# Computing across Trust Boundaries using Distributed Homomorphic Cryptography

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

***Abstract*—In this work, we advance the conceptual and technical aspects of Secure Multiparty Computation (SMC). We approach SMC as a computational problem and propose a novel formulation of this problem in terms of *trust boundaries*. From this formulation, we derive a general framework that enables a more comprehensive characterization of both the SMC problem and its solutions. Existing SMC solutions are commonly seen as diametrically different and incompatible, but we show how they can be mapped to particular instances of our framework, hence enabling their analysis under a common and unified basis. In this framework, the core component of an SMC solution is a *distributed homomorphic cryptosystem*. We show that the features this cryptosystem provides determine the need for interaction and overall efficiency of the corresponding SMC solutions. Based on this analysis, we introduce a practical instantiation of our framework by proposing a distributed version of the Brakerski-Fan-Vercauteren (BFV) lattice-based homomorphic cryptosystem. We analyze the security, noise overhead, and computational costs of this scheme. Due to its conceptual simplicity and efficiency, our solution has great potential for addressing highly relevant scenarios, such as secure data-sharing and machine-learning. Hence, this work constitutes a step forward in secure computation, by enabling computation across trust boundaries.**

***Index Terms*—Secure Multiparty Computation, Distributed Homomorphic Cryptography, Distributed Systems**

## I. Introduction

In a *secure multiparty computation* (SMC) setting, a group of parties are able to *securely* carry out computations over their joint inputs. Although the exact definition of security depends on how the adversary is modeled, the most common requirement, *input privacy*, informally states that parties should not obtain more information about the other parties' inputs than what can be deduced from the computation output. Interestingly, an SMC setting can model almost any secure interactive protocol, due to the generality of its formulation. Combining this generality with strong security guarantees, SMC is highly relevant, both from an academic and a technical perspective. In fact, the last decade has seen this established theoretical field evolve into an applied one, notably due to its potential in securing *data-sharing* scenarios in the financial [1]–[3], biomedical [4]–[6], and law enforcement [7], [8] sectors.

However, the cultural and expertise gaps between cryptographic research and its application domains remain significant obstacles to this transition. One potential reason could be the community's focus on making SMC protocols efficient, to address what has long been their most criticized aspect. Although being a successful endeavor, this had the downside of leaving the potential users of these protocols with a fragmented and intricate design space that comprises many highly-optimized yet specialized protocols, ultimately making their integration and implementation difficult for non-experts. In response to this issue, several code-to-protocol compilers were proposed in order to provide a *technical* layer of abstraction [9]–[12] on top of complex MPC protocols. But, as Hastings et al. point out, these tools are themselves research projects that do not yet provide the required level of usability for non-expert end-users [13]. As a result, and despite substantial efforts in improving the bandwidth and overall interaction requirements of SMC protocols, their integration in concrete systems remains a significant challenge.

In this work, we take a different approach by proposing a set of *conceptual* abstractions for SMC solution. We propose an abstract, general-purpose SMC framework and show its potential to unify existing solutions in the SMC design space. We also propose a *concrete* instantiation of this framework, at the core of which lies a distributed version of the Brakerski-Fan-Vercauteren (BFV) homomorphic cryptosystem [14] that we introduce. This instantiation, as we demonstrate, has great potential to be adopted in privacy engineering, due to its efficiency and conceptual simplicity.

### *Approach and Outline*

We view SMC settings as particular instances of a computational problem that we refer to as *the secure-multiparty-computation problem* (formalized in Section IV-A). In particular, we carefully distinguish between the SMC problem and its existing solutions, sometimes directly and ambiguously referred to as *multiparty computation* or *MPC*. The benefit of a problem-centric formulation is to account for both the general and recursive nature of SMC: we can use this definition to model a broad range of well-known SMC instances, ranging from simple oblivious transfer to large scale data sharing, but also to model the less-obvious ones, such as simple secret message exchange, key agreement, or privacy-preserving cloud computing. As another example, the scenario of evaluating a smart contract over private inputs [15] that are provided by at least two mutually distrusting parties instantiates an SMC problem [16]. We illustrate the recursive nature of SMC problems in Section IV, where we consider examples of how SMC problems can be *reduced* to other, simpler, ones.

We introduce the concept of *trust boundary* as trust-modeling primitive that can abstract a cryptographic setting, and relate the SMC problem to making data cross these trust boundaries. This formulation enables a novel view on the known theoretical impossibility of a purely centralized SMC solution [16], where a single party would obliviously carry out the computation on behalf of the other parties.

We show how this view naturally leads to the somewhat less acknowledged yet generic SMC solution that relies on *distributed homomorphic encryption* to protect input data. Such a solution uses secret-sharing techniques [17] to distribute the secret encryption key among the parties. We formalize this approach as a generic framework for SMC solutions. Interestingly, we show that the *predominant* approach, consisting in directly secret-sharing the input data, can be seen as specific instance of this framework.

We demonstrate how the characteristics of the distributed homomorphic cryptosystem determine the efficiency of the corresponding framework instance. For this purpose, we introduce a distributed version of the Brakerski-Fan-Vercauteren homomorphic cryptosystem [14] that, based on the *Ring Learning With Errors* (R-LWE) assumptions, supports computation across trust boundaries. We make the following contributions:

- We introduce the concept of *trust boundary* and reformulate the secure multiparty computation problem in terms of this new primitive (Section III).
- We formulate a generic SMC framework based on distributed homomorphic encryption and show both its conceptual and technical relevance (Section IV).
- We propose a distributed version of the BFV scheme [14], as a concrete realization of the general SMC framework (Section V).
- We analyze the effect of transitioning from a centralized to a distributed cryptosystem in terms of noise growth (Section VI).
- We provide security proofs for the protocols that define the distributed cryptosystem (Section VII).
- We implement our scheme and provide an evaluation of its core functions (Appendix A).

In this paper, we bridge a gap between the SMC research community and its application domains, by providing a framework that is extensible, flexible, and strongly consistent across its abstraction levels.

## II. SMC Background

In the last decade, a substantial amount of concrete software solutions to SMC problem instances were proposed [13], demonstrating the evolution of a consolidated, mostly theoretical field into an applied one. Oblivious-transfer based solutions, such as *garbled circuits*, are a good example of ideas that date back to the late eighties [18]–[20] and that are now applied in concrete, usable frameworks [21], [22]. Garbled circuits enable two parties to privately compute an arbitrary function over their inputs but are not directly applicable, in an efficient way, to the general $N$-party case.

We identify and briefly introduce three main categories or approaches in $N$-party SMC solutions that have emerged as outcomes from cryptographic research. These solutions are often perceived and positioned as diametrically different, which might stem from the lack of a common qualitative comparison basis. Therefore, we examine their actual differences when considering them as instances of our framework, in Section IV-E.

*(a) Data-Level Secret Sharing:* The first and predominant category of techniques, sometimes viewed as the de-facto approach when referring to SMC solutions, consists in applying *secret-sharing* [17] to input data. Archer et al. surveyed state-of-the-art systems based on this approach [23] and their concrete applications. In these approaches, the evaluation of arithmetic circuits is generally enabled by the homomorphism of the secret-sharing scheme [24], or by making use of protocols based on multiplication-triplets protocols [25] when no such homomorphism exists for the multiplication. These techniques are computationally efficient, but they impose a significant overhead on the network, as the execution of an interactive protocol is required for each multiplication gate in the circuit. Moreover, the triplet-based multiplication protocol requires prior distribution of one-time triplets, which can be performed either by a trusted third-party or by the parties themselves. Interestingly, the latter can also be formulated as a *smaller* SMC problem, which demonstrates its recursive nature.

*(b) Multi-key Encryption Schemes:* Unlike the previous approach, categories (b) and (c) both make use of an homomorphic scheme to encrypt and exchange the computation data, which can then be operated on, in a non-interactive way, by using encrypted arithmetic. The idea of homomorphic encryption as a means of reducing the volume of interaction in SMC can be traced back to a work by Franklin and Haber [26], later improved on by Cramer et al. [27]. However, at the time, the lack of homomorphic schemes that preserve two distinct algebraic operations ruled out complete non-interactivity at the evaluation phase. This changed in 2009, when Gentry proposed the first *fully homomorphic* encryption (FHE) scheme [28]. The main difference between approaches (b) and (c) lies in the key(s) under which the computation is performed, and thus in their decryption procedure.

The approach (b) uses a *multi-key* homomorphic encryption scheme, as introduced by López-Alt et al. [29]. It enables the parties to provide their input data encrypted under their own locally generated key, and to operate on them directly (in a non-interactive way). As a consequence of the scheme being both message-homomorphic and key-homomorphic, the computation result is encrypted under an *on-the-fly* joint key, thus requiring the collaboration only between the involved parties at the decryption phase. Although conceptually appealing, the *on-the-fly* feature of multi-key HE comes at prohibitive performance cost, both in terms of time and space, limiting its applicability in real use-cases. Notably, current multi-key schemes lack the compactness property: the size of their ciphertexts and the timing of arithmetic operations are, respectively, linear and quadratic in the number of keys in the input. Furthermore, they require significant interactive pre-computation of potentially large keys [30], [31] (usually growing quadratically or cubically in the number of parties). Hence, multi-key approaches, although conceptually appealing, are not yet practical for many concrete SMC scenarios.

*(c) Distributed Encryption Schemes:* The third SMC approach uses only one encryption key that is secret-shared among the parties. The decryption of the input and output data then requires a collaboration between the parties. We use the term *distributed cryptosystem* when referring to this construction. This is the idea behind the joint line of work by Asharov et al. [32], [33] and López-Alt et al. [34] that is highly related to our technical contribution. These authors construct a distributed cryptosystem based on the *Learning With Errors* (LWE) problem [35], along with the SMC solution it enables. Unfortunately, this whole line of work could not find its echo in applications, and has remained, until now, of only theoretical interest. One possible reason is the lack, at the time, of available and practical LWE-based scheme implementations. This would have required system designers to become familiar with the fairly recent and fast-moving research field of lattice-based cryptography. Today, however, multiple ongoing efforts aim at standardizing homomorphic encryption [36] and at making it available to a broader public [37]–[40]. This new generation of homomorphic cryptosystems is mostly based on the hardness of the *Ring Learning With Errors* (R-LWE) problem [41]. Interestingly, one of the most important improvements brought to the approach (a), known as the SPDZ protocol [42], actually relies on the approach (c) to solve the Beaver-triples generation subproblem.

On the system research side, we notice that, in order to benefit from the SMC security guarantees and to cope with the protocols overhead, many proposed system models increase (*trust splitting*) or reduce (*trust delegation*) the number of parties to two [5], [23], [43], [44]. This is often achieved at the cost of introducing auxiliary actors (e.g., *two-cloud* model [5], [45] or decryption proxies [46]), new assumptions (e.g. non-collusion), and complexity in the trust models of these systems, thus rendering them less realistic and difficult to integrate, hence severely harming adoption. However, these solutions have, so far, remained predominantly based on approach (a). [23].

We argue that, on the other hand, approach (c) deserves more than theoretical interest. Not only does its evolution directly benefit

SPDZ-like solutions, which are still extensively studied [47], [48], it also enables scenarios where an interactive circuit evaluation does not fit the system model (e.g., parties going offline during evaluation), or cannot be supported by the network capacity. Additionally, we argue that approach (c) is conceptually natural in solving SMC problem instances, which we now demonstrate throughout Sections III and IV.

## III. Trust Boundaries

In this section, we introduce the concept of *trust boundaries* as a primitive for modeling the trust setting in systems that handle sensitive data. We focus the definition of this concept to the context of the SMC problem so that we can rely on the introduced terminology and abstractions throughout the rest of this work. We show how this leads to a novel SMC problem formulation: computing across trust boundaries.

### A. *Tracing Trust boundaries*

We define a trust boundary as the perimeter of an *information domain* that is enforced by some access control mechanism. We refer to the delimited information domain as a *trust domain*. The first and most natural trust boundary is physical: In theory, it is possible to control the entirety of a system's information-domain by controlling its inputs and outputs. However, such physical compartmentalization is not possible in today's fundamentally inter-connected systems, where data might need to transit through, or be processed in, untrusted environments. In this work, we are mainly interested in cryptography, that offers a logical way of extending a trust domain to these untrusted environments.

We can consider the ciphertext space of an encryption scheme as an information domain, whereas the knowledge of the secrets parameterizing the scheme represents the necessary condition to read information from that domain. Thus, the key generation algorithm of this scheme creates a trust domain for which the access conditions (i.e., the trust boundary enforcement) are specific to the cryptosystem and its parameters. For example, the ciphertext space defined by a symmetric encryption scheme instantiates a trust boundary for which read and write access both require the knowledge of a secret key. In other words, multiple parties willing to read/write data from/to this trust domain need to trust each other in the complete enforcement of its boundary. Asymmetric encryption, however, enables further granularity, in the sense that a secret-key holder can derive a public *gadget* (the *public key*) that grants its recipient the write operation to the trust domain. These gadgets must not leak the secrets on which the boundary enforcement is based on; i.e., they should not grant read access to the trust domain. For homomorphic cryptosystems, these gadgets are not limited to enabling encryption, but can also enable evaluation of arithmetic circuits, through the so-called *evaluation keys*, directly on the encrypted data.

Although describing trust models by ownership of cryptographic keys can be adequate for simple systems, it quickly becomes difficult to reason about when considering schemes with complex encryption or decryption structures such as distributed, attribute-based or even multi-key cryptosystems. In the context of secure multiparty computation, a trust domain also needs to provide evaluation capabilities, which can make this trust model even more complex. In data-level secret sharing approaches, the actions of writing, evaluating and reading in trust domains are expressed in a different way. The trust boundary concept enables reasoning about a system's trust model by abstracting away the specifics of the cryptographic primitive (or even the protection mechanism) in use. Therefore, we propose to trace the trust boundaries as a preliminary step in designing a multiparty system; only then should cryptographic protocols (or other mechanisms) be defined and shown to be enforcing the desired trust boundaries. Picturing the boundaries of existing or work-in-progress designs can nevertheless be useful for exposing their potential weakest-links or undesirable trust relationships, and can also unveil instances of the SMC problem.

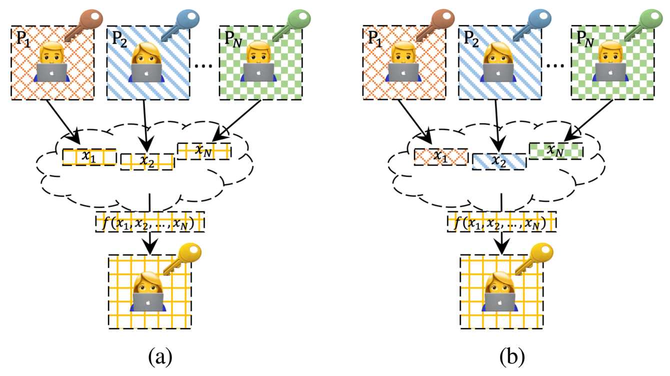

Fig. 1: Van Dijk and Juels' classes of private cloud computing applications [16]: (a) depicts the single-client class, where only one trust domain is involved in the computation, and (b) shows the multi-client class, which contains the solutions to SMC problems.

Note that, although this work focuses on cryptography, the trust boundary concept is flexible enough to model other data protection mechanisms such as digital signatures and differential privacy.

### B. *Computing across Trust Boundaries*

In a different context and formulation, Van Dijk and Juels introduced a classification of private cloud-computing applications [16], by defining two classes: *single-client* and *multi-client*. They consider a *honest-but-curious* cloud provider that, provided with the encrypted input data of its multiple clients, is tasked to homomorphically carry out the computation of some function on these data, and output the result. Their functionality and privacy requirements, although formulated for secure cloud-computing, are the requirements of a secure two- and multi-party computation problem. We can characterize these classes in terms of cross-trust boundary computation, as illustrated in Figure 1. In the first class (Figure 1a), the cloud is tasked to compute a function where both input and output lie in the same trust domain: that of the computation result receiver. This is a typical and straightforward use of homomorphic encryption, as it implies only one trust domain in which the clients have to provide their input data, by encrypting them with the domain's public key. These encryptions can be safely provided to the computing party (the cloud), which can obliviously evaluate the required function and output the result to its intended recipient. However, this is insufficient to fulfill the input privacy requirement of SMC: A collusion between the cloud provider and the recipient of the result (which is part of the passively-secure SMC threat model) would enable decryption of the input ciphertexts. The second class (Figure 1b), which is hypothetical, would permit the clients to provide their inputs to the cloud encrypted under their own key. The cloud, without interacting with the clients any further, would output the result in the trust boundary of its intended recipient. Hence, this class would contain applications that could solve SMC problem instances using a single centralized computer. However, this class is shown to be theoretically impossible to realize in a general way, even with access to an hypothetical, perfect, homomorphic scheme. As the emulation of a single computing node is the target of any SMC solution, the conclusions of Van Dijk and Juels are pivotal. Using our terminology, they conclude that, for a majority of useful functionalities, achieving SMC requires the parties to have access to a *commonly trusted information domain*.

We extend this result by means of our framework, described in Section IV, and we argue that, in the absence of a *physical* common trust-domain, the involved parties have to instantiate and enforce a *logical* trust-domain, for which the trust boundary provides sufficient guarantees. These guarantees can be given to a party only when a trust-boundary crossing of data is formulated as an interactive protocol in which the party chooses whether to participate or not; indeed, any non-interactive kind of access control would trivially realize the impossible multi-client class.

The actions of instantiating and enforcing trust boundaries have a particular conceptual meaning for privacy, and this meaning is well conveyed by our new formulation. Ultimately, the proposed abstraction makes an SMC problem easier to isolate, and makes its solution easier to integrate in larger designs.

## IV. SMC Framework

We introduce a novel framework in which various SMC solutions, in particular the approaches (a), (b) and (c) described in Section II, can be modeled and compared. At the center of this framework is the tight, yet somewhat misunderstood, connection between homomorphic encryption and the SMC problem, that we tackle in this section: We first formalize the SMC problem and its parameters in Section IV-A. We define distributed cryptosystems (Section IV-B) and the basic framework for SMC solutions these cryptosystems enable (Section IV-C). Then, we discuss how this framework is adapted depending on the features offered by the cryptosystem (Section IV-D), and how this can unify the existing approaches to SMC solutions (Section IV-E).

### A. The SMC Problem

Our approach considers a problem-centric formulation of secure multiparty computation, which we view as particularly suited to capture its general and recursive nature.

**Definition 1.** Let $\mathcal{P} = \{P_1, P_2, \dots, P_N\}$ be a set of $N$ parties respectively holding inputs $(x_1, x_2, \dots, x_N)$ and let $\mathcal{A}$ be an adversary, which can have a non-empty intersection with $\mathcal{P}$. Let $f(x_1,x_2,...,x_N) = (y_1,y_2,...,y_N)$ be a function (*ideal functionality*) over the parties' inputs, where $y_i$ denotes the output for party $P_i$.
The *Secure Multiparty Computation Problem* consists in computing $f(x_1,x_2,...,x_N)$, while preserving specific security properties in the presence of $\mathcal{A}$.

The solution to an SMC problem is a protocol $\pi_f$ realizing the problem's ideal functionality $f()$. We observe that Definition 1 intentionally leaves aside the concrete definition of $\mathcal{A}$, and, therefore, the nature (e.g., set of corrupted parties, external eavesdropper,...) and behavior (e.g., active, passive, rational,...) of the adversary. Whereas the sough security properties (e.g., correctness, input/output privacy,...) will define the control that an acceptable solution should enact on the inputs/outputs (e.g., exact/approximate outputs, zero/limited leakage of inputs and outputs).

In this work, we instantiate this problem and consider a semi-honest adversary $\mathcal{A}$ that is assumed capable of statically corrupting up to $N-1$ parties (we slightly abuse notation by using $\mathcal{A} \cap \mathcal{P}$ also as the set of corrupted parties). Our security property is that of a cryptographic approach: it states that $\mathcal{A}$ must learn *nothing* more about $\{x_i,y_i\}_{P_i \notin \mathcal{A}}$ than what can be deduced from its own input and output $\{x_i,y_i\}_{P_i \in \mathcal{A}}$. Equivalently, it requires that the execution of $\pi_f$ emulates an ideal setting where parties are provided with an incorruptible environment that, given their inputs, will carry out the computation of $f$ and that will provide them with their respective outputs. This naturally translates into our trust-boundary formulation, by seeing each party input as belonging to the party's trust domain. The functional and security requirements come from Van Dijk and Juels' result: The ideal scenario assumes the existence of a commonly trusted trust-domain that the real system needs to emulate.

Definition 1 does not specify any system or network model, so we define one for the scope of this work. We assume that the parties have access to a reliable and authenticated broadcast channel to the other parties in $\mathcal{P}$. The parties are assumed to have access to a uniformly random *Common Reference String* (CRS) [49].

### B. Distributed Homomorphic Cryptosystems

A *distributed* (or *threshold*) cryptosystem $E^S$ securely splits the secret key of a cryptosystem $E$ among a group of $N$ parties, according to a secret-sharing scheme $S$ [50]. Hence, operations interacting with this secret key are formulated as multiparty protocols in the distributed scheme. Definition 2 formalizes this intuition, using our terminology, for an asymmetric encryption scheme.

**Definition 2.** Let $E = (\texttt{SecKeyGen}, \texttt{PubKeyGen}, \texttt{Enc}, \texttt{Dec})$ be an encryption scheme and let $S = (\texttt{Share}, \texttt{Combine})$ be an $N$-party secret sharing scheme.
The associated *distributed cryptosystem* is defined as the tuple $E^S = (\pi_{\texttt{SecKeyGen}}, \pi_{\texttt{PubKeyGen}}, E.\texttt{Enc}, \pi_{\texttt{Dec}})$ of multiparty protocols for which the ideal functionalities for party $P_i$ are

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

$$f_{i,\pi_{\texttt{PubKeyGen}}}(\texttt{sk}_1, \texttt{sk}_2, \dots, \texttt{sk}_N) = E.\texttt{PubKeyGen}(\widetilde{\texttt{sk}}),$$

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

where $\widetilde{\texttt{sk}} = S.\texttt{Combine}(\texttt{sk}_1, \dots, \texttt{sk}_N)$ is the ideal secret key: the operations that depend on it are implemented in the real world as SMC protocols.

A direct consequence from Definition 2 is that $E$ and $E^S$ are *compatible*: secret-sharing a valid secret key for $E$ results in a valid instantiation of $E^S$ that can decrypt ciphertexts produced by $E$. Conversely, reconstructing the ideal secret key of $E^S$ yields a valid secret key for $E$ that can decrypt ciphertexts of $E^S$. Additionally, the output of the $\pi_{\texttt{PubKeyGen}}$, denoted $\texttt{cpk}$ or *collective public key*, is a valid public key for $E$. We rely on this property in our general SMC protocol in Section IV-C.

Since the $S.\texttt{Combine}$ operation has to be embedded in $f_{\pi_{\texttt{Dec}}}$, the secret-sharing scheme defines an access structure [17] that directly characterizes the access structure of $E^S$. For example, using additive secret-sharing of the secret key results in a scheme where all parties must collaborate to decrypt a ciphertext (i.e., to *use* the secret key), whereas only a *threshold* number of them would be required when using Shamir secret-sharing [17].

In this work, we consider distributed homomorphic cryptosystems[1], so we augment $E$ and $E^S$ with the $\texttt{Eval}$ procedure that enables encrypted arithmetic on their ciphertexts. Hence, by using this augmented scheme

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

[1]We use additive secret-sharing of the keys, so the threshold is set to $N$. Hence, we use the term *distributed* cryptosystem to avoid ambiguities with respect to the more general *threshold* cryptosystems.

the parties are able to emulate an ideal environment where $E$ can be used to encrypt and operate on data, whereas decryption is achieved through collaboration, according to the access structure defined by $S$.

In terms of trust boundaries, the formulation is very intuitive: By relying on $E^*$, the parties are able to instantiate a collectively enforced trust boundary where computation is allowed. We show next that realizing $E^*$ is theoretically sufficient to solve any instance of the SMC problem. In Section V, we construct a distributed version of the Brakerski-Fan-Vercauteren somewhat-homomorphic encryption scheme [14], where the secret key is additively shared among the parties.

### C. *Generic SMC Protocol Overview*

Protocol 1: **`Generic_SMC`**

**Public input**: $f$
**Private input**: $x_i$ for each $P_i \in \mathcal{P}$
**Output** for $P_i$: $y_i = f(x_1, x_2, ..., x_N)$

1) The parties create a jointly enforced, publicly writable trust boundary
$$\mathtt{sk}_i = E^*.\pi_{\mathtt{SecKeyGen}}(\lambda),$$
$$\mathtt{cpk} = E^*.\pi_{\mathtt{PubKeyGen}}(\mathtt{sk}_1, ..., \mathtt{sk}_N),$$
2) each $P_i$ writes its input in the trust boundary as
$$c_i = E^*.\mathtt{Encrypt}(\mathtt{cpk}, x_i),$$
3) any party can compute the required encrypted outputs
$$(c'_1, c'_2, ..., c'_n) = E^*.\mathtt{Eval}(f, c_1, c_2, ..., c_n),$$
4) for each $c'_i$, the parties execute the decryption protocol
$$y_i = E^*.\pi_{\mathtt{Dec}}(\mathtt{sk}_1, ..., \mathtt{sk}_N, c'_i).$$

We introduce the generic SMC protocol arising from the definition of the distributed cryptosystem $E^*$. First, we present the protocol as defined in [27], [33], but formulated using our trust boundaries abstraction. Then, we explain how this protocol can be adapted based on the characteristics of its underlying cryptosystem $E^*$. By following this approach, we achieve a two-fold effect: We unify seemingly different SMC techniques, and we make the framework more flexible from an application point of view.

Protocol 1 describes the generic SMC protocol, and Figure 2 illustrates its trust-boundary representation. Step 1 is a setup phase that instantiates the common trust boundary; to do so, the parties run the $\pi_{\mathtt{SecKeyGen}}$ protocol to create a ciphertext space for which each party knows only one share $\mathtt{sk}_i$ of the ideal secret key $\widetilde{\mathtt{sk}}$. Then, they use the $\pi_{\mathtt{PubKeyGen}}$ protocol to derive a collective public key $\mathtt{cpk}$ associated to the ideal secret key. Given a choice of parameters and a set of players $\mathcal{P}$, this step is independent of the rest of the protocol, as it has to be run only once for a given set of parties (the case of dynamic sets is addressed by our extended framework). Hence, the produced key is not limited to one function evaluation, as garbled circuits would be. The complexity of this step does not depend on the number of circuit evaluations to be performed, contrarily to the Beaver-triples generation, but only on the circuit complexity of $f$. This is because $E^*$ has to be instantiated with a scheme parameterized with sufficient homomorphic capacity to support the evaluation of the circuit.

In Step 2, the parties use the public (i.e., non-interactive, due to the existence of the public key $\mathtt{cpk}$) encryption procedure of $E^*$ to write

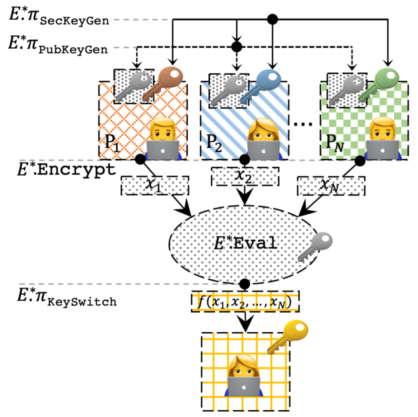

Fig. 2: Trust-boundary based representation of the extended `General_SMC` protocol, where a distributed homomorphic cryptosystem $E^*$ is used by the parties to instantiate a common trust domain with evaluation capabilities.

data within the trust domain. Then, the parties can send the encrypted input values to the entity/entities performing the next step (evaluation).

Step 3 consists in the evaluation of the ideal functionality, by exploiting the homomorphic property of the scheme to carry out the computation of the required encrypted outputs. This step is non-interactive and can be performed by any entity, even outside of $\mathcal{P}$. Consequently, expensive homomorphic computations can be outsourced to an untrusted (yet semi-honest) infrastructure, such as a cloud-computing provider. Hence, this feature opens up this SMC solution to a broader variety of computing scenarios requiring input privacy (e.g., smart contracts evaluation on sensitive, multiparty inputs). Note that this step illustrates how the *trust domain* concept brings this general SMC protocol *conceptually closer* to the ideal-world scenario where parties receive help from a trusted third party.

Step 4 enables the parties to decrypt their respective outputs. Indeed, this requires collaboration according to the access structure defined by the sharing of the ideal secret key. Hence, when the outputs are different for each party, this phase corresponds to $N$ runs of the $\pi_{\mathtt{Dec}}$ multiparty protocol, each of them outputting $y_i$ to $P_i$ (and nothing to the other parties). This can be considered as collectively authorized read operations from the common trust-domain.

We highlight that, if $E^*$ is a distributed homomorphic encryption scheme as defined in Section IV-B, the `Generic_SMC` protocol is nearly optimal in terms of communication, as only those steps related to trust boundary creation and crossing (steps 1 and 4) require interaction. Further message exchanges might be required by the sought functionality itself, and thus are in exact correspondence with those of an equivalent insecure system not using encryption. Hence, this general protocol confines the need for interaction to the actual problem it solves: the protection of a trust domain.

### D. *Extended General SMC Protocol*

We discuss how the `Generic_SMC` protocol can be adapted according to the feature set provided by $E^*$. More precisely, we discuss how *more interaction* can *compensate* for cryptosystems that do not provide a public encryption algorithm (1) and/or homomorphic arithmetic covering the sought functionality (2). We also discuss how the *outsourced* SMC scenario is modeled within our framework (3).

*1) Interactive vs. Public Encryption:* For cryptosystems that have no public key (i.e., symmetric schemes), the encryption algorithm of Step 2 must also be replaced by a secure multiparty protocol. Such an encryption protocol can also be used when the message to be encrypted is secret-shared among the encrypting parties, thus providing a way to transform a secret-shared value (e.g., from an SMC solution using approach (a) in Section II) in a secure way, in order to process it under homomorphic encryption.

*2) Interactive vs. Public Evaluation:* Non-interactive processing in Step 3 is only possible if $E^*$ supports the sufficient homomorphic arithmetic to execute the required functionality. When this requirement is not completely met, interaction can be used to *compensate* for it. That is the case, for example, of the well-known multiplication protocol for additive homomorphic schemes described in [27]. This observation eventually relates the need for interaction during Step 3 to the cryptosystem's inability to keep the dimension of its key-space constant throughout the function evaluation: In Cramer's protocol example, the product between two encryptions in an additively homomorphic cryptosystem could be interpreted as a set of encryptions in an increased key space, where the original keys are combined with additional ephemeral secrets (which can only be removed by Cramer's interactive protocol). In other words, the need of interaction in Step 3 is driven by the ability (commonly referred to as *relinearization*) of $E^*$ to evaluate parts of its decryption function homomorphically, for the higher dimensions of the key-space that result from arithmetic operations. This idea is illustrated in Section IV-E for the case of traditional data-level secret sharing, and in Section V-E for the specific example of our distributed BFV scheme.

*3) Outsourced SMC:* It is common to find SMC propositions where parties holding the input data delegate the SMC protocol execution to a different, usually smaller, set of nodes. These scenarios, often referred to as *outsourced* SMC settings [23], [51], [52] differentiate between the parties that provide the inputs (*input parties*) and those that evaluate the function and output its result (*computing parties*). This is relevant from a performance point of view, as it enables the use of SMC solutions that would not scale to a large number of parties. Also, it enables system models where the input parties can go offline after providing their inputs, and do not need to be themselves capable of running secure computation. However, such delegation of SMC tasks introduces the new assumption that the computing parties will not collude (in attacking the input parties' data), hence that they have to be collectively trusted by the input parties (consistently with Van Dijk and Juels' impossibility result). Thus, in terms of *trust model*, introducing the role of *input party* does not make the framework more general; conversely, it simply introduces a new assumption that indeed reduces generality.

*4) General Trust-Boundary Crossing:* Another common SMC scenario considers an output party (the *receiver*) that receives the computation result, without providing any private input. This scenario could also be viewed as a larger SMC problem by extending the set of parties with the receiver (which would have the empty bit-string as input). In the simple SMC framework, the receiver would then hold a share of the collective secret key that would thus involve her in the decryption protocol execution. However, such solution involves the receiver in the enforcement of a trust boundary in which she wrote no data.

A more appropriate solution would be to enable the computing parties to transfer the computation output from their collectively-enforced trust domain directly to the receiver's trust-domain. In cryptographic terms, we require the cryptosystem $E^*$ to provide a *key switching* protocol $\pi_{\texttt{CKS}}$, which enables the parties to re-encrypt a ciphertext that originally decrypts under a shared secret key $\texttt{sk}$ into a new ciphertext that decrypts under the receiver's secret key $\texttt{sk}'$. This protocol generalizes $\pi_{\texttt{Dec}}$; i.e., decryption (or the ability to decrypt) maps to the particular case of setting $\texttt{sk}'=0$ (or any publicly known value). Given that $\texttt{pk}'$ is a public key for the secret key $\texttt{sk}'$, the ideal functionality of key switching,

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

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

Consequently, by considering general trust boundary crossing in our extended framework, we make it more general (from the trust-model point of view) than the simple one. This has technical benefits too, as the computing parties can now execute the $\pi_{\texttt{CKS}}$ protocol and, at a later point in time, the receiver can come online and retrieve her result with only a single interaction with a computing party holding the key-switched ciphertext. Our concrete distributed cryptosystem proposition (see Section V) features such general trust boundary crossing, thus enabling outsourced SMC scenarios.

## *E. Unifying SMC Solutions*

As we highlighted in Section II, solutions based on data-level secret-sharing (approach (a) in Section II) and those based on homomorphic encryption (approaches (b) and (c)), are usually seen as diametrically different. But they could, in fact, be constructed as specific instances of our framework. We illustrate that by discussing (1) how our framework is instantiated with a multi-key homomorphic scheme and (2) how data-level secret sharing can be seen as a special case of this instantiation. Explicitly representing garbled-circuits solutions is also a promising extension of this work.

*1) Multi-key Cryptosystems:* A multi-key homomorphic cryptosystem enables the parties to send their input encrypted under their own locally generated key. Hence, Step 1 in protocol 1 becomes non-interactive. This would be conceptually ideal from a system design point of view, but comes with prohibitive costs for all existing multi-key schemes [30]: (i) The multi-key ciphertext length grows linearly or quadratically with the number of parties $N$. (ii) Multi-key arithmetic operations require evaluation keys whose size is quadratic or cubic in $N$. (iii) The homomorphic capacity decreases exponentially in $N$ [29], limiting practical use of these schemes.

Hence, while multi-key schemes are of high theoretical relevance, we argue that they are still *too costly for what they bring* to concrete practical applications. We underline that, in fact, this framework instantiation only delays the complexity of establishing the necessary common trust-domain to the trust-boundary crossing step (Step 4), at the cost of a more complex decryption function.

*2) Data-level Secret Sharing:* SMC solutions based on secret-sharing of the data (approach (a)) can be expressed as instances of our framework. In the scope of this work, we provide a concise and thus relatively informal sketch for this reduction for an additive secret-sharing scheme over $\mathbb{Z}_q$.

We proceed by instantiating our framework with $E^*$ being a secret-shared one-time-pad. Let $m \in \mathbb{Z}_q$ be a message and $k \in \mathbb{Z}_q$ be a one-time key, both known to party $P_1 \in \mathcal{P}$. The ciphertext $c = m + k \mod q$ can be disclosed and operated on in a multi-key fashion, provided that a multiparty decryption protocol exists. Such decryption protocol will require the key $k$, and all other one-time-keys involved in the computation, to be known collectively by the parties, in order to allow trust-boundary crossing. This is achieved by secret-sharing this key, as

$$S = \{k_1, k_2, ..., k_N\} \text{ s.t. } m = c + \sum_S k_i \mod q,$$

which is equivalent to the sharing $S' = \{c + k_1, k_2, ..., k_N\}$ of $m$, where $P_1$ simply stores his share as its partial decryption of $c$, instead of storing them both separately. Thus, sharing $S'$ corresponds to a *traditional* sharing of the value $m$.

Due to the one-time key nature of $E^*$, each ciphertext is encrypted under a new key, which makes this approach inherently multi-key. This instantiation, although computationally efficient due to its simple integer arithmetic, suffers from heavy bandwidth requirements needed to compensate for its lack of homomorphic properties. More specifically, the used $E^*$ is only additively homomorphic: decryption of a sum requires the sum of the involved keys (one element of $\mathbb{Z}_q$ per party). However, no public relinearization key exists for the on-the-fly key that results from a product operation, which requires interactive relinearization to be performed using Beaver's protocol. This yields an interesting analogy between the pre-computed evaluation keys in multi-key schemes and the Beaver-triples *reserve* in SPDZ-like approaches.

### F. Discussion

We have shown the central role of the distributed encryption scheme within the proposed SMC framework, and highlighted several properties that are crucial for this scheme to realize an efficient SMC solution. In particular, $E^*$ should enable:

- public-key generation,
- two distinct encrypted arithmetic (ring) operations,
- public (non-interactive) ciphertext-space relinearization,
- general and efficient trust-boundary crossing.

Additionally, from a system model perspective, these properties have to scale to realistic numbers of parties $N$ (at least up to the scale at which the secret key access structure still represents realistic scenarios). By deliberately giving away the multi-key feature, we require parties to interact in order to create a commonly enforced trust boundary. However, as we show in Appendix A, this interaction can be very efficient, and it can substantially reduce the complexity of the SMC problem posed by the decryption (i.e., the delivery) of the result.

## V. Distributing the BFV Scheme

In this section, we introduce a distributed version of the Brakerski-Fan-Vercauteren (BFV) cryptosystem [14] that features the properties of Section IV-F. Hence, it can be used to instantiate the SMC framework described in Section IV. The BFV cryptosystem is a Ring Learning with Errors (R-LWE) [41] scheme that supports both additive and multiplicative homomorphic operations. Due to its practicality, it has been implemented in most of the emerging lattice-based cryptographic libraries [37], [39], [40] and is also included as part of the current standardization effort of homomorphic encryption [36]. It is worth noting that, although formulated for the BFV scheme, the introduced protocols can be straightforwardly adapted to other R-LWE-based cryptosystems, such as BGV [53], or the more recent CKKS [54], that enables homomorphic approximate arithmetic.

### A. 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. When applied to polynomials, these operations are performed coefficient-wise. We use regular letters for integers and polynomials; and bold letters for vectors of integers and of polynomials. $\boldsymbol{a}^T$ denotes the transpose of a vector $\boldsymbol{a}$. Given a probability distribution $\alpha$ over a ring $R$, $a \leftarrow \alpha$ denotes the sampling of an element $a \in R$ according to $\alpha$, and $a \leftarrow R$ denotes uniform sampling in $R$. For a polynomial $a$ we denote $||a||$ its infinity norm.

TABLE I: BFV Parameters

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

TABLE II: BFV Symbols

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

Scheme 1: **BFV**

**BFV.SecKeyGen**$(1^\lambda)$: Sample $s \leftarrow R_3$. Output: $\mathtt{sk} = s$

**BFV.PubKeyGen**$(\mathtt{sk})$:
Let $\mathtt{sk} = s$. Sample $p_1 \leftarrow R_q$, and $e \leftarrow \chi$. Output:

$$\mathtt{pk} = (p_0, p_1) = (-(sp_1 + e), p_1)$$

**BFV.RelinKeyGen**$(\mathtt{sk}, \mathbf{w})$:
Let $\mathtt{sk} = s$. Sample $\mathbf{r}_1 \leftarrow R_q^l$, $\mathbf{e} \leftarrow \chi^l$. Output:

$$\mathtt{rlk} = (\mathbf{r}_0, \mathbf{r}_1) = (s^2\mathbf{w} - s\mathbf{r}_1 + \mathbf{e}, \mathbf{r}_1)$$

**BFV.Encrypt**$(\mathtt{pk}, m)$:
Let $\mathtt{pk} = (p_0, p_1)$. Sample $u \leftarrow R_3$ and $e_0, e_1 \leftarrow \chi$. Output:

$$\mathtt{ct} = (\Delta m + up_0 + e_0 \,,\, up_1 + e_1)$$

**BFV.Decrypt**$(\mathtt{sk}, \mathtt{ct})$:
Let $\mathtt{sk} = s$, $\mathtt{ct} = (c_0, c_1)$. Output:

$$m' = [\lfloor \frac{t}{q}[c_0 + c_1 s]_q \rceil]_t$$

**BFV.Add**$(\mathtt{ct}, \mathtt{ct}')$:
Let $\mathtt{ct} = (c_0, c_1)$ and $\mathtt{ct}' = (c_0', c_1')$ Output:

$$\mathtt{ct}_{\text{add}} = (c_0 + c_0', c_1 + c_1')$$

**BFV.Multiply**$(\mathtt{ct}, \mathtt{ct}')$:
Let $\mathtt{ct} = (c_0, c_1)$ and $\mathtt{ct}' = (c_0', c_1')$ Output:

$$\mathtt{ct}_{\text{mul}} = [\lfloor \frac{t}{q}(c_0c_0' \,,\, c_0c_1' + c_0'c_1 \,,\, c_1c_1') \rceil]_q$$

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

$$\mathtt{ct}_{\text{relin}} = (c_0 + \sum_{b=0}^{l} \mathbf{r}_{0,b} c_2^{(b)} \,,\, c_1 + \sum_{b=0}^{l} \mathbf{r}_{1,b} c_2^{(b)})$$

### B. The Brakerski-Fan-Vercauteren Encryption Scheme

We first recall the original (centralized) BFV encryption scheme [14], in its most common instantiation. 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 class modulo $q$. Unless otherwise stated, we consider the arithmetic in

$R_q$, and polynomial reductions are omitted in the notation. The relinearization operation requires an intermediary base $w < q$, in which ciphertexts are temporarily decomposed. We write $l = \lceil \log_w(q) \rceil$ and $\mathbf{w} = (w^0, w^1, ..., w^l)^T$.

Let the plaintext space be the ring $R_t = \mathbb{Z}_t[X]/(X^n+1)$ for $t < q$. Messages are elements in $R_t$, and the encrypted arithmetic operations preserve the plaintext arithmetic. We denote $\Delta = \lfloor q/t \rfloor$, the integer division of $q$ by $t$.

In practical instantiations of its normal form, 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 R-LWE error distribution $\chi$ over $R_q$ has coefficients distributed according to a centered discrete Gaussian with standard deviation $\sigma$ and truncated support over $[-B,B]$.

The security of BFV is based on the hardness of the R-LWE problem [41], that is informally stated as follows: Given a uniformly random ring element $a \leftarrow R_q$, 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 $b \leftarrow R_q$.

Table I summarizes the cryptosystem parameters, and Table II summarizes the various symbols used in the encryption scheme formulation. The cryptosystem operations are detailed in Scheme 1. We can map this set of functionalities to the formulation introduced in Section III: The trust-domain instantiation comprises three procedures: `BFV.SecKeyGen` creates the trust-boundary across which public writes are enabled by the output of `BFV.PubKeyGen`. The `BFV.Add` and `BFV.Multiply` operations permit public, non-interactive evaluation of functions within the delimited trust domain. Public relinearization is enabled by the `BFV.Relinearize` operation and its associated *relinearization key* (`rlk`). We use the more specific *relinearization key* term instead of *evaluation key*: This is to emphasize that, even though the multiplication operation produces a ciphertext in $R_q^3$, decryption of higher-dimensional ciphertexts is, at some cost, still possible (similarly as in multi-key schemes).

In the BFV scheme, decryption can be seen as a two-step process. The first and main step performs the trust boundary crossing and requires the secret key to compute

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

where $e_{\mathtt{ct}}$ is the ciphertext overall error, or *ciphertext noise*. In the second step, the message is decoded from the noisy term by rescaling and rounding

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

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

### C. The Distributed BFV Scheme

We construct a distributed cryptosystem based on the BFV scheme. We use additive secret-sharing to distribute the BFV secret key, denoted as $s$ in what follows, among the $N$ parties in $\mathcal{P}$. We denote $s_i$ the secret key share of party $P_i$, thus

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

Hence, ciphertexts in our scheme can be decrypted only through the collaboration of all parties. Our scheme hence tolerates up to $N-1$ colluding corrupted nodes in the passive adversary model.

To emphasize its purpose, we refer to the original *centralized* scheme as *the ideal scheme*: the ideal centralized functionality that needs to be emulated in a distributed 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. We reformulate the operations of the original BFV scheme that involve this key into secure $N$-party protocols.

*1) Ideal-Secret-Key Generation:* We propose a simple ideal-secret-key generation procedure, in which each party independently samples its own share as $s_i = \mathtt{BFV.SecKeyGen}(1^\lambda)$. Thus, the ideal secret-key is generated in a non-interactive way. Observe that, although Eq. (3) applies, this is not a *usual* sharing of $s$, in the sense that the shares' distribution is not uniform in $R_q$. This is not a concern because, as discussed in Section VII, the security of our scheme does not rely on this property. It might be possible to design more complex protocols resulting in *proper* additive shares of $s$ in the original ternary key-distribution. These would be compatible with our protocols and would represent interesting extensions.

*2) Collective-Key Generation (`CKG`):* The Collective Key Generation, detailed in Protocol 2, emulates the `BFV.PubKeyGen` procedure. Besides 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 can sample its coefficients from the CRS, which is easily achieved in the passive adversary model.

Protocol 2: **`CKG`**

**Public Input**: $p_1$
**Private Input** for $P_i$: $s_i = \mathtt{sk}_i$
**Output**: $\mathtt{cpk} = (p_0, p_1)$

Each party $P_i$:
1) samples $e_i \leftarrow \chi$ and broadcasts $p_{0,i} = [-(p_1 s_i + e_i)]_q$
2) waits for $p_{0,j}$ from every $P_j$, and outputs $\mathtt{cpk} = ([\sum_{P_j \in \mathcal{P}} p_{0,j}]_q \ , p_1)$

After the execution of the `CKG` protocol, each party has access to a copy of the collective public key

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

which has the same form as the ideal public key `pk` in Scheme 1, with different worst-case norms $\|s\|$ and $\|e\|$. The ciphertext noise growth (detailed in Section VI) is only linear in $N$. Hence, very large number of parties can be practically accommodated by choosing an appropriate modulus $q$ for the sought bit-security [36]. This is a significant advantage over the existing multi-key schemes, where this dependency is exponential [29], hence limiting $N$ to very small values. Another notable feature of the `CKG` protocol is that it would apply to any kind of additive sharing of $s$, including *traditional* uniformly random ones. In fact, in our scheme, the magnitude of the secret key shares is irrelevant, only their sum in $R_q$ is.

Protocol 3: **RKG**

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

Each party $P_i$:

1) samples $\mathbf{e}_{0,i} \leftarrow \chi^l$, $u_i \leftarrow R_3$ and broadcasts $\mathbf{h}_i = -u_i\mathbf{a} + s_i\mathbf{w} + \mathbf{e}_{0,i}$
2) waits for $\mathbf{h}_j$ from all $P_j$, samples $\mathbf{e}_{1,i}, \mathbf{e}_{2,i} \leftarrow \chi^l$ and broadcasts
$$\mathbf{h}'_{0,i} = s_i \sum_{P_j \in \mathcal{P}} \mathbf{h}_j + \mathbf{e}_{1,i} \quad \text{and} \quad \mathbf{h}'_{1,i} = s_i\mathbf{a} + \mathbf{e}_{2,i}$$
3) waits for $\mathbf{h}'_{0,j}, \mathbf{h}'_{1,j}$ from all $P_j$, sets $\mathbf{h}'_0 = \sum_{P_j \in \mathcal{P}} \mathbf{h}'_{0,j}$ and $\mathbf{h}'_1 = \sum_{P_j \in \mathcal{P}} \mathbf{h}'_{1,j}$, samples $\mathbf{e}_{3,i} \leftarrow \chi^l$ and broadcasts $\mathbf{h}''_i = (u_i - s_i)\mathbf{h}'_1 + \mathbf{e}_{3,i}$
4) waits for $\mathbf{h}''_j$ from all $P_j$ and outputs $\mathtt{rlk} = (\mathbf{r}_0, \mathbf{r}_1) = (\mathbf{h}'_0 + \sum_j \mathbf{h}''_j \ , \mathbf{h}'_1)$

*3) Relinearization-Key Generation (*RKG*):* In order to support public and non-interactive relinearization within a trust-domain, our distributed scheme enables the parties to generate a *relinearization key* `rlk` associated with a given collective secret key $s$. Protocol 3 (RKG) emulates the centralized `BFV.RelinKeyGen`; for this, it produces pseudo-encryptions of $s^2w^b$ for each power $b = 0...l$ of the decomposition basis $w$. The protocol requires a public input $\mathbf{a}$, uniformly sampled in $R_q^l$ by the parties, using the CRS. We use vector notation to express that these pseudo-encryptions are generated in parallel for every element of $\mathbf{w} = (w^0, w^1, ..., w^l)^T$.

Asharov et al. proposed a method to produce relinearization keys for distributed schemes based on the LWE problem [33]. This method could be adapted to our scheme, but would result in significantly increased noise in the `rlk` (thus, resulting in more noise to be introduced 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 pseudo-encryptions of $s^2$. Therefore, based on the observation that using the public key is unnecessary when the secret key is collectively known, we propose Protocol 3 as an improvement on Asharov et al.'s approach (we compare both approaches in Appendix B).

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

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

which, similarly as for the `cpk`, has the same form as `rlk` in the original scheme, although with larger $||e_{rlk}||$ for each component, and can be used to perform the `BFV.Relinearize` operation. We analyze the resulting relinearization noise in Section VI.

A convenient feature of the proposed RKG 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 [14], or those based on the *Chinese Remainder Theorem*, as proposed by Bajard et al. [55].

*4) Collective Key-Switching:* As discussed in Section IV-D, the key-switching functionality extends the general SMC framework by allowing the computation result to be re-encrypted under an arbitrary key. This, we recall, enables the *receiver* of a multiparty computation to be decoupled from the SMC problem. Hence, given a ciphertext `ct` decrypting under some *input key* $s$ and an *output key* $s'$, the key-switching procedure produces $\mathtt{ct}'$ such that $\mathtt{Dec}(s, \mathtt{ct}) = \mathtt{Dec}(s', \mathtt{ct}')$. The instantiation of the procedure, in our distributed setting, depends on whether the parties performing re-encryption are (collectively) entrusted with the output secret key, or only with its corresponding public-key. Therefore, we provide protocols that perform key-switching from $s$ to $s'$ for these two trust models:

a) Protocol 4 (**CKS**) is used when $s'$ is collectively known,
b) Protocol 5 (**PCKS**) is used when only a public key is known.

Both protocols require some fresh noise terms to be sampled from a special noise distribution $\chi_{CKS}$ that depends on the ciphertext being key-switched. This fresh noise implements the *smudging* technique, as introduced by Asharov et al. [33], the need and implementation of which we discuss in Section V-D. This technique assumes that the system keeps track of the ciphertext noise-level throughout the `General_SMC` protocol, which should be the case anyway to ensure correctness of the computation. Given a ciphertext `ct`, we denote `var(ct)` the variance of its noise term (see Eq. (1)).

*a) Collective Key-Switching (*CKS*):* Protocol 4 details the steps for key switching under the assumption that input parties collectively know the output secret key $s'$. In the SMC setting, this means the output party provided the input parties with shares of its key $s'$, which is safe for the receiver, as long as she trusts at least one input party (e.g., in *anytrust* types of models). The `CKS` protocol can also be used to handle the case of parties leaving and joining the system.

Protocol 4: **CKS**

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

Each party $P_i$:

1) samples $e_i \leftarrow \chi_{\mathtt{CKS}}(\sigma_{\mathtt{ct}})$ and broadcasts $h_i = [(s_i - s'_i)c_1 + e_i]_q$
2) waits for $h_j$ from every $P_j$ and outputs $\mathtt{ct}' = (c'_0, c_1) = ([c_0 + \sum_j h_j]_q, c_1)$

Protocol 5: **PCKS**

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

Each party $P_i$:

1) samples $u_i \leftarrow R_3$, $e_{0,i} \leftarrow \chi_{\mathtt{CKS}}(\sigma_{\mathtt{ct}})$, $e_{1,i} \leftarrow \chi$ and broadcasts
$$h_{0,i} = [s_i c_1 + u_i p'_0 + e_{0,i}]_q \, \text{and} \, h_{1,i} = [u_i p'_1 + e_{1,i}]_q$$
2) waits for $h_{0,j}$, $h_{1,j}$ from every $P_j$ and outputs $\mathtt{ct}' = (c'_0, c'_1) = ([c_0 + \sum_j h_{0,j}]_q, [\sum_j h_{1,j}]_q)$

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

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

where $e_{\texttt{CKS}} = \sum_j e_j$, and the last equality holds, provided that $\|e_{\texttt{ct}} + e_{\texttt{CKS}}\| < q/(2t)$. Hence, the correctness property holds as long as the output ciphertext noise is kept within decryptable bounds.

Consistently with our framework definition, the decryption protocol is simply a special case of the $\texttt{CKS}$ protocol where $s_j' = 0 \forall P_j \in \mathcal{P}$. In the SMC problem instance we consider, the computation output must be private, which is achieved by asserting that only the output party is provided with all the decryption shares. In a setting where the output party is in $\mathcal{P}$, it is enough that she does not reveal its own share.

*b) Collective Public-Key Switching ($\texttt{PCKS}$):* Protocol 5 details the steps for key switching when the input parties only know a public key for the output secret key $s'$. Contrarily to the $\texttt{CKS}$ protocol, the output party does not need to make any assumption about the presence of an honest input party in $\mathcal{P}$. This covers the scenario of our SMC framework extension described in Section IV-D, where the output party can be outside of the set of input parties.

After the execution of the $\texttt{PCKS}$ protocol on an input ciphertext $\texttt{ct} = (c_0, c_1)$ for which $c_0 + sc_1 = \Delta m + e_{\texttt{ct}}$, and a target public key $\texttt{pk} = (p_0', p_1')$ such that $p_0' = -(s'p_1' + e_{\texttt{pk}})$, the parties have access to $\texttt{ct}'$ satisfying

$$\begin{aligned}&\texttt{BFV.Dec}(s',\texttt{ct}') = \lfloor \frac{t}{q}[c_0' + s'c_1']_q \rceil \\ &= \lfloor \frac{t}{q}[c_0 + \sum_j \big(s_jc_1 + u_jp_0' + e_{0,j}\big) + s'\sum_j \big(u_jp_1' + e_{1,j}\big)]_q \rceil \\ &= \lfloor \frac{t}{q}[c_0 + sc_1 + up_0' + s'up_1' + e_0 + s'e_1]_q \rceil \\ &= \lfloor \frac{t}{q}[\Delta m + e_{\texttt{ct}} + e_{\texttt{PCKS}}]_q \rceil = m, \end{aligned} \quad (7)$$

where $e_d = \sum_j e_{d,j}$ for $d = 0,1$, $u = \sum_j u_j$, and the total added noise $e_{\texttt{PCKS}} = e_0 + s'e_1 + ue_{\texttt{pk}}$ depends on both the protocol-induced and the public key noises. The last equality holds if $\|e_{\texttt{ct}} + e_{\texttt{PCKS}}\| < q/(2t)$. Observe that the $\texttt{PCKS}$ protocol is more costly than $\texttt{CKS}$: It involves two more polynomial multiplications per party, and requires $N$ more ring elements to be sent and aggregated. It also introduces more noise, as it *emulates* the public encryption algorithm. This is the price for removing the *anytrust* assumption in $\texttt{CKS}$.

## D. Smudging

R-LWE-based cryptosystems have the fundamental property of decrypting to noisy plaintext messages (Eq. (1)), that are then *decoded* by the decryption algorithm (Eq. (2)). As the noise depends on the evaluation circuit and its intermediate values (this dependency is characterized in section VI), this cryptosystem family does not ensure *circuit privacy*. This has an important implication for our distributed scheme, where Equations (6) and (7) show that both $\texttt{CKS}$ and $\texttt{PCKS}$ allow error terms to cross trust boundaries. Two important facts must be considered: (a) The hardness assumption for the cryptosystem only holds if the error in the R-LWE samples is unknown to the adversary; and (b) the key-switched ciphertext-noise distribution depends on the source secret key, and can depend on the plaintext messages when the ciphertext is not fresh (see Section VI). Exploiting the aforementioned dependencies, an attacker with read access to the output trust-boundary (i.e., with knowledge of the output key) could attempt to extract information about the input key or some intermediate plaintext values in the computation.

We address this problem by means of *Smudging* techniques, as introduced by Asharov et al. [33]. Conceptually, smudging with a noise distribution $\chi_{\texttt{CKS}}$ ensures that the decryption of a key-switched ciphertext, before quantization and rounding, is indistinguishable from the decryption of a *fresh* one, that was encrypted using $\chi_{\texttt{CKS}}$ as error distribution. This is achieved by sampling part of the noise introduced during the $\texttt{CKS}$ and $\texttt{PCKS}$ protocols from a $\chi_{\texttt{CKS}}(\sigma_{\texttt{ct}})$ distribution, which must have significantly *larger* variance $\sigma^2_{\text{smg}}$ than that of the input ciphertext noise distribution (represented by $\sigma^2_{\texttt{ct}}$). Concretely, choosing

$$\sigma^2_{\text{smg}} = 2^{\lambda}\sigma^2_{\texttt{ct}} \quad (8)$$

guarantees that this is the case. As for the *fresh* noise distribution, the smudging noise can also be drawn from a truncated Gaussian.

As opposed to the method of Asharov et al. that introduce smudging noise also during evaluation, we introduce smudging noise exclusively during the $\texttt{CKS}$ protocols (i.e., at the trust-boundary crossing), where it is a single additive term in the $c_0'$ term. We detail the corresponding security argument in Section VII-C.

## E. Discussion

Our distributed scheme is flexible and has the potential to be extended. We showcase this flexibility by sketching how our protocols can support a variety of system models and extensions.

*1) Generalized Trust-Boundary Crossing:* The ability to perform some form of key-switching is a requirement for an SMC solution (see Section IV-D). In simple scenarios involving no external receiver, a decryption procedure, which we view as a particular case of key switching in our extended SMC framework, suffices. This is also a special case of our $\texttt{CKS}$ protocol, where each party can set its destination key share $s_i'$ to 0. Conversely, CKS can also be particularized into a distributed encryption protocol, by means of which the parties obtain the BFV encryption of a secret-shared message. Hence, as for $\pi_{\texttt{CKS}}$ in our general SMC framework, the $\texttt{CKS}$ protocol implements trust-boundary crossing in a general way.

*2) Dynamic Systems:* The versatility of our key-switching procedures can be very useful from a system design standpoint, to enable parties to dynamically join or leave the system. We consider the case of transferring data from the collective trust boundary of $\mathcal{P}$ to that of $\mathcal{P}'$. If $\mathcal{P} \subset \mathcal{P}'$, the $\texttt{PCKS}$ protocol can be used to re-encrypt data under the new collective key. Conversely, if $\mathcal{P}' \subset \mathcal{P}$, the $\texttt{CKS}$ protocol can be used with $s_i' = 0$ for each departing $P_i$. More efficient solutions are also possible under more stringent trust assumptions; e.g., a leaving party could additively share its collective secret-key share among the remaining participants.

*3) Generalized Ciphertext Space:* In our protocols, we consider ciphertexts comprising two ring elements, which corresponds to their definition in the original BFV cryptosystem. In fact, the BFV ciphertext space can be generalized as the ring $R_q[S]$ of polynomials in $S$ with coefficients in $R_q$. The encrypted arithmetic operations then correspond to the ring operations in $R_q[S]$. The decryption of a ciphertext $\texttt{ct}$, encrypted under key $s$, is performed as $m = [\lfloor \frac{t}{q}[\texttt{ct}(s)]_q \rceil]_t$, i.e., its evaluation in $S = s$ (followed by quantization and rounding). Thus, ciphertexts are allowed to grow during evaluation, which increases the flexibility in choosing when to apply a costly and noisy relinearization step. For this reason, this possibility is offered in most of the available BFV scheme implementations.

Our distributed scheme also features this flexibility; we chose not to integrate it directly into the protocol formulation for the sake of conciseness. More specifically, we can generalize our protocols in the following way: (a) The key-switching protocols can be extended to support ciphertexts of *degree* $d \geq 2$ (i.e., ciphertexts of $d+1$ ring elements), and (b) the $\texttt{RKG}$ procedure can be extended to produce relinearization keys for these ciphertexts (i.e., pseudo-encryptions of $s^d$).

*a) Degree-$d$ Key-Switching:* The trust-boundary crossing of an arbitrary size ciphertext $(c_0, c_1, ..., c_d)$ can be achieved using the $\texttt{CKS}$ protocol, by observing that its Step 1 can be iterated over (with intermediate aggregation) to raise the term in $s$ to the appropriate powers. Then, this augmented protocol can be used to perform collective decryption for each pair $(c_0, c_i)$ $i = 2...d$ and the usual $\texttt{CKS}$ protocol for $(c_0, c_1)$, thus cancelling the terms in higher powers of $s$. Notice how the result of this augmented protocol is equivalent to performing $d-2$ *interactive relinearization* steps followed by the actual trust boundary crossing. The possibility of interactive relinearization also provides ways to trade-off noise growth against computational and network overhead.

*b) Degree-$d$ Relinearization-Key:* Relinearization keys for a degree-$d$ polynomial of the ideal secret (i.e., ciphertexts) key can be generated using the $\texttt{RKG}$ protocol (Protocol 3), by iterating $d-1$ times its Step 2 (with intermediate aggregation) to raise the initial pseudo-encryption of $s$ to the appropriate powers. Notice that this would indeed increase the noise in the produced relinearization key.

*4) Flexible Network-Topology:* We assumed a public broadcast-based communication, thus formulating the protocols in a way that abstracts the communication patterns and network topology. We observe that this also has the interesting side effect of leaving total flexibility for the implementation of efficient communication patterns.

For example, the $N$ parties could be arranged in a tree-like topology, where each node in the tree would interact solely with its parent and children. We observe that, for all the protocols, the shares are always combined by computing their sum, which is commutative. Hence, for a given party in our protocols, a round would consist in

1) Receiving a message from the parent,
2) Forwarding it to its children while computing its share,
3) Aggregating its children's share and sending the result *up the tree* to its own parent.

Such network topology enables parallel computation of the shares and significantly reduces network traffic with respect to a naive broadcast approach. As for the ciphertexts, the public nature of protocols' transcripts also enable the parties to securely outsource the storage of intermediate values in the protocols' execution (e.g., to a cloud provider). Doing so, communication in our protocols becomes asynchronous, which yields a significant fault tolerance advantage.

*5) Fault-Tolerant Access Structures:* The secret-sharing scheme we chose for our distributed scheme requires that all parties interact at the decryption phase. This is by design: in an anytrust model, we want to protect against collusion of up to $N-1$ corrupted parties. However, for large systems, this guarantee would be too strong with respect to the risk of faulty nodes going offline for an undefined period. A *threshold* version of our distributed scheme using *$T$-out-of-$N$* Shamir secret-sharing [17] to share the secret key would enable such fault-tolerance capabilities, by allowing subsets of $T$ parties to perform key-switching. This would indeed cover less strict trust models, by relaxing the constraints on trust-boundary crossing. Although the computational cost of such a scheme would be higher, we believe it would be acceptable, for two main reasons: (a) As opposed to data-level secret-sharing approaches, the additional complexity would be confined to trust-boundary creation and enforcement (i.e., it would not affect the circuit evaluation phase). (b) Even though combining the shares introduces an overhead quadratic in $N$ (compared to a linear one in the *additive* case), the extrapolation of the results from Appendix A suggests that the scheme would remain efficient.

Note that, regardless of the secret-sharing scheme, our approach requires a single, asynchronous, interaction per-party, once the non-interactive evaluation phase finishes. Hence, our approach has a signifiant advantage over online SMC-solutions, by gracefully tolerating nodes going offline for a limited amount of time.

## VI. Noise Analysis

We now analyze the behavior of the distributed version of the BFV cryptosystem in terms of noise growth. We recall that our public- and relinearization-keys only differ from the original scheme in the magnitude of the noise they contain. Hence, the analysis of [14] still applies, with a larger worst-case error norm that we express as a function of the number of parties $N$. The derivations for the equations presented in this section can be found in Appendix C.

We represent ciphertexts as elements of $R_q[S]$ (see Section V-E3). The infinity norm of a polynomial $p$ (i.e., its largest coefficient in absolute value) is denoted $\|p\|$ ($\|p\| \leq q/2$ for $p \in R_q$). We also recall that, since the polynomial modulus in $R_q$ is a degree-$n$ power of 2 cyclotomic, we have $\|ab\| \leq n\|a\|\|b\|$. We consider an instantiation of our distributed BFV scheme with $N$ parties.

As a result of the secret-key generation procedure, where each $s_i$ is sampled from $R_3$ (see Section V-C1), we know that $\|s\| \leq N$. It must be noted that, by assuming a trusted dealer or a more complex key-generation protocol, we could lower or completely remove the factor $N$, so we keep the norm of $s$ as a parameter in the following, for the sake of generality. As a result of our $\texttt{CKG}$ protocol, the collective public key noise is $e_{\texttt{cpk}} = \sum_{i=1}^{N} e_i$ (see Eq. (4)), which implies that $\|e_{\texttt{cpk}}\| \leq NB$, where $B$ is the worst-case norm for an error term sampled from the R-LWE error distribution $\chi$.

### A. Fresh Encryption

Let $\texttt{ct}= (c_0, c_1)$ be a fresh encryption of a message $m$ under $\texttt{cpk}$, such that $\frac{t}{q}(c_0 + sc_1) = m + v_{\mathit{fresh}}$; we have

$$\|v_{\mathit{fresh}}\| \leq -\frac{r_t(q)}{q}\|m\| + \frac{t}{q}B(nN + n\|s\| + 1), \tag{9}$$

where $r_t(q)$ denotes the remainder of the division of $q$ by $t$. Thus, for a key generated by the $\texttt{CKG}$ protocol, the worst-case fresh ciphertext noise is linear in the number $N$ of parties.

### B. Arithmetic Operations

We consider $\texttt{ct}_1$ and $\texttt{ct}_2$, two encryptions such that $\frac{t}{q}[\texttt{ct}_i(s)]_q = m_i + v_i$ with $\|v_i\| < B_i$ for $i = 1,2$.

Let $\texttt{ct}_{\mathit{add}}$ be the homomorphic sum of $\texttt{ct}_1$ and $\texttt{ct}_2$, such that $\frac{t}{q}[\texttt{ct}_{\mathit{add}}(s)]_q = m_1 + m_2 + v_{\mathit{add}}$ with

$$\|v_{\mathit{add}}\| \leq B_1 + B_2 + t\frac{r_t(q)}{q}(m_{\mathit{add}} - [m_{\mathit{add}}]_t),$$

where $m_{add} = m_1 + m_2$, and the last term is non-zero only in the case where $\|m_{\text{add}}\| > t$. This follows directly from the analysis in [14].

Let $\mathtt{ct}_{mul}$ be the homomorphic product of $\mathtt{ct}_1$ and $\mathtt{ct}_2$, such that $\frac{t}{q}[\mathtt{ct}_{mul}(s)]_q = [m_1 m_2]_t + v_{mul}$. Then, by relying on the upper bound given by Lemma 2 in [14], we have

$$\|v_{mul}\| < nt(B_1+B_2)(n\|s\|+1) + 2t^2 n^2(\|s\|+1).$$

As for the homomorphic addition, the additional noise depends only on the operands' inherent noise and the magnitude of the secret key (there is no additional noise source). Therefore, the homomorphic multiplication noise-growth before relinearization does not directly depend on the number of parties. However, it does, indirectly, when $s$ is the sum of $N$ elements of $R_3$, as proposed in Section V-C1.

### C. Relinearization (Type I)

We analyze the noise resulting from a Type I relinearization [14] performed using a key generated by the RKG protocol (Protocol 3). Thus, for a ciphertext $\mathtt{ct} = (c_0, c_1, c_2)$, we can write its 2-component equivalent as $\mathtt{ct}_{\mathtt{relin}} = (c_0', c_1')$, where $\frac{t}{q}[\mathtt{ct}_{\mathtt{relin}}(s)]_q = m + v_{fresh} + v_{relin}$ with

$$\|v_{relin}\| \leq \frac{wnt}{2q}(l+1)((n\|s\|+2)N + nN^2)B. \tag{10}$$

Therefore, the noise introduced by relinearization increases by a factor that is quadratic in $N$; this factor stems from the noise introduced in the relinearization key (see Eq. (14)). This result is consistent, as the relinearization keys generated by our RKG protocol only differ from those of the centralized scheme in the magnitude of the noise term, in their $\mathbf{r}_0$ component. Analogously to the original scheme, the noise introduced by the relinearization is independent of the noise already present in the input ciphertext.

### D. Collective Key-Switching

Let $\mathtt{ct} = (c_0, c_1)$ be an encryption of $m$ under the collective secret key $s$, and $\mathtt{ct}' = (c_0', c_1)$ be the output of the CKS protocol on $\mathtt{ct}$ with target key $s'$. Then, $\frac{t}{q}\mathtt{ct}'(s') = m + v_{fresh} + v_{\mathtt{CKS}}$ with

$$\|v_{\mathtt{CKS}}\| \leq \frac{t}{q} B_{smg} N, \tag{11}$$

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

### E. Public Collective Key-Switching

Let $\mathtt{ct} = (c_0, c_1)$ be an encryption of $m$ under the collective secret key $s$, and $\mathtt{ct}' = (c_0', c_1')$ be the output of the PCKS protocol on $\mathtt{ct}$ and target public key $\mathtt{pk}' = (p_0', p_1')$, such that $p_0' = -sp_1' + e_{\mathtt{pk}'}$. Then, $\frac{t}{q}\mathtt{ct}'(s') = m + v_{fresh} + v_{\mathtt{PCKS}}$ with

$$\|v_{\mathtt{PCKS}}\| \leq \frac{t}{q} N(nB_{\mathtt{pk}'} + n\|s'\|B + B_{\text{smg}}) \tag{12}$$

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

### F. Discussion

The noise analysis shows that our distributed scheme, in contrast to multi-key schemes, keeps the noise within manageable bounds even for a large number of parties. The worst overhead is quadratic in $N$, in the case of relinearization. However, this operation is carried out after a multiplication that has a significantly larger impact on the ciphertext noise already in the original scheme.

## VII. Security Analysis

In this section, we analyze the security of the proposed scheme in the passive adversary model. Let $\mathcal{A}$ denote the adversary, defined as a subset of at most $N-1$ corrupted parties in $\mathcal{P}$. We prove the security of our multiparty protocols in the ideal/real simulation paradigm [56]. That is, for every possible $\mathcal{A}$, we prove by construction that 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 of SMC, we require that $\mathcal{A}$ must not be able to distinguish the real view (generated from the honest parties' inputs) from the simulated one (generated with the adversary's input only). For a given value $x$, we denote $\tilde{x}$ its simulated equivalent. Unless otherwise stated, we consider computational indistinguishability between distributions, denoted $\tilde{x} \overset{c}{\equiv} x$.

Our threat model implies that at least one honest player exists, which we denote $P_h$. The choice for $P_h$ when multiple honest parties exist is irrelevant and does not reduce generality. It does, however, help simplify the formulation of the security argument. We denote $\mathcal{H}$ the set $\mathcal{P} \backslash (\mathcal{A} \cup \{P_h\})$ 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 $\mathcal{A}$, attacking the CKG protocol as 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 started). Thus, we model the functionality of the CKG protocol as $f_{\mathtt{CKG}}(\{s_i, e_i \mid P_i \in \mathcal{P}\}) = \mathtt{cpk}$, where $\mathtt{cpk} = (p_0, p_1)$ is the output for all parties, as defined in Eq. (4).

We observe that the view of each party in the execution of the CKG protocol consists in 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 distinguish from the real ones). $S$ can compute this sharing as

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

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 $\mathcal{A}$ distinguishing, with non-negligible advantage, between real and simulated shares of those players in $\mathcal{H}$ would directly yield a distinguisher for the decision-R-LWE problem [41]. For the share of player $P_h$, we consider two cases: (1) When $\mathcal{H} \neq \emptyset$, the share $p_{0,h}$ is a uniformly random element in $R_q$ because $[\sum_{P_j \in \mathcal{H}} p_{0,j}]_q$ is itself so, and the same indistinguishability argument as above applies. (2) In the presence of $N-1$ adversaries, $\mathcal{H} = \emptyset$ and $S$ computes the real value for the honest party's share, hence outputting the real view.

### B. Collective Relinearization-Key Generation

The private input for each party $P_i$ in the RKG protocol consists in 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, its ephemeral secret, and the error terms added in each round. The output for each party is $f(x_1, \ldots, x_N) = (\mathbf{r}_0, \mathbf{r}_1)$, the generated relinearization key defined in Eq. (5). Throughout the protocol execution, the parties compute the public values $\mathbf{h}$, $\mathbf{h}' = (\mathbf{h}_0', \mathbf{h}_1')$ and $\mathbf{h}''$. These values can be simulated, with the constraints $\mathbf{r}_0 = \mathbf{h}'_0 + \mathbf{h}''$ and $\mathbf{r}_1 = \mathbf{h}'_1$. For every round, the parties' view in the protocol comprises additive sharings of these values, which $S$ can simulate as

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

$$\widetilde{\mathbf{h}}'_{0,i} = \begin{cases} [s_i\widetilde{\mathbf{h}}+\mathbf{e}_{1,i}]_q & P_i \in \mathcal{A} \\ \leftarrow R_q^l & P_i \notin \mathcal{A} \end{cases},$$

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

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

We first show that the simulated view cannot be distinguished from the real one, by considering the *combined* view of the adversary,

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

We show that given $\mathbf{a}$, this view is indistinguishable from the uniform distribution over $R_q^{4\times l}$. As $\mathbf{h}'_0 - \mathbf{h}'' \approx s^2\mathbf{a} + s^2\mathbf{w}$, this is equivalent to Assumption 1.

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

$$(a, \quad s_1a+e_1, \quad s_2a+e_2, \quad s_2s_1a+e_3, \quad s_1^2a+e_4) \tag{13}$$

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

Hence, the security of the RKG protocol relies on similar assumptions as the original, centralized, BFV scheme. More specifically, the first two elements of Eq. (13) 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 $u = s_2$. The last element, approximated as $\mathbf{h}'_0 - \mathbf{h}''$ in the protocol, corresponds to the reasonable assumption that indistinguishability holds, even when generating further samples with higher degrees of the secret key.

## C. Collective Key-Switching

The security argument for the CKS 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 distributed lattice-based schemes, often formulated as an ad-hoc security measure. Due to space limitations, we show the analysis only for the CKS protocol. However, the argument is very similar for its public key version and directly applies to the special case of $s' = 0$ (collective decryption).

Given a ciphertext $\mathtt{ct} = (c_0, c_1)$ decrypting under $s$ the ideal functionality of the CKS protocol (Protocol 4) is to compute $\mathtt{ct}' = (c'_0, c_1)$ decrypting under secret key $s'$. We can formulate this functionality implicitly, as the computation of $c'_0$ satisfying

$$c_0 + sc_1 - e = c'_0 + s'c_1 - e',$$

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

$$\hat{h} = \hat{f}_{\mathtt{CKS}}(\{s_i, s'_i, e'_i | \forall P_i \in \mathcal{P}\}) = c'_0 - c_0 = (s - s')c_1 - \hat{e} + e',$$

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 should look fresh, even for an adversary that knows all shares of $s'$, which is allowed in the CKS protocol. As this term cannot be efficiently computed in practice, the real output differs from the ideal one. Simulation-based proofs allow this difference, as long as we prove 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 having access to the real output, the adversary cannot distinguish the simulated view from the real one (Property 2). Therefore, showing both Properties (1) and (2) implies

$$(\widetilde{h}_1, \widetilde{h}_2, \ldots, \widetilde{h}_N, \hat{h}) \overset{c}{\equiv} (h_1, h_2, \ldots, h_N, h),$$

i.e., that the CKS protocol 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 R-LWE-like structure of $h$. More specifically, 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 from that of the ideal output, where it is *fresh*:

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

As $e$ is the key-switched ciphertext error, it follows a centered Gaussian distribution whose variance we denote $\sigma_{\mathtt{ct}}^2$. Whereas, $e'$ is the sum of all the noise terms protecting the key switching shares, which are sampled according to the $\chi_{\mathtt{CKS}}(\sigma_{\mathtt{ct}})$ distribution that has variance $\sigma_{\mathtt{CKS}}^2$. Thus, as long as the ratio $\sigma_{\mathtt{ct}}^2/\sigma_{\mathtt{CKS}}^2$ 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 CKS protocol is an additive sharing $(h_1, h_2, \ldots, h_N)$ of $h$, which $S$ can simulate as

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

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

**Assumption 2.** *An input ciphertext $(c_0, c_1)$ to the CKS protocol is such that $c_0 + sc_1 = \Delta m + e_{ct}$ where $e_{ct} = e_{\mathcal{A}} + e_h$ includes a term $e_h$ that is unknown to, and independent from, the adversary. Furthermore $e_h$ follows a distribution according to the R-LWE hardness assumptions.*

If Assumption 2 holds, we know that $\mathcal{A}$ can only approximate the term $e_h$ up to an error $e_{\mathtt{ct},h}$, which is enough to make $(h_h,c_1)$ indistinguishable from $(a \leftarrow R_q, c_1)$, even if the adversary controls $N-1$ parties. We remark that as long as all parties provide at least one input (for which the noise is fresh) to the homomorphic function evaluation, the requirement of Assumption 2 is satisfied.

## VIII. Conclusions

In this work, we have introduced a novel perspective on the Secure Multiparty Computation Problem and proposed a general framework for its solutions. The concept of *trust boundary* assumes a crucial role in modeling multiparty trust settings, and enables the isolation and characterization of the SMC problem in a way that abstracts the specifics of the protection mechanisms.

We have proposed a view in which distributed cryptosystems are at the core of an SMC solution, and have formalized this view into a generic SMC framework. We showed that this framework is capable of modeling and analyzing the characteristics of SMC solutions that are often seen as fundamentally different and difficult to compare. By formulating existing SMC solutions (such as those based on secret-sharing the computation data) as particular instances of our framework, we extracted the desirable features that a distributed cryptosystem should offer in order to instantiate an efficient SMC solution.

As an example of such cryptosystem, we have introduced a distributed version of a widespread lattice-based cryptosystem (BFV), which can instantiate our framework in an efficient way, even for very large number of parties. Notably, we have shown that the noise overhead with respect to the original (centralized) scheme is only quadratic in the number of parties (in the worst case), which provides a significant advantage over multi-key schemes, where it is exponential. We also analyzed the security of its underlying protocols in the passive adversary model. We have implemented the distributed BFV (see Appendix A), and demonstrated that, besides being theoretically practical, our scheme can provide a flexible and efficient SMC solution.

On the technical side, the proposed SMC solution can benefit complex and highly relevant SMC scenarios, ranging from data sharing and distributed machine learning to oblivious smart contract evaluation and efficient generation of Beaver triples for the SPDZ protocol. We are currently exploring concrete instantiations of our SMC solution in these application areas. These instantiations comprise also extensions of our distributed cryptosystem implementation to other R-LWE variants, such as the recent CKKS scheme [54].

On the conceptual side, it is worth noting that the trust boundary concept can be extended to other adversarial models and security mechanisms, hence opening up new possibilities in system and trust modeling. This can make SMC easier to integrate into larger and more complex designs, also from a conceptual point of view.

In the near future, cloud-services providers could offer storage and evaluation services on their clients' encrypted sensitive data. By combining our technical and conceptual contributions, these services could be extended to multi-clients scenarios, hence bridging the gap between SMC research and its adopters.

## Acknowledgments

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

## References

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

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

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

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

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

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

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

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

[9] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[10] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 359–376.

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

[12] A. Aly, M. Keller, E. Orsini, D. Rotaru, P. Scholl, N. Smart, and T. Wood, "SCALE–MAMBA v1. 3: Documentation," 2019.

[13] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "SoK: General purpose compilers for secure multi-party computation," in *SoK: General Purpose Compilers for Secure Multi-Party Computation*. IEEE, 2019, p. 0.

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

[15] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution," *arXiv preprint arXiv:1804.05141*, 2018, last accessed February 2019.

[16] M. Van Dijk and A. Juels, "On the impossibility of cryptography alone for privacy-preserving cloud computing." *HotSec*, vol. 10, pp. 1–8, 2010.

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

[18] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 1986, pp. 162–167.

[19] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, pp. 218–229.

[20] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 1990, pp. 503–513.

[21] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits." in *USENIX Security Symposium*, vol. 201, no. 1, 2011, pp. 331–335.

[22] S. Gueron, Y. Lindell, A. Nof, and B. Pinkas, "Fast garbling of circuits under standard assumptions," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 567–578.

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

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

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

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

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

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

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

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

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

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

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

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

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

[36] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic Encryption Security Standard," HomomorphicEncryption.org, Toronto, Canada, Tech. Rep., November 2018.

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

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

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

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

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

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

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

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

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

[46] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 85–100.

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

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

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

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

[51] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing Multi-Party Computation." *IACR Cryptology ePrint Archive*, vol. 2011, p. 272, 2011.

[52] A. Peter, E. Tews, and S. Katzenbeisser, "Efficiently outsourcing multiparty computation under multiple keys," *IEEE transactions on information forensics and security*, vol. 8, no. 12, pp. 2046–2058, 2013.

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

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

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

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

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

## Appendix A
## Evaluation

We demonstrate the practicality of a distributed R-LWE cryptosystem. For this purpose, we implemented both the centralized and our distributed versions of the BFV scheme using Go language. Our polynomial arithmetic implementation uses CRT [55] (with 60-bit limbs) and NTT [57] optimizations. For this evaluation, we time all the operations with no parallelization.

We executed our benchmarks on a 2.5 GHz Intel Core i5-7200U processor and 8 Gb of memory running Go 1.12 (linux/amd64). Table III shows the parameter sets used in our benchmarks, all guaranteeing a security level of 128 bits [36]. These parameters are application dependent, but their choice represents typical sizes of $q$ that correspond to different *homomorphic capacities* (e.g., set `P8192` allows for approximately twice the noise level as `P4096`). For relinearization, we use a base $w$ of 60 bits.

For reference, Table IV shows the timings in the centralized scheme. While still being under development, our implementation is comparable, in terms of operations timings, to the currently available C++ libraries. Table V shows the computation time per operation and per party in our distributed scheme instantiated with two parties. We observe that the number of parties plays very little role in these timings, as it only influences the share aggregation that consists in computing their sum in $R_q$. This is not a costly operation, and depends on the actual network topology, which could rely on intermediate aggregation as discussed in Section V-E4. For the same reason, and because they also depend on the network performance, we omit network-related timings. In order to evaluate the communication overhead, Table VI shows the sizes of the output for each step, which correspond to the theoretical minimum amount of data to be sent, at that step, by each party.

It can be seen that distribution of the scheme is practical, and that the timings of its operations are comparable to the ones in the centralized scheme. The fact that the distributed scheme apparently performs better in generating the `cpk` and `rlk` is due to the experimental setup. The `CKS` protocol is also comparable with the centralized `Decrypt` operation, which is consistent with the definition of the latter as a special case of key switching.

These results are promising, from an application point of view: The most costly operations, those of `RKG`, are naturally parallelizable for each elements of $\mathbf{r}_0$ and $\mathbf{r}_1$. Moreover, this procedure, along with the `CKG` protocol, needs to be run only once as a part of the trust boundary creation process. On the other hand, the `CKS` protocol has to be run for each result decryption. However, we observe that it is efficient, and consist in a single round that can be made asynchronous, by relying on a central storage accessible to each party. We recall that this storage, along with the computation of homomorphic arithmetic operation, can be outsourced to a third party without making further trust assumptions.

Overall, we conclude that, when instantiated with our distributed scheme, the framework described in Section IV can be an efficient SMC solution.

TABLE III: Benchmarking parameter sets

| Set | $n$ | $\log_2 q$ |
|---|---|---|
| P4096 | 4096 | 120 |
| P8192 | 8192 | 240 |
| P16384 | 16384 | 480 |
| P32768 | 32756 | 960 |

TABLE IV: Centralized BFV Operation timings (ms)

| | | P4096 | P8192 | P16384 | P32768 |
|---|---|---|---|---|---|
| | `KeyGen` | 14.52 | 46.76 | 162.05 | 607.86 |
| Encryption | `RelinKeyGen` | 18.47 | 138.89 | 1103.43 | 8878.57 |
| | `Encrypt` | 7.40 | 17.08 | 42.65 | 127.28 |
| | `Decrypt` | 0.83 | 3.66 | 15.01 | 63.57 |
| | `Add` | 0.05 | 0.26 | 0.82 | 3.29 |
| Arithmetic | `Mul` | 16.11 | 62.49 | 259.03 | 1306.21 |
| | `Relin` | 3.37 | 16.41 | 93.35 | 623.81 |

TABLE V: Distributed BFV Operation timings (ms)

| | | P4096 | P8192 | P16384 | P32768 |
|---|---|---|---|---|---|
| | Step 1 | 0.98 | 2.66 | 8.00 | 27.99 |
| `CKG` | Step 2 | 0.05 | 0.18 | 0.73 | 3.29 |
| | **Total** | **1.03** | **2.84** | **8.73** | **31.27** |
| | Step 1 | 2.00 | 10.50 | 63.96 | 517.00 |
| | Step 2 | 4.04 | 22.17 | 140.87 | 1017.66 |
| `RKG` | Step 3 | 2.25 | 13.12 | 81.64 | 617.79 |
| | Step 4 | 0.34 | 2.98 | 21.85 | 175.71 |
| | **Total** | **8.64** | **48.77** | **308.32** | **2328.16** |
| | Step 1 | 1.23 | 3.86 | 13.21 | 52.04 |
| `CKS` | Step 2 | 0.10 | 0.39 | 1.64 | 6.77 |
| | **Total** | **1.33** | **4.24** | **14.84** | **58.81** |
| | Step 1 | 7.80 | 18.45 | 48.55 | 148.84 |
| `PCKS` | Step 2 | 0.14 | 0.55 | 2.38 | 9.58 |
| | **Total** | **7.94** | **19.00** | **50.93** | **158.42** |

TABLE VI: Distributed BFV Operation output size (kB)

| | | P4096 | P8192 | P16384 | P32768 |
|---|---|---|---|---|---|
| | Step 1 | 66 | 263 | 1,049 | 4,195 |
| `CKG` | Step 2 | 0 | 0 | 0 | 0 |
| | **Total** | **66** | **263** | **1,049** | **4,195** |
| | Step 1 | 66 | 263 | 1,049 | 4,195 |
| | Step 2 | 132 | 525 | 2,098 | 8,389 |
| `RKG` | Step 3 | 66 | 263 | 1,049 | 4,195 |
| | Step 4 | 0 | 0 | 0 | 0 |
| | **Total** | **263** | **1,049** | **4,195** | **16,778** |
| | Step 1 | 66 | 263 | 1,049 | 4,195 |
| `CKS` | Step 2 | 0 | 0 | 0 | 0 |
| | **Total** | **66** | **263** | **1,049** | **4,195** |
| | Step 1 | 132 | 525 | 2,098 | 8,389 |
| `PCKS` | Step 2 | 0 | 0 | 0 | 0 |
| | **Total** | **132** | **525** | **2,098** | **8,389** |

## Appendix B
## Comparison between RKG and SIMPLE_RKG

We first show how to adapt the classic method of [33] to our scheme, resulting in the Protocol 6.

Protocol 6: **SIMPLE_RKG**

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

Each party $P_i$:

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

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

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

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

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

TABLE VII: Timing benchmark RKG vs. SIMPLE_RKG (ms)

| | | P4096 | P8192 | P16384 | P32768 |
|---|---|---|---|---|---|
| SIMPLE_RKG | Step 1 | 14.84 | 67.59 | 338.17 | 1969.06 |
| | Step 2 | 15.40 | 68.65 | 357.19 | 2194.63 |
| | Step 3 | 0.29 | 2.51 | 18.61 | 156.83 |
| | **Total** | **30.54** | **138.74** | **713.98** | **4320.51** |
| RKG | Step 1 | 2.00 | 10.50 | 63.96 | 517.00 |
| | Step 2 | 4.04 | 22.17 | 140.87 | 1017.66 |
| | Step 3 | 2.25 | 13.12 | 81.64 | 617.79 |
| | Step 4 | 0.34 | 2.98 | 21.85 | 175.71 |
| | **Total** | **8.64** | **48.77** | **308.32** | **2328.16** |

## Appendix C
## Derivation of Noise-Growth Equations

### A. Derivation of Eq. (9)

From the ideal decryption of a fresh encryption of $m$ under the collective public key $\mathtt{cpk}=(p_0,p_1)$:

$$\begin{aligned}\frac{t}{q}(c_0+sc_1)&=\frac{t}{q}(\Delta m+p_0u+e_0+sp_1u+se_1)\\ &=\frac{t}{q}(\frac{q-r_t(q)}{t}m-ue_{\mathtt{cpk}}+e_0+se_1)\\ &=m-\frac{r_t(q)}{q}m+\frac{t}{q}(-ue_{\mathtt{cpk}}+e_0+se_1),\end{aligned}$$

where we substituted the expression of BFV.Encrypt and used $\Delta=\frac{q-r_t(q)}{t}$. As $\|u\|=1$ and $\|e_i\|\leq B$ for $i=0,1$, Eq. (9) follows.

### B. Derivation of Eq. (10)

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

$$\|e^{(i)}_{\mathtt{rlk}}\|<((n\|s\|+2)N+nN^2)B. \tag{14}$$

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

$$\begin{aligned}\frac{t}{q}(c'_0+sc'_1)&=\frac{t}{q}(c_0+\mathbf{c}_2\cdot\mathbf{r}_0+s(c_1+\mathbf{c}_2\cdot\mathbf{r}_1)\\ &=\frac{t}{q}(c_0+sc_1+\mathbf{c}_2\cdot(\mathbf{r}_0+s\mathbf{r}_1))\\ &=\frac{t}{q}(c_0+sc_1+\mathbf{c}_2\cdot s^2\mathbf{w}+\mathbf{c}_2\cdot\mathbf{e}_{\mathtt{rlk}})\\ &=m+v_{\mathit{fresh}}+\frac{t}{q}(\mathbf{c}_2\cdot\mathbf{e}_{\mathtt{rlk}}),\end{aligned}$$

where the upper bound for the inner product term is derived from the expression for $\mathbf{e}_{\mathtt{rlk}}$ in Eq. (14), by observing that each of the $l+1$ elements in $\mathbf{c}_2$ have coefficients in $[-\frac{w}{2},\frac{w}{2}]$.

### C. Derivation of Eq. (11)

From the decryption expression of $\mathtt{ct}'$,

$$\begin{aligned}\frac{t}{q}(c'_0+s'c_1)&=\frac{t}{q}(c_0+\sum_j((-s'_j+s_j)c_1+e_{\mathtt{CKS},j})+s'c_1)\\ &=\frac{t}{q}(c_0+sc_1+\sum_j e_{\mathtt{CKS},j})\\ &=m+v_{\mathit{fresh}}+\frac{t}{q}(\sum_j e_{\mathtt{CKS},j}).\end{aligned}$$

As $e_{\mathtt{CKS},j}\leq B_{smg}$, Eq. (11) follows.

## D. *Derivation of Eq.* (12)

From the decryption expression of $\texttt{ct}'$,

$$\begin{aligned}\frac{t}{q}(c_0'+s'c_1')=&\frac{t}{q}\Big(c_0+\sum_j(s_jc_1+u_jp_0'+e_{0,j})\\&+s'\sum_j(u_jp_1'+e_{1,j})\Big)\\=&\frac{t}{q}(c_0+sc_1+up_0'+s'up_1'+\sum_j e_{0,j}+se_{1,j})\\=&m+v_{\textit{fresh}}+\frac{t}{q}(\sum_j u_je_{\texttt{pk}'}+e_{0,j}+s'e_{1,j}),\end{aligned}$$

and Eq. (12) follows.