Publicly Accountable Robust Multi-Party Computation

Marc Rivinius, Pascal Reisert, Daniel Rausch and Ralf Küsters
Institute of Information Security
University of Stuttgart
Stuttgart, Germany
{marc.rivinius, pascal.reisert, daniel.rausch, ralf.kuesters}@sec.uni-stuttgart.de

2022 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-1316-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SP46214.2022.9833608

***Abstract*—In recent years, lattice-based secure multi-party computation (MPC) has seen a rise in popularity and is used more and more in large scale applications like privacy-preserving cloud computing, electronic voting, or auctions. Many of these applications come with the following high security requirements: a computation result should be publicly verifiable, with everyone being able to identify a malicious party and hold it accountable, and a malicious party should not be able to corrupt the computation, force a protocol restart, or block honest parties or an honest third-party (client) that provided private inputs from receiving a correct result. The protocol should guarantee verifiability and accountability even if all protocol parties are malicious. While some protocols address one or two of these often essential security features, we present the first publicly verifiable and accountable, and (up to a threshold) robust SPDZ-like MPC protocol without restart. We propose protocols for accountable and robust online, offline, and setup computations. We adapt and partly extend the lattice-based commitment scheme by Baum et al. (SCN 2018) as well as other primitives like ZKPs. For the underlying commitment scheme and the underlying BGV encryption scheme we determine ideal parameters. We give a performance evaluation of our protocols and compare them to state-of-the-art protocols both with and without our target security features: public accountability, public verifiability and robustness.**

I. Introduction

In recent years, secure multi-party computation (MPC) has evolved from a theoretical concept to a technology with more and more industrial scale applications including, for example, complex machine learning (ML) tasks [1]–[6]. One major contribution to this success are efficient two-phase protocols like SPDZ [7], [8] and "SPDZ-like" protocols [9]–[16] which consist of an input-independent offline phase and a highly efficient input-dependent online phase.[1] Most of these protocols provide security with abort (the output is correct or the protocol aborts without output), often even with a dishonest majority.

While in many situations this is sufficient, applications following the client-server model generally require stronger security properties. In client-server applications, servers are responsible for running the MPC protocol and clients provide inputs to and receive outputs from the servers such that individual servers do not learn the inputs and, depending on the application, also not the outputs. Many important real-life applications follow this client-server model, e.g., auctions [17], [18], e-voting protocols [19], [20], and cloud services for privacy preserving computation [21]–[24] – including ML tasks [1], [5], [25]–[27]. These client-server applications often require *publicly identifiable abort* [11], [14], [28]: it must be possible to verify whether the outputs of the servers are correct[2] and, if the result is not correct and hence the protocol aborts, then one must be able to identify at least one misbehaving server that can be held *accountable* for causing the abort. Here "public" means that not just the servers running the MPC protocol, but rather everyone, including clients and external parties, can verify the results and hold misbehaving servers accountable. This not only allows clients and external parties to trust the final output (e.g., the result of an election or an auction) but, when coupled with a financial or contractual penalty [35], [36], serves as a strong incentive for malicious servers to honestly follow the protocol instead of causing aborts. Unlike in the traditional (non-client-server) setting, where at least one honest MPC participant is assumed, publicly identifiable abort should still hold true even if all servers are malicious since clients might not trust any of the servers. In what follows, we call a security notion *strong* if it holds even in this setting. To the best of our knowledge, the SPDZ-like protocol of Cunningham et al. [14] is the only efficient two-phase protocol with strong publicly identifiable abort.

If only a certain (small) fraction of participants of an MPC protocol is corrupted, then it is desirable to prevent the malicious parties from causing an abort in the first place, rather than merely blaming them after the fact (if too many parties are corrupted, it is generally impossible to prevent an abort [37]). This property is called *robustness* or *guaranteed output delivery* [38], [39] which is a highly useful property both in traditional and client-server applications. For example, a single malicious server should not be able to prevent the computation of the winner of an election or the result of an auction. Since a number of misbehaving parties above the threshold can still cause aborts even for robust protocols, (strong) publicly identifiable abort still serves as a desirable backup security mechanism. This mechanism can even be strengthened as follows: In case of an abort, one should identify not just a single but at least a number of malicious parties that is needed to cause an abort, i.e., more than the threshold. Also, even if a protocol did not abort, one might

[1] We use "SPDZ-like" to refer to protocols that improve and/or extend the SPDZ protocol.

[2] This property is called *public/universal verifiability* [10], [29]–[33] and is implied by publicly identifiable abort [34].

DOI 10.1109/SP46214.2022.00091

still be able to identify some misbehaving parties that tried but failed to undermine robustness. In what follows, we use the term (strong) *public accountability* [34] to refer to this strengthened notion of (strong) publicly identifiable abort.

Clearly, both (strong) public accountability and robustness are often highly desirable security properties. Yet the combination of both properties has not been considered for efficient two-phase protocols so far (cf. Table I; note that SPDZ-like protocols are only a subset of the two-phase protocols we examined). The protocol that probably comes closest is Cunningham et al. [14] which is a SPDZ-like protocol with strong publicly identifiable abort. A straightforward method to add robustness to such a protocol is to restart the whole protocol without the previously misbehaving parties whenever a result does not verify [13], [28], [40]. But this method is actually insecure in certain application contexts such as auctions, since an adversary can see the betting behavior of other parties in previous (aborted) rounds and adapt their strategy accordingly (cf. [13]). It seems however possible to combine Cunningham et al.'s protocol securely with a more advanced method of this iteration technique, namely the so-called best-of-both-worlds (BoBW) protocols [41], [42]. These best-of-both-worlds protocols add robustness to an MPC protocol with identifiable abort by iterating the MPC protocol, while also adding another layer of secret-sharing to prevent the security issues caused by the straightforward approach, i.e., the adversary no longer learns the result of aborted runs. While the BoBW protocols [41], [42] consider only non-publicly identifiable abort in a traditional setting (non-client-server), using the same approach should likely also preserve strong publicly identifiable abort or even strong public accountability (in the client-server setting) while adding robustness. However, using such an iteration technique comes with serious downsides. Firstly, in the presence of misbehaving parties there is a drastic loss of performance (runtime, network communication, and communication rounds) by a factor of up to $\mathcal{O}(n)$ if one malicious server is identified in each round, where n is the number of participants/servers. Most importantly, since the material generated in the offline phase of Cunningham et al. depends on the set of participants and since this set changes with each protocol iteration, also costly offline material has to be re-generated multiple times *during the online phase* if an abort happens there.[3] Secondly, clients have to be involved in each re-run of the protocol to provide new inputs, which is unacceptable in many applications such as elections where a rerun would erode trust into the election system.

In summary, while it seems possible to make MPC protocols with (strong) publicly identifiable abort or even (strong) public accountability robust by iterating/restarting the protocol, and hence, combine identifiable abort and robustness, this comes with severe performance penalties, and most importantly, is simply unacceptable for several client-server applications.

Our Contribution. Our goal therefore is to obtain the first efficient two-phase protocol that combines strong accountability and robustness while avoiding the downsides of protocol iteration. The protocol should provide efficiency comparable to other efficient two-phase protocols and full support for deployment in the client-server setting.

[3]Alternatively, one would have to generate offline material $\mathcal{O}(2^n)$ times during the offline phase to prepare for all possible subsets of parties.

TABLE I
SECURITY PROPERTIES OF RELATED TWO-PHASE PROTOCOLS

Protocol	Publicly verifiable[a]	Publicly id. abort[a]	Robust
most[b]	−	−	−/+
[10], [43]	+	−	−
[44]	+	−	+
[11], [28]	∘	∘	−
[14]	+	+	−
[41]	should work for and inherit properties of any		+
[42]	MPC protocol with identifiable abort		+
ours	+	+	+

[a] + indicates a strong property that holds even when all MPC participants/servers are corrupted, whereas ∘ holds only for partial corruption.
[b] Including, e.g., SPDZ [7], [8], most SPDZ-like protocols [12], [15], protocols with (non-public) identifiable abort [13], [40], etc.

We propose the first MPC protocol that meets all of the above goals and prove its security. Our protocol is SPDZ-like and can therefore benefit from future improvements to the components and primitives used in this class of protocols. The protocol consists of three main subprotocols, namely a setup, an offline, and an online protocol. We follow a holistic approach where we design and analyze publicly accountable and robust (sub-)protocols for all three components, unlike many other works (e.g. [10], [11], [14], [28]) which often consider the setup component to be out of scope and which assume that the keys and other setup components are already distributed at the start of the protocol. In doing so, we had to adapt almost all (and even extend some) components used in standard protocols like SPDZ [7] or the state-of-the-art lattice-based commitment scheme by Baum et al. [45], to realize an accountable and robust protocol, guarantee security in our extended setup, and remain reasonably efficient.

The core idea of our protocol is to extend traditional SPDZ-like protocols to support threshold secret-sharing and make them compatible with a suitable homomorphic commitment scheme. The commitments then allow external parties to verify the correctness of the computations of every individual server without learning any confidential information. The threshold secret-sharing scheme allows us to obtain robustness without iterating the protocol. Specifically, the threshold t determines the number of parties that are needed to reconstruct shares, and hence, if up to $n-t$ parties are malicious, the remaining set of parties can still reconstruct shares and continue the MPC protocol. Note that there is a tradeoff between privacy and robustness since t malicious parties could break privacy. For instance, in an election[4] a (small) subset of malicious parties, say $n-t = n/3$, should (and with our protocol will) not be able to abort the election and force a rerun.

[4]We note that there are also many other aspects and security properties that have to be taken into account to obtain a secure e-voting system, such as coercion resistance in high-stakes elections. Our MPC protocol therefore does not serve as a ready to use election system on its own. It can, however, be used, e.g., to instantiate the MPC component of Ordinos [20] to obtain an end-to-end verifiable tally-hiding voting system for lower-stakes elections.

Furthermore, as long as there are less than $t = 2n/3$ corrupted parties,[5] the privacy of the voters is guaranteed. Note that this trade-off does not affect the necessity for *public* verifiability or accountability: A voter or any external observer must be confident that the result is correct in all cases, no matter how many compute parties are corrupted.

A priori there are several homomorphic commitment schemes that might be considered for building a suitable protocol. Indeed, Cunningham et al. [14] achieve publicly identifiable abort in a SPDZ-like protocol by using Pedersen commitments. However, in this work we chose a lattice-based commitment scheme since it offers better synergy with the lattice-based BGV encryption scheme [46] used by SPDZ in the offline phase (see Section VII) and offers additional advantages for the offline phase and setup (Section V-B and Appendix B). There is also the prospect of making our scheme future-proof. Indeed, since we use lattice-based primitives and avoid rewinding in our proofs, we expect our protocol (possibly after small modifications) to be post-quantum secure; we, however, leave a detailed analysis of this aspect to future work.

While the general ideas of our protocol seem natural, constructing a workable protocol that combines lattice-based commitments with a robust secret-sharing scheme while retaining efficiency of the underlying SPDZ structure required us to tackle a number of challenges and to avoid pitfalls of straightforward approaches. Often, these direct approaches cause a subtle loss of security or lead to increased parameter sizes as well as a drastic loss in performance. Throughout the paper, we highlight where and why a straightforward approach either does not work or leads to a suboptimal protocol, thereby providing insights also beyond our protocol. We then propose solutions as well as optimizations that address these issues. As part of this, we develop several new constructions and techniques, some of which are interesting also in their own right, e.g.: i) We propose a modification to the state-of-the-art commitment scheme by Baum et al. [45] which allows for improving its homomorphic properties without increasing the underlying plaintext space (cf. Section VI). ii) By using the additional commitments we construct more efficient zero-knowledge proofs for verified ciphertext multiplication, key generation, and decryption (cf. Section V-B1 and Appendix B). iii) We design a computationally secure online phase that significantly increases the performance compared to a more straightforward information-theoretically hiding approach (cf. Section V-A).

To demonstrate the efficiency of our protocol, we perform a quantitative analysis of the parameters used in our protocol. As part of this analysis, we provide deeper insights in the combination of BGV with classical Pedersen commitments as compared to our lattice-based scheme, which yields smaller BGV parameters. We analyze parameters for a variant of zero-knowledge proof aggregation (combining classical and rejection sampling methods) for non-interactive zero-knowledge proofs (NIZKPs). To our knowledge, this is the first concrete and detailed analysis of this technique in a SPDZ-like setting which also provides new insights into other existing protocols of this class. Based on our analysis, we evaluate the concrete efficiency and practicality of our MPC protocol. To our knowledge, this is the first time that concrete bandwidths or benchmarks of a (SPDZ-like) protocol with publicly identifiable abort have been computed. Our results show that, with reasonably more communication, memory and runtime compared to plain SPDZ, it is possible to also obtain public accountability (and hence, publicly identifiable abort as well as public verifiability) and robustness.

In summary, we make the following contributions:

- The first two-phase MPC protocol with strong public accountability and robustness without restarts (cf. Sections III to V). Our protocol has asymptotic and concrete complexity comparable to other state-of-the-art SPDZ-like protocols with weaker properties (cf. Sections VIII and IX).
- A quantitative analysis of secure parameters as well as benchmarks for our protocol which illustrates the practicality of our protocol and provides insights that might be useful also for related SPDZ-like protocols (cf. Sections VII and IX).
- We further propose improvements of primitives and subprotocols which are of independent interest, e.g., a generalized version of the lattice-based commitment scheme by Baum et al. [45] (cf. Section VI), new ways to handle lattice-based commitments efficiently (in a SPDZ-like context; cf. Section V), and an accountable multiplication protocol for BGV ciphertexts (cf. Section V-B1).

Full details are available in the full version of the paper [47].

II. Notation

Let p be an odd prime and $\mathbb{F}_p$ be the corresponding prime field. As usual $\mathbb{Z}_q$ is the ring of integers modulo $q \in \mathbb{N}_{\geq 2}$. We use $R \coloneqq \mathbb{Z}[X]/\Phi_m(X)$ to denote integer polynomials modulo the m-th cyclotomic polynomial Φ_m. To simplify notation we restrict ourselves to $m = 2N$ a power of 2 and hence $\Phi_m(X) = X^N + 1$. Furthermore, we define R_p to be R with coefficients modulo p (we use representatives from $\{-(p-1)/2, \ldots, (p-1)/2\}$). Elements of R and R_p can also be seen as N-tuples of $\mathbb{Z}$ and $\mathbb{Z}_p$, respectively. This also induces the standard L^k-norm for $k \in \{1, 2, \ldots, \infty\}$ of R by taking the respective norm of the coefficient vector.

We use lowercase bold and uppercase bold letters for vectors and matrices, e.g. $\boldsymbol{x}, \mathbf{M}$. We write $\boldsymbol{x}[i]$ and $\mathbf{M}[i, j]$ to index the i-th and (i, j)-th element of vectors and matrices, respectively, where indices start from zero. The $n \times m$ zero-matrix will be denoted by $\mathbf{0}_{n \times m}$; the $n \times n$ identity-matrix by $\mathbf{I}_n$. We use $\boldsymbol{x} \xleftarrow{\$} U(S)$ to say that $\boldsymbol{x}$ is sampled uniformly at random from a set S. D_σ is used instead of U if $\boldsymbol{x}$ is sampled from a discrete Gaussian distribution with standard deviation σ.

We use $\mathcal{P}_i \in \mathcal{P} \coloneqq \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ to identify a compute party, i.e., a server in the client-server terminology. We use $\mathcal{C} \subseteq \mathcal{P}$ for the set of statically corrupted (compute) parties and $\mathcal{H} \coloneqq \mathcal{P} \setminus \mathcal{C}$ for the set of honest (compute) parties. Input parties (which can be clients, servers, or a mixture of both) are denoted with $\mathcal{I}_i \in \mathcal{I}$ where we identify one input party with one input. Input parties can also be statically corrupted;

[5] Corrupted parties include *malicious* and *honest-but-curious* parties.

all of our results are independent of the exact set of corrupted input parties.

An arithmetic circuit consists of addition and multiplication gates, where inputs and outputs of gates are identified by unique identifiers ("id_x"). A valid circuit can be deterministically traversed such that every identifier is *set* only once (as external input or the output of some gate) and, whenever a circuit needs to be computed, then the identifiers used as inputs are already defined.

We write $[x]_i$ to denote the share of party $\mathcal{P}_i$ (obtained by secret-sharing the value x). We consider Shamir secret-sharing where a share $[x]_i := f_x(i) = x + \sum_{l=1}^{t-1} i^l \cdot c_l$ is the evaluation of a polynomial f_x with constant term x and the remaining coefficients c_l sampled uniformly at random. This is a t-out-of-n secret-sharing scheme, i.e., t shares are sufficient to reconstruct x. We also use full-threshold (or "additive") secret-sharing, which is defined as $x = \sum_{i=1}^{n} x_i$ for a secret x and shares x_i. We explicitly mention whenever we use this n-out-of-n scheme; otherwise, we use Shamir secret-sharing.

As many SPDZ-like protocols, e.g., [7], [8], [15], [16], we use the BGV encryption scheme [46]. Specifically, we use an instantiation that is somewhat homomorphic, i.e., allows for addition and up to one multiplication of (plaintexts in) ciphertexts. We present details of BGV, as far as needed, while describing the offline phase of our protocol (cf. Section V-B, with more details in Appendix A-B). We refer to the commitment scheme of Baum et al. [45] as BDLOP scheme in what follows and give a detailed description in Section VI. $\mathsf{Enc}_k(x)$ denotes a ciphertext of x constructed with randomness $\mathsf{R_E}(x)$ and $\mathsf{Com}_{par}(x, \mathsf{R_C}(x))$ $\big($or just $\mathsf{Com}_{par}(x)\big)$ denotes a commitment for x with randomness $\mathsf{R_C}(x)$. Hence, $(x, \mathsf{R_C}(x))$ is the decommitment/opening for $\mathsf{Com}(x)$. Verify_{par} is the corresponding verification algorithm. We omit the public key k and the commitment parameters *par* if they are clear from the context. To simplify notation, we define additions of commitments and public (plaintext) values as $\mathsf{Com}(x) + c := \mathsf{Com}(x) + \mathsf{Com}(c, 0)$. Additionally, we define $\top$ to be an "invalid" commitment for which every linear operation yields $\top$ (e.g., $\top + c = \top$) and $\mathsf{Verify}(\cdot, \cdot, \top) = 0$. Consequently, we also define ZKPs (such as in Fig. 5, Line 12) to always fail verification if statements for $\top$ are proven.

III. Overview

As mentioned in the introduction, our protocol builds on and extends SPDZ. In particular, it also consists of an offline and online phase, where the former computes correlated randomness for the latter. We present our protocol in such a way that it can be understood without prior knowledge of SPDZ. However, a short summary of SPDZ is given in Appendix A-A. We want the inputs of our protocol to be (BGV) ciphertexts, i.e., clients/input parties can simply encrypt their secret inputs and then provide the resulting ciphertexts to the servers/compute parties of our protocol. We think this provides great utility in the client-server setting but other constructions are possible as well, e.g., privately opening input masks to clients who then use this to publish their masked inputs as in [10]. The servers first transform the ciphertext into a secret-sharing and then use SPDZ-like techniques to compute an arithmetic circuit on those shares. Then they recombine shares to compute outputs. For simplicity of presentation (but without loss of generality; cf. Remark 1), we consider the case where outputs are public, i.e., may be revealed to everyone. We use bulletin boards to publish data just as almost all other protocols with public verifiability/accountability, e.g., [10], [11], [14], [28]. While it might be desirable to not handle all communication through bulletin boards (to improve efficiency), bulletin boards seem to be necessary so that communication is transparent for all parties, importantly, including verifiers.

Online Phase: Like other SPDZ-like protocols, we can compute any arithmetic circuit with a linear secret-sharing scheme by utilizing Beaver's technique [48] (we explain this later in Section V-A). Our online phase becomes robust by using a threshold secret-sharing scheme (with the mentioned tradeoff for privacy). To get accountability, we add publicly known commitments for each party's shares. With this, everyone can check if the parties computed results and intermediate results correctly by verifying the decommitments on the bulletin board. We describe the resulting protocol for the online phase in Section V-A.

There are two main hurdles in designing this online protocol: Firstly, we have to transform the initial ciphertexts into shares, including commitments on those shares, in a publicly accountable manner. Secondly, adding a lattice-based commitment scheme to SPDZ introduces several new challenges (in the security proofs and in practice). For example, it needs to offer a sufficient homomorphic structure to support the Beaver multiplication sub-step for circuits of practical sizes. As it turns out, existing lattice-based schemes do not provide all properties required by our protocol simultaneously. In Section VI, we therefore modify the state-of-the-art BDLOP scheme. Our modification improves the homomorphic properties of BDLOP (both to support larger arithmetic circuits and to make multiplications with Beaver triples secure in the first place), which is of independent interest. Additionally, we show (in Section VII) that we can drastically reduce the amount of data communicated in the online phase (for this and similar commitment schemes) by replacing the information-theoretically secure online phase with a computationally secure one.

Offline Phase: The online phase relies on correctly generated correlated randomness from the offline phase, which therefore also needs to be publicly accountable and robust. We propose a protocol for the offline phase in Section V-B that uses published NIZKPs to show correctness of the critical steps. There are various hurdles we had to overcome in order to develop this protocol, e.g., the protocol has to allow for a simulation-based MPC security proof, requires NIZKPs that also work for our modified commitment scheme, and needs to retain a high efficiency even with several additional NIZKPs. To achieve good efficiency, we employ state-of-the-art NIZKPs utilizing commitments for verified ciphertext multiplication, key generation and decryption. We construct our protocol in a suitable way to keep (encryption and commitment) parameters small and support additional features, e.g. robustness. Furthermore, for certain other NIZKPs employed by our offline protocol (e.g., the one for showing correct encryption), we use

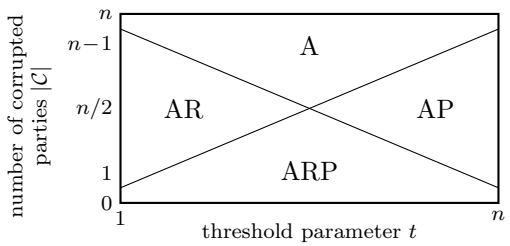


Fig. 1. Security guarantees of our protocol. Public accountability is indicated with "A" (recall that this also implies public verifiability), robustness/guaranteed output delivery with "R", and full privacy with "P".

a variant of ZKP aggregation that combines classical aggregation [49] and rejection sampling [50] to improve efficiency. We evaluate and compare this technique in terms of resulting parameter sizes in Section VII. As mentioned, this is the first concrete analysis of this technique in a SPDZ-like protocol and thereby provides insights that are also useful within this wider class of protocols. We further propose a new multiplication algorithm for BGV ciphertexts (cf. Section V-B1) which is adapted to the specific constructions used in our protocols and which can be used to bootstrap the distributed key generation algorithm (see below) from the linear homomorphic version of BGV. This accountable multiplication algorithm is likely to prove useful as a component for protocols beyond ours.

Setup Components: The offline phase requires a distributed key generation and a distributed decryption protocol. While these components are often considered out of scope, in our case it is crucial that they are also publicly accountable. We therefore provide publicly accountable robust algorithms for both components in the full version. A major challenge was to find solutions which avoid the problems that can be caused by the Shamir secret-sharing scheme. For example, naive solutions might not achieve robust decryption or can introduce a cubic factor in the runtime of decryption.

Security Properties: Fig. 1 summarizes the security properties that our protocol (including all of the above-mentioned sub-protocols) achieves depending on the number of corrupted compute parties/servers $|\mathcal{C}| \leq n$ and the threshold $1 \leq t \leq n$ of parties that can decrypt/recombine shares. These results hold independently of the number of corrupted external input parties, where by external we mean that the input party is not a compute party/server, i.e., it is a pure client. The parameter t can be fine tuned depending on the application to offer better privacy or better robustness. We emphasize that even in those cases where privacy and/or robustness no longer hold, our protocol still provides public accountability and thereby public verifiability, including the case that *all* compute parties/servers are malicious.

Next, we first summarize our security model along with the central security properties (Section IV) and then present our protocol (Section V).

IV. Security Model

In this section we define our security properties: robustness in Definition 1 and public accountability in Definition 2. We will prove the security of our protocols in the universal composability (UC) [51] setting, i.e., our protocols are indistinguishable from idealized protocols (functionalities) that naturally satisfy our security properties. We also use several functionalities to model setup assumptions: All communication is handled through a bulletin board modelled by $\mathcal{F}_{\text{BB}}$ and $\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{PKI}}$ are used to model sampling from a common reference string (CRS), a random oracle, and a public key infrastructure, respectively.[6]

Our protocols should achieve strong public accountability and robustness. That is, for such a protocol Π, we additionally consider a judge $\mathcal{J}$ – a polynomial time algorithm with access to a transcript of all public information (e.g., the bulletin board communication and the CRS) – that has two outputs: A set of parties that are blamed for misbehavior and an overall protocol output, which can be $\perp$ (i.e., "abort") if a run cannot be verified. This output is called a *verdict*. Using the judge, the security properties are defined as follows:

Definition 1 (Robustness)**.** *Let $1 \leq t \leq n$ be a threshold parameter. Let f be a circuit with inputs $\boldsymbol{x}$. We call Π t-*robust *if $\mathcal{J}$ outputs a correct result (in particular, no abort) with overwhelming probability (over all protocols runs of Π, with polynomially bounded environments $\mathcal{E}$) whenever $|\mathcal{P} \setminus \mathcal{C}| \geq t$.*

Definition 2 (Public Accountability)**.** *Let f be a circuit with inputs $\boldsymbol{x}$. A t-*robust *protocol Π is called* publicly accountable *if the following holds (except with negligible probability over all protocols runs of Π, with polynomially bounded environments $\mathcal{E}$): i) $\mathcal{J}$ outputs $\mathcal{M} \subseteq \mathcal{C} \subseteq \mathcal{P}$ (i.e., no honest party is falsely blamed). ii) If $\mathcal{J}$ outputs $\boldsymbol{y}$, it is correct, i.e., $\boldsymbol{y} = f(\boldsymbol{x})$. iii) If $\mathcal{J}$ outputs abort, then $|\mathcal{M}| > n - t$.*

Since we are working in the UC setting, we specify three ideal functionalities, one each for the setup, offline, and online protocols, that meet both the standard properties such as privacy as well as the additional ones from Definitions 1 and 2 by definition. Any real protocol that realizes such an ideal functionality then has all of these properties as well. We provide the ideal functionality for the online phase in Fig. 2 with the other ones available in Fig. 12 and in the full version. The set of possible verdicts

$$\mathcal{V} = \{A \subseteq \mathcal{C}, b \in \mathcal{B} \mid (b = \texttt{abort}) \Rightarrow |A| > n - t\} \quad (1)$$

with $\mathcal{B} = \{\texttt{ok}, \texttt{abort}\}$ is used to define the ideal functionalities. With this, the functionalities only accepts messages from the adversary that identify enough malicious parties to justify the abort of a t-robust protocol.

In our descriptions we add the modifier "(once)" to some *phases* of functionalities and protocols to say that this phase is run once and subsequent calls to it are ignored. Phases are strictly ordered, i.e., in Fig. 4, the preprocessing comes before the input phase, which comes before the compute phase.

Remark 1. *Observe that the definitions of robustness and public accountability cover a general setup, including SPDZ-like protocols but also realizations with less phases or realizations that further subdivide phases. Further extensions,*

[6]For clarity we work with all four functionalities $\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{PKI}}$. We remark however that this set is not minimal, e.g., $\mathcal{F}_{\text{CRS}}$ can be implemented with a call to $\mathcal{F}_{\text{RO}}$. For more details, see the full version.

Prepare (once): On input $(\texttt{prep})$ by each $\mathcal{P}_i \in \mathcal{P}$:
  Send $(\texttt{prep})$ to adversary $\mathcal{A}$ and receive $(\mathcal{M}, \beta) \in \mathcal{V}$ (cf. Eq. (1)).
Input (once): On input $(\texttt{input}, x_j)$ by each $\mathcal{I}_j \in \mathcal{I}$ and input $(\texttt{input})$ by each $\mathcal{P}_i \in \mathcal{P}$:
  Get x_j for corrupted $\mathcal{I}_j$ from $\mathcal{A}$.
  Pack all x_j into $\boldsymbol{x}$.
  if $\beta = \texttt{ok}$ **then**
    Send $(\texttt{input})$ to $\mathcal{A}$. If $|\mathcal{C}| \geq t$, also send $\boldsymbol{x}$.
    Receive $(\mathcal{M}', \beta') \in \mathcal{V}$ from $\mathcal{A}$ with $\mathcal{M}' \supseteq \mathcal{M}$ (overriding the previous values of $\mathcal{M}$ and β).
Compute (once): On input $(\texttt{comp}, f)$ by each $\mathcal{P}_i \in \mathcal{P}$:
  if $\beta = \texttt{ok}$ **then**
    Compute the result $\boldsymbol{y} \coloneqq f(\boldsymbol{x})$.
    Send $(\texttt{comp}, \boldsymbol{y})$ to $\mathcal{A}$.
    Receive $(\mathcal{M}'', \beta'') \in \mathcal{V}$ from $\mathcal{A}$ with $\mathcal{M}'' \supseteq \mathcal{M}$ (overriding the previous values of $\mathcal{M}$ and β).
    if $\beta = \texttt{ok}$ **then reply** $\mathcal{M}, \boldsymbol{y}$ to $\mathcal{P}_i$.
  reply $\mathcal{M}, \perp$ *to* $\mathcal{P}_i$.
Audit: On input $(\texttt{audit}, \texttt{comp}, f)$ by $\mathcal{J}$:
  if $\beta = \texttt{ok}$ **then reply** $\mathcal{M}, \boldsymbol{y}$ **else reply** $\mathcal{M}, \perp$.

Fig. 2. Online functionality $\mathcal{F}_{\text{online}}$.

like the support for private outputs can be done by standard techniques (the parties compute $f'(\boldsymbol{x}, r_i) = f(\boldsymbol{x}) + r_i$ where party $\mathcal{P}_i$ has an additional input r_i and is thus able to compute $f(x)$ while no other party can do so).

V. Our Accountable Robust Protocol

We now describe our accountable and robust MPC protocol and prove its security. The focus in this section will be on the online protocol (Section V-A) and the offline protocol (Section V-B). We present the setup protocol in Appendix B. All three protocols are accountable and robust. The respective security proofs can be found in Section V-A and in Appendix D.

A. Accountable Robust Online Phase

The online phase of our protocol is depicted in Fig. 4. It uses $\mathcal{F}_{\text{offline}}$ for the offline phase (main properties are discussed later; we provide a realization in Section V-B). Additionally $\mathcal{F}_{\text{CRS}}$ is used to sample the commitment parameters *par* from the common reference string (CRS), and $\mathcal{F}_{\text{BB}}$ (bulletin board) is used for all communication.

To perform efficient computations in the online phase, the offline phase prepares correlated randomness in advance following Beaver's classical approach [48]. A *view* of a shared value x is

$$\langle x \rangle_i \coloneqq ([x]_i, \mathsf{R_C}([x]_i), \mathsf{Com}([x]_1), \ldots, \mathsf{Com}([x]_n))$$

for each party $\mathcal{P}_i$ with $\mathsf{Com}_{par}([x]_j, \mathsf{R_C}([x]_j)) = \mathsf{Com}([x]_j)$ for all parties $\mathcal{P}_j$, i.e., parties hold commitments to shares of all parties and decommitments for their own share. The shares are computed with a t-out-of-n secret-sharing scheme (t parties are required to reconstruct the secret). We also define a public view $\langle x \rangle_{\text{audit}}$ used by the judge $\mathcal{J}$, consisting of only the commitments. For linear secret-sharing schemes, linear operations on views are done as in (2) to (4) (cf. Fig. 3).[7] The offline phase has to produce views for random $\boldsymbol{r}, \boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{b}$. As mentioned, the initial inputs are given in the form of BGV ciphertexts, i.e., $\mathsf{Enc}(x_i)$, $\mathcal{I}_i \in \mathcal{I}$, which allows external clients to easily provide secret inputs to our protocol. For more details on BGV encryption scheme see also Appendix A-B.

We use an accountable and robust subroutine that we developed for our offline protocol to (privately) compute a vector of masked inputs $\boldsymbol{m} \coloneqq \boldsymbol{x} - \boldsymbol{r}$ from the list of encrypted inputs. This allows us to transform ciphertexts into secret-sharings in an accountable manner, solving one of the main tasks in designing the online protocol.

Now, let us explain Π_{online}. After invoking the offline phase to get the initial views and after processing the inputs, the input-independent views $\langle \boldsymbol{r} \rangle_i$ can now be used to get views for the inputs. One can compute these views as $\langle \boldsymbol{x} \rangle_i \coloneqq \langle \boldsymbol{r} \rangle_i + \boldsymbol{m}$. Note that at this point, an initial set of malicious parties might have already been identified by $\mathcal{F}_{\text{offline}}$ while computing the initial views or while computing $\boldsymbol{m}$ from the inputs.

Similarly, the provided multiplication triples $\langle a \rangle_i, \langle b \rangle_i, \langle c \rangle_i$ can be used in the online phase. They are used to multiply with only linear operations as follows:

$$\langle x \cdot y \rangle_i \coloneqq \langle c \rangle_i + u \cdot \langle a \rangle_i + v \cdot \langle b \rangle_i + u \cdot v \tag{5}$$

for values x and y that should be multiplied. $u \coloneqq x - b, v \coloneqq y - a$ are *opened* values (described below). Further linear operations (additions, subtractions, and multiplications with publicly known constants) are done locally on views. Hence, our lattice-based commitment scheme has to provide a level of additive homomorphic structure that not only supports the above multiplication but more generally is suitable for practical circuit sizes that contain large numbers of additions (and multiplications). Note that the multiplicative depth of the circuit is not relevant for the commitment scheme but rather the maximal amount of linear operations between multiplication. This is because Beaver multiplication "resets" commitments as can be seen in (5) (the commitments of $\langle x \cdot y \rangle_i$ are a linear combination of commitments of $\langle a \rangle_i, \langle b \rangle_i, \langle c \rangle_i$, i.e., "fresh" commitments from the preprocessing). The chosen BDLOP scheme, which fits our (other) requirements best, does not satisfy this requirement, since it loses its security properties after just one Beaver multiplication. We discuss and solve this issue by proposing a generalization of BDLOP in Section VI.

The only operation left for the online phase is *opening* views, which is required for multiplications (u and v above are results of the openings for $\langle x - b \rangle_i$ and $\langle y - a \rangle_i$) and for obtaining the final outputs of the circuit. To open a view, every party $\mathcal{P}_i$ publishes the decommitment/opening contained in their view and other parties $\mathcal{P}_j$ check if the decommitment verifies w.r.t. the locally computed commitment for $\mathcal{P}_i$ in $\mathcal{P}_j$'s view. Shares for parties that could not provide valid decommitments (or were identified by the offline phase as malicious) are ignored in the reconstruction. Note that all

[7] With Shamir secret-sharing, addition of shares and public values is done by adding the value to the share of each party. For the much-used full-threshold secret-sharing, it should be only added to the share of a single party.

$$\langle x\rangle_i + \langle y\rangle_i := ([x]_i + [y]_i, \mathsf{R_C}([x]_i) + \mathsf{R_C}([y]_i), \mathsf{Com}([x]_1) + \mathsf{Com}([y]_1), \ldots, \mathsf{Com}([x]_n) + \mathsf{Com}([y]_1)) \quad (2)$$

$$\langle x\rangle_i + c := ([x]_i + c, \mathsf{R_C}([x]_i), \mathsf{Com}([x]_1) + c, \ldots, \mathsf{Com}([x]_n) + c) \quad (3)$$

$$c \cdot \langle x\rangle_i := (c \cdot [x]_i, c \cdot \mathsf{R_C}([x]_i), c \cdot \mathsf{Com}([x]_1), \ldots, c \cdot \mathsf{Com}([x]_n)) \quad (4)$$

Fig. 3. Linear operations on views (for public constant values c). Subtraction works analogously to addition.

1 ***Prepare*** **(once):** On input $(\texttt{prep})$ by each $\mathcal{P}_i \in \mathcal{P}$:
2 Parties setup commitment parameters *par* via $\mathcal{F}_{\text{CRS}}$.
3 Parties send $(\texttt{prep}, par)$ to $\mathcal{F}_{\text{offline}}$ and get the (initial) set of malicious parties $\mathcal{M}$, sufficient input views $\langle \boldsymbol{r}\rangle_i$ and triples $\langle \boldsymbol{a}\rangle_i, \langle \boldsymbol{b}\rangle_i, \langle \boldsymbol{c}\rangle_i$.
4 **if** $\mathcal{F}_{\text{offline}}$ *outputs* $\perp$ **then abort**.
5 ***Input*** **(once):** On input $(\texttt{input}, x_j)$ by each $\mathcal{I}_j \in \mathcal{I}$ and input $(\texttt{input})$ by each $\mathcal{P}_i \in \mathcal{P}$:
6 The parties forward their inputs to $\mathcal{F}_{\text{offline}}$ and each $\mathcal{P}_i$ gets $\mathcal{M}'$ and masks $\boldsymbol{m}$. $\mathcal{M}'$ is added to $\mathcal{P}_i$'s $\mathcal{M}$.
7 **if** $\boldsymbol{m} = \perp$ **then abort**.
8 Input views are computed as $\langle \boldsymbol{x}\rangle_i := \langle \boldsymbol{r}\rangle_i + \boldsymbol{m}$.
9 ***Compute*** **(once):** On input $(\texttt{comp}, f)$ by each $\mathcal{P}_i \in \mathcal{P}$:
10 Assign the identifiers id_x for inputs of f to $\langle \boldsymbol{x}\rangle_i$.
11 **foreach** *gate* $g \in f$ *in topological order* **do**
12 **case** g *is linear* **do**
13 Compute g locally as in Fig. 3.
14 **case** $(\texttt{mul}, \text{id}_x, \text{id}_y, \text{id}_z)$ **do**
15 Get the next triple $\langle a\rangle_i, \langle b\rangle_i, \langle c\rangle_i$.
16 $\langle u\rangle_i := \langle x\rangle_i - \langle b\rangle_i$ and $\langle v\rangle_i := \langle y\rangle_i - \langle a\rangle_i$.
17 Open $\langle u\rangle_i$ and $\langle v\rangle_i$ with $\texttt{Open}$ (cf. Fig. 5).
18 $\langle z\rangle_i := \langle c\rangle_i + u \cdot \langle a\rangle_i + v \cdot \langle b\rangle_i + u \cdot v$.
19 **case** $(\texttt{output}, l, \text{id}_x)$ **do**
20 $x_l := \texttt{OpenOutput}(\langle x\rangle_i)$ (cf. Fig. 5).
21 Pack all outputs x_l of $\texttt{output}$-gates into $\boldsymbol{y}$.
22 **reply** $\mathcal{M}, \boldsymbol{y}$.
23 ***Audit*****:** On input $(\texttt{audit}, \texttt{comp}, f)$ by $\mathcal{J}$:
24 Audit the ***Prepare*** and ***Input*** phases (of $\mathcal{F}_{\text{offline}}$).
25 Perform the ***Compute*** phase on public data (compute on $\langle\cdot\rangle_{\text{audit}}$-representation of views, only retrieve data from $\mathcal{F}_{\text{BB}}$, and only verify NIZKPs).

Fig. 4. Online protocol Π_{online}.

operations in an arithmetic circuit become linear if we realize multiplications as in (5). Hence the commitments for every opening can be computed locally (from the view provided by the offline functionality and the already opened and verified intermediate results).

SPDZ-like protocols that already use commitments, like the protocols of Baum et al. [10] or Cunningham et al. [14], usually follow the straightforward approach of handling all openings the same way, i.e., openings during multiplication and openings of outputs are simply decommitted. We decided to treat final outputs differently in order to achieve better performance (see Section IX). In our protocol (see Fig. 5), parties pick new commitments for their final shares and prove in zero-knowledge that these and the (publicly known) commitments in other parties' views commit to the same share. Then, the resulting new views with these commitments are opened. This way, we do not require perfectly/statistically hiding commitments and equivocation for commitments to prove security of the online phase as in the related protocols [10], [14]. There, equivocation is necessary for the simulation-based security proof and the perfect/statistical hiding property is needed so the simulator can equivocate to every possible plaintext. Instead, we can use commitments that are computationally hiding *and* computationally binding with tighter commitment parameters (more in Sections VII and IX). No longer requiring equivocation and statistically hiding commitments is the main factor why our performance can be substantially better. To illustrate this advantage, we can construct a protocol version Π_{equiv} that uses the same openings for multiplications and outputs (and thus requires equivocation).[8] The two versions are then compared in Section IX. A security proof of Π_{equiv} is included in the full version.

Any (external) auditor $\mathcal{J}$ can recompute the above operations on commitments and openings (i.e., everything except for operations directly performed on the individual secret shares $[x]_i$ of each party, or the respective randomness). It blames any party that did not provide a valid opening. This provides accountability because malicious parties can alter the output only by providing an incorrect share during an opening phase (i.e., during multiplication or while opening the final output). By the binding property of the commitment scheme or the soundness of the ZKPs, this would be detected. Formally we obtain:

Theorem 1. *The protocol Π_{online} is a publicly accountable t-robust MPC protocol for arithmetic circuit evaluation. That is, Π_{online} UC-realizes the functionality $\mathcal{F}_{\text{online}}$ in the $(\mathcal{F}_{\text{offline}}, \mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{BB}})$-hybrid model under the assumption that the used homomorphic commitment scheme is computationally binding and hiding.*

Proof (Sketch). The proof can be split in two parts, depending on the number of corrupted parties. If $|\mathcal{C}| \geq t$, the proof is trivial as the simulator gets all necessary data from the functionality. Note that accountability is still given, even if all parties are corrupted, as the commitment parameters are sampled from the CRS and an adversary is unable to break the commitments or the ZKPs.

For $|\mathcal{C}| < t$, the simulator sets up a local simulation of the offline phase, samples commitment parameters and

[8] Π_{equiv} is mostly equivalent to Π_{online} but $\texttt{Open}$ is used instead of $\texttt{OpenOutput}$ for the outputs in Line 20 of Fig. 4. The parameters of the commitment scheme, however, will be vastly different as discussed in Section IX.

macro $\texttt{Open}(\langle v\rangle_i, \mathcal{M}' = \emptyset)$**:**
  Send $[v]_i$ and $\mathsf{R_C}([v]_i)$ to $\mathcal{F}_{\text{BB}}$.
  Retrieve $[v]_j$ and $\mathsf{R_C}([v]_j)$ from other $\mathcal{P}_j \in \mathcal{P}$.
  Add all $\mathcal{P}_j$ to $\mathcal{M}'$ and to $\mathcal{M}$ for which $\mathsf{Verify}([v]_j, \mathsf{R_C}([v]_j), \mathsf{Com}([v]_j)) = 0$.
  if $|\mathcal{P} \setminus \mathcal{M}'| < t$ **then abort**.
  Reconstruct v while ignoring shares of $\mathcal{P}_j \in \mathcal{M}'$.
  return v.
macro $\texttt{OpenOutput}(\langle v\rangle_i)$**:**
  Commit to $[v]_i$ to get $\mathsf{Com}([w]_i)$.
  Prove in ZK that $\mathsf{Com}([v]_i)$ and $\mathsf{Com}([w]_i)$ contain the same plaintext. Let the NIZKP be z_i.
  Send z_i to $\mathcal{F}_{\text{BB}}$ and retrieve z_j from other $\mathcal{P}_j \in \mathcal{P}$.
  Add all $\mathcal{P}_j$ to a new set $\mathcal{M}'$ and to $\mathcal{M}$ where verification of z_j failed.
  return $\texttt{Open}(\langle w\rangle_i, \mathcal{M}')$.

Fig. 5. Opening subprotocols for Π_{online} (cf. Fig. 4) at $\mathcal{P}_i \in \mathcal{P}$.

picks random data for all honest input parties. Controlling the simulation allows it to extract data for corrupted parties.

Up until the final openings, the simulation and a real protocol instance are indistinguishable because only random data is communicated ((shares of) uniformly random masked values and decommitments with randomness chosen as in the protocol). For the final openings of the outputs, the simulator first gets the outputs from the functionality. With knowledge of the shares of corrupted parties (which can be computed locally by the simulator), shares for the honest parties can be chosen in a way that reconstructs the same outputs as given by the functionality. Afterwards, the required ZKP can be faked for honest parties utilizing the random oracle. The final decommitments are then distributed equally to the ones in the protocol as they are constructed in the same way (fresh randomness and shares that reconstruct to the outputs). Finally the simulator collects all corrupted parties for which messages failed verification in $\mathcal{M}$ and sets $\beta := \texttt{abort}$ if the computation in the simulation aborted, or sets $\beta := \texttt{ok}$ otherwise. The simulator sends $(\mathcal{M}, \beta)$ to $\mathcal{F}_{\text{online}}$ and $\mathcal{F}_{\text{online}}$ provides the final output. For the complete proof, see the full version. There, we also show how one can guarantee security of the straightforward construction Π_{equiv} that utilizes statistically hiding commitments and trapdoors for equivocation. An added difficulty there (compared to [10], [14]) is that the distribution of the final decommitments might reveal that we are in a simulation since decommitments using a trapdoor and the decommitments of linear combinations of views (as in the protocol) have differently distributed randomness. □

Remark 2. *Our proof shows that* every *party in our protocol that sends a message which cannot be verified is identified and blamed by our judge $\mathcal{J}$, and that corrupted parties can only prevent a correct output by sending such a message. Hence, in our protocol* every *corrupted party that tries to manipulate the output is identified and blamed, even if the protocol does not actually abort. This is a stronger security property than formally required by Definition 2, which only requires that the judge outputs a sufficiently large number of corrupted parties, not everyone, and only if the protocol aborts. We note that it does not seem possible to formalize this stronger security property via a general ideal functionality. Such a functionality is not aware of whether a corrupted party actually misbehaves or still follows the protocol honestly since this is part of the simulation within the simulator.*

B. Accountable Robust Offline Phase

As described above, the offline protocol (see Fig. 6) has to produce *views* of random values, views of Beaver triples, and a masked input $\boldsymbol{m}$, all in an accountable and robust way. It uses $\mathcal{F}_{\text{CRS}}$ to compute a second set of commitment parameters *par*′ to commit to (components of) ciphertexts, encryption randomness, and commitment randomness (of *par*-commitments). Commitments and commitment randomness w.r.t. *par*′ is denoted with $\mathsf{Com}'(\cdot)$ and $\mathsf{R}'_{\mathsf{C}}(\cdot)$, respectively. Commitments with this second parameter set need different properties than the ones that use *par* – they need to be statistically binding (and computationally hiding) so the simulator in our simulation-based proof can extract decommitments of corrupted parties. Other papers [10], [14] accomplish this by reuse of the BGV encryption scheme. In our situation, however, certain values that we want to be able to extract, e.g., randomness for commitments, might be larger than the plaintext space of BGV, so encryption might not correctly recover these values. We address this by using the same commitment scheme with different parameters *par*′ that has a sufficiently large input space to commit to the above mentioned values. We present the specific scheme in Section VI.

We obtain and leverage several synergy effects due to these commitments (using both *par* and *par*′). Firstly, using lattice-based commitments (with *par*) we can consistently associate a ciphertext and a commitment to the same plaintext, to create so-called "committing ciphertexts", since both the encryption scheme and the commitment scheme can be constructed on the same plaintext space R_p. In contrast, a commitment scheme on $\mathbb{F}_p$ will result in N commitments per ciphertext (at least if used naively on the coefficients of the plaintext); generally schemes on other plaintext spaces than BGV will need a suitable transformation that would still have to guarantee our security properties. These "committing ciphertexts" allow more natural descriptions and ZKPs. The same can be said for commitments with *par*′. Secondly, the use of *par*-commitments allows us to choose BGV parameters that are independent of the parameters of the commitment scheme (cf. Section VII). Lastly, we can use the *par*′-commitments to make ZKPs more efficient (cf. Section V-B1).

Another primitive used in the offline phase is the setup component $\mathcal{F}_{\text{PK}}$ for accountable threshold cryptography (in particular, this is also robust; its realization is presented in Appendix B), i.e., computing parties can compute a public key k and can perform distributed decryption together. When we say "decryption" in what follows, we mean this kind of decryption where parties that misbehave during any decryption are detected and the protocol aborts if too many parties misbehaved, considering the misbehavior during this decryption.

To get the necessary views, we have to generate shares and commitments thereof. The commitments should be public,

macro $\texttt{Decrypt}(\mathsf{Enc}(x))$**:**
  Decrypt with $\mathcal{F}_{\text{PK}}$ to get the decryption x and $\mathcal{M}'$ for this decryption. Add $\mathcal{M}'$ to each $\mathcal{P}_i$'s $\mathcal{M}$.
  if $x = \perp$ **then abort else return** x.
***Prepare* (once):** On input $(\texttt{prep}, par)$ by each $\mathcal{P}_i \in \mathcal{P}$:
  Parties setup commitment parameters par' via $\mathcal{F}_{\text{CRS}}$.
  Parties setup a key k via $\mathcal{F}_{\text{PK}}$ and get a set $\mathcal{M}$.
  if $k = \perp$ **then abort**.
  Sample coefficients $\mathbf{W}_i \xleftarrow{\$} U(R_p^{(I+3\cdot M)\times t})$ to (later) construct shares and masks $\boldsymbol{y}_i \xleftarrow{\$} U(R_p^{I+3\cdot M})$.
  Encrypt these value to get $\mathsf{Enc}(\mathbf{W}_i), \mathsf{Enc}(\boldsymbol{y}_i)$.
  Commit to $\boldsymbol{y}_i$ with $\mathsf{R_C}(\boldsymbol{y}_i)$ to get $\mathsf{Com}(\boldsymbol{y}_i)$.
  Commit to the decommitments $\boldsymbol{y}_i, \mathsf{R_C}(\boldsymbol{y}_i)$ with par' to get $\mathsf{Com}'(\boldsymbol{y}_i), \mathsf{Com}'(\mathsf{R_C}(\boldsymbol{y}_i))$.
  Prove Line 9 to 11 in ZK. Let the NIZKP be z_i.
  Send z_i to $\mathcal{F}_{\text{BB}}$ and retrieve z_j for other $\mathcal{P}_j \in \mathcal{P}$.
  Add $\mathcal{P}_j$ to a new set $\mathcal{M}'$ and to $\mathcal{M}$ where verification of z_j failed.
  Use $\mathsf{Com}(\boldsymbol{y}_j) = \top$ for $\mathcal{P}_j \in \mathcal{M}'$ and combine coefficients to $\mathsf{Enc}(\mathbf{W}) \coloneqq \sum_{\mathcal{P}_j \in \mathcal{P}\setminus\mathcal{M}'} \mathsf{Enc}(\mathbf{W}_j)$.
  foreach $\mathcal{P}_j \in \mathcal{P} \setminus \mathcal{M}'$ **do**
    Define the encrypted share of $\boldsymbol{v} = \mathbf{W}[\,\cdot\,, 0]$ as $\mathsf{Enc}([\boldsymbol{v}]_j) \coloneqq \sum_{l=0}^{t-1} j^l \cdot \mathsf{Enc}(\mathbf{W})[\,\cdot\,, l]$.
    $\boldsymbol{m}_j \coloneqq \texttt{Decrypt}(\mathsf{Enc}(\boldsymbol{y}_j) - \mathsf{Enc}([\boldsymbol{v}]_j))$.
  Construct the view $\langle \boldsymbol{v} \rangle_i \coloneqq (\boldsymbol{y}_i - \boldsymbol{m}_i, \mathsf{R_C}(\boldsymbol{y}_i), \mathsf{Com}(\boldsymbol{y}_1) - \boldsymbol{m}_1, \ldots, \mathsf{Com}(\boldsymbol{y}_n) - \boldsymbol{m}_n)$.
  Split $\langle \boldsymbol{v} \rangle_i$ and $\mathsf{Enc}(\boldsymbol{v})$ in parts of size I, M, M, M to get views and ciphertexts for $\boldsymbol{r}, \boldsymbol{a}, \boldsymbol{b}$, and $\boldsymbol{d}$.
  Compute $\mathsf{Enc}(\boldsymbol{c})$ with SHE or $\texttt{Multiply}$ (Fig. 7).
  $\langle \boldsymbol{c} \rangle_i \coloneqq \langle \boldsymbol{d} \rangle_i + \texttt{Decrypt}(\mathsf{Enc}(\boldsymbol{c}) - \mathsf{Enc}(\boldsymbol{d}))$.
  reply $\mathcal{M}, \langle \boldsymbol{r} \rangle_i, \langle \boldsymbol{a} \rangle_i, \langle \boldsymbol{b} \rangle_i, \langle \boldsymbol{c} \rangle_i$ to $\mathcal{P}_i$.
***Input* (once):** On input $(\texttt{input}, x_j)$ by each $\mathcal{I}_j \in \mathcal{I}$ and input $(\texttt{input})$ by each $\mathcal{P}_i \in \mathcal{P}$:
  Each $\mathcal{I}_j$ audits $\mathcal{F}_{\text{offline}}$ and $\mathcal{F}_{\text{PK}}$. $\mathcal{I}_j$ also gets key k.
  if $\mathcal{F}_{\text{offline}}$ *outputs* $\perp$ **then abort**.
  Each $\mathcal{I}_j$ sends $\mathsf{Enc}_k(x_j)$ to $\mathcal{F}_{\text{BB}}$.
  Each $\mathcal{P}_i$ retrieves $\mathsf{Enc}(x_j)$ from $\mathcal{F}_{\text{BB}}$.
  Pack all $\mathsf{Enc}(x_j)$ into $\mathsf{Enc}(\boldsymbol{x})$.
  reply $\mathcal{M}, \boldsymbol{m} \coloneqq \texttt{Decrypt}(\mathsf{Enc}(\boldsymbol{x}) - \mathsf{Enc}(\boldsymbol{r}))$ to $\mathcal{P}_i$.
***Audit*:** On input $(\texttt{audit}, \texttt{prep}, par)$ by $\mathcal{J}$:
  Perform the ***Prepare*** phase on public data.
***Audit*:** On input $(\texttt{audit}, \texttt{input})$ from $\mathcal{J}$:
  Perform the ***Prepare*** and ***Input*** phases on publ. data.

Fig. 6. Offline protocol Π_{offline}.

with decommitments known only to a single party. We achieve this by using a (linear) homomorphic encryption scheme to construct (ciphertexts of) shares. If enough parties are honest (exactly when there are not enough corrupted parties to decrypt ciphertexts on their own), the shared values will be uniformly random values as honest parties contribute uniform randomness. With each party producing its own "committing ciphertext" (a ciphertext and a commitment for the same plaintext), we can construct views from these ciphertext shares. Note that each party needs to prove correctness for these committing ciphertexts in order to safely construct the views.

More specifically, given a committing ciphertext, i.e., $\mathsf{Enc}(y_j)$ and $\mathsf{Com}(y_j)$ for plaintext y_j of party $\mathcal{P}_j$, and an encrypted share $\mathsf{Enc}([v]_j)$ (we describe below how this can be obtained), we can get $m_j \coloneqq y_j - [v]_j$ from decrypting $\mathsf{Enc}(y_j) - \mathsf{Enc}([v]_j)$. Then, we have $\mathsf{Com}(y_j) - m_j = \mathsf{Com}([v]_j)$, while letting $\mathcal{P}_j$ compute $[v]_j \coloneqq y_j - m_j$. This gets us views $\langle v \rangle_i$ for all $\mathcal{P}_i$ and is enough to generate the required views for $\boldsymbol{r}, \boldsymbol{a}, \boldsymbol{b}$ (for the inputs and parts of the triples). As this requires a well-formed commitment from party $\mathcal{P}_j$, we cannot use $\mathsf{Com}(y_j)$ and thus $\mathsf{Com}([v]_j)$ if the ZKP for committing ciphertexts did not verify. We use $\top$ instead for $\mathcal{P}_j$, making sure that shares of this party are not used in the online phase (as openings will always fail).

For the above to work, everyone (also external parties who later verify the computation) has to know the encrypted shares $\mathsf{Enc}([v]_i)$. This requires a different construction in our case with Shamir secret-sharing compared to the standard case of full-threshold secret-sharing. We can utilize the linear homomorphic property of the encryption scheme to first let each party $\mathcal{P}_i$ construct a matrix $\mathsf{Enc}(\mathbf{W}_i)$ of ciphertexts with t columns. The number of rows corresponds to the number of views we want to produce in the offline phase (we assume this or an upper bound for this is known when the offline phase is executed). Adding them up (for parties that could prove correct encryption for their $\mathsf{Enc}(\mathbf{W}_i)$) to $\mathsf{Enc}(\mathbf{W}) \coloneqq \sum_i \mathsf{Enc}(\mathbf{W}_i)$ makes sure that we get ciphertexts for uniformly random plaintexts, provided not too many parties are corrupted (or they could decrypt alone, know the plaintexts of others, and adjust their own ciphertexts accordingly). This matrix of ciphertexts can be used to construct ciphertexts of shares in the following way: $\mathsf{Enc}([v]_i) \coloneqq \sum_{l=0}^{t-1} i^l \cdot \mathsf{Enc}(\mathbf{w})[\,\cdot\,, l]$. This is a share of $\mathsf{Enc}(\boldsymbol{v}) \coloneqq \mathsf{Enc}(\mathbf{W})[\,\cdot\,, 0]$, which is known to everyone (by having only communication through $\mathcal{F}_{\text{BB}}$).

Lattice-based cryptographic primitives like the BGV encryption scheme (cf. Appendix A-B) use "noise" to hide the content of a message. If the noise becomes too big then the message can no longer be recovered. Moreover, BDLOP commitments require "small" randomness for decommitments (without a bound on the randomness, commitments are not binding). In our protocols we use ZKPs to prove that committing ciphertext and (normal) ciphertexts were constructed correctly and in particular, that their noise is acceptably small. Since homomorphic operations usually increase the noise, a small initial noise allows us to perform more homomorphic operations on the instances that are proven correct. For lattice-based primitives, this also means that the contained "noise" is small (to allow for more homomorphic operations on the instances that are proven correct). To increase efficiency, a standard way to do these proofs is to use a classical aggregation technique [49] but this comes with an additional noise growth ("slack") from the ZKPs that is exponential in the (statistical) security parameter η. Instead, we chose to combine this with rejection sampling [50][9] which decreases the slack by approximately a factor of 2^η. While this does not solve the

[9]This possibility was already mentioned in [52], [53] but, to our knowledge, was not used before. We show in Section VII that it can improve parameters in certain settings and formalize the construction in the full version.

problem of exponential slack, the potential decreases in noise for both ciphertexts and commitments gives improvements where other ZK techniques are not applicable (see Appendix E for more details and Section VII for an evaluation).

Alternatively, one might consider using *approximate* ZKPs, i.e., proofs where only approximate relations are proven (e.g., for BDLOP [45]). *Approximate* ZKPs usually come with a comparably small slack. However, they have worse homomorphic properties than the exact ZKPs, which makes them inapplicable in our setup.

One aspect of the offline phase is still open: generating the final component of multiplication triples. Assuming we have an additional view $\langle d \rangle_i$ and ciphertexts $\mathsf{Enc}(d), \mathsf{Enc}(c) = \mathsf{Enc}(a \cdot b)$, we can do something similar to the view generation and compute $\langle d \rangle_i + e = \langle c \rangle_i$ where e is the decryption of $\mathsf{Enc}(c) - \mathsf{Enc}(d)$. Getting the ciphertext $\mathsf{Enc}(c)$ can be done in two ways for our protocol. First, as we already have ciphertexts $\mathsf{Enc}(a), \mathsf{Enc}(b)$ from generating the views for a and b, we can get $\mathsf{Enc}(c)$ by using the somewhat homomorphic nature of the encryption scheme. The second, more elaborate, way is to use a special ciphertext multiplication protocol. We give a new construction of such a protocol in Fig. 7 (described in Section V-B1). This is useful if the encryption scheme does not support ciphertext multiplications natively (if one wants to adapt our protocol to other encryption schemes) or, more importantly, to bootstrap our key generation. We describe this in Appendix B. We note that our MPC protocol is currently the only (efficient) accountable and robust MPC protocol that can be used for bootstrapping key generation.

We finally note that, as described in Section V-A, we use some subroutines of the offline protocol within the input phase of the online protocol. This has the advantage that we can re-use the encryption scheme from the offline phase for providing encrypted inputs. To get the value $\boldsymbol{m} = \boldsymbol{x} - \boldsymbol{r}$ needed in the online phase, we can simply compute it as the decryption of $\mathsf{Enc}(\boldsymbol{x}) - \mathsf{Enc}(\boldsymbol{r})$ for a ciphertext $\mathsf{Enc}(\boldsymbol{x})$ of the inputs. Supporting encrypted inputs is a feature that is very useful for future deployment in a wide range of client-server settings, e.g., in e-voting where voters/clients often provide their votes in the form of ciphertexts. For the specific application of e-voting, we describe several extensions of our protocol in the full version. These allow us, e.g., to enforce additional constraints on inputs and to reduce the number of inputs our protocol has to process.

Theorem 2. *The protocol Π_{offline} is a publicly accountable and t-robust preprocessing protocol (w.r.t. Π_{online}). That is, Π_{offline} UC-realizes the functionality $\mathcal{F}_{\text{offline}}$ in the $(\mathcal{F}_{\text{PK}}, \mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{BB}})$-hybrid model under the assumption that the used homomorphic encryption scheme is CPA-secure and that the homomorphic commitment schemes are binding and hiding.*[10]

1) Linear Ciphertext Multiplication: While the above construction using somewhat homomorphic encryption (SHE) is enough for an accountable and robust offline phase, we also support an optional subprotocol for ciphertext multiplication (and thus triple generation) that uses only the linearly homomorphic variant of the BGV scheme. The already mentioned benefit of using this for the key generation is expanded upon in Appendix B. The subprotocol (pictured in Fig. 7) works in the spirit of Overdrive's LowGear protocol [15] (or BDOZ [9]): We construct the ciphertext for c by multiplying shares of a with the ciphertext of b and add noise. However, in our construction we can use the distributed decryption and the homomorphic commitment scheme to make it more efficient and accountable. We prove the correctness of our new multiplication protocol.

As a first step, the protocol in Fig. 7 computes ciphertexts for $[a]_i \cdot \mathsf{Enc}(b) + \mathsf{Enc}_k(0, r)$, i.e., multiplications of shares of a with ciphertexts of b plus noise $\mathsf{Enc}_k(0, r)$ (with additional randomness $r = (v, \boldsymbol{e})^{\mathsf{T}}$). We call this process re-encryption since we get a new ciphertext for the plaintext $[a]_i \cdot b$. The results of our re-encryption can then be combined with Rec (the algorithm to reconstruct a secret from shares; recall, this is a weighted sum of t ciphertexts with coefficients in $\mathbb{F}_p$). By the linear homomorphic properties of BGV (multiplying $[a]_i$ with a ciphertext is a linear operation), we get that the result is a ciphertext for $c = a \cdot b$. Obviously, we should only use ciphertexts that are constructed in such a way or the result might be wrong. By only requiring t correctly multiplied ciphertexts, we get robustness. To get accountability, we use ZKPs again.

With this novel construction, we can utilize efficient ZKPs. As most parts are already committed to (for the simulator in the security proof), we only have to additionally commit to the encryption randomness (and the re-encryption result that will be decommitted immediately). Proving correct commitments for the encryption randomness is done in an aggregated way using the mentioned ZKP aggregation technique; it can be done ahead-of-time for all multiplications. The efficient proofs of linear relations [45], [54] for the commitment scheme (discussed in Section VI) make sure that we can prove the correctness of the whole multiplication and re-encryption operation in a single one-shot proof. This improves on, e.g., BDOZ [9], where η proofs with one-bit challenges have to be combined to get a soundness error that is negligible in η. Selective failure attacks as in [9], [15] are not possible as the ZKPs in our protocol make sure that the ciphertexts that are actually used (and decrypted) later are in a valid range. Additionally, only masked ciphertexts are decrypted, so no extra randomness has to be added in the re-encryption, and this is a public (distributed) decryption instead of letting parties decrypt their own shares of c. See Section VII for more details on the differences (w.r.t. parameter sizes) when using this LHE multiplication or SHE BGV directly. The way we get the key material to support these versions of BGV is discussed in Appendix B.

VI. Lattice-Based Commitments

Our protocol requires a suitable homomorphic commitment scheme. As we will see in Section VII, the Pedersen commitment scheme that is used in related works [10], [14] does not combine very well with BGV encryption, i.e., it

[10] Commitments using the commitment parameters *par* need to be binding and hiding as for the online phase and commitments with *par'* need to be statistically binding and computationally hiding.

macro `Multiply:` `// to multiply BGV ciphertexts` $\mathsf{Enc}(\boldsymbol{a})$ `and` $\mathsf{Enc}(\boldsymbol{b})$ `using only LHE`

Let $\boldsymbol{y}'_j, \boldsymbol{m}'_j$ be the subset of $\boldsymbol{y}_j, \boldsymbol{m}_j$ used to construct $\langle \boldsymbol{a} \rangle_j$.

Set views with *par'*: $[\boldsymbol{a}]_i := \boldsymbol{y}'_i - \boldsymbol{m}'_i$, $\mathsf{R}'_{\mathsf{C}}([\boldsymbol{a}]_i) := \mathsf{R}'_{\mathsf{C}}(\boldsymbol{y}'_i)$, and $\mathsf{Com}'([\boldsymbol{a}]_j) := \mathsf{Com}'(\boldsymbol{y}'_j) - \boldsymbol{m}'_j$ for $\mathcal{P}_j \in \mathcal{P} \setminus \mathcal{M}'$.

Generate re-encryption randomness $(\boldsymbol{v}_i, \boldsymbol{e}_i) \xleftarrow{\$} D_{\sigma_v}(R_q^M) \times D_{\sigma_e}(R_q^M)^2$.

Compute re-encrypted share of product $\mathsf{Enc}([\boldsymbol{c}]_i) := [\boldsymbol{a}]_i \cdot \mathsf{Enc}(\boldsymbol{b}) + \mathsf{Enc}_k(0, (\boldsymbol{v}_i, \boldsymbol{e}_i)^{\mathrm{T}}) = \begin{pmatrix} [\boldsymbol{a}]_i \cdot \mathsf{Enc}(\boldsymbol{b})[0] + k[1] \cdot \boldsymbol{v}_i + p \cdot \boldsymbol{e}_i[0] \\ [\boldsymbol{a}]_i \cdot \mathsf{Enc}(\boldsymbol{b})[1] + k[0] \cdot \boldsymbol{v}_i + p \cdot \boldsymbol{e}_i[1] \end{pmatrix}$.

Commit to $\boldsymbol{v}_i, \boldsymbol{e}_i, \mathsf{Enc}([\boldsymbol{c}]_i)$ with *par'*.

Let $\mathsf{Com}'(\boldsymbol{v}_i), \mathsf{Com}'(\boldsymbol{e}_i), \mathsf{Com}'(\mathsf{Enc}([\boldsymbol{c}]_i))$ be the resulting commitments.

Prove Line 5 to 7 in ZK. Let the NIZKP be z_i.

Send z_i and the decommitment for $\mathsf{Com}'(\mathsf{Enc}([\boldsymbol{c}]_i))$ to $\mathcal{F}_{\mathrm{BB}}$. Retrieve z_j and the decommitment for other $\mathcal{P}_j \in \mathcal{P}$.

Add $\mathcal{P}_j$ to $\mathcal{M}$ and $\mathcal{M}'$ if verification of z_j failed or $\mathsf{Verify}_{par'}(\mathsf{Enc}([\boldsymbol{c}]_j), \mathsf{R}'_{\mathsf{C}}(\mathsf{Enc}([\boldsymbol{c}]_j)), \mathsf{Com}'(\mathsf{Enc}([\boldsymbol{c}]_j)) = 0$.

return $\mathsf{Enc}(\boldsymbol{c}) := \mathsf{Rec}(\mathsf{Enc}([\boldsymbol{c}]_1), \ldots, \mathsf{Enc}([\boldsymbol{c}]_n))$ using $\mathbb{F}_p$-coefficients while ignoring shares of $\mathcal{P}_j \in \mathcal{M}'$.

Fig. 7. Multiplication subprotocol for Π_{offline} (cf. Fig. 6) at $\mathcal{P}_i \in \mathcal{P}$.

requires increasing the modulus of the underlying plaintext space and thereby negatively impacts performance. To address this mismatch (and to get a fully lattice-based protocol), we propose using lattice-based commitments instead.

There exists a wide variety of lattice-based commitment schemes, e.g., [45], [55]–[58]. The overall best suited scheme appears to be the BDLOP commitment scheme [45], which offers efficient zero-knowledge proofs for our offline phase and can be instantiated, using different parameters, to be either statistically hiding or statistically binding as required by our offline and online phases (cf. Section V). However, we cannot use BDLOP directly since it offers only limited homomorphic properties. In what follows, we first recall BDLOP and then propose a modification that improves the homomorphic property to be sufficient for our protocol while keeping the modulus of the plaintext space small (as opposed to Pedersen commitments).

The BDLOP scheme is based on the Module-Short Integer Solution (M-SIS) and Module-Learning With Errors (M-LWE) problems [46], [59]. The (public) commitment parameters consist of two matrices $\mathbf{A}_0 := (\mathbf{I}_{d_1} \ \mathbf{A}'_0)$ and $\mathbf{A}_1 := (\mathbf{0}_{1 \times d_1} \ \mathbf{I}_1 \ \mathbf{A}'_1)$ where $\mathbf{A}'_0 \in R_p^{d_1 \times (1+d_2)}$ and $\mathbf{A}'_1 \in R_p^{1 \times d_2}$ are uniformly random sub-matrices. With $par := (\mathbf{A}_0, \mathbf{A}_1)$, we can define the commitment procedure for $x \in R_p$ with small randomness r as

$$\mathsf{Com}_{par}(x, r) := c = \begin{pmatrix} c[0] \\ c[1] \end{pmatrix} = \begin{pmatrix} \mathbf{A}_0 \cdot r \\ \mathbf{A}_1 \cdot r + x \end{pmatrix},$$

while verification $\mathsf{Verify}(x, r, c)$ checks if $\mathsf{Com}(x, r) = c$ and $\|r[i]\| \leq B_r, 0 \leq i < d_2 + d_1 + 1$.[11] Generally, B_r should not be too large as otherwise the underlying M-SIS problem becomes easy and the scheme is no longer binding. This can be prevented by increasing the modulus p and thereby increasing the hardness of the M-SIS problem. Further information on zero-knowledge proofs and the already mentioned approximate commitments is provided in the full version.

Homomorphic operations increase the randomness/noise of the resulting commitment. To allow for more operations, one can increase the bound B_r. E.g., using $2 \cdot B_r$ allows for decommitting to $\mathsf{Com}(x_1, r_1) + \mathsf{Com}(x_2, r_2)$ with $x_1 + x_2$ and $r_1 + r_2$. However, using Beaver triples for multiplication as in our protocol requires a homomorphic computation of $d := u \cdot a + v \cdot b + c$ for commitments a, b, c, d and uniformly random $u, v \in R_p$. The multiplication with u, v introduces a factor p into the noise of d, i.e., the noise of d is upper bounded by *at least* $B_r \cdot (1+p)$ where B_r is the bound for a, b, c (cf. [45] for more details on the norm estimates). For practical choices of B_r (or σ_r), this bound allows trivially decommitting d to arbitrary values, i.e., d is not binding. Increasing the modulus p does not solve this problem since the upper bound of the noise of d also linearly depends on p.

Intuitively, to solve the issue we have to use two independent moduli for (the noise of) a, b, c, d and the (masked) values u, v. This would allow us to increase the modulus of the commitments without also increasing the noise bound of d (which then only depends on the now independent modulus of u, v) such that d can become binding. However, for Beaver multiplication to hide the inputs of the commitments, we need that the modulus used for the plaintexts in a, b, c, d is the same as the modulus for u, v. We thus propose a modification of the above scheme that uses two different moduli for commitments. That is, for the randomness r and the first component $c[0]$ we use a modulus p' that is an integer multiple of the prime p, while all message-related components (i.e., $c[1] = \mathbf{A}_1 \cdot r + x$) are modulo p.[12] For the scheme with the above modifications, we get the following theorem.

Theorem 3 (Generalization of BDLOP)**.** *The BDLOP commitment scheme with the above generalization is binding and hiding. The strength of the binding property (computationally or statistically) is based on the hardness of M-SIS. The strength of the hiding property (computationally or statistically) is based on the hardness of M-LWE.*

In the full version, we provide full details of our construction, show that security of our construction can still be reduced to the M-SIS and M-LWE assumptions, and that only p', but not p, needs to be increased to improve the binding property.

[11] In the computationally secure case, r is sampled uniformly at random with L^∞-norm at most B_r. For statistically binding (and extractable) commitments, r is sampled from $D_{\sigma_r}(R^{d_2+d_1+1})$. The latter also induces a bound $B_r \geq \sigma_r \cdot \sqrt{2 \cdot N}$ on the L^2-norm of $r[i]$.

[12] This idea is similar to [54], but [54] considers the case of two primes with $p' < p$. In contrast, we consider and argue security of the case $p \mid p'$.

Hence, we can simply increase p' to a level such that the results d of our Beaver multiplication (with the verification bound $B_r \cdot (1+p)$) can be verified but are still binding. Observe that the plaintexts in this construction indeed remain hidden in Beaver multiplication since they use the same modulus p as the masked plaintexts u, v.

Note that our construction is not only useful for Beaver multiplications. More generally, it can be used to improve the homomorphic properties of the commitment scheme by changing p' without changing/affecting the modulus p of the plaintext space, which is unlike for the original commitment. Hence, one can simply increase p' to support a larger number of homomorphic additions (e.g., to the level required by circuits for our MPC protocol) without affecting other primitives that use the same plaintext space (such as the BGV scheme in our protocol). Thus, our construction might also be useful for other protocols and even in contexts outside of MPC.

To be suitable for simulation-based MPC security proofs, the above commitment scheme additionally requires trapdoors for the simulator to equivocate and extract messages from the above commitment scheme. Note that, due to our construction for opening outputs in the online phase, we only need the second property, i.e., to be able to extract messages. However, we will additionally provided a protocol Π_{equiv} that uses equivocation like some of the related work, e.g. [10], [14], and compare the two approaches in Section IX. We add such trapdoors by following and adapting the construction of Damgård et al. [60] to the generalized variant of BDLOP (see the full version for more details). In our protocol we use our modification of BDLOP to commit to values in R_p during the online phase. To commit to values from R_q in the offline phase, where the modulus of the input space is already much bigger and, in particular, no Beaver multiplications are required, we can simply use the original scheme (with trapdoors added as in [60], [61]).

VII. Parameters

To illustrate and judge practicality of our protocol, here we compute the necessary parameters for the BGV encryption scheme and the (generalized) commitment scheme of Section VI. The asymptotic and concrete complexity of our protocol is analyzed in Section VIII. Our methodology is described in Appendix F and more detailed results are given in the full version.

BGV Parameters: The main parameter that determines practicality of the BGV scheme is the ciphertext modulus q. We have computed the parameters for an LHE and an SHE version of our protocol, with the results shown in Fig. 8. As parameters for other efficient two-phase protocols with (publicly) identifiable abort are not available, we instead use the BGV parameters of LowGear [15] and TopGear [16] (two recent very efficient protocols without identifiable abort) in addition to classical SPDZ [7], [8] as a baseline for a comparison of our parameters. As can be seen from this comparison, the additional zero-knowledge proofs of correct decryption used to obtain public accountability of our protocol result in somewhat larger parameters. However, the parameters are still in a practical range that is rather close to those highly

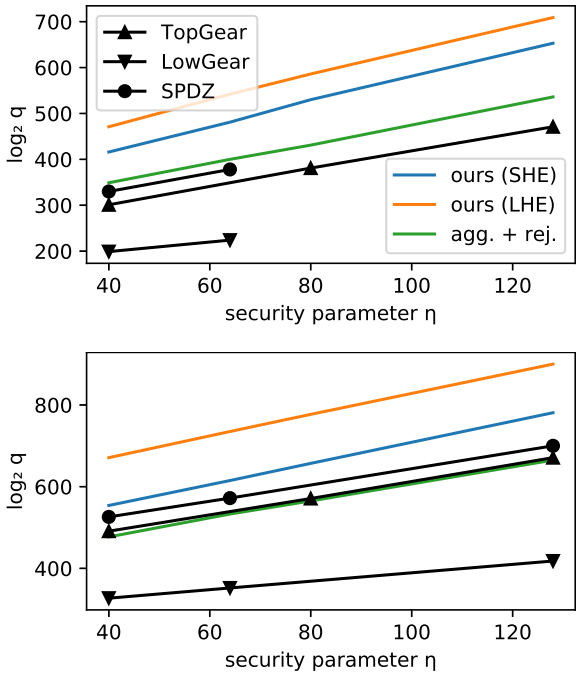


Fig. 8. Comparison of BGV parameters against TopGear, LowGear, and SPDZ for $\log p = 64$ (top) and $\log p = 128$ (bottom). Parameters are essentially independent of the choice of t and n, with the above values being for $n = t = 2$. E.g., for $\eta = 80$ and $\log p = 64$, increasing $n = t$ from 2 to an extreme value of 4096 increases $\log q$ only by relatively few bits from 529 to 584.

efficient protocols. Also, the parameters show a near identical slope, suggesting that our protocol adds only a "constant" overhead compared to current SPDZ-like protocols.

Recall that our offline phase combines zero-knowledge proof aggregation [49] with rejection sampling [50]. To evaluate the benefits of this techniques also for other SPDZ-like protocols, we have also computed the BGV parameters of a theoretical version of our protocol without proofs of correct decryption. The resulting protocol is denoted by "agg. + rej.". Our evaluation allows for the first time to estimate the advantage of this combined technique if employed in other (non-accountable) SPDZ-like protocols. To summarize, for the bigger plaintext space ($\log p = 128$) the combined aggregation technique yields slightly tighter parameters than TopGear and classical SPDZ. We suspect that this is even more pronounced for larger plaintext sizes, which might make this technique worth considering also for other SPDZ-like protocols that cannot use techniques employed in TopGear (aggregating proofs over all parties; this prevents identifying individual misbehaving parties) to decrease parameters.

We have also computed the BGV parameters necessary to use our construction with Pedersen commitments (based on elliptic curves or quadratic residues; as suggested by Cunningham et al. [14]) instead of our lattice-based commitment scheme, see Fig. 9. The comparison shows that the choice of a lattice-based commitment scheme indeed synergizes with the lattice-based BGV encryption scheme, leading to lower parameters. Note that, due to our modification to the commitment scheme (cf. Section VI), we can improve the binding

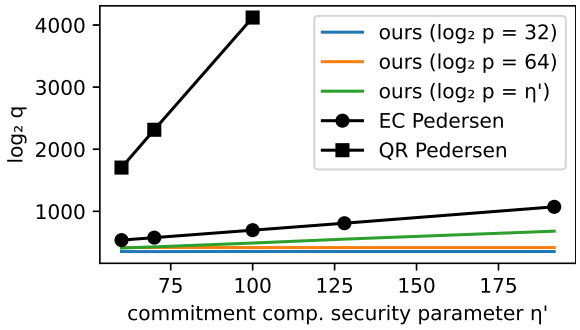


Fig. 9. Comparison of BGV parameters against Cunningham et al. [14] with elliptic curve (EC) and quadratic residue (QR) Pedersen commitments.

TABLE II
COMMITMENT PARAMETERS FOR R_p AND R_q

				for R_p with par			for R_q with par'		
$\log p$	η	N	$\log h^a$	d_2	d_1	$\log p'$	d_2	d_1	$\log q$
32	40	16384	59	1	1	120	2	1	352
32	64	16384	59	1	1	134	2	2	416
32	80	16384	59	1	1	142	2	2	458
32	128	32768	60	1	1	170	1	1	589
64	40	16384	91	1	1	152	2	2	418
64	64	16384	91	1	1	166	2	2	480
64	80	32768	92	1	1	176	1	1	530
64	128	32768	92	1	1	202	1	1	653
128	40	32768	156	1	1	218	1	1	553
128	64	32768	156	1	1	232	1	1	616
128	80	32768	156	1	1	240	2	2	658
128	128	32768	156	1	1	266	2	2	780

[a] homomorphic factor, i.e., the longest chain of linear operations while evaluating ResNet152 is equivalent to summing up h fresh commitments

property and the homomorphic properties of our scheme without affecting the plaintext size. That is, we can actually use a plaintext space with constant size independent of the security parameter (blue and orange line), further improving the resulting BGV parameters.

BDLOP Parameters: Since the commitment parameters are not completely circuit independent (p' depends on the number of homomorphic operations required by the circuit as illustrated in Section VI), we also had to estimate the number of commitment operations in the online phase. For this, we chose ResNet152 [62] as example of a non-trivial circuit.

To summarize Table II, we achieve small (or even minimal) dimensional parameters d_2, d_1 for commitments. Furthermore, the modulus p' remains at practical sizes and increase only moderately in η and p (N and q are the same as for, and thus determined by, the BGV scheme). Notably, by our modification to the commitment scheme (Section VI), the modulus p' used for commitments in the particularly critical online phase can be chosen to be the product of multiple machine-word-sized primes, which makes very efficient implementations possible [63]. We also computed parameters for the generalized commitment scheme but with equivocation. The results can be found in the full version and the impact on our protocol's performance is analyzed in Section IX. Altogether, our analysis shows that our parameters are well within the realm needed for complex applications.

TABLE III
COMPARISON OF MPC PROTOCOLS

	ours	[41], [42][a]	[14]	[11]	[28]
Pub. acc.	+	Π	+	+	+
Strong prop.	+	Π	+	−	−
Priv. thresh.	t	$\max\{t, n-\lvert\mathcal{M}\rvert\}^b$	n	n	n
Abort thresh.	$n-t+1$	$n-t+1$	1	1	1
Online com.[c]	$n\cdot\lvert f\rvert$	$n\cdot\Pi$	$n\cdot\lvert f\rvert$	$n^2\cdot\lvert f\rvert$	$n^2\cdot\lvert f\rvert$
Online cmp.[d]	$n\cdot\lvert f\rvert$	$n\cdot\Pi$	$n\cdot\lvert f\rvert$	$n^2\cdot\lvert f\rvert$	$n^2\cdot\lvert f\rvert$
Online rnd.	$\lvert f\rvert$	$n\cdot\Pi$	$\lvert f\rvert$	$\lvert f\rvert$	1
Offline com.[c]	$n^2\cdot\lvert f\rvert$	Π_{off}	$n\cdot\lvert f\rvert$	$n^3\cdot\lvert f\rvert$	$n^2\cdot\lvert f\rvert$

[a] Internally restart resp. run in parallel up to $\mathcal{O}(n)$ instances of another protocol. Most properties depend on the underlying protocol, denoted by the placeholder Π. Π_{off} denotes the offline phase of Π. For complexities, Π denotes the combined complexity of *both* the online and the offline phase. In case of [42], certain protocols allow for reducing the online complexity from Π to Π_{on} by performing (some) additional steps in the offline phase. But this optimization is not applicable for [14].
[b] $\mathcal{M}$ is the set of parties that caused an abort and thus a restart.
[c] Number of broadcasts / stores on the bulletin board in $\mathcal{O}$-notation; $\lvert f\rvert$ denotes the number of multiplication resp. AND gates in a circuit f.
[d] Operations *per party* in $\mathcal{O}$-notation; $\lvert f\rvert$ denotes the number of gates.

VIII. DISCUSSION AND COMPARISON

As already discussed in the introduction (cf. Table I), to the best of our knowledge the combination of publicly identifiable abort (or, more generally, public accountability) and robustness has not been considered for efficient two-phase protocols. The protocol that comes closest to this goal is a combination of Cunningham et al.'s protocol [14], which is a SPDZ-like protocol that offers strong publicly identifiable abort, with a best-of-both-worlds protocol [41], [42] that provides robustness. While not formally proven,[13] this combination, which we denote by $\text{BoBW}_{[14]}$, likely provides strong publicly identifiable abort (and also strong public accountability) while additionally being robust.

Table III provides an overview of the properties of our protocol, BoBW protocols [41], [42], and Cunningham et al.'s protocol [14], where the properties of $\text{BoBW}_{[14]}$ can be derived from the combination of [41], [42] and [14]. The table shows the number of corrupted parties that are needed to break privacy and robustness. We also indicate whether the protocols provide accountability (resp. publicly identifiable abort) and whether this property is strong, i.e., still holds even if all protocol participants/servers are malicious. We further give the asymptotic complexity for the overall communication during online and offline phases, for the number of communication rounds during the online phase, and for the computations of the online phase. In what follows, we discuss the differences between our protocol and $\text{BoBW}_{[14]}$ in detail.

Comparison of Properties: The security properties and thresholds of our protocol and $\text{BoBW}_{[14]}$ are mostly identical, except that the privacy property of $\text{BoBW}_{[14]}$ can tolerate a larger number of corrupted parties in certain cases, namely if the set of parties $\mathcal{M}$ that have caused an abort is small (observe that this is therefore not a static bound but rather depends on a specific run). Note that, in situations where one

[13] [41], [42] consider and give proofs for the traditional non-client-server setting where at least one participant is honest.

expects only a very small number of parties to try to abort the protocol (due to the deterrence factor of accountability coupled with strong contractual or financial incentives), one generally would choose a large threshold t, in which case there is only a small potential difference.

The advantage in terms of privacy of BoBW$_{[14]}$ comes at the cost of using protocol iteration, which not only negatively impacts performance (see below) but also makes BoBW$_{[14]}$ unsuitable for certain client-server applications. In contrast, our protocol avoids protocol iteration entirely. It is therefore the first and only efficient two-phase protocol with public accountability and robustness that is suitable even for client-server applications where clients cannot be expected to deal with the downsides of protocol iteration.

Asymptotic Performance Comparison: In terms of asymptotic communication, computation, and round complexity, our protocol outperforms BoBW$_{[14]}$ (at least) by a factor of $\mathcal{O}(n)$ in all aspects except for offline communication, where we require an additional factor of $\mathcal{O}(n)$. More specifically, we achieve the same online complexity as Cunningham et al. but manage to additionally provide robustness while avoiding the iteration technique used by the best-of-both-world protocols [41], [42]. Hence, in the optimal situation for BoBW$_{[14]}$ where no malicious parties cause an abort and thus the online phase of BoBW$_{[14]}$ requires just a single iteration of Cunningham et al.'s protocol, both BoBW$_{[14]}$ and our protocol have identical online complexity. However, the performance of the online phase of BoBW$_{[14]}$ progressively deteriorates with every abort. Notably, each rerun of [14] due to an abort requires first rerunning the entire (expensive) offline phase within the online phase of BoBW$_{[14]}$, which is impractical.[14] In contrast, our protocol retains the same level of efficiency independently of the number of malicious parties trying to cause aborts and without rerunning its offline phase within the online phase.

Above, we have compared our protocol with the only other protocol that might provide both public accountability and robustness. Next, we compare our protocol with other two-phase protocols to show that our protocol achieves the desired security while also retaining the advantages of the underlying SPDZ-like structure.

Table III provides a comparison of our protocol with three efficient two-phase protocols that offer public accountability but no robustness [11], [14], [28]. Compared to these protocols, we achieve identical or better asymptotic complexity, except for the online round complexity of Baum et al.'s protocol [28] and, as mentioned, the offline communication complexity of Cunningham et al.'s protocol [14], which are better. Note that [11], [28] do not achieve strong publicly identifiable abort (cf. Table I) and are based on primitives whose security breaks down if all servers are corrupted (e.g., information theoretic signatures). So they cannot be adapted to the fully corrupted case without redesigning the protocols.

More generally, even when compared to highly-efficient SPDZ-like protocols without public accountability such as [7]–[9], [12], [15], [16], our protocol still manages to achieve comparable asymptotic complexity. The main difference lies in the concrete computational overhead introduced by the commitments used for accountability, whereas the simple field operations and information theoretic MACs used by most SPDZ-like protocols are computationally less costly. We discuss the concrete performance of our protocol next.

[14]It might be possible to precompute the offline phase of [14] for all subsets of parties, but this introduces an additional factor of $\mathcal{O}(2^n)$ to the offline phase of BoBW$_{[14]}$.

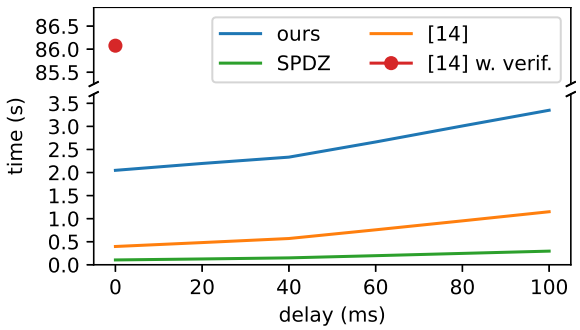
IX. Concrete Evaluation

We compare the concrete performance of our protocol to SPDZ, a state-of-the-art protocol without our additional security properties, and Cunningham et al.'s protocol [14]. The comparison with [14] serves as an approximation of the so far theoretical BoBW$_{[14]}$, discussed in Section VIII: The online phase of [14] (without performing lazy verification of commitments) is essentially the online phase of BoBW$_{[14]}$ in an ideal case, i.e., without restarts. The combination of several offline and online phases of [14] (including verification for aborted online phases) corresponds to BoBW$_{[14]}$ with restarts.

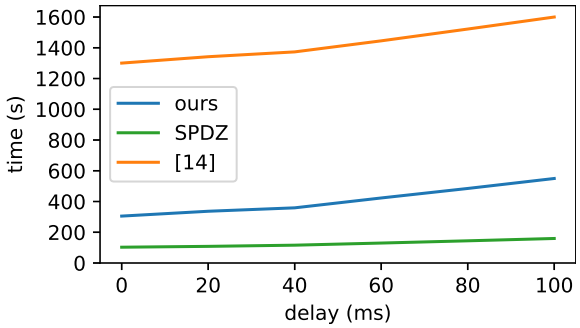
We have experimentally evaluated the runtime of all protocols for a concrete setting, where a small neural network ("network A" as in MP-SPDZ [64], [65], introduced by Mohassel and Zhang [1]) is evaluated N times on a batch of separate inputs. Such batch processing can fully utilize the N slots available in BDLOP commitments. A discussion on amortizing the cost of commitments if we do use batches of size N can be found in the full version. The precise setting and resulting benchmarks for all protocols are given in Fig. 10 with more details available in Appendix G. These benchmarks were obtained using our own implementations of all protocols to ensure a fair comparison. More specifically, for the online phase we have implemented and benchmarked the circuit evaluation, including verification for our protocol and [14]. For the offline phase, we have implemented and performed microbenchmarks that we then extrapolate to approximate the overall runtime. We have validated these approximations by verifying that, for SPDZ, they yield similar results as prior benchmarks obtained for the widespread MP-SPDZ implementation. This validation as well as full details of our experimental setup, further benchmark results, and additional discussions of our results can be found in the full version.

To summarize key findings, the overhead of the additional security properties offered by our protocol compared to basic SPDZ is a factor between 20 (0 ms network delay) to 11 (100 ms network delay) in the online phase. While the online phase of [14] (i.e., BoBW$_{[14]}$ without restarts) is faster than our protocol as long as no error occurs and hence no lazy verification is performed, the online phase of [14] becomes slower than ours if a misbehaving party needs to be identified (cf. red dot in Fig. 10(a)). If [14] is then restarted after such an error (i.e., BoBW$_{[14]}$), which requires rerunning the offline phase, then the difference is even more pronounced (cf. Fig. 10(c) for one restart).

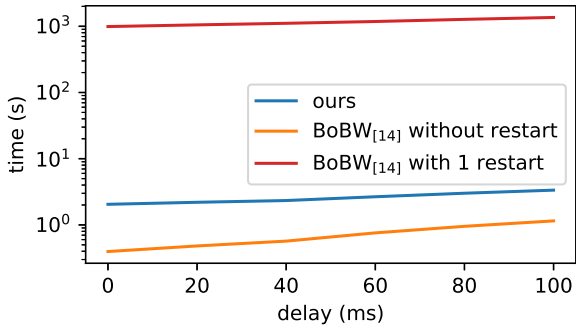
The concrete communication cost per multiplication gate and party for our protocol, SPDZ, and [14] is given in Table IV and Fig. 11. For the specific setting considered in Fig. 10, this results in an amortized communication cost per party and neural network evaluation in the online phase



(a) Amortized online runtime without restarts.



(b) Amortized offline runtime without restarts.



(c) Amortized online runtime with restarts.

Fig. 10. Runtime for evaluating "network A" [1], [64] (118016 addition and 118272 multiplication gates, with batch size $b = N = 32768$; see Appendix G) in the online phase and runtime of the offline phase to prepare the necessary Beaver triples. Timings are amortized for an evaluation in the following setting: $n = 3$; $t = 2$; $\log p = 128$; single-threaded computation (AMD EPYC 7443 CPU); bandwidth limited to $1\,\mathrm{Gbit\,s^{-1}}$; statistical and computational security parameters are 40 bit and 128 bit, respectively.

TABLE IV
ONLINE COMMUNICATION COST (IN BITS) PER PARTY AND MULTIPLICATION

$\log p$	η	SPDZ	ours	Π_{equiv} [a]	[14]
64	40	128	722	3020	1008
64	64	128	722	3128	1008
64	80	–[b]	728	3260	1008
64	128	–[b]	728	3476	1008
128	40	256	1240	4504	1008
128	64	256	1240	4600	1008
128	80	256	1240	4672	1008
128	128	256	1240	4888	1008

[a] with trapdoor for equivocation (cf. Section V-A)
[b] not secure with statistical security η as MAC error is $2/p$

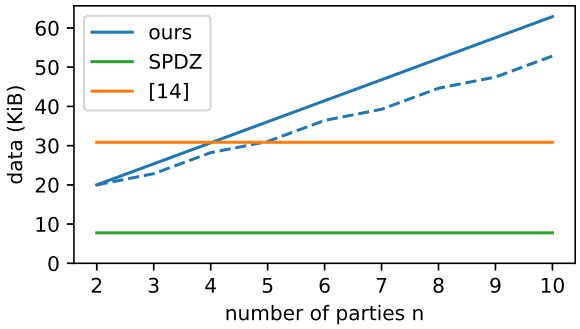


Fig. 11. Offline communication cost per party and multiplication. The solid line for our protocol is for $t = n$, the dashed line is for $t = \lfloor n/2 \rfloor + 1$.

of 17.48 MiB for our protocol, 3.61 MiB for SPDZ, and 14.21 MiB for [14]. For the corresponding offline phases (to prepare a single evaluation of the neural network), the amortized concrete communication cost is 2.58 GiB, 897 MiB, and 3.48 GiB, respectively. Additionally, we show the online communication cost of Π_{equiv} in Table IV—the variant of our protocol that uses equivocal BDLOP commitments. Not only is more communication required for this variant, initial tests also indicate that this variant is about 4 times slower than our protocol in the online phase, which is why we omit a full runtime analysis. The NIZKPs added in the output phase of our protocol to avoid equivocation account for less than 0.02 % of the overall online runtime shown in Fig. 10(a).

We note that we use the parameters from Section VII for the BDLOP commitments, which were computed to be sufficient for a much larger arithmetic circuit. While these parameters could be decreased for the smaller "network A" to further improve performance of our protocol, we nevertheless used those parameters to show that the runtime overhead due to parameter size is practical also for larger circuits. For [14], we use Curve25519-based commitments to reach the same computational security level of 128 bit. The statistical security parameter is set to $\eta = 40$ for all protocols (as in, e.g., [7], [15], [16]).

Altogether, our protocol performs significantly better than (so-far theoretical) BoBW$_{[14]}$ in a malicious setting while offering the same security guarantees and while being applicable even when restarts are not an option, e.g., because inputs cannot be provided repeatedly. The additional security properties of our protocol over basic SPDZ still come at a cost, but the resulting performance remains practical relative to other approaches (with weaker security properties).

ACKNOWLEDGMENT

We thank our anonymous reviewers and our shepherd for their invaluable feedback. We also thank Andrés Bruhn and Azin Jahedi from the Institute for Visualization and Interactive Systems at the University of Stuttgart for providing the computational resources and assistance with running our experiments.

The research was supported by the DFG through grant KU 1434/11-1 and by the CRYPTECS project which has received funding from the German BMBF through grant 16KIS1441 and from the French ANR through grant ANR-20-CYAL-0006.

REFERENCES

[1] P. Mohassel and Y. Zhang, "SecureML: A System for Scalable Privacy-Preserving Machine Learning," in *SP 2017*. IEEE Computer Society, 2017, pp. 19–38.

[2] D. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. Pagter, N. Smart, and R. Wright, "From keys to databases—real-world applications of secure multi-party computation," *Comput. J.*, vol. 61, pp. 1749–1771, 2018.

[3] V. Chen, V. Pastro, and M. Raykova, "Secure Computation for Machine Learning With SPDZ," *CoRR*, vol. abs/1901.00329, 2019.

[4] I. Damgård, D. Escudero, T. K. Frederiksen, M. Keller, P. Scholl, and N. Volgushev, "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning," in *SP 2019*. IEEE, 2019, pp. 1102–1120.

[5] A. P. K. Dalskov, D. Escudero, and M. Keller, "Secure Evaluation of Quantized Neural Networks," *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 4, pp. 355–375, 2020.

[6] H. Chen, M. Kim, I. P. Razenshteyn, D. Rotaru, Y. Song, and S. Wagh, "Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning," in *ASIACRYPT 2020*. Springer, 2020, pp. 31–59.

[7] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic Encryption," in *CRYPTO 2012*. Springer, 2012, pp. 643–662.

[8] 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 *ESORICS 2013*. Springer, 2013, pp. 1–18.

[9] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *EUROCRYPT 2011*. Springer, 2011, pp. 169–188.

[10] C. Baum, I. Damgård, and C. Orlandi, "Publicly Auditable Secure Multi-Party Computation," in *SCN 2014*. Springer, 2014, pp. 175–196.

[11] C. Baum, E. Orsini, and P. Scholl, "Efficient Secure Multiparty Computation with Identifiable Abort," in *TCC 2016-B*, 2016, pp. 461–490.

[12] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer," in *CCS 2016*. ACM, 2016, pp. 830–842.

[13] G. Spini and S. Fehr, "Cheater Detection in SPDZ Multiparty Computation," in *ICITS 2016*, 2016, pp. 151–176.

[14] R. K. Cunningham, B. Fuller, and S. Yakoubov, "Catching MPC Cheaters: Identification and Openability," in *ICITS 2017*. Springer, 2017, pp. 110–134.

[15] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ Great Again," in *EUROCRYPT 2018*. Springer, 2018, pp. 158–189.

[16] C. Baum, D. Cozzo, and N. P. Smart, "Using TopGear in Overdrive: A More Efficient ZKPoK for SPDZ," in *SAC 2019*. Springer, 2019, pp. 274–302.

[17] F. Benhamouda, S. Halevi, and T. Halevi, "Supporting Private Data on Hyperledger Fabric with Secure Multiparty Computation," in *IC2E 2018*. IEEE, 2018, pp. 357–363.

[18] J. Cartlidge, N. P. Smart, and Y. T. Alaoui, "MPC Joins The Dark Side," in *AsiaCCS 2019*. ACM, 2019, pp. 148–159.

[19] B. Adida, "Helios: Web-based Open-Audit Voting," in *USENIX Security '08*, P. C. van Oorschot, Ed. USENIX Association, 2008, pp. 335–348.

[20] R. Küsters, J. Liedtke, J. Müller, D. Rausch, and A. Vogt, "Ordinos: A Verifiable Tally-Hiding E-Voting System," in *EuroS&P 2020*. IEEE, 2020, pp. 216–235.

[21] I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft, "Confidential Benchmarking Based on Multiparty Computation," in *FC 2016*. Springer, 2016, pp. 169–187.

[22] A. Bestavros, A. Lapets, and M. Varia, "User-centric distributed solutions for privacy-preserving analytics," *Commun. ACM*, vol. 60, no. 2, pp. 37–39, 2017.

[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," *Comput. J.*, vol. 61, no. 12, pp. 1749–1771, 2018.

[24] A. B. Alexandru, M. Morari, and G. J. Pappas, "Cloud-Based MPC with Encrypted Data," in *CDC 2018*. IEEE, 2018, pp. 5014–5019.

[25] P. Li, J. Li, Z. Huang, T. Li, C. Gao, S. Yiu, and K. Chen, "Multi-key privacy-preserving deep learning in cloud computing," *Future Gener. Comput. Syst.*, vol. 74, pp. 76–85, 2017.

[26] X. Liu, R. H. Deng, Y. Yang, N. H. Tran, and S. Zhong, "Hybrid privacy-preserving clinical decision support system in fog-cloud computing," *Future Gener. Comput. Syst.*, vol. 78, pp. 825–837, 2018.

[27] J. So, B. Güler, and A. S. Avestimehr, "CodedPrivateML: A Fast and Privacy-Preserving Framework for Distributed Machine Learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 441–451, 2021.

[28] C. Baum, E. Orsini, P. Scholl, and E. Soria-Vazquez, "Efficient Constant-Round MPC with Identifiable Abort and Public Verifiability," in *CRYPTO 2020*. Springer, 2020, pp. 562–592.

[29] B. Schoenmakers and M. Veeningen, "Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems," in *ACNS 2015*. Springer, 2015, pp. 3–22.

[30] G. Asharov and C. Orlandi, "Calling Out Cheaters: Covert Security with Public Verifiability," in *ASIACRYPT 2012*. Springer, 2012, pp. 681–698.

[31] I. Damgård, C. Orlandi, and M. Simkin, "Black-Box Transformations from Passive to Covert Security with Public Verifiability," in *CRYPTO 2020*. Springer, 2020, pp. 647–676.

[32] S. Faust, C. Hazay, D. Kretzler, and B. Schlosser, "Generic Compiler for Publicly Verifiable Covert Multi-Party Computation," in *EUROCRYPT 2021*. Springer, 2021, pp. 782–811.

[33] P. Scholl, M. Simkin, and L. Siniscalchi, "Multiparty Computation with Covert Security and Public Verifiability," Cryptology ePrint Archive, Tech. Rep. 2021/366, 2021.

[34] R. Küsters, T. Truderung, and A. Vogt, "Accountability: definition and relationship to verifiability," in *CCS 2010*. ACM, 2010, pp. 526–535.

[35] A. Kiayias, H. Zhou, and V. Zikas, "Fair and Robust Multi-party Computation Using a Global Transaction Ledger," in *EUROCRYPT 2016*. Springer, 2016, pp. 705–734.

[36] C. Baum, B. David, and R. Dowsley, "Insured MPC: Efficient Secure Computation with Financial Penalties," in *FC 2020*. Springer, 2020, pp. 404–420.

[37] R. Cleve, "Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract)," in *STOC 1986*. ACM, 1986, pp. 364–369.

[38] O. Goldreich, S. Micali, and A. Wigderson, "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority," in *STOC 1987*. ACM, 1987, pp. 218–229.

[39] R. Cohen and Y. Lindell, "Fairness versus Guaranteed Output Delivery in Secure Multiparty Computation," in *ASIACRYPT 2014*. Springer, 2014, pp. 466–485.

[40] Y. Ishai, R. Ostrovsky, and V. Zikas, "Secure Multi-Party Computation with Identifiable Abort," in *CRYPTO 2014*. Springer, 2014, pp. 369–386.

[41] M. Hirt, C. Lucas, and U. Maurer, "A Dynamic Tradeoff between Active and Passive Corruptions in Secure Multi-Party Computation," in *CRYPTO 2013*. Springer, 2013, pp. 203–219.

[42] A. Patra and D. Ravi, "Beyond Honest Majority: The Round Complexity of Fair and Robust Multi-party Computation," in *ASIACRYPT 2019*. Springer, 2019, pp. 456–487.

[43] F. Baldimtsi, A. Kiayias, T. Zacharias, and B. Zhang, "Crowd Verifiable Zero-Knowledge and End-to-End Verifiable Multiparty Computation," in *ASIACRYPT 2020*. Springer, 2020, pp. 717–748.

[44] S. Kanjalkar, Y. Zhang, S. Gandlur, and A. Miller, "Publicly Auditable MPC-as-a-Service with succinct verification and universal setup," in *EuroS&P Workshops 2021*. IEEE, 2021, pp. 386–411.

[45] C. Baum, I. Damgård, V. Lyubashevsky, S. Oechsner, and C. Peikert, "More Efficient Commitments from Structured Lattice Assumptions," in *SCN 2018*. Springer, 2018, pp. 368–385.

[46] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012*. ACM, 2012, pp. 309–325.

[47] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, "Publicly Accountable Robust Multi-Party Computation," Cryptology ePrint Archive, Tech. Rep. 2022/436, 2022.

[48] D. Beaver, "Efficient Multiparty Protocols Using Circuit Randomization," in *CRYPTO '91*. Springer, 1991, pp. 420–432.

[49] R. Cramer and I. Damgård, "On the Amortized Complexity of Zero-Knowledge Protocols," in *CRYPTO 2009*. Springer, 2009, pp. 177–191.

[50] V. Lyubashevsky, "Lattice Signatures without Trapdoors," in *EUROCRYPT 2012*. Springer, 2012, pp. 738–755.

[51] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," in *FOCS 2001*. IEEE Computer Society, 2001, pp. 136–145.

[52] C. Baum, I. Damgård, K. G. Larsen, and M. Nielsen, "How to Prove Knowledge of Small Secrets," in *CRYPTO 2016*. Springer, 2016, pp. 478–498.

[53] R. Cramer, I. Damgård, C. Xing, and C. Yuan, "Amortized Complexity of Zero-Knowledge Proofs Revisited: Achieving Linear Soundness Slack," in *EUROCRYPT 2017*, 2017, pp. 479–500.

[54] R. del Pino, V. Lyubashevsky, and G. Seiler, "Lattice-Based Group Signatures and Zero-Knowledge Proofs of Automorphism Stability," in *CCS 2018*. ACM, 2018, pp. 574–591.
[55] A. Jain, S. Krenn, K. Pietrzak, and A. Tentes, "Commitments and Efficient Zero-Knowledge Proofs from Learning Parity with Noise," in *ASIACRYPT 2012*. Springer, 2012, pp. 663–680.
[56] X. Xie, R. Xue, and M. Wang, "Zero Knowledge Proofs from Ring-LWE," in *CANS 2013*. Springer, 2013, pp. 57–73.
[57] F. Benhamouda, S. Krenn, V. Lyubashevsky, and K. Pietrzak, "Efficient Zero-Knowledge Proofs for Commitments from Learning with Errors over Rings," in *ESORICS 2015*. Springer, 2015, pp. 305–325.
[58] C. Boschini, J. Camenisch, and G. Neven, "Relaxed Lattice-Based Signatures with Short Zero-Knowledge Proofs," in *ISC 2018*. Springer, 2018, pp. 3–22.
[59] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Des. Codes Cryptogr.*, vol. 75, no. 3, pp. 565–599, 2015.
[60] I. Damgård, C. Orlandi, A. Takahashi, and M. Tibouchi, "Two-Round n-out-of-n and Multi-signatures and Trapdoor Commitment from Lattices," in *PKC 2021*. Springer, 2021, pp. 99–130.
[61] D. Micciancio and C. Peikert, "Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller," in *EUROCRYPT 2012*. Springer, 2012, pp. 700–718.
[62] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," in *ECCV 2016*. Springer, 2016, pp. 630–645.
[63] V. Lyubashevsky, C. Peikert, and O. Regev, "A Toolkit for Ring-LWE Cryptography," in *EUROCRYPT 2013*. Springer, 2013, pp. 35–54.
[64] CSIRO Data61 Engineering & Design, "MP-SPDZ," https://github.com/data61/MP-SPDZ, 2022.
[65] M. Keller, "MP-SPDZ: A Versatile Framework for Multi-Party Computation," in *CCS 2020*. ACM, 2020, pp. 1575–1590.
[66] E. Orsini, "Efficient, Actively Secure MPC with a Dishonest Majority: A Survey," in *WAIFI 2020*. Springer, 2020, pp. 42–71.
[67] 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 *EUROCRYPT 2012*. Springer, 2012, pp. 483–501.
[68] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai, "Threshold Cryptosystems from Threshold Fully Homomorphic Encryption," in *CRYPTO 2018*. Springer, 2018, pp. 565–596.
[69] T. Attema, V. Lyubashevsky, and G. Seiler, "Practical Product Proofs for Lattice Commitments," in *CRYPTO 2020*. Springer, 2020, pp. 470–499.
[70] D. Rotaru, N. P. Smart, T. Tanguy, F. Vercauteren, and T. Wood, "Actively Secure Setup for SPDZ," Cryptology ePrint Archive, Tech. Rep. 2019/1300, 2019.
[71] D. Unruh, "Post-quantum Security of Fiat-Shamir," in *ASIACRYPT 2017*. Springer, 2017, pp. 65–95.
[72] C. Baum, J. Bootle, A. Cerulli, R. del Pino, J. Groth, and V. Lyubashevsky, "Sub-linear Lattice-Based Zero-Knowledge Arguments for Arithmetic Circuits," in *CRYPTO 2018*. Springer, 2018, pp. 669–699.
[73] R. del Pino and V. Lyubashevsky, "Amortization with Fewer Equations for Proving Knowledge of Small Secrets," in *CRYPTO 2017*. Springer, 2017, pp. 365–394.
[74] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic Evaluation of the AES Circuit," in *CRYPTO 2012*. Springer, 2012, pp. 850–867.
[75] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of Learning with Errors," *J. Math. Cryptol.*, vol. 9, no. 3, pp. 169–203, 2015.
[76] M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer, "Estimate All the {LWE, NTRU} Schemes!" in *SCN 2018*. Springer, 2018, pp. 351–367.
[77] D. Micciancio and O. Regev, *Lattice-based Cryptography*. Springer, 2009, pp. 147–191.
[78] M. Rückert and M. Schneider, "Estimating the Security of Lattice-based Cryptosystems," Cryptology ePrint Archive, Tech. Rep. 2010/137, 2010.

APPENDIX A
PRELIMINARIES

For completeness, we present the most important preliminaries for our protocols in this section – including SPDZ and the BGV encryption scheme. A description of the BDLOP commitment scheme can be found in Section VI.

A. SPDZ

The SPDZ protocol by Damgård et al. [7] has given rise a to a line of efficient MPC protocols in the dishonest majority setting (see [66] for an overview). It computes arbitrary functions that are representable as arithmetic circuits by separating the secure computation in a very efficient online phase and a more demanding but input-independent offline phase. The latter is also function-independent in the sense that only the size of the circuit (number of multiplications and inputs) has to be known in the offline phase.

In most SPDZ-like protocols, the online phase uses a combination of full-threshold secret-sharing and so-called information theoretic MACs. This combination results in authenticated shares $[\![x]\!]_i = ([x]_i, [\alpha \cdot x]_i)$ for MAC key α. Linear operations on these authenticated shares can be computed locally (without communication) and very efficiently as the used secret-sharing scheme is linear. Multiplications of shares are computed with Beaver's technique [48] (analogously to (5)): $[\![x \cdot y]\!]_i := [\![c]\!]_i + u \cdot [\![a]\!]_i + v \cdot [\![b]\!]_i + u \cdot v$ is computed to multiply $[\![x]\!]_i$ and $[\![y]\!]_i$. This requires triples $[\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i = [\![a \cdot b]\!]_i$ and opened values $u := x - b, v := y - a$. For this and final outputs, shares have to be *opened*, i.e., all parties get to know x for $[\![x]\!]_i$. In the protocol, this means that all parties publish their shares $[x]_i$. The MACs $[\alpha \cdot x]_i$ can be used to verify all opened shares in an aggregated way at the end of the online phase [8].

To use Beaver's technique, the above precomputed triples $[\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i$. Ensuring that a and b are uniformly random, makes Beaver's technique perfectly private (as only $[\![x - b]\!]_i$ and $[\![y - a]\!]_i$ are opened and this masks x and y perfectly). The correlated randomness ($c = a \cdot b$) implies correctness and verifiability (as MACs allow us to verify openings of linear combinations of authenticated shares and all operations are now linear).

Several ways to compute these triples have been proposed, e.g., MASCOT [12] uses an OT-based offline phase, Overdrive [15] and TopGear [16] compute triples with the linear homomorphic BGV encryption scheme (improving on original use of BGV in SPDZ [7], [8]).

These building blocks (linear authenticated secret-sharing, Beaver's technique, and a secure way to compute triples) allow for the evaluation of arbitrary arithmetic circuits in a dishonest majority setting. Most of the computation and all expensive cryptographic primitives are moved to the offline phase, leaving only a very efficient linear online phase.

B. BGV

To better describe our protocol based on LHE BGV in Section V-B1, we summarize the most important details of the BGV encryption scheme [46] here. We use the version of BGV without modulus switching. This is easier to present and analyze. Additionally, Keller et al. [15] show little difference when comparing parameter sizes for SPDZ without modulus switching [7] and SPDZ with modulus switching [8]. However, our analysis can be adjusted for other variants of BGV in a straightforward way.

The public key $k := (k[0], k[1]) = (k[0], k[0] \cdot s + p \cdot \epsilon) \in R_q^2$ (for a different prime $q > p$) is constructed from a private

key $s \in R_q$ and small noise $\epsilon \in R_q$ sampled from D_{σ_s} and D_{σ_ϵ}, respectively, while $k[0]$ is sampled uniformly at random. Encryption of a plaintext $x \in R_p$ is then defined as

$$\mathsf{Enc}_k(x, (v, \boldsymbol{e})^{\mathrm{T}}) \coloneqq \begin{pmatrix} k[1] \cdot v + p \cdot \boldsymbol{e}[0] + x \\ k[0] \cdot v + p \cdot \boldsymbol{e}[1] \end{pmatrix}$$

with encryption randomness $(v, \boldsymbol{e}) \in R_q \times R_q^2$ sampled from D_{σ_v} and D_{σ_e}, respectively.[15] The second component $k[1]$ of the public key and both components of the ciphertext (bar the addition of x) form Ring-LWE samples. Thus, the private key stays hidden due to the hardness of Ring-LWE and the plaintext x is hidden because we mask it with a value from a distribution that is indistinguishable from random. By construction we get that $(\mathsf{Enc}(x)[0] - s \cdot \mathsf{Enc}(x)[1]) \bmod p$ recovers x.

The above description of BGV is linear homomorphic as we can add up ciphertexts (component-wise) to get a ciphertext that encrypts the sum of the plaintexts (of the summed up ciphertexts). BGV can also be used as a somewhat homomorphic encryption (SHE) scheme. For this, we define a ciphertext to be an element of R_q^3 instead of R_q^2 with encryption defined as above but with the third component as zero. Two ciphertexts with zero third components can be multiplied as

$$\mathsf{Enc}(x) \cdot \mathsf{Enc}(y) = a \cdot b \coloneqq \begin{pmatrix} a[0] \cdot b[0] \\ a[1] \cdot b[0] + a[0] \cdot b[1] \\ -a[1] \cdot b[1] \end{pmatrix}.$$

Decryption of $\mathsf{Enc}(x) = c$ is now $(c[0] - s \cdot c[1] - s^2 \cdot c[2]) \bmod p$. For unmultiplied ciphertexts, this is exactly the decryption of LHE BGV.

Appendix B
Accountable Robust Setup Phase

Our setup requires the generation of commitment parameters (for the online and offline phases) and a threshold public key (PK) cryptosystem (for the offline phase). The commitment keys can be produced by a standard CRS (or random oracle) functionality (as in [10], [14]). For public key cryptography, we present a robust and accountable protocol based on Asharov et al.'s work [67]—but again, we benefit of the capabilities of current commitment schemes (discussed in Section VI) and get a more efficient protocol. We do this for the linear homomorphic version of the BGV encryption scheme, while the extension to SHE BGV (and related schemes) is straightforward.

The key generation and distributed decryption procedure is described in the full version in more detail. For the lack of space, we also provide the protocol and functionality descriptions only there.

We note that the use of Shamir secret-sharing in combination with lattice cryptography comes with many potential pitfalls in the setup phase, particularly for naive implementations. For example, a main concern with lattice-based encryption is the so-called noise that we want to keep small. Using standard masking techniques for distributed decryption [7], [8], [15], [16] introduces unpredictable noise increase and makes decryption impossible. Techniques that avoid this unpredictable behavior can still increase the decryption noise by a factor of about $\mathcal{O}((n!)^3)$ and would lead to a comparable increase of q [68]. Natural alternative techniques (pre-agreeing on shares of decryption masks) would instead increase the communication complexity of each decryption to $\mathcal{O}(n^2 \cdot t)$. By building on Asharov et al.'s protocol [67], we are able to avoid the above issues. Additionally, our construction provides better efficiency by utilizing commitments and better NIZKPs. We proceed with a description of our construction in what follows.

The main idea of our construction is to have a full-threshold sharing of the BGV private key, while each party also has a sharing of the (full-threshold) key shares of all other parties. This provides robustness and the efficiency of the full-threshold variant. To get accountability, we use commitments to all shares and ZKPs to make sure that values were constructed in the right way. For linear homomorphic BGV, one needs range proofs to make sure that the private keys are short. We can do this by extending recent range proof constructions [69] from $\mathbb{Z}_q$ to R_q (see Appendix E for details). This new alternative construction could be of independent interest.

The other proofs (for the rest of the key generation and distributed decryption) boil down to proving linear relations on commitments. For somewhat homomorphic BGV, non-linear components appear. A natural way to handle this is to run our complete protocol (using LHE multiplication in Π_{offline} as in Section V-B1) to get the data needed to run SHE BGV. This is similar to the core idea of using MPC to generate BGV keys by Rotaru et al. [70]. However, as we want a (strong) accountable setup, our protocol is the only (efficient) protocol that can be used for this purpose. This is because other protocols with identifiable abort, such as [11], [14], [28], do not provide strong public accountability and/or use SHE BGV themselves. In sum, we get the following theorem.

Theorem 4. *The protocol Π_{PK} is a publicly accountable and t-robust protocol for BGV key generation and distributed decryption. That is, Π_{PK} UC-realizes the functionality $\mathcal{F}_{\text{PK}}$ in the $(\mathcal{F}_{\text{BB}}, \mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{PKI}})$-hybrid model under the assumption that the used commitment scheme is statistically binding and computationally hiding.*

Appendix C
Functionalities and Protocols

We provide the remaining protocols and functionalities that were left out in the main part. This includes Fig. 12 for the functionality of the offline phase. The protocol and functionality for key generation and distributed decryption are available in the full version.

Appendix D
Security Proofs

A. Proof of Theorem 2

Here, we sketch the proof for Π_{offline} realizing $\mathcal{F}_{\text{offline}}$.

Proof (Sketch). The main difference to [10], [14] is the way shares are constructed, the used ZKPs, and that inputs play

[15] There is an additional third zero-component of the encryption used for SHE BGV; for LHE BGV, this can be omitted.

***Prepare* (once):** On input $(\texttt{prep}, par)$ by each $\mathcal{P}_i \in \mathcal{P}$:
  Send par to the $\mathcal{A}$ and receive $(\mathcal{M}, \beta) \in \mathcal{V}$.
  if $\beta = \texttt{abort}$ **then reply** $\mathcal{M}, \bot$ to $\mathcal{P}_i$.
  if $|\mathcal{C}| < t$ **then**
    Pick $\boldsymbol{r}, \boldsymbol{a}, \boldsymbol{b}$ randomly and set $\boldsymbol{c} \coloneqq \boldsymbol{a} \cdot \boldsymbol{b}$.
    Pick shares $[\boldsymbol{r}]_j, [\boldsymbol{a}]_j, [\boldsymbol{b}]_j, [\boldsymbol{c}]_j$ for all $\mathcal{P}_j$.
    Compute commitments for all honest shares.
    Send shares for corrupted parties $\mathcal{P}_j$ and commitments for honest parties to $\mathcal{A}$.
  else
    Send $(\texttt{shares})$ to $\mathcal{A}$.
    Let the adversary pick $\boldsymbol{r}, \boldsymbol{a}, \boldsymbol{b}$, and $\boldsymbol{c} \coloneqq \boldsymbol{a} \cdot \boldsymbol{b}$.
    Let $\mathcal{A}$ also pick shares for all parties.
    Compute commitments for all honest shares.
    Send commitments for honest parties to $\mathcal{A}$.
  Receive $(\mathcal{M}', \beta') \in \mathcal{V}$ with $\mathcal{M}' \supseteq \mathcal{M}$ (overriding the previous values of $\mathcal{M}$ and β), commitments, and decommitments for corrupted parties $\mathcal{P}_j$.
  Set the commitments of parties with invalid decommitments to $\top$. Also add these parties to $\mathcal{M}$ and $\mathcal{M}'$.
  if $\beta = \texttt{abort}$ **then reply** $\mathcal{M}, \bot$ to $\mathcal{P}_i$.
  else reply $\mathcal{M}, \langle \boldsymbol{r} \rangle_i, \langle \boldsymbol{a} \rangle_i, \langle \boldsymbol{b} \rangle_i, \langle \boldsymbol{c} \rangle_i$ to $\mathcal{P}_i$.
***Input* (once):** On input $(\texttt{input}, x_j)$ by each $\mathcal{I}_j \in \mathcal{I}$ and input $(\texttt{input})$ by each $\mathcal{P}_i \in \mathcal{P}$:
  Get x_j for corrupted $\mathcal{I}_j$ from $\mathcal{A}$.
  Pack all x_j into $\boldsymbol{x}$.
  if $\beta = \texttt{ok}$ **then**
    if $|\mathcal{C}| \geq t$ **then** Send $(\texttt{input}, \boldsymbol{x})$ to $\mathcal{A}$.
    else Send $(\texttt{mask}, \boldsymbol{m} \coloneqq \boldsymbol{x} - \boldsymbol{r})$ to $\mathcal{A}$.
    Receive $(\mathcal{M}'', \beta'') \in \mathcal{V}$ with $\mathcal{M}'' \supseteq \mathcal{M}$ (overriding the previous values of $\mathcal{M}$ and β).
    if $\beta = \texttt{ok}$ **then reply** $\mathcal{M}, \boldsymbol{m}$ to $\mathcal{P}_j$.
  reply $\mathcal{M}, \bot$ to $\mathcal{P}_j$.
***Audit*:** On input $(\texttt{audit}, \texttt{prep}, par)$ by $\mathcal{J}$:
  if $\beta' \neq \texttt{ok}$ **then reply** $\mathcal{M}', \bot$.
  else reply $\mathcal{M}', \langle \boldsymbol{r} \rangle_{\text{audit}}, \langle \boldsymbol{a} \rangle_{\text{audit}}, \langle \boldsymbol{b} \rangle_{\text{audit}}, \langle \boldsymbol{c} \rangle_{\text{audit}}$.
***Audit*:** On input $(\texttt{audit}, \texttt{input})$ by $\mathcal{J}$:
  if $\beta'' = \texttt{ok}$ **then reply** $\mathcal{M}'', \boldsymbol{m}$ **else reply** $\mathcal{M}'', \bot$.

Fig. 12. Offline functionality $\mathcal{F}_{\text{offline}}$.

a part in the offline phase. Correctly constructing shares follows a similar approach by constructing ciphertexts of shares. The relations that the ZKPs prove are chosen in a way to get the necessary properties for the protocol to work. Also, extracting the decommitments for corrupted parties from (extractable) commitments instead of ciphertexts still works. Involving the offline protocol in the input phase gives us usability improvements for the clients (only a ciphertext has to be published) while revealing no additional information (the inputs are masked with values that are uniformly random by construction; assuming not too many parties are corrupted, in which case we do not have to hide the inputs at all). For the complete proof, see the full version. □

B. Proof of Theorem 4

Here, we sketch the proof for Π_{PK} realizing $\mathcal{F}_{\text{PK}}$.

Proof (Sketch). In the LHE case, the protocol is structured similarly to Asharov et al. [67] but ZKPs are replaced by efficient commitment-based variants. This does not influence the security.

In the SHE case, we base the key generation on our MPC protocol and thus get the required security properties. For the decryption, only an additional term is added to the ZKPs and thus the results of the LHE case can be reused. For the complete proof, see the full version. □

Appendix E
Zero-Knowledge Proofs

To make our offline phase secure, we use zero-knowledge proofs. Σ-protocols are used with the Fiat-Shamir transform to generate non-interactive zero-knowledge proofs (NIZKPs) that everyone can verify. We furthermore refer to Unruh [71] for a discussion of Σ-protocols in the post-quantum setting. To prove the correctness of the encryption and the commitment for the same plaintext we use a combination of rejection sampling [50] and the aggregation technique of [49].

Additionally, zero-knowledge proofs that only involve commitments can be made very efficient – without aggregation techniques for the BDLOP scheme. We use this to prove correct committing [45], as well as products [69] and linear relations using commitments [45], [54]. The latter two are combined to get range proofs in an accountable encryption scheme key generation.

Zero-knowledge proofs are used in two settings that were not described yet (how to prove plaintext-ciphertext multiplications was discussed in Section V-B1 and the proofs used in the online phase are straight adaptions of the ones for linear relations [45]). The first is range proofs for the BGV key generation. Secondly, we prove many *committing ciphertexts* statements (and other aggregated proofs) in the offline phase. Both are discussed in the next sections.

A. Range Proofs

To get range proofs for R_q, we combine the existing NIZKPs [45], [54], [69] in the following way.

1) A value $x \in R_q$ is split in its bits $\boldsymbol{b}, \boldsymbol{b}[i] \in \{0,1\}^N$. The negated bits $\bar{\boldsymbol{b}}$ are computed.
2) A product proof [69] is used to prove $\boldsymbol{b} \cdot \bar{\boldsymbol{b}} = \boldsymbol{0}$.
3) The generalized sum proof [54] (as a generalization of the proof by Baum et al. [45]) is used to prove $\boldsymbol{b} + (\bar{\boldsymbol{b}} - 1) = \boldsymbol{0} \Leftrightarrow \bar{\boldsymbol{b}} = 1 - \boldsymbol{b}$. The generalization allows us to prove this sum for the extended commitments from the product proof (the commitment key for the product proof has a third matrix $\mathbf{A}_2$ similar to $\mathbf{A}_1$).
4) A proof of sum [45], [54] is used to prove $x = \sum_{i=0}^{l-1} 2^i \cdot \boldsymbol{b}[i] - 2^l \cdot \boldsymbol{b}[k]$.

Note that we do not try to prove the length of a single integer and pack it in the coefficients of an R_q element as in [69]. Instead, we want to prove a bound on $\|x\|_\infty$. As in [69], the steps 2 and 3 prove that $\boldsymbol{b}[i]$ has binary coefficients. Step 4 then implies $\|x\|_\infty \leq 2^l$. We do not want to introduce any slack

in these range proofs to give a key generation protocol that can also be used in applications where tighter parameters are needed. As the key generation is done only once in the offline phase, we opted for this kind of construction. Non-power-of-two bounds could be supported by adding more terms.

B. Aggregated Proofs

The adapted zero-knowledge proofs we use are based on rejection sampling [50] and classical aggregation [49]. Details on this can be found in the full version. We do not get proofs that have a slack as small as the one obtained by Baum et al. [72], but their proof also does not prove exact relations but one that is off by a factor of two (i.e., for the ring case, it proves $\mathbf{A}\boldsymbol{s} = 2\boldsymbol{t}$ instead of $\mathbf{A}\boldsymbol{s} = \boldsymbol{t}$).

Other techniques are possible as well – and give a smaller slack than our approach – but they also have some downsides. The approach of [52] uses a cut-and-choose technique which we want to avoid as this was identified to be impractical by Keller et al. [15]. Cramer et al. [53] need to amortize over many instances (η^2 or $\eta^{3/2}$) to be efficient. The work by del Pino and Lyubashevsky [73] builds upon [52], [53] to reduce the large number of instances required by Cramer et al. [53] but it is still a cut-and-choose proof. Another reason why we opted for the classical aggregation technique [49] is its simplicity and use in other protocols. This should make it straightforward to implement our adaptions to it if our MPC protocol is to be implemented.

We do not claim that our approach is a new idea – in fact, it was mentioned in [52], [53] that one can construct a zero-knowledge proof in such a way – but it was not formalized, to our knowledge. Additionally, we get reasonably good parameters, even with the exponential slack (as seen in Section VII).

Also note that the operations for the combination of rejection sampling and classical aggregation are the same. The runtime can increase as parts of the proof have to be repeated but also the aggregation in SPDZ [7] has a probability of repeating the proof: With probability $1/32$ the proof has to be discarded.

Appendix F
Parameters (Continued)

We use a statistical security of $40\,\text{bit}$ and a computational security level of $128\,\text{bit}$ in our parameter search in Section VII, unless stated otherwise. Also, if we do not give results for both, we consider the variant of our protocol with somewhat homomorphic encryption (and not with linear homomorphic encryption).

For getting the BGV parameters, we use a technique to assess the security level of various parameter sizes that is similar to [8], [74] but we consider the worst-case bounds for elements that are distributed w.r.t. Gaussian distributions (instead of average-case bounds).

We chose the parameters for BDLOP scheme by searching for the combination that achieves a required security level (we fixed the computational security, while varying the statistical security parameter η, e.g., used for zero-knowledge proofs)

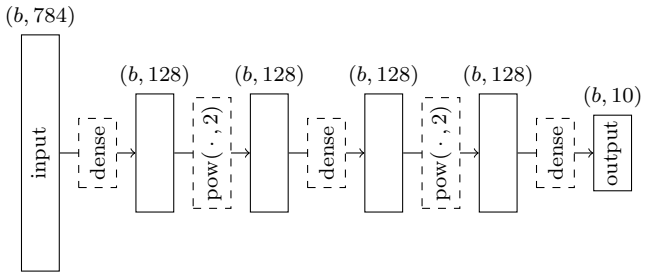


Fig. 13. Architecture of "network A" [1], [64]. The shape of the input and (intermediate) feature tensors is given above each tensor. b represents the batch size. Dashed boxes represent the network layers (dense layers, nonlinearities).

while minimizing the total size of a plaintext-randomness-commitment tuple. By optimizing for the size of this combined tuple, we avoid favoring one aspect over others. To obtain our results for commitment parameters, which are given in Table II, we used the LWE Estimator [75] with cost models from [76] to estimate the hardness of M-LWE. Additionally, we used results from the literature [75], [77], [78] to estimate the hardness of M-SIS. We also made sure that the constraints for the trapdoors are met (see the full version for more details). Extra parameters (for example for the commitments in Π_{equiv}) can also be found in the full version.

Appendix G
Benchmarks

To compare the runtime of our protocol to SPDZ and BoBW$_{[14]}$, we implemented a benchmark that emulates the online and offline phase. For the offline phase, we run all parts of the triple generation (except sacrificing for SPDZ, i.e., we assume that sacrificing is free; additionally, preparation for inputs is not considered here) and the core operations in the online phase (we leave out the input phase for all protocols but the output phase is implemented for our protocol's online phase, where this entails NIZKPs unlike in the other protocols; MAC checks for the other protocols are considered to be free, i.e., not implemented). The runtime of the benchmarks for the offline phase are then extrapolated to get as many triples as multiplications needed for the circuit we consider in the online phase. The results of this are shown in Fig. 10(b). Further parts where our benchmark differs from a full implementation is that we use broadcast channels instead of a bulletin board and sampling of randomness was changed to uniformly random sampling to simplify the implementation. The latter does not effect the runtime behavior of the protocols we want to compare. For the online phase, we evaluate the arithmetic core of "network A" (see Fig. 13). The final argmax layer is left out as the operations there are not purely arithmetic. Additional details on our experiments, as well as an extended discussion, can be found in the full version.

We ran our benchmark on a single server (AMD EPYC 7443 CPU, 24 cores, $2.85\,\text{GHz}$, $512\,\text{GB}$ RAM) to get the presented results. All experiments were done on a single machine running the code for all parties. We emulate the network behavior with the netem functionality.