wise product of large integer vectors, for which we discuss its application as a simple multiparty private-set-intersection protocol (Section VII-B). (ii) A multiparty input selection circuit, and its application to multiparty private-information-retrieval circuits (Section VII-C). We compare the performance for both circuits against a baseline system that uses a data-level secret-sharing approach: The MP-SPDZ library implementation [57] of the Overdrive protocol [26] for the semi-honest, dishonest majority setting. This system generates multiplication triples using plain homomorphic encryption (*offline* phase), then proceeds to compute the circuit using secret-shared arithmetic (the *online* phase).

In the peer-to-peer setting, we consider the generation of multiplication triples, commonly referred to as the "offline" phase of data-level secret-sharing-based approaches (Section VII-D). Finer-grained benchmarks at the primitive-operation level are presented in Appendix B. All experiments operate on private vector inputs for which the embedding in the plaintext space R_t is implicitly using *packed* representation.

A. Experimental Setup

For the cloud-assisted setting, the client-side timings were measured on a MacBook Pro with a 3.1 GHz Intel i5 processor. The server-side timings were measured on a 2.5 GHz Intel Xeon E5-2680 v3 processor (2x12 cores). For the peer-to-peer setting, we used the latter machine for all parties. We evaluate the network-related cost in terms of number of communicated bytes, to make it independent from the actual setting (note that this could slightly favor the baseline system, as it is more sensitive to the round-trip time delays in its online phase). Hence, we run all benchmarks over the localhost interface.

B. Component-Wise Vector Product

We consider a scenario in which each of the N input parties holds a private vector $\boldsymbol{x}_i$ of 32-bit integers. The ideal functionality consists in providing an *external* receiver

$\mathcal{R}$ (with secret key $s_{\mathcal{R}}$), with the component-wise product between the N private vectors. Thus, $f_{\mathcal{R}}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \boldsymbol{x}_1 \odot \boldsymbol{x}_2 \odot \cdots \odot \boldsymbol{x}_N = \boldsymbol{y}$ where $\odot$ denotes the component-wise product over integer vectors.

This circuit could be used, for example, to implement efficient multiparty private-set-intersection for very large number of parties. In its most simple instantiation, the parties could encode their sets as binary vectors and use this functionality to compute the bit-wise AND between them.

The steps in the MHE–MPC protocol unfolds as follow:

Setup The parties use the EncKeyGen and RelinKeyGen protocols to produce the public encryption cpk and relinearization rlk keys for to their joint secret key $s_{\mathcal{P}}$.

In Each input party $P_i \in \mathcal{P}$ encodes its input vector $\boldsymbol{x_i}$ as a polynomial x_i using *packed* plaintext encoding. Then, it encrypts this vector under the collective public key and sends $\mathsf{Enc}_{s_{\mathcal{P}}}(x_i)$ it to the cloud.

Eval The cloud computes the overall product using the BFV.Mul operation (with intermediary BFV.Relinearize operations). This results in $\mathsf{Enc}_{s_{\mathcal{P}}}(y)$ where y is the *packed* representation of $\boldsymbol{y}$. The cloud sends $\mathsf{Enc}_{s_{\mathcal{P}}}(y)$ to the input parties.

Out The input parties use the PubColKeySwitch protocol to reencrypt $\mathsf{Enc}_{s_{\mathcal{P}}}(y)$ into $\mathsf{Enc}_{s_{\mathcal{R}}}(y)$. Then, the receiver can come online and decrypt the result.

We set the vector size to $d = 2^{14}$. The communication is the sum of upstream and downstream. The baseline system [57] was configured for computation over the domain $\mathbb{Z}_p$ with a 32-bit p. The MHE system computes the same circuit with the multiparty BFV scheme, instantiated with $n = 2^{14}$, 438-bit q and plaintext modulus t of 32 bits. Table I reports on the evaluation of our implementation of the MHE solution against the baseline. We report the cost repartition per phase of the computation in Table II. The setup is the same as for Table I, only with a 16-bit plaintext modulus. This illustrates how the MHE–MPC protocol can solve large secure-multiparty-computation problems, even for resource-constrained clients, by delegating all the heavy computation and the storage requirements. We also show that parallelizing the evaluation of the homomorphic circuit can yield even lower response times.

Transferred to the PSI application, the results can, to some extent, be compared with the special purpose multiparty PSI protocol by Kolesnikov et al. [58]. For the standard semi-honest model with dishonest majority setting, set size 2^{12} and 15 parties (the largest evaluated value in [58]) in the LAN setting, the MHE solution was 1029 times faster and required 15.3 times less communication to compute the intersection. However, encoding the sets as binary vectors limits the application to finite sets. More advanced encodings should be investigated to match the flexibility of the approach by Kolesnikov et al.

C. Multiparty Input Selection

We consider a simple yet powerful *multiparty input selection* functionality where a party P_1 selects one among $N-1$ other parties' $P_2, \ldots, P_N$ inputs $x_2, \ldots, x_N$ while keeping the selector r private. This corresponds to the private ideal functionality $f_1(r, x_2, \ldots, x_N) = x_r$ for player P_1.

This selection circuit can be seen as generalizing an oblivious transfer functionality to the N-party setting, and can directly implement an N-party PIR system where a requestor party can retrieve a row in a database partitioned across multiple parties. We represent inputs as d-dimensional vectors in $\mathbb{Z}_p$ for p a 32-bit prime. We denote $\boldsymbol{u_i}$ the vector having all-zero coefficients but its i-th one, which is set to 1.

To compute the ideal functionality, the parties engage in the MHE–MPC protocol, the steps of which unfold as follows:

Setup The parties run EncKeyGen, RelinKeyGen and RotKeyGen to produce the encryption-, relinearization- and rotation- keys for their joint secret key s.

In Each *Provider* P_i encodes its input in R_t, encrypts it using the cpk as ct_i and sends it to the cloud.
The *Requester* generates its selector as the $\boldsymbol{u_r}$ vector, encrypts it under the cpk as ct_r and sends it to the cloud.

Eval For each provider input i, the cloud computes an encrypted mask m_i by (1) multiplying ct_r with u_i using ciphertext-plaintext multiplication and (2) replicating the i-th encrypted slot to the other slots by repeated column rotation and addition. Hence, m_i always encrypts the zero vector, except for $i = r$, for which it is all-ones. The cloud then multiplies each provider input x_i with the mask m_i using BFV.Mul, aggregates all N resulting ciphertexts with BFV.Add and applies the BFV.Relinearization to the resulting ciphertext ct_{out}.

TABLE I: Component-wise product: Baseline comparison

		Time [s]			Com./party [MB]		
# Parties		2	4	8	2	4	8
[57]	Offline	0.21	1.19	5.33	3.42	29.13	156.06
	Online	0.02	0.04	0.10	1.05	6.29	29.36
	Total	**0.24**	**1.24**	**5.52**	**4.47**	**35.42**	**185.42**
MHE	Setup	0.18	0.20	0.25	25.17	25.17	25.17
	Circuit	**0.29**	**0.41**	**0.64**	**4.72**	**4.72**	**4.72**

TABLE II: Component-wise product: Phases costs

	Party		Cloud		
	Time [ms]	Com. [MB]	CPU time (Wall time) [s]		
# Parties	*indep.*	*indep.*	32	64	128
Setup	96.41	25.17	0.49	0.85	1.99
In	20.02	1.57	0.04	0.04	0.15
Eval	0.00	0.00	4.5(0.8)	10.3(1.0)	22.7(1.5)
Out	25.38	3.15	0.05	0.10	0.21

TABLE III: Input selection: Baseline comparison

		Time [s]			Com./party [MB]		
# Parties		2	4	8	2	4	8
[57]	Offline	0.35	1.04	3.56	6.58	25.74	101.82
	Online	0.02	0.04	0.07	1.31	4.72	17.83
	Total	**0.37**	**1.08**	**3.66**	**7.89**	**30.46**	**119.65**
MHE	Setup	0.59	0.58	0.69	42.93	42.93	42.93
	Circuit	**0.27**	**0.28**	**0.31**	**1.31**	**1.31**	**1.31**

TABLE IV: Input selection: Phase Costs

	Party		Cloud		
	Time [ms]	Com. [MB]	Wall time / CPU time [s]		
# Parties	*indep.*	*indep.*	32	64	128
Setup	262.58	42.93	0.85	1.68	3.38
In	6.22	0.52	0.01	0.01	0.02
Eval	0.00	0.00	8.1(0.4)	23.4(0.8)	62.1(1.6)
Out	3.34	0.79	0.01	0.02	0.02

Out The providers engage in the ColKeySwitch protocol (excluding the receiver) with target ciphertext ct_{out}, input key s and output key 0. They send their decryption shares to the cloud, that can then aggregate them to produce an output ciphertext encrypting x_r under the receiver secret-key share (as he did not participate in the ColKeySwitch protocol).

We set the vector size to $d = 2^{13}$ and a p of 32 bits. The communication is the sum of upstream and downstream. We used the same parameters for the baseline as for the previous experiment. The MHE system was instantiated with $n = 2^{13}$, 218-bit q and 32-bit plaintext modulus t. Table III shows a comparison with the baseline system. The MHE-based system matches the response time of the baseline in the two-party setting, and it is more efficient in terms of network usage. The generation of rotation keys accounts for approximately 75% of the setup cost and is the main overhead of the protocol. However, they enable the unpacking of the receiver query filter from a single ciphertext during the evaluation phase. When considering the case with 8 parties, the MHE setup cost is already $5.2\times$ faster and requires $2.4\times$ less communication than the baseline's offline phase, yet is a one-time setup.

We report on the per-phase cost for the MHE-based solution in Table IV, for larger number of parties. Again, the parallelization of the circuit computation over multiple core yields a very low response-time, regardless of the algorithmic complexity of homomorphic operations. Our choice for t enables 32.8 kilobytes to be packed into each ciphertext. For the 8-party setting, this yields a throughput of 117.1 kB/s and an expansion of only $40\times$ the communication cost of an insecure plaintext system. We ran the same experiment for $N = 8000$ and the response time was 61.7 seconds.

D. Multiplication Triples Generation

We evaluated how the MHE-based system can be used to produce multiplication triples in a peer-to-peer setting. This use-case is of particular interest, as triple generation is the bottleneck cost for secret-sharing-based MPC approaches. We consider the following functionality:

Let $\mathbf{x_i} = (\mathbf{a_i}, \mathbf{b_i}) \in \mathbb{Z}_p^{n\times 2}$ be the input of party P_i, where n is the number of generated triples and p is a prime. The *joint ideal functionality* is $F_{\mathcal{P}}(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_i}) = (\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_N})$ such that $\mathbf{c} = \sum_{i=1}^{N} \mathbf{c_i} = (\sum_{i=1}^{N} \mathbf{a_i}) \odot (\sum_{i=1}^{N} \mathbf{b_i}) = \mathbf{a} \odot \mathbf{b}$, where $\odot$ denotes the coefficient-wise product. The MHE–MPC protocol is instantiated as follows:

Setup The parties run the RelinKeyGen protocol to generate a relinearization key rlk.

In The parties use the Share2Enc protocol to produce encryptions of $\mathbf{a}$ and $\mathbf{b}$. The aggregation of the shares is done along the tree so that this phase ends with the root having access to $\mathsf{ct}_a = \mathsf{Enc}(\mathbf{a})$ and $\mathsf{ct}_b = \mathsf{Enc}(\mathbf{b})$.

Eval The root computes $\mathsf{ct}_c = \mathsf{Relin}(\mathsf{Mult}(\mathsf{ct}_a, \mathsf{ct}_b), \mathsf{rlk})$ and sends it down the tree.

Out The parties use the Enc2Share protocol to produce an additive sharing of c from $\mathsf{Enc}(\mathbf{c})$. The aggregation is done along the tree with the root being P_1.

Figure 1 shows a comparison with two currently prevalent techniques: oblivious-transfer-based and plain homomorphic-encryption-based. We implemented both the plain HE and our MHE-based approaches with our library [43]. For the OT-based one, we used the *Multi-Protocol SPDZ* library [57] that provides an implementation of the SPDZ2K [59] in the semi-honest setting. Both HE-based generators where parameterized to produce triples in $\mathbb{Z}_p$ for p a 32-bit prime and the OT-based generator to produce triples in $\mathbb{Z}_{2^{32}}$[1]. For the HE-based generators, we used polynomial degree $n = 2^{13}$, coefficient modulus size of 218 bits, and plaintext modulus $t = p$. Thus, after the setup phase, the parties can loop over the In-Eval-Out steps to produce a stream of triples in batches of 2^{13}. To report on the steady regime of the systems, we do not include the setup phase costs in the measurements in Figure 1.

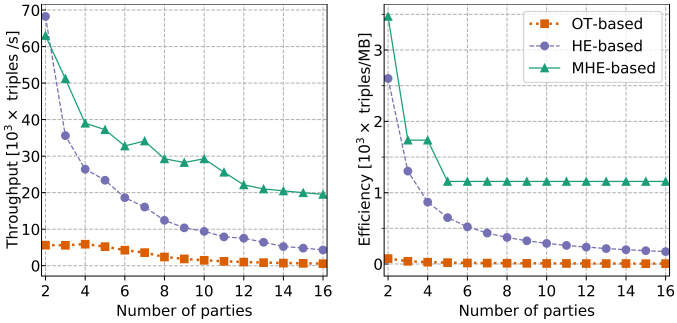

Fig. 1: Beaver triple generation in terms of number of triples per second (throughput, left) and triples per megabyte of communication (efficiency, right).

E. Discussion

The presented experiments confirm that the asymptotic cost-reduction of the MHE-based solution yields a significant advantage over secret-sharing-based systems, both in terms of response time and communication cost. Additionally, we discover that this advantage appears for surprisingly small circuit and low number of parties. We observe that, in general, MHE-based solutions have a setup cost higher than the secret-sharing-based ones due to the need for evaluation keys (e.g., relinearization, rotation). Hence, in scenarios where a single evaluation of a circuit with few multiplication gates and small number of input parties, the MHE-based solution would not be efficient. However, we observe that, as this setup is a one-time cost, it is quickly amortized when considering circuits with a few thousands multiplication gates and more than two parties. Evaluating where the decision-boundary stands regarding which system to use for smaller use-cases is a crucial question to be investigated as a future work.

VIII. Conclusions

In this work, we have proposed a novel solution to multiparty computation problems, based on *multiparty homomorphic encryption* (MHE). To support this approach, we have introduced and implemented a multiparty version of

[1]At the time of writing, the MP-SPDZ library does not implement a standalone benchmark for OT-based generation of triples in a prime field.

the BFV encryption scheme. We have analyzed the features, security, and performance of the proposed solution, and have confirmed that re-balancing the cost of MPC toward computation time brings crucial advantages: First, our MHE-based solution substantially reduces the latency of MPC tasks, mainly because the computation costs can be conveniently amortized by parallelization over multiple cores or nodes, whereas this cannot be the case for network costs in current approaches. In fact, our implementation in three example use-cases shows a net improvement ranging between 1 and 2 orders of magnitude in both response time and communication complexity compared to the current approaches. Second, MHE enables new computation models for MPC, that go beyond peer-to-peer, including outsourced cloud-assisted models that can reduce the communication cost per party to be constant in the number of parties. Moreover, the cloud-assisted model enables new opportunities for MPC-as-a-service, which we view as a promising application and adoption driver for both the HE and the MPC research.

Future research directions comprise using Shamir secret-sharing [11] to enable flexible threshold access structures, and extending the MHE-based approach to malicious settings.

Our proposed MHE-based solution brings MPC at a new level of scalability and flexibility, by supporting secure computation among thousands of parties in a wide spectrum of outsourced and distributed scenarios.

ACKNOWLEDGMENTS

The authors would like to thank Jean-Philippe Bossuat for the implementation of the distributed cryptosystem, and Henry Corrigan-Gibbs for the valuable reviews and comments. This work was supported in part by the grant #2017-201 of the Strategic Focal Area "Personalized Health and Related Technologies (PHRT)" of the ETH Domain.

REFERENCES

[1] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter *et al.*, "Secure multiparty computation goes live," in *International Conference on Financial Cryptography and Data Security*. Springer, 2009, pp. 325–343.

[2] D. Bogdanov, R. Talviste, and J. Willemson, "Deploying secure multiparty computation for financial data analysis," in *International Conference on Financial Cryptography and Data Security*. Springer, 2012, pp. 57–64.

[3] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk, and R. Talviste, "Students and taxes: a privacy-preserving study using secure computation," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 3, pp. 117–135, 2016.

[4] H. Cho, D. J. Wu, and B. Berger, "Secure genome-wide association analysis using multiparty computation," *Nature biotechnology*, vol. 36, no. 6, p. 547, 2018.

[5] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano, "Deriving genomic diagnoses without revealing patient genomes," *Science*, vol. 357, no. 6352, pp. 692–695, 2017.

[6] J. L. Raisaro, J. Troncoso-Pastoriza, M. Misbach, J. S. Sousa, S. Pradervand, E. Missiaglia, O. Michielin, B. Ford, and J.-P. Hubaux, "MedCo: Enabling secure and privacy-preserving exploration of distributed clinical and genomic data," *IEEE/ACM transactions on computational biology and bioinformatics*, 2018.

[7] D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht, "How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 227–234.

[8] J. Kroll, E. Felten, and D. Boneh, "Secure protocols for accountable warrant execution," *See http://www. cs. princeton. edu/felten/warrant-paper. pdf*, 2014.

[9] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart, and R. N. Wright, "From Keys to Databases—Real-World Applications of Secure Multi-Party Computation," *The Computer Journal*, vol. 61, no. 12, pp. 1749–1771, 2018.

[10] "Transforming trust by changing how cryptographic keys are secured," Online: https://www.unboundtech.com/, 2020.

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

[12] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 334–348.

[13] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *2017 38th IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[14] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 259–282.

[15] A. B. Alexandru, M. Morari, and G. J. Pappas, "Cloud-based MPC with encrypted data," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 5014–5019.

[16] S. Englehardt, "Next steps in privacy-preserving telemetry with prio," https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/, 2019.

[17] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption." *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[18] 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.

[19] G. Asharov, A. Jain, and D. Wichs, "Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE," Cryptology ePrint Archive, Report 2011/613, 2011, https://eprint.iacr.org/2011/613.

[20] 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.

[21] 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.

[22] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "SoK: General purpose compilers for secure multi-party computation," in *Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1220–1270.

[23] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 192–206.

[24] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 643–662.

[25] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 1–18.

[26] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: making SPDZ great again," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 158–189.

[27] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Annual International Cryptology Conference*. Springer, 1991, pp. 420–432.

[28] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified vss and fast-track multiparty computations with applications to threshold cryptography," in *podc*, vol. 98. Citeseer, 1998, pp. 101–111.

[29] M. Keller, E. Orsini, and P. Scholl, "Mascot: faster malicious arithmetic secure computation with oblivious transfer," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 830–842.

[30] Y. G. Desmedt, "Threshold cryptography," *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 449–458, 1994.

[31] M. Franklin and S. Haber, "Joint encryption and message-efficient secure computation," *Journal of Cryptology*, vol. 9, no. 4, pp. 217–232, 1996.

[32] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2001, pp. 280–300.

[33] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 724–738.

[34] D. Froelicher, J. R. Troncoso-Pastoriza, J. S. Sousa, and J. Hubaux, "Drynx: Decentralized, secure, verifiable system for statistical queries andmachine learning on distributed datasets," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2020.

[35] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford University Stanford, 2009, vol. 20, no. 09.

[36] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "Cloud-Assisted Multiparty Computation from Fully Homomorphic Encryption." *IACR Cryptology ePrint Archive*, vol. 2011, p. 663, 2011.

[37] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, p. 34, 2009.

[38] 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., November 2018.

[39] "Microsoft SEAL (release 3.2)," https://github.com/Microsoft/SEAL, Feb. 2019, microsoft Research, Redmond, WA.

[40] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption library," August 2016, https://tfhe.github.io/tfhe/.

[41] C. Aguilar-Melchor, J. Barrier, S. Guelton, A. Guinet, M.-O. Killijian, and T. Lepoint, "NFLlib: NTT-based fast lattice library," in *Cryptographers' Track at the RSA Conference*. Springer, 2016, pp. 341–356.

[42] Y. Polyakov, K. Rohloff, and G. W. Ryan, "PALISADE lattice cryptography library," https://git.njit.edu/palisade/PALISADE, 2018.

[43] "Lattigo 1.3.1," Online: http://github.com/ldsec/lattigo, Feb. 2020, EPFL-LDS.

[44] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.

[45] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. ACM, 2012, pp. 1219–1234.

[46] H. Chen, I. Chillotti, and Y. Song, "Multi-Key Homomophic Encryption from TFHE."

[47] L. Chen, Z. Zhang, and X. Wang, "Batched multi-hop multi-key FHE from ring-LWE with compact ciphertext extension," in *Theory of Cryptography Conference*. Springer, 2017, pp. 597–627.

[48] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 395–412.

[49] R. Canetti and M. Fischlin, "Universally composable commitments," in *Annual International Cryptology Conference*. Springer, 2001, pp. 19–40.

[50] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, p. 13, 2014.

[51] D. Rotaru and N. P. Smart, "Actively secure setup for spdz," *IACR Cryptology ePrint Archive*, 2019.

[52] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca, "A full RNS variant of FV like somewhat homomorphic encryption schemes," in *International Conference on Selected Areas in Cryptography*. Springer, 2016, pp. 423–442.

[53] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 360–384.

[54] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*. Springer, 2017, pp. 277–346.

[55] G. Seiler, "Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography." *IACR Cryptology ePrint Archive*, vol. 2018, p. 39, 2018.

[56] K. Han and D. Ki, "Better bootstrapping for approximate homomorphic encryption," *IACR Cryptology ePrint Archive*, 2019.

[57] "MP-SPDZ," Online: https://github.com/data61/MP-SPDZ/, Jan. 2020.

[58] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, "Practical multi-party private set intersection from symmetric-key techniques." in *ACM Conference on Computer and Communications Security*, 2017, pp. 1257–1272.

[59] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "SPD$\mathbb{Z}_{2^k}$: Efficient mpc mod 2^k for dishonest majority," in *Annual International Cryptology Conference*. Springer, 2018, pp. 769–798.

APPENDIX A
SYMBOL TABLE

Tables V and VI respectively show the notation and a brief description for the parameters and symbols used in this work.

TABLE V: BFV Parameters

Parameter	Description
q	Coefficient modulus in ciphertext space
t	Coefficient modulus in plaintext space
n	Polynomial degree
w	Intermediary relinearization base
σ	Error standard deviation
B	Error-norm upper bound

TABLE VI: BFV Symbols

Symbol	Description
Δ	Quotient of the integer division of q by t
$[q]_t$	Remainder of the integer division of q by t
R_q	Ciphertext space ring $\mathbb{Z}_q[X]/(X^n+1)$
R_t	Plaintext space ring $\mathbb{Z}_t[X]/(X^n+1)$
R_3	Key space ring $\mathbb{Z}_3[X]/(X^n+1)$
l	Length $\lceil \log_w(q) \rceil$ of the base-w decomposition of $a \in R_q$
$\mathbf{w}$	$(w^0, w^1, ..., w^l)^T$ base-w reconstruction vector
χ	Error distribution, discrete normal $\mathcal{N}(0, \sigma^2)$ over $[-B, B]$

APPENDIX B
LOCAL OPERATIONS BENCHMARKS

We executed our benchmarks on an Intel i5 processor at 3.1 GHz, with 16 GB of memory, running Go 1.13.4 (darwin/amd64). The open-source library includes Go benchmarks for reproducibility; as a design decision, we considered that the most efficient parallelization happens at the application layer, so to give freedom to the application developer, all the primitives in the library run in one thread.

TABLE VII: Benchmarking parameter sets

Set	n	$\log_2 q$	$\log_2 w$
P8192	8192	218	60
P16384	16384	438	110
P32768	32756	881	180

TABLE VIII: Centralized BFV operation performance (ms)

		P8192	P16384	P32768
Encryption	Encrypt	4.91	18.16	69.42
	Decrypt	1.93	8.06	34.58
Evaluation	Add	0.07	0.29	1.26
	Multiply	15.15	71.59	390.53
	Relin	5.64	30.03	157.31
	Rotate	5.77	31.15	154.67

Table VII shows the parameter sets used in our benchmarks, all guaranteeing a security level of at least 128 bits [38]. These parameters are application dependent, but their choice represents typical sizes of q that correspond to different *homomorphic capacities* (e.g., set `P8192` enables approximately twice the noise level as `P4096`).

Table VIII shows the timings for the operations of the centralized scheme that are used in our MHE-based solution.

TABLE IX: Distributed BFV local operations performance (ms)

		P8192	P16384	P32768
EncKeyGen	Gen	1.80	5.72	19.65
	Agg	0.05	0.18	0.68
	Out	0.11	0.37	1.45
RelinKeyGen	Gen	21.95	70.46	319.61
	Agg	0.55	2.27	11.34
	Out	0.62	2.78	13.68
RotKeyGen	Gen	6.07	20.45	90.62
	Agg	0.14	0.54	2.92
	Out	0.27	1.20	6.09
ColBootstrap	Gen	5.97	21.20	82.43
	Agg	0.07	0.28	1.17
	Out	2.13	8.60	36.43
ColKeySwitch	Gen	3.73	11.69	41.16
	Agg	0.03	0.14	0.55
	Out	0.03	0.14	0.60
PubColKeySwitch	Gen	7.33	25.39	97.59
	Agg	0.07	0.27	1.14
	Out	0.05	0.21	0.93

TABLE X: Cryptographic objects size (MB)

	P8192	P16384	P32768
Ciphertext	0.39	1.57	6.29
Public key	0.39	1.57	6.29
Relin. key	1.57	6.29	31.46
Rot. key	1.57	6.29	31.46
EncKeyGen-share	0.26	1.05	3.93
RelinKeyGen-share	3.15	12.58	62.91
RotKeyGen-share	0.79	3.15	15.73
ColBootstrap-share	0.39	1.57	6.29
ColKeySwitch-share	0.20	0.79	3.15
PubColKeySwitch-share	0.39	1.57	6.29

For each protocol of the distributed scheme, Table IX shows the following values:

Gen corresponds to the cost for a given party to generate its own public share in the protocol. Hence, this cost is independent of the number of parties. For the RelinKeyGen protocol, the party's share consists in multiple round shares and this cost is aggregated across all rounds.

Agg corresponds to the cost of combining two shares in the protocol. The way these costs are reflected on the system and the involved network traffic depends on the chosen system model. Table X shows the share size for each protocol, from which the network cost of a given system model can be easily derived: In peer-to-peer settings, a party having N_c children in the tree will receive and aggregate N_c shares, aggregate its own share if it is in $\mathcal{P}$, and send 1 share to its parent. In the cloud-assisted model, the cloud takes care of the aggregation for all the N parties, so parties do not have inbound traffic and only need to send a single share.

Out corresponds to the cost of computing the final output of the protocol (i.e., obtaining the collective secret key, the key-switched ciphertext, etc.) when provided with the aggregate of all the shares (i.e., after the *Aggr* phase completed). In a tree-like topology, this task is normally performed by the root, hence, by the cloud in the helper-cloud model.

APPENDIX C
NOISE ANALYSIS

In BFV-like lattice-based cryptosystems, the correct decryption of a given ciphertext is guaranteed only if the magnitude of its error term, called *noise*, is kept below a certain threshold fraction of q (Eq. 2). Whereas each ciphertext contains only a very small initial amount of noise that is introduced by the BFV.Encrypt procedure, the ciphertext noise grows with homomorphic operations. Hence, it is crucial to ensure that evaluating a given homomorphic circuit will not result in the noise growing out of bounds, which is usually done by computing the corresponding worst-case noise magnitude and by choosing q to accommodate this bound.

We analyze the effect that distributing the BFV cryptosystem has on the ciphertext noise. As distribution affects only the magnitude of the scheme's secrets (key and noise), the original cryptosystem analysis [17] directly applies, though with a larger worst-case error norm that we express as a function of the number of parties N in the following.

A. Ideal Secret Key

The magnitude of the secret key plays a major role in the noise growth of BFV-like cryptosystems. For the distributed scheme, it is crucial that the magnitude of the ideal key, which would be output by the S.Combine operation (see Def. 2), remains small even for a large number of parties. As a result of the secret-key generation procedure, where each additive share s_i is sampled from R_3 (see Section IV-A), we know that $\|s\| \leq N$. This linear dependency is the main advantage over multi-key schemes (approach (b) in Section II), for which the flexibility brought by on-the-fly keys requires a much more complex Combine operation for which the dependency with the input keys is multiplicative in most cases.

As a result of the EncKeyGen protocol, the collective public key noise is $e_{\mathsf{cpk}} = \sum_{i=1}^{N} e_i$ (see Eq. (4)), which implies that $\|e_{\mathsf{cpk}}\| \leq NB$, where B is the worst-case norm for an error term sampled from the RLWE error distribution χ.

B. Fresh Encryption

Let $\mathsf{ct}= (c_0, c_1)$ be a fresh encryption of a message m under a collective public key. The first step of the decryption (Eq. (1)) under the *ideal* secret key outputs $c_0 + sc_1 = \Delta m + e_{fresh}$, where

$$\|e_{fresh}\| \leq B(2nN + 1). \tag{9}$$

Thus, for a key generated by the EncKeyGen protocol, the worst-case fresh ciphertext noise is linear in the number N of parties.

C. Collective Key-Switching

Let $\mathsf{ct} = (c_0, c_1)$ be an encryption of m under the collective secret key s, and $\mathsf{ct}' = (c'_0, c_1)$ be the output of the ColKeySwitch protocol on ct with target key s'. Then, $c'_0 + s'c_1 = m + e_{fresh} + e_{\mathsf{CKS}}$ with

$$\|e_{\mathsf{CKS}}\| \leq B_{smg}N, \tag{10}$$

where B_{smg} is the bound of the smudging distribution. We observe that the additional noise does not depend on the destination key s'.

D. Public Collective Key-Switching

Let $\mathsf{ct} = (c_0, c_1)$ be an encryption of m under the collective secret key s, and $\mathsf{ct}' = (c'_0, c'_1)$ be the output of the PubColKeySwitch protocol on ct and target public key $\mathsf{pk}' = (p'_0, p'_1)$, such that $p'_0 = -sp'_1 + e_{\mathsf{pk}'}$. Then, $c'_0 + s'c'_1 = m + e_{fresh} + e_{\mathsf{PCKS}}$ with

$$\|e_{\mathsf{PCKS}}\| \leq N(nB_{\mathsf{pk}'} + n\|s'\|B + B_{\text{smg}}), \tag{11}$$

where $\|e_{\mathsf{pk}'}\| \leq B_{\mathsf{pk}'}$, and B_{smg} is the bound on the smudging noise. Note that in this case, the smudging noise should dominate this term.

E. Arithmetic Operations

Let ct_1 and ct_2 be two ciphertexts with worst-case noise norm B_1 and B_2 and let $\mathsf{ct}_{\text{add}} = \mathsf{BFV.Add}(\mathsf{ct}_1, \mathsf{ct}_2)$. The noise e_{add} of ct_{add} is such that

$$\|e_{\text{add}}\| \leq B_1 + B_2,$$

whose overhead is thus independent of N.

Let $\mathsf{ct}_{\text{mul}} = \mathsf{BFV.Multiply}(\mathsf{ct}_1, \mathsf{ct}_2)$ and e_{mul} be the error term of ct_{mul}. Then, by relying on the upper bound given by Lemma 2 in [17], we have

$$\|e_{\text{mul}}\| < nt(B_1 + B_2)(nN + 1) + 2t^2n^2(N + 1)^2.$$

Hence, as for the original scheme, the noise after a multiplication is expanded by a factor $\|s\|$, which brings a linear dependence in N (it also adds a quadratic, yet less significant, term in N). Note that this dependency is due solely to the magnitude of the secret key (i.e., no fresh noise is added by homomorphic operations).

F. Relinearization (Type I)

We analyze the noise resulting from a Type I relinearization [17] performed with a key generated by the RelinKeyGen protocol (Protocol 3). For a ciphertext $\mathsf{ct} = (c_0, c_1, c_2)$, we can write its 2-component equivalent as $\mathsf{ct}_{\mathsf{relin}} = (c'_0, c'_1)$, where $c'_0 + sc'_1 = m + e_{fresh} + e_{relin}$ with

$$\|e_{relin}\| \leq \frac{wn}{2}(l + 1)BN(2nN + 2). \tag{12}$$

Therefore, the noise introduced by the relinearization increases by a factor that is quadratic in N; this factor stems from the noise introduced in the relinearization key (see Eq. (13)): The RelinKeyGen protocol outputs relinearization keys that have noise in their $\mathbf{r}_0$ component only, as opposed to the approach by Asharov et al. [20], for which the noise added in the $\mathbf{r}_1$ component introduces noise in c'_1. As the latter noise term is multiplied by the secret key at decryption, their approach results in a significantly larger e_{relin} than ours. Analogously to the original scheme, the noise introduced by the relinearization is independent of the noise already present in the input ciphertext.

G. Discussion

The distributed BFV scheme keeps the noise within manageable bounds, even for a large number of parties. The noise overhead is predominantly linear in N, with only the relinearization-incurred noise being quadratic (but is still small w.r.t. the multiplication-incurred overhead). Therefore, for a fixed size of the modulus q, the multiparty version of BFV can accommodate much larger number of parties than the multi-key counterparts. In Section VII-B, we show that the framework can be used with hundreds of parties in an efficient way.

H. Derivation of Noise-Growth Equations

This appendix details the derivations of the noise growth equations presented in the previous sections. The infinity norm of a polynomial p (i.e., its largest coefficient in absolute value) is denoted $\|p\|$ ($\|p\| \leq q/2$ for $p \in R_q$). We also recall that, since the polynomial modulus in R_q is a degree-n power of 2 cyclotomic, we have $\|ab\| \leq n\|a\|\|b\|$. We consider an instantiation of our distributed BFV scheme with N parties.

1) Derivation of Eq. (9)*:* From the ideal decryption of a fresh encryption of m under the collective public key $\mathsf{cpk} = (p_0, p_1)$:

$$\begin{aligned} c_0 + sc_1 &= \Delta m + p_0u + e_0 + sp_1u + se_1 \\ &= \Delta m - ue_{\mathsf{cpk}} + e_0 + se_1, \end{aligned}$$

where we substituted the expression of BFV.Encrypt. As $\|u\| = 1$ and $\|e_i\| \leq B$ for $i = 0, 1$, Eq. (9) follows.

2) Derivation of Eq. (10)*:* From the decryption expression of ct',

$$\begin{aligned} c'_0 + s'c_1 &= c_0 + \sum_j ((-s'_j + s_j)c_1 + e_{\mathsf{CKS},j}) + s'c_1 \\ &= c_0 + sc_1 + \sum_j e_{\mathsf{CKS},j} \\ &= \Delta m + e_{fresh} + \sum_j e_{\mathsf{CKS},j}. \end{aligned}$$

As $e_{\mathsf{CKS},j} \leq B_{smg}$, Eq. (10) follows.

3) Derivation of Eq. (11)*:* From the decryption expression of ct',

$$\begin{aligned} c'_0 + s'c'_1 &= c_0 + \sum_j (s_jc_1 + u_jp'_0 + e_{0,j}) + s' \sum_j (u_jp'_1 + e_{1,j}) \\ &= c_0 + sc_1 + up'_0 + s'up'_1 + \sum_j e_{0,j} + se_{1,j} \\ &= \Delta m + e_{fresh} + \sum_j u_je_{\mathsf{pk}'} + e_{0,j} + s'e_{1,j}, \end{aligned}$$

and Eq. (11) follows.

4) Derivation of Eq. (12)*:* Let $\mathsf{rlk} = (\mathbf{r}_0, \mathbf{r}_1)$ be the collectively generated relinearization key. It has the same form as in the original scheme, except for the magnitude of its $\mathbf{e}_{rlk}$ components:

$$\|e^{(i)}_{\mathsf{rlk}}\| < ((n\|s\| + 2)N + nN^2)B\,. \tag{13}$$

Thus, the same analysis as in the original scheme applies. Let $\mathbf{c}_2$ be the base-w decomposition vector of c_2, such that

Protocol 7: NoisyRelinKeyGen

Public Input: $\mathsf{cpk} = (p_0, p_1)$, $\mathbf{w}$
Private Input for P_i: $\mathsf{sk} = s_i$
Output:$\mathsf{rlk} = (\mathbf{r}_0, \mathbf{r}_1)$

Each party P_i:

1) samples $\mathbf{e}_{0,i}$, $\mathbf{e}_{1,i} \leftarrow \chi^l$, $\mathbf{u}_i \leftarrow R_3^l$ and broadcasts
$$\mathbf{h}_{0,i} = p_0\mathbf{u}_i + s_i\mathbf{w} + \mathbf{e}_{0,i} \quad \text{and} \quad \mathbf{h}_{1,i} = p_1\mathbf{u}_i + \mathbf{e}_{1,i}$$
2) waits for $\mathbf{h}_{0,j}$, $\mathbf{h}_{1,j}$ from all P_j, computes
$$\mathbf{h}_0 = \sum_{P_j \in \mathcal{P}} \mathbf{h}_{0,j} \quad \text{and} \quad \mathbf{h}_1 = \sum_{P_j \in \mathcal{P}} \mathbf{h}_{1,j}$$
samples $\mathbf{e}_{2,i}, \mathbf{e}_{3,i} \leftarrow \chi^l$, $\mathbf{v}_i \leftarrow R_3^l$ and broadcasts
$$\mathbf{h}'_{0,i} = s_i\mathbf{h}_0 + p_0\mathbf{v}_i + \mathbf{e}_{2,i} \quad \text{and} \quad \mathbf{h}'_{1,i} = s_i\mathbf{h}_{1,i} + p_1\mathbf{v}_i + \mathbf{e}_{3,i}$$
3) waits for $\mathbf{h}'_{0,j}$, $\mathbf{h}'_{1,j}$ from all P_j, outputs $\mathsf{rlk} = (\sum_j \mathbf{h}'_{0,j}\ ,\ \sum_j \mathbf{h}'_{1,j})$

the inner product $\mathbf{c}_2 \cdot \mathbf{w}$ equals c_2. We have

$$\begin{aligned} c'_0 + sc'_1 &= c_0 + \mathbf{c}_2 \cdot \mathbf{r}_0 + s(c_1 + \mathbf{c}_2 \cdot \mathbf{r}_1) \\ &= c_0 + sc_1 + \mathbf{c}_2 \cdot (\mathbf{r}_0 + s\mathbf{r}_1)) \\ &= c_0 + sc_1 + \mathbf{c}_2 \cdot s^2\mathbf{w} + \mathbf{c}_2 \cdot \mathbf{e}_{\mathsf{rlk}} \\ &= \Delta m + e_{\mathit{fresh}} + \mathbf{c}_2 \cdot \mathbf{e}_{\mathsf{rlk}}, \end{aligned}$$

where the upper bound for the inner product term is derived from the expression for $\mathbf{e}_{\mathsf{rlk}}$ in Eq. (13), by observing that each of the $l+1$ elements in $\mathbf{c}_2$ have coefficients in $[-\frac{w}{2}, \frac{w}{2}]$.

Appendix D
Comparison between RelinKeyGen and previous work

We show here how to adapt the classic method of [20] to our scheme, resulting in the Protocol 7.

After the execution of NoisyRelinKeyGen, each party holds a copy of the public rlk corresponding to the collective secret key s. This enables them and, potentially, other external entities, to use the BFV.Relinearize algorithm in the trust domain defined by s. The resulting key is of the form

$$\begin{aligned} &\mathsf{rlk} = (\mathbf{r}_0, \mathbf{r}_1) \\ &= (p_0(s\mathbf{u}+\mathbf{v}) + s^2\mathbf{w} + s\mathbf{e}_0 + \mathbf{e}_2\ ,\ p_1(s\mathbf{u}+\mathbf{v}) + s\mathbf{e}_1 + \mathbf{e}_3) \\ &= (-s\mathbf{b} + s^2\mathbf{w} - (s\mathbf{u}+\mathbf{v})e_{\mathsf{cpk}} + s\mathbf{e}_0 + \mathbf{e}_2\ ,\ \mathbf{b} + s\mathbf{e}_1 + \mathbf{e}_3), \end{aligned}$$

where $\mathbf{b} = p_1(s\mathbf{u} + \mathbf{v})$. Hence, with respect to the key produced by BFV.RelinKeyGen, rlk holds a significantly increased noise, not only in $\mathbf{r}_0$, but also in $\mathbf{r}_1$, which is not *noisy* when generated in a centralized way.

By producing a noise-free $\mathbf{r}_1$ term and a less noisy $\mathbf{r}_0$ term, our solution significantly improves on the simple method. Although it requires one more round of communication, our protocol has the same volume of network traffic, and is computationally less expensive.

Appendix E
Collective Relinearization-Key Generation Security

The private input for each party P_i in the RelinKeyGen protocol is the tuple $x_i = (s_i, u_i, e_{0,i}, e_{1,i}, e_{2,i}, e_{3,i})$: its ideal secret-key share s_i, its *ephemeral* secret u_i, and the error terms added in each round. The output for each party is $f(x_1, \ldots, x_N) = (\mathbf{r}_0, \mathbf{r}_1)$, the generated relinearization key defined in Eq. (5). Throughout the protocol execution, the parties compute the public values $\mathbf{h}$, $\mathbf{h}' = (\mathbf{h}'_0, \mathbf{h}'_1)$ and $\mathbf{h}''$. These values can be simulated, with the constraints $\mathbf{r}_0 = \mathbf{h}'_0 + \mathbf{h}''$ and $\mathbf{r}_1 = \mathbf{h}'_1$. For every round, the parties' view in the protocol comprises additive sharings of these values, which S can simulate as

$$\widetilde{\mathbf{h}}_i = \begin{cases} [-u_i\mathbf{a} + s_i\mathbf{w} + \mathbf{e}_{0,i}]_q & \text{if } P_i \in \mathcal{A} \\ \leftarrow R_q^l & \text{if } P_i \notin \mathcal{A} \end{cases},$$

$$\widetilde{\mathbf{h}}'_i = \begin{cases} ([s_i\widetilde{\mathbf{h}} + \mathbf{e}_{1,i}]_q\ ,\ [s_i\mathbf{a} + \mathbf{e}_{2,i}]_q) & \text{if } P_i \in \mathcal{A} \\ \leftarrow R_q^{2\times l} & \text{if } P_i \in \mathcal{H} \\ (\leftarrow R_q^l\ ,\ [\mathbf{r}_1 - \sum_{P_j \in \mathcal{A}\cup\mathcal{H}} \widetilde{\mathbf{h}}'_{1,j}]_q) & \text{if } P_i = P_h \end{cases},$$

$$\widetilde{\mathbf{h}}''_i = \begin{cases} [(u_i - s_i)\widetilde{\mathbf{h}}'_1 + \mathbf{e}_{3,i}]_q & \text{if } P_i \in \mathcal{A} \\ \leftarrow R_q^l & \text{if } P_i \in \mathcal{H} \\ [\mathbf{r}_0 - \widetilde{\mathbf{h}}'_0 - \sum_{P_j \in \mathcal{A}\cup\mathcal{H}} \widetilde{\mathbf{h}}''_j]_q & \text{if } P_i = P_h \end{cases}.$$

The indistinguishability argument for the shares is similar to the one of Section VI-A. To prove indistinguishability for their composition, we consider the *combined* view of the adversary,

$$\begin{pmatrix} \mathbf{h} \\ \mathbf{h}'_0 \\ \mathbf{h}'_1 \\ \mathbf{h}'' \end{pmatrix} = \begin{pmatrix} -u\mathbf{a} + s\mathbf{w} + \mathbf{e}_0 \\ -su\mathbf{a} + s^2\mathbf{w}, +s\mathbf{e}_0 + e_1 \\ s\mathbf{a} + \mathbf{e}_2 \\ (u-s)s\mathbf{a} + (u-s)\mathbf{e}_2 + \mathbf{e}_3 \end{pmatrix}.$$

Lemma 1 extracts the sought property for the transcript (note that $\mathbf{h}'_0 - \mathbf{h}'' \approx s^2\mathbf{a} + s^2\mathbf{w}$).

Lemma 1. *Let $a \leftarrow R_q$, $s_1, s_2 \leftarrow R_3$ be two RLWE secrets, and $e_1, e_2, e_3, e_4 \leftarrow \chi$ be four RLWE error terms. The distribution*

$$(a, \quad s_1a + e_1, \quad s_2a + e_2, \quad s_2s_1a + e_3, \quad s_1^2a + e_4) \quad (14)$$

is computationally indistinguishable from the uniform distribution over R_q^5 for any adversary not knowing the secrets and error terms.

We do not provide the proof for Lemma 1, as the assumption that we can *chain* RLWE sample generators is already required by all mainstream RLWE-based cryptosystems. For an intuition, note that the first two elements of Eq. (14) correspond to a public key with secret key $s = s_1$, and the next two (together) can be seen as an encryption of 0 under this key, with randomness $u = s_2$.