Multiparty Homomorphic Encryption from Ring-Learning-With-Errors

Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat and Jean-Pierre Hubaux
Laboratory for Data Security, EPFL
firstname.lastname@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 our solution, we introduce a multiparty version of the Brakerski-Fan-Vercauteren homomorphic cryptosystem, implement it in an open-source library, and evaluate its performance. We show the main advantages that MHE-based MPC solutions have over current approaches: Their public transcripts and non-interactive circuit-evaluation capabilities enable a broad variety of computing paradigms, 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 solutions 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*. When evaluated on the first circuit with eight parties, our approach is 8.6 times faster and requires 39.3 times less communication than the state-of-the-art. The input selection circuit over eight thousand parties requires only 1.31 MB of communication per party and completes in 61.7 seconds.**

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. The exact definition of security depends on how the adversary is modeled, but 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 a general functionality makes the study of MPC techniques highly relevant. 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 [13], [12], biomedical [30], [42] and law enforcement [10], [34] sectors, as well as for protecting digital assets [5].

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 [46] of the input data according to some linear secret-sharing scheme (LSSS), and interactive circuit evaluation. These approaches have two practical limitations: (i) standard protocols require many rounds of communication over private channels between the parties, which makes them inadequate for low-end devices and unreliable networks. (ii) 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 becomes a bottleneck for large circuits and/or number of parties. As a result, many MPC applications propose *outsourced* models where the actual computation is delegated to two non-colluding clouds [40], [30], [39], [18], [4]. The passive-adversary model is indeed realistic in concrete data-sharing scenarios [42] and is typically assumed for *honest-but-curious* cloud-services. However, the non-collusion assumptions might not be realistic in some contexts and, thus, the N-party to two-party reduction not always possible. Hence, we believe that an efficient and scalable N-party passive-adversary solution that is compatible with the cloud-service paradigm is needed.

Homomorphic encryption (HE) techniques have been known to reduce the communication complexity of MPC for more than two decades now [26], [20]. By exploiting the new possibilities offered by lattice-based cryptography, Asharov et al. proposed a generic HE-based MPC protocol based on the Learning-with-Errors (LWE) assumption [6]. Until today, however, no concrete MPC solution implements the generic HE-based MPC protocol, and the use of HE in MPC is mostly confined to the *offline* pre-computations (i.e., not 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.

We propose, implement, and evaluate a new instance of the HE-based MPC protocol in the passive-adversary with dishonest-majority model. We show that this protocol, based on multiparty homomorphic encryption (MHE), can support computations among several thousands of parties without relying on additional non-collusion assumptions. The per-party communication complexity is reduced to being linear in the circuit's inputs and outputs. Furthermore, we discuss the additional features of the MHE-based approaches, such as their compatibility and complementarity with existing LSSS-based MPC, and how they effectively remove the need for private party-to-party communication channels. Hence, in addition to the traditional peer-to-peer setting, our MHE-based solution can operate in a broad range of computing platforms, from cloud-outsourcing to smart-contract technologies.

Our solution is based on a multiparty version of the BFV homomorphic encryption-scheme [25] and it can be extended to other schemes of the same family, such as CKKS [17]. We use secret-sharing [46] to distribute a secret-key of this scheme among the parties, as proposed by Asharov et al. [6]. By bringing these techniques to ring-learning-with-errors (RLWE) cryptosystems and to practice we make the following contributions:

- We propose a novel multiparty version of the BFV homomorphic encryption scheme (Section IV). Beyond Asharov et al.'s construct, our scheme introduces novel single-round protocols to bridge with LSSS-based approaches and for bootstrapping a BFV ciphertext in multiparty settings.
- We discuss the features of the MHE-based approach and how it can both be integrated into existing MPC solutions, and be used directly as a standalone one that supports both the peer-to-peer and cloud-assisted models. (Section V).
- We implement our scheme in the Lattigo open-source library [1] and demonstrate its concrete efficiency for three example circuits (Section VI).

Through these contributions, this work bridges the gap between the existing theoretical work on MHE-based MPC and their application as powerful privacy-enhancing technologies.

II. Related Work

We classify N-party dishonest-majority MPC approaches in two categories: (a) Linear Secret-Sharing at Data-level (for short: LSSS-based), which is predominantly implemented in generic MPC solutions [29], [5], consists in applying *secret-sharing* [46] to input data. (b) *Multiparty encryption schemes (for short: MHE-based)*, use an homomorphic scheme to encrypt and exchange the input data, that can then be operated on non-interactively with encrypted arithmetic.

(a) LSSS-based. Most of the available generic MPC solutions, such as Sharemind [11] and SPDZ [22], [21], [32], apply secret-sharing to the input data. The evaluation of these arithmetic circuits is generally enabled by the homomorphism of the LSSS, or by interactive protocols [9] (when no such homomorphism is available), the most widely implemented protocol being Beaver's triple-based protocol [9]. The strength of approach (a) is to enable evaluation through only simple and efficient primitives in terms of which the circuit can be decomposed by code-to-protocol compilers, thus strengthening usability. However, this approach imposes two practical constraints: First, the interactive protocols at each multiplication gate require 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. Indeed, 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 [29] by using techniques such as oblivious transfer [31] or plain homomorphic encryption [22], [32] in an *offline phase*.

(b) MHE-based. In this category, the parties use an HE scheme to encrypt their inputs, and the computations are performed using the scheme's homomorphic operations. To preserve the inputs privacy, the scheme's secret key is securely distributed among the parties using some secret-sharing scheme. Hence, decryption requires the collaboration between the parties, according to the access structure of the used secret-sharing scheme. Such constructions are commonly referred to as *threshold* [24] or *distributed* encryption, 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 [26], later improved by Cramer et al. [20]. 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 less attractive than approach (a). Nevertheless, task-specific instances that use multiparty additive-homomorphic encryption have recently been successful in supporting use-cases in distributed machine learning [47], [27], highlighting the potential a generic and usable *fully homomorphic* encryption (FHE) [28] solution would have.

This is the idea behind the line of work by Asharov et al. [6] and López-Alt et al. [36]. These contributions propose various multiparty schemes in which the *ideal* secret-key is additively shared among the parties, and they analyze the theoretical MPC solution these schemes enable. Although of great interest, this line 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* [43] (LWE) -based homomorphic schemes, in terms of which these schemes were presented. Today, multiple ongoing efforts aim at standardizing homomorphic encryption [3] and at making its implementations available to a broader public. This new generation of schemes is mostly based on the *Ring Learning with Errors* (RLWE) problem [38] and has brought HE from being practical to being efficient. Therefore, we argue that MHE-based approaches are now mature and usable enough to support more than the offline phase of LSSS-based approaches.

Multi-key encryption schemes, as introduced by López-Alt et al. [37], are an important class of MHE schemes, where the ideal secret-key does not have to be defined before the computation. Instead, the parties provide their input data encrypted under their own secret key. 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 parties involved in a given computation are required to participate in the decryption sub-protocol. Unfortunately, when the number of involved parties is large, the performance cost of using these schemes can be prohibitive. We will consider a setting where all the parties have an input in the computation, which does not require the flexibility and cost of a multi-key approach. But the two techniques are compatible and could be used in conjunction to provide better trade-offs between setup, evaluation and decryption costs.

We will show that, in addition to being 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, outsourced scenarios) or the network model (e.g., low-end devices, publicly visible channels). Hence, we propose and build an open-source MPC solution that brings the theoretical work on multiparty schemes [6] to RLWE cryptography and to practice.

III. Background

We define our problem statement and recall the high-level building block of the MHE-based MPC solution.

A. Problem and Security Models

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*). Definition 1 formulates the *secure-multiparty-computation problem* we consider for the scope of this work.

Definition 1. Let $\mathcal{P} = \{P_1, P_2, \dots, P_N\}$ be a set of N parties respectively holding inputs $(x_1, x_2, \dots, x_N)$ (*input parties*) and let $\mathcal{R}$ be a party that can be either inside or outside of $\mathcal{P}$ (*receiver party*). Let $f(x_1, x_2, \dots, x_N) = y$ 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 $y = f(x_1, x_2, \dots, x_N)$, 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, the execution of π_f emulates an ideal setting where parties have collective access to an oracle 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 $\mathcal{P}$. That is, 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 inputs to f should have such role.

We assume that the parties in $\mathcal{P}$ have access to a uniformly random *Common Reference String* (CRS) [15], and they are connected through authenticated channels, which are not required to be confidential.

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

B. Multiparty Homomorphic Encryption

A *multiparty* encryption-scheme securely splits the secret key of an encryption scheme among N parties, according to a secret-sharing scheme. Hence, the operations in the original scheme that depend on the secret key are implemented in the multiparty scheme as special-purpose secure-multiparty protocols. Definition 2 formalizes this intuition for a HE scheme.

In addition to the security-related parameters, HE schemes require proper parameterization to support the evaluation of the target ideal functionality. We model this dependency by introducing an abstract *homomorphic capacity* parameter, which we denote κ. In practice, κ represents a choice of cryptographic parameters that determine the maximum width (i.e., the size of the plaintext space) and depth of the homomorphic circuits the scheme can evaluate before some kind of re-encryption is needed.

Definition 2. Let $E = (\mathsf{SecKeyGen}, \mathsf{PubKeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be an abstract homomorphic encryption scheme, λ a security parameter, κ a homomorphic-capacity parameter, and let $S = (\mathsf{Share}, \mathsf{Combine})$ be an N-party secret sharing scheme. The associated *multiparty homomorphic 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

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

where $\pi_{\mathsf{SecKeyGen}}$, $\pi_{\mathsf{PubKeyGen}}$, π_{Dec} are multiparty protocols that have the following *private ideal functionalities* for each party P_i:

Ideal secret-key generation:

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

Collective public-key generation:

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

Collective decryption:

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

As the $S.\mathsf{Combine}$ operation has to be embedded in $f_{\pi_{\mathsf{Dec}}}$, the secret-sharing scheme defines an access structure [46] that characterizes the access structure of E^*. For example, an 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 are required when using Shamir secret-sharing [46]. In this work, we focus on additive secret sharing of the key, which can be replaced by Shamir's in scenarios with less strict requirements.

In Section IV, we construct a multiparty version of the Brakerski-Fan-Vercauteren somewhat-homomorphic encryption scheme [25], where the secret key is additively shared among the parties.

C. 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], [6]. 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.

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 provider can assume this role. Although it is frequent to define the role of *computing party* in current MPC applications [30], [4], [5], it is usually a part of the N-party to 2-party problem reduction that introduces non-collusion assumptions. In the $\mathsf{MHE-MPC}$ protocol, the computing parties are not required to be part of the computation data access structure, removing the need for such assumptions.

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 this step can resemble the *offline* phase of the LSSS-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 directly depend 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 (and its keys) have to be parameterized to support a sufficient *homomorphic capacity* κ.

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

The *Eval* step consists in the evaluation of the *ideal functionality*, using the homomorphic property of the scheme to compute the encrypted output. As this step requires no secret input from the parties, it can 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 $\mathcal{R}$. Thus, whenever $\mathcal{R}$ has no input in the computation, it is fully decoupled from the secure multiparty computation problem.

Protocol 1. $\mathsf{MHE-MPC}$

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

Setup: the parties instantiate the MHE scheme E^*

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

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

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

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

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

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

We propose a concrete, practical and open-source instance of the $\mathsf{MHE-MPC}$ protocol. In this instance, E^* is the Brakerski-Fan-Vercauteren (BFV) homomorphic scheme [25]. Hence, this instance supports the computation of any function that can be represented or approximated by a polynomial over the plaintext space of BFV (defined in Section III-E).

D. Mathematical Notation

We denote $[\cdot]_q$ the reduction of an integer modulo q, and $\lceil \cdot \rceil$, $\lfloor \cdot \rfloor$, $\lfloor \cdot \rceil$ 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 $\|a\|$.

E. The BFV Encryption Scheme

The Brakerski-Fan-Vercauteren cryptosystem [25] is a ring-learning-with-errors [38] 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 [45], [41], [1] and is part of the draft HE standard [3].

We first recall the original and most common instantiation of the (centralized) BFV encryption scheme 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 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 $[-B, B]$.

The security of BFV is based on the hardness of the *decisional-RLWE* problem [38], 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. We denote BFV.Add and BFV.Mul the homomorphic addition and multiplication respectively, and refer the reader to [25] for their implementation. The BFV.Mul operation outputs a ciphertext consisting of three R_q elements, that 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 this degree back to one, by using a BFV.Relinearize operation [25]. This operation is public but requires the generation of a special type of public key, referred to as 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

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

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

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

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. To preserve this condition when multiplying with the rlk (as a part of BFV.Relinearize), ciphertexts are temporarily decomposed in a basis $w < q$ and the product is performed on each element of the decomposition [25]. We write $l = \lceil \log_w(q) \rceil$ the number of coefficients in this decomposition, and $\mathbf{w} = (w^0, w^1, ..., w^l)^T$ the base-w reconstruction vector.

Scheme 1. BFV

```
BFV.SecKeyGen($1^\lambda$): 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:
        $m' = [\lfloor \frac{t}{q} [c_0 + c_1 s]_q \rceil]_t$
```

IV. The Multiparty BFV Scheme

We introduce a novel multiparty version of the Brakerski-Fan-Vercauteren (BFV) cryptosystem [25] that supports the MHE−MPC protocol (Protocol 1, Section III-C). 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 [14] or the more recent CKKS [17], that enables homomorphic approximate arithmetic. In fact, we implemented both multiparty versions for the BFV and CKKS schemes in the Lattigo open-source library [1]. Even though our high-level approach follows the blueprint of the LWE-based protocols by Asharov et al. [6], we introduce several improvements to their schemes. In particular, we propose a novel procedure for generation of relinearization keys that introduces significantly less noise in the key than the procedure proposed by Asharov et al.. Additionally, we propose a generalization of the distributed decryption procedure, from which we derive novel protocols that bridge between the MHE-based and LSSS-based MPC protocols, and that enable the practical bootstrapping of a BFV ciphertext.

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

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

Protocol 2. EncKeyGen

Public Input: p_1 (common random polynomial)
Private Input for P_i: $s_i = \mathsf{sk}_i$ (secret key share)
Public Output: $\mathsf{cpk} = (p_0, p_1)$ (collect. encrypt. 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 $\mathsf{cpk} = (p_0\ ,\ p_1)$

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. Thus, when used as E^* in the MHE$-$MPC protocol, this scheme can solve secure-multiparty-computation problems in the strictest dishonest-majority setting: 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. We detail the security arguments in Appendix A. 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 samples its own share as $s_i = \mathsf{BFV.SecKeyGen}(\lambda, \kappa)$ independently. Thus, the ideal secret-key is generated in a non-interactive way. Eq. (3) applies, but 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 the security of our scheme (analyzed in Appendix A) does not rely on this property. However, the norm of the resulting ideal secret key grows with $\mathcal{O}(N)$, which has an effect on the noise growth (analyzed in Appendix C).

By using techniques such as those described in [44], 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

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). In the passive-adversary model, the CRS can be implemented by any keyed pseudorandom function. We used BLAKE2b [7] in our implementation.

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

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

that has the same form as the ideal public key pk in Scheme 1, with larger worst-case norms $\|s\|$ and $\|e\|$. The norm grows linearly in N hence is not a concern (as shown in Appendix C), even for large number of nodes. Another notable feature of the EncKeyGen protocol is that it would apply to any kind of linear sharing of s, as long as the shares are valid RLWE secrets and the norm of the reconstruction is small enough. This includes uniformly random shares of a traditional BFV secret key in R_3.

C. Collective Relinearization-Key Generation

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 $b = 0 \ldots l$ 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 [6]. 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. By observing that the collective encryption key is not needed for this purpose (because the secret key is collectively known), we propose Protocol 3 as an improvement over the method by Asharov et al.

After completing the RelinKeyGen protocol, the parties have access to a relinearization key of the form

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

where $\mathbf{b} = s\mathbf{a} + \mathbf{e}_2$ and $e_k = \sum_j e_{k,j}$ for $k = 0, 1, 2, 3$. As opposed to the technique by Asharov et al., $\mathbf{r}_1$ contains no error. This significantly reduces the output noise, as this component (and its error terms) is multiplied by the base-w decomposed ciphertext during relinearization.

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 [25], those based on the Chinese Remainder Theorem, as proposed by Bajard et al. [8] and Cheon et al [16], and even hybrids of these two approaches (which we use in our implementation).

Protocol 3. RelinKeyGen

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

Each party P_i:

1) samples $u_i \leftarrow R_3$, $\mathbf{e}_{0,i}, \mathbf{e}_{1,i} \leftarrow \chi^l$ and discloses $(\mathbf{h}_{0,i}\ ,\ \mathbf{h}_{1,i}) = (-u_i\mathbf{a} + s_i\mathbf{w} + \mathbf{e}_{0,i}\ \ ,\ \ s_i\mathbf{a} + \mathbf{e}_{1,i})$
2) 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 $\mathbf{e}_{2,i}, \mathbf{e}_{3,i} \leftarrow \chi^l$ and discloses $(\mathbf{h}'_{0,i}\ ,\ \mathbf{h}'_{1,i}) = (s_i\mathbf{h}_0 + \mathbf{e}_{2,i}\ \ ,\ \ (u_i - s_i)\mathbf{h}_1 + \mathbf{e}_{3,i})$

Out: from $\mathbf{h}'_0 = \sum_{P_j \in \mathcal{P}} \mathbf{h}'_{0,j}$ and $\mathbf{h}'_1 = \sum_{P_j \in \mathcal{P}} \mathbf{h}'_{1,j}$, outputs $\mathsf{rlk} = (\mathbf{h}'_0 + \mathbf{h}'_1\ ,\ \mathbf{h}_1)$

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. When only a public key is known, the PubColKeySwitch protocol is used.

*1) Collective Key-Switching (*ColKeySwitch*):* Protocol 4 details the steps for performing 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 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 parties with secret-shares of a secret key it controls.

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

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

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 $s'_j = 0 \quad \forall P_j \in \mathcal{P}$, and is the basis for bridging MHE-based and LSSS-based approaches, as explained in Section IV-E.

Protocol 4. ColKeySwitch

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

Each party P_i:

1) samples $e_i \leftarrow \chi$ and discloses $h_i = (s_i - s'_i)c_1 + e_i$

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

*2) Collective Public-Key Switching (*PubColKeySwitch*):* The use of the ColKeySwitchprotocol is limited to the cases where parties have collective knowledge of the output secret key s'. Indeed, that may not be the case when considering an external receiver. This situation would require confidential channels between the receiver and each party in $\mathcal{P}$, in order to either (i) collect decryption shares from all parties, or (ii) distribute an additive sharing of its secret key to the system. However, (i) would quickly become expensive for a large number of parties, and (ii) would require $\mathcal{R}$ to trust at least one party in $\mathcal{P}$. Moreover, confidential point-to-point channels might not fit the system model (e.g., on public smart-contract technologies). 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 secure-multiparty-computation 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 $p'_0 = -(s'p'_1 + e_{\mathsf{pk}})$, the parties hold ct' satisfying

$$\begin{aligned}\mathsf{Dec}(s', \mathsf{ct}') &= \lfloor \frac{t}{q}[c_0 + \sum_j (s_jc_1 + u_jp'_0 + e_{0,j}) + s'\sum_j (u_jp'_1 + e_{1,j})]_q \rceil \\ &= \lfloor \frac{t}{q}[c_0 + sc_1 + up'_0 + s'up'_1 + e_0 + s'e_1]_q \rceil \\ &= \lfloor \frac{t}{q}[\Delta m + e_{\mathsf{ct}} + e_{\mathsf{PCKS}}]_q \rceil = m, \qquad (7)\end{aligned}$$

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 target-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 LSSS-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

Protocol 5. $\mathsf{PubColKeySwitch}$

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

Each party P_i:

1) samples $u_i \leftarrow R_3$, $e_{0,i} \leftarrow \chi$, $e_{1,i} \leftarrow \chi$, and discloses
$$(h_{0,i}\ ,\ h_{1,i}) = (s_i c_1 + u_i p'_0 + e_{0,i}\ \ ,\ \ u_i p'_1 + e_{1,i})$$

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

additive sharing of m over R_t by masking their share in the decryption (i.e., $\mathsf{ColKeySwitch}$ 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 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 ($\mathsf{Share2Enc}$*):* Given a secret shared value $m \in R_t$ such that $m = \sum_{i=1}^{N} M_i$, the parties produce an encryption $\mathsf{ct} = (c_0, c_1)$. To do so, each party P_i samples a from the CRS and produces a $\mathsf{ColKeySwitch}$ 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 $\mathsf{Share2Enc}$ and $\mathsf{Enc2Share}$ protocols can be combined into a *multiparty bootstrapping* procedure (Protocol 6), enabling the parties to reduce the ciphertext noise back to a *fresh*-like one, which enables further computation even when reaching the homomorphic capacity limits. This is a crucial functionality for the BFV scheme, for which the bootstrapping procedure is expensive.

Intuitively, the $\mathsf{ColBootstrap}$ protocol consists in a conversion from an encryption to secret-shares and back, implemented as a parallel execution of the $\mathsf{Enc2Share}$ and $\mathsf{Share2Enc}$ protocols. It is an efficient single-round protocol that the parties can use during the evaluation phase, instead of a computationally heavy bootstrapping procedure. Note, however, that making use of that functionality introduces interaction at the evaluation step of the $\mathsf{MHE}{-}\mathsf{MPC}$ 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. But the $\mathsf{ColBootstrap}$ protocol offers a trade-off between computation and communication (demonstrated in Section VI-B), and more flexibility when the computation circuit is not known in advance.

Protocol 6. $\mathsf{ColBootstrap}$

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

Each party P_i:

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

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

G. 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 element-wise plaintext arithmetic. Provided with public *rotation keys*, any semi-honest party can operate arbitrary rotations over the vector components [16], which opens up homomorphic function evaluation to a broad kind of non-linear functions. Generating these rotation keys (that are pseudo-encryptions of the equivalent rotation on the secret-key coefficients) can be done in the multiparty scheme, by means of an $\mathsf{RotKeyGen}$ sub-protocol. We do not detail this protocol, as it is a straightforward adaptation of $\mathsf{EncKeyGen}$. We will make use of rotations in the input-selection example circuit in Section VI-C.

V. Features Analysis

We now discuss the high-level aspects of the $\mathsf{MHE}{-}\mathsf{MPC}$ protocol, when instantiated with the multiparty BFV scheme of Section IV.

A. Introduced trade-offs

We first discuss the trade-offs inherent to our construct, and briefly show how to optimize them.

1) Circuit Privacy: 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 cryptosystem family does not directly ensure *circuit privacy*. This has an important implication for our multiparty scheme, where Equations (6) and (7) show that both $\mathsf{ColKeySwitch}$ and $\mathsf{PubColKeySwitch}$ permit the final error term to be obtained by the receiver. By exploiting these dependencies, an attacker with the output secret key could try to extract information about the input key or the intermediate plaintext values in the computation.

The solution proposed by Asharov et al. [6], commonly referred to as *smudging* or *noise flooding*, consists in adding fresh noise to the ciphertext and to the decryption shares. We could adapt this approach to our scheme by sampling the noise introduced during the $\mathsf{ColKeySwitch}$ and $\mathsf{PubColKeySwitch}$ protocols from a distribution that has a variance σ^2_{smg} significantly *larger* than that of the input ciphertext noise distribution (represented by σ^2_{ct}). Concretely, choosing $\sigma^2_{\mathrm{smg}} = 2^{\lambda}\sigma^2_{\mathsf{ct}}$ bounds the advantage of an attacker trying to extract information from the output noise to $2^{-\lambda}$.

While theoretically ensuring circuit privacy, this technique will, in practice, lead to less efficient parameters for circuits of large depth. Hence, whereas our initial approach used smudging as placeholder circuit-privacy technique (in our security argument of Appendix A), it should be replaced by a more efficient solution. A promising direction was recently explored by de Castro et al. in the context of oblivious linear function evaluation using the RNS variant of BFV [23] (which we use in our implementation). Their technique, based on a special RNS modular-reduction and rounding, generalizes to non-linear functions and can be used in our context.

2) Arithmetic Circuits: The MHE-based MPC solution is indeed limited to arithmetic functions over the plaintext space ring of the MHE scheme (in our case, $(R_t, +, \times)$). Whereas analytic function such $\sin(x)$ or e^x can be efficiently evaluated through polynomial approximation, it is not the case for non-arithmetic functions such as comparisons. In LSSS-based MPC, however, several solutions are already available to compute such functions and enable branching programs at the cost of leaking (information about) the conditional variable. Hence, the ability to switch between the two representations with the $\mathsf{Enc2Share}$ and $\mathsf{Share2Enc}$ protocols is currently pivotal, as this enables LSSS-based approaches to cover for the current limitations of the MHE-based approach. Indeed, these limitations are being increasingly mitigated, as MHE-based solutions directly benefit from the advances in homomorphic encryption research that make HE schemes increasingly versatile.

MHE-based solution can also cover for limitations of the LSSS-based approaches. In the rest of this section, we discuss the particular features of the MHE-based solution and relate these features to the efficient system models they enable.

B. 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 output or collective bootstrapping is needed, the circuit evaluation procedure is non-interactive and can be performed by any semi-honest entity. This not only enables the evaluation to be efficiently distributed among the parties in the usual peer-to-peer setting, but also enables new computation models for MPC:

Cloud-Outsourced Model: The homomorphic circuit evaluation can be outsourced to a cloud-like service, by providing it with the inputs and necessary 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.

Smart Contracts: A special case of an outsourced MPC task is the execution of a smart contract over private data, which becomes feasible by means of the MHE-based MPC solution. In this scenario, the contract shareholders are the MHE secret-key owners, and the smart-contract platform acts as an oblivious contract evaluator. Indeed, depending on the platform, further adaptation would be required beyond the system and threat models of this work.

C. Public-Transcript Protocols

All the protocols of Section IV have public transcripts, which removes the need for direct party-to-party communication. Hence, not only the evaluation step, but the whole $\mathsf{MHE{-}MPC}$ protocol can be executed over any public authenticated channel. This also brings new possibilities in designing MPC systems:

Efficient Peer-to-Peer Communication Pattern: The presented protocols rely solely on the ability for the parties to publicly *disclose* their shares and to aggregate them. This gives flexibility for using efficient communication patterns: The parties can be organized in a topological way, as nodes in a tree, where each node interacts 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 consists in

Gen: computing its own share in the protocol,
Agg: collecting and aggregating the share of each of its children and its own share,
Out: sending the result *up the tree* to its parent, or outputting it.

Such an execution enables the parties to compute their shares in parallel and results in a network traffic that is constant at each node. By trading-off some latency, the 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. As 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 VI-D.

Cloud-Assisted MPC Model: The special case of a single root node holding no share of the key can be mapped to a cloud-assisted setting where parties run the protocols interacting solely with a central node. This model complements the circuit evaluation outsourcing feature by removing the need for synchronous and private party-to-party communication and the need for the input parties to be online and active for the protocol to progress. Hence, the cloud-assisted $\mathsf{MHE{-}MPC}$ protocol has a clear advantage in terms of tolerance to unreliable parties, which is a significant step toward large-scale MPC. We use the cloud-assisted model for the first two example circuits of Section VI and demonstrate its practicality for computations involving thousands of parties. Adapting the multiparty BFV scheme (Section IV) to a T*-out-of-*N threshold scheme is a natural next step (provided that the system tolerates the weaker threat-model) to address the challenge of parties going offline for an arbitrary amount of time.

VI. Performance Analysis

We implemented the multiparty BFV scheme and integrated it in the Lattigo open-source library [1]. 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 [8]. In order to analyze the performance of the MHE−MPC protocol in both the cloud-assisted and the peer-to-peer settings, we chose to evaluate three generic yet powerful circuits. Thus, we can compare the results with those of a baseline system for generic MPC in a reproducible way. Indeed, these circuits represent common building blocks for more complex functionalities that we briefly discuss. All the parameters we used have an equivalent security of at least 128 bits [3].

In the cloud-assisted setting, we consider two example circuits: (i) The element-wise product of integer vectors, and its application as a simple multiparty private-set-intersection protocol (Section VI-B). (ii) A multiparty input selection circuit, and its application to multiparty private-information-retrieval (Section VI-C). We compare the performance for both circuits against a baseline system that uses a LSSS-based approach: The MP-SPDZ library implementation [2] of the Overdrive protocol [32] for the semi-honest, dishonest majority setting. In the peer-to-peer setting, we consider the generation of multiplication triples, commonly referred to as the "offline" phase of data-level LSSS-based approaches (Section VI-D). We compare the performances against the SPDZ2K [19] Oblivious-Transfer-based and the Overdrive [32] HE-based triple-generation protocols.

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 run all parties on the latter machine. We do not include the network-related delays in our measurements and, instead, evaluate the network-related cost in terms of number of communicated bytes (upstream + downstream), to make it independent from the actual setting. Hence, we run the benchmarks over the localhost interface (note that this could slightly favor the baseline system, as its online phase is more sensitive to the round-trip time delays).

B. Element-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 element-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 element-wise product over integer vectors. This is a demanding circuit, as its multiplicative depth is equal to $\lceil \log N \rceil$.

TABLE I: Element-wise product: Baseline comparison

		Time [s]			Com./party [MB]		
#Parties		2	4	8	2	4	8
[2]	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
	Circ.	**0.29**	**0.41**	**0.64**	**4.72**	**4.72**	**4.72**

TABLE II: Element-wise product: Phases costs

	Party		Cloud		
	Time [ms]	Com. [MB]	Wall time/CPU 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	0.8/4.5	1.0/10.3	1.5/22.7
Out	25.38	3.15	0.05	0.10	0.21

The steps in the MHE−MPC protocol unfolds as follows:

Setup: The parties use the EncKeyGen and RelinKeyGen protocols to produce the public encryption and relinearization 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 $\mathsf{Enc}_{s_\mathcal{P}}(x_i)$ 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 $\mathsf{Enc}_{s_\mathcal{R}}(y)$, which the receiver retrieves.

We set the vector size to $d = 2^{14}$. The baseline system [2] 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. The input vectors are encoded in the plaintext space using packed encoding. Table I reports the evaluation of our implementation of the MHE solution against the baseline, for up to 8 parties. We report the cost per phase of the MHE−MPC protocol for up to 128 parties 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. We also show that parallelizing the homomorphic evaluation can yield even lower response times. Finally, we run the experiment for $N = 1024$ parties, which represents a circuit of multiplicative depth 10. The above parameterization of the MHE scheme would not enable such a large depth, so we

TABLE III: Element-wise product: $N = 1024$ parties

	Party				Cloud	
	CPU Time [ms]		Com. [MB]		Wall/CPU Time [h:m:s]	
	(i)	(ii)	(i)	(ii)	(i)	(ii)
Setup	467.5	110.2	121.8	25.2	57s	13s
In	78.4	21.6	6.3	1.6	3s	1s
Eval	0.00	202.4	0.0	18.9	29s/19m14s	6s/3m45s
Out	107.5	27.2	12.6	3.1	4.3s	1.2s

have two options. The first one (i) is to adapt the parameters to enable larger depth; more specifically, we use a polynomial degree $n = 2^{15}$, and a coefficient size q of 880 bits. The second option (ii) is to keep the same parameters and use the ColBootstrap protocol to refresh the ciphertexts. The results in Table III illustrate how the ColBootstrap protocol introduces a trade-off between network usage and CPU usage. In this case, for an additional 4.7 MB of communication per party in the online phase, refreshing ciphertexts is more cost-effective (for bandwidth and CPU, by a factor between $4\times$ and $5\times$) than using larger parameters, even if it requires one more communication round.

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. Transferring the results to this application, we can compare with the special purpose multiparty PSI protocol by Kolesnikov et al. [33]. For the standard semi-honest model with dishonest majority, the set size 2^{12} and 15 parties (the largest evaluated value in [33]), the MHE solution was $1029\times$ faster (in the LAN setting) and required $15.3\times$ less communication to compute the intersection. However, our encoding of sets 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 ideal functionality $f_1(r, x_2, \ldots, x_N) = x_r$ for player P_1.

This selection circuit can be seen as a generalization of an oblivious transfer functionality to the N-party setting, and can directly implement an N-party PIR system where a *requester* party retrieves a row in a database partitioned across multiple *provider* parties. We represent inputs as d-dimensional vectors in $\mathbb{Z}_p^d$ for p a 32-bit prime. We denote u_i the polynomial having all-zero coefficients but its degree i one, which is 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 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}.

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).

TABLE IV: Input selection: Baseline comparison

		Time [s]			Com./party [MB]		
#Parties		2	4	8	2	4	8
[2]	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
	Circ.	**0.27**	**0.28**	**0.31**	**1.31**	**1.31**	**1.31**

TABLE V: 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	0.4/8.1	0.8/23.4	1.6/62.1
Out	3.34	0.79	0.01	0.02	0.02

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 in Section VI-B. The MHE system was instantiated with $n = 2^{13}$, 218-bit q and 32-bit plaintext modulus t. Table IV 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. In the 8 parties case, 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.

Table V shows the per-phase cost for the MHE-based solution for larger number of parties. Again, the parallelization of the circuit computation over multiple threads 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 105.7 kB/s (baseline: 9.0 kB/s) and an expansion of only $\sim 40\times$ (baseline: $\sim 3650\times$) w.r.t. the communication cost of an insecure plaintext system. We ran the same experiment for $N = 8000$ parties 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 is of particular interest, as triple generation is the bottleneck cost for LSSS-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 $F_{\mathcal{P}}(\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_i}) = (\mathbf{c_1}, \mathbf{c_2}, \dots, \mathbf{c_N})$ 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 $\mathbf{b}$. Hence, this phase ends with the root node holding $\mathsf{ct}_a = \mathsf{Enc}(\mathbf{a})$ 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 the Lattigo library [1]. For the OT-based one, we used the *Multi-Protocol SPDZ* library [2] that provides an implementation of the SPDZ2K [19] 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 of all methods in the measurements.

E. Discussion

We observe that, in general, the main cost of MHE-based solutions is the network load of their setup phase, primarily due to the generation of 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 as efficient as a LSSS-based approach generating triples *on-the-fly*. However, since the MHE setup is only performed once, it is quickly amortized when considering circuits with a few thousands multiplication gates and more than two parties; in that scenario, the cost of the LSSS-based approach is dominated by the generation of multiplication triples. 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.

[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.

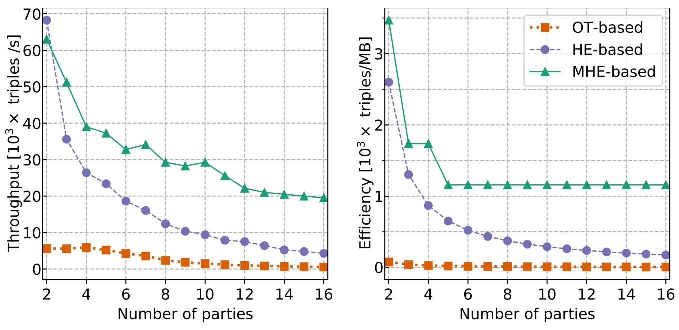

Fig. 1: Number of generated triples per second (throughput, left) and per megabyte of communication (efficiency, right).

VII. Conclusions

In this work, we have introduced a novel MHE scheme based on the BFV cryptosystem, and demonstrated its capacity to solve multiparty computation problems. We have analyzed the features of the proposed solution, and observed how the public-transcript nature of the MHE$-$MPC protocol enables new computation models for MPC that go beyond the traditional peer-to-peer setting. Notably, this includes outsourced cloud-assisted models that reduce the communication cost per party to be constant in the number of parties without relying on non-collusion assumptions. We also analyzed the performance of the cloud-based solution, and noticed a net improvement ranging between 1 and 2 orders of magnitude in both response time and communication complexity compared to the LSSS-based approaches. Hence, this cloud-assisted model enables new opportunities for large scale MPC-as-a-service, which we view as a promising application and a driver for adoption of HE and MPC solutions in privacy-enhancing technologies.

Acknowledgments

The authors would like to thank 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] Lattigo 1.3.0. Online: http://github.com/ldsec/lattigo, December 2019. EPFL-LDS.
[2] MP-SPDZ. Online: https://github.com/data61/MP-SPDZ/, January 2020.
[3] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic Encryption Security Standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[4] Andreea B Alexandru, Manfred Morari, and George J Pappas. Cloud-based MPC with encrypted data. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 5014–5019. IEEE, 2018.

[5] David W Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P Smart, and Rebecca N Wright. From Keys to Databases—Real-World Applications of Secure Multi-Party Computation. *The Computer Journal*, 61(12):1749–1771, 2018.

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

[7] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. Blake2: simpler, smaller, fast as md5. In *International Conference on Applied Cryptography and Network Security*, pages 119–135. Springer, 2013.

[8] Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*, pages 423–442. Springer, 2016.

[9] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[10] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril 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*, pages 227–234. Springer, 2015.

[11] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[12] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis. In *International Conference on Financial Cryptography and Data Security*, pages 57–64. Springer, 2012.

[13] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes live. In *International Conference on Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

[14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):13, 2014.

[15] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Annual International Cryptology Conference*, pages 19–40. Springer, 2001.

[16] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.

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

[18] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 259–282, 2017.

[19] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD$\mathbb{Z}_{2^k}$: Efficient mpc mod 2^k for dishonest majority. In *Annual International Cryptology Conference*, pages 769–798. Springer, 2018.

[20] Ronald Cramer, Ivan Damgård, and Jesper B Nielsen. Multiparty computation from threshold homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–300. Springer, 2001.

[21] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

[22] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 643–662. Springer, 2012.

[23] Leo de Castro, Chiraag Juvekar, Analog Devices, and Vinod Vaikuntanathan. Fast vector oblivious linear evaluation from ring learning with errors. *IACR Cryptology ePrint Archive*, 2020.

[24] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–458, 1994.

[25] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[26] Matthew Franklin and Stuart Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232, 1996.

[27] 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*, pages 1–1, 2020.

[28] Craig Gentry and Dan Boneh. *A fully homomorphic encryption scheme*, volume 20. Stanford University Stanford, 2009.

[29] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. SoK: General purpose compilers for secure multi-party computation. In *Symposium on Security and Privacy (SP)*, pages 1220–1270. IEEE, 2019.

[30] Karthik A Jagadeesh, David J Wu, Johannes A Birgmeier, Dan Boneh, and Gill Bejerano. Deriving genomic diagnoses without revealing patient genomes. *Science*, 357(6352):692–695, 2017.

[31] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

[32] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making SPDZ great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018.

[33] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *ACM Conference on Computer and Communications Security*, pages 1257–1272, 2017.

[34] J Kroll, E Felten, and Dan Boneh. Secure protocols for accountable warrant execution. 2014.

[35] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

[36] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. Cloud-Assisted Multiparty Computation from Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2011:663, 2011.

[37] Adriana López-Alt, Eran Tromer, and Vinod 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*, pages 1219–1234. ACM, 2012.

[38] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[39] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 38th IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[40] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 334–348. IEEE, 2013.

[41] Yuriy Polyakov, Kurt Rohloff, and Gerard W Ryan. PALISADE lattice cryptography library. https://git.njit.edu/palisade/PALISADE, 2018.

[42] Jean Louis Raisaro, Juan Troncoso-Pastoriza, Mickaël Misbach, João Sá Sousa, Sylvain Pradervand, Edoardo Missiaglia, Olivier Michielin, Bryan Ford, and Jean-Pierre Hubaux. MedCo: Enabling secure and privacy-preserving exploration of distributed clinical and genomic data. *IEEE/ACM transactions on computational biology and bioinformatics*, 16(4):1328–1341, 2018.

[43] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[44] Dragos Rotaru, Nigel P Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for spdz. *IACR Cryptol. ePrint Arch.*, 2019:1300, 2019.

[45] Microsoft SEAL (release 3.2). https://github.com/Microsoft/SEAL, February 2019. Microsoft Research, Redmond, WA.

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

[47] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 724–738. IEEE, 2019.

Appendix

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* [38]. For a more thorough analysis, we refer the reader to the works by Asharov et al. [6]. We provide arguments in terms of the ideal/real simulation formalism [35].

We prove by construction that, for every possible $\mathcal{A}$ defined as a subset of at most $N-1$ corrupted polynomial-time parties in $\mathcal{P}$, 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 from the simulated one. 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 $\mathsf{EncKeyGen}$ 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 $\mathsf{EncKeyGen}$ 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 $\mathsf{EncKeyGen}$ 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 [38]. For the share of player P_h, we consider two cases: (1) When $\mathcal{H} \neq \emptyset$, the share $p_{0,h}$ is uniformly random 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 share of P_h, hence outputting the real view.

B. Collective Key-Switching

The security argument for the $\mathsf{ColKeySwitch}$ 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 $\mathsf{ColKeySwitch}$ protocol only. However, the argument 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 $\mathsf{PubColKeySwitch}$ and $\mathsf{ColBootstrap}$ protocols.

Given a ciphertext $\mathsf{ct} = (c_0, c_1)$ decrypting under s, the ideal functionality of the $\mathsf{ColKeySwitch}$ 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' that satisfies $c_0 + sc_1 - e = c_0' + s'c_1 - e'$, where e and e' are the noise terms that result from the decryption of ct and ct', respectively. We consider its explicit form as an equivalent 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 $\mathsf{ColKeySwitch}$ 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 $(\widetilde{h}_1, \widetilde{h}_2, ..., \widetilde{h}_N,\ \hat{h}) \overset{c}{\equiv} (h_1, h_2, ..., h_N,\ h)$: that $\mathsf{ColKeySwitch}$ securely computes its functionality.

1) Output indistinguishability: We want to show that

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

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 $\overset{s}{\equiv}$) from the fresh noise of the ideal output: $e' = e - \hat{e} + e' \overset{s}{\equiv} e + e'$.

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 $\hat{h} \overset{c}{\equiv} h$.

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: $(-s_i c_1 + e'_i, c_1)$ 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 who has access to s' can extract $e + e'$ from the output and might be able to estimate e'_i for $i \notin \mathcal{A}$. Thus, 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. We formalize this application-related requirement as:

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.

C. 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 $f(x_1, \dots, x_N) = (\mathbf{r}_0, \mathbf{r}_1)$, the generated relinearization key defined in Eq. (5). Throughout the protocol execution, the parties compute the public values $\mathbf{h} = (\mathbf{h}_0, \mathbf{h}_1)$ and $\mathbf{h}' = (\mathbf{h}'_0, \mathbf{h}'_1)$. These values can be simulated, with the constraints $\mathbf{r}_0 = \mathbf{h}'_0 + \mathbf{h}'_1$ 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

$$\widetilde{\mathbf{h}}_i = \begin{cases} ([-u_i \mathbf{a} + s_i \mathbf{w} + \mathbf{e}_{0,i}]_q \;\;,\;\; [s_i \mathbf{a} + \mathbf{e}_{1,i}]_q) & \text{if } P_i \in \mathcal{A} \\ \leftarrow R_q^{2 \times l} & \text{if } P_i \in \mathcal{H} \\ (\leftarrow R_q^l \;\;,\;\; [\mathbf{r}_1 - \sum\limits_{P_j \in \mathcal{A} \cup \mathcal{H}} \widetilde{\mathbf{h}}_{1,j}]_q) & \text{if } P_i = P_h \end{cases},$$

$$\widetilde{\mathbf{h}}'_i = \begin{cases} ([s_i \widetilde{\mathbf{h}}_0 + \mathbf{e}_{2,i}]_q \;\;,\;\; [(u_i - s_i)\widetilde{\mathbf{h}}_1 + \mathbf{e}_{3,i}]_q) & \text{if } P_i \in \mathcal{A} \\ \leftarrow R_q^{2 \times l} & \text{if } P_i \in \mathcal{H} \\ (\mathbf{b} \leftarrow R_q^l \;\;,\;\; [\mathbf{r}_0 - \mathbf{b} - \sum\limits_{P_j \in \mathcal{A} \cup \mathcal{H}} \widetilde{\mathbf{h}}'_j]_q) & \text{if } P_i = P_h \end{cases},$$

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

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

Lemma 1 extracts the sought property for the transcript (note that $\mathbf{h'_0} - \mathbf{h'}_1 \approx s^2\mathbf{a} + s^2\mathbf{w}$).

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*

$$(a, \quad s_1 a + e_1, \quad s_2 a + e_2, \quad s_2 s_1 a + e_3, \quad s_1^2 a + e_4) \quad (8)$$

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

We omit 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. (8) 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 $u = s_2$.

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 [25] 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.

Ideal Secret-Key and Encryption-Key: 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 $\|s\| \leq N$.

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 χ.

Fresh Encryption: Let ct= (c_0, c_1) be a fresh encryption of a message m under a collective public key. The first step

TABLE VI: Benchmarking parameter sets

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

of the decryption (Eq. (1)) under the *ideal* secret key outputs $c_0 + sc_1 = \Delta m + e_{\textit{fresh}}$, where

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

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

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, $c_0' + s'c_1 = m + e_{\textit{fresh}} + e_{\mathsf{CKS}}$ with

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

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

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, $c_0' + s'c_1' = m + e_{\textit{fresh}} + e_{\mathsf{PCKS}}$ with

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

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

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), on a single core. Table VI shows the parameter sets used in our benchmarks, all guaranteeing a security level of at least 128 bits [3]. These parameters are application dependent, but their choice represents typical sizes of q that correspond to different *homomorphic capacities*. Table VII shows the timings for the operations of the centralized scheme that are used in our MHE-based solution. For each protocol of the distributed scheme, Table VIII shows the following values: *Gen* is to the cost for a given party to generate its own public share in the protocol (aggregated over all rounds for RelinKeyGen). *Agg* corresponds to the cost of combining two shares in the protocol. *Out* corresponds to the cost of computing the final output of the protocol when provided with the aggregate of all the shares Table IX 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.

TABLE VII: 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 VIII: 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 IX: 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