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

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.

An extended abstract of this paper appeared in the Proceedings of the 2022 IEEE Symposium on Security and Privacy [1].

Contents

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 [2]–[7]. One major contribution to this success are efficient two-phase protocols like SPDZ [8], [9] and "SPDZ-like" protocols [7], [10]–[21] 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 [22], [23], e-voting protocols [24], [25], and cloud services for privacy preserving computation [18], [26]–[28] – including ML tasks [2], [6], [29]–[31]. These client-server applications often require *publicly identifiable abort* [12], [15], [32]: 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 [39], [40], 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. [15] 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 [41]). This property is called *robustness* or *guaranteed output delivery* [33], [42]–[52] 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 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* [38] 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. [15] 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 [14], [32], [53]. 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. Section VIII and [14]). 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 [51], [54]. 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 [51], [54] 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.

[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* [11], [33]–[37] and is implied by publicly identifiable abort [38].

[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]	–	–	–/+
[11], [55]	+	–	–
[56]	+	–	+
[12], [32]	○	○	–
[15]	+	+	–
[51]	should work for and inherit properties of any		+[c]
[54]	MPC protocol with identifiable abort		+[c]
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 [8], [9], most SPDZ-like protocols [13], [16], [19]–[21], protocols with (non-public) identifiable abort [14], [53], etc.
[c] Restarts or runs multiple parallel instances of the (inner) MPC protocol.

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.

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. [11], [12], [15], [32]) 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 [8] or the state-of-the-art lattice-based commitment scheme by Baum et al. [57], 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. 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. [15] 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 [58] 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. [57] which allows for improving its homomorphic properties without increasing the underlying plaintext space (cf. Section VI and Appendix E). 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).

[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 [25] to obtain an end-to-end verifiable tally-hiding voting system for lower-stakes elections.

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

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. [57] (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).

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 $R := \mathbb{Z}[X]/\Phi_m(X)$ 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 $\big($we use representatives from $\{-(p-1)/2, \dots, (p-1)/2\}\big)$. 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 $k \in \{1, 2, \dots, \infty\}$ 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 (definition in Appendix E). We write $\boldsymbol{x} \xleftarrow{\$} A(\dots)$ if $\boldsymbol{x}$ is sampled by running a probabilistic algorithm A.

We use $\mathcal{P}_i \in \mathcal{P} := \{\mathcal{P}_1, \dots, \mathcal{P}_n\}$ 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 $\mathcal{H} := \mathcal{P} \setminus \mathcal{C}$ 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; all of our results are independent of the exact set of corrupted input parties.

We consider arithmetic circuits f, where I is the number of inputs (i.e., $|\mathcal{I}| = I$). A 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]–[9], [16], [17], [21], we use the BGV encryption scheme [58]. 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 setup and offline phases of our protocol (cf. Section V-B and Appendices A-B and B). We refer to the commitment scheme of Baum et al. [57] 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.[6] 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, we consider the case where outputs are public, i.e., may be revealed to everyone. One can use standard masking techniques to reveal private outputs only to a specific party (for completeness we describe this extension in Appendix G); all our results carry over. We use bulletin boards to publish data just as almost all other protocols with public verifiability/accountability, e.g., [11], [12], [15], [32]. 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 [59] (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 a variant of ZKP aggregation that combines classical aggregation [60] and rejection sampling [61] 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.

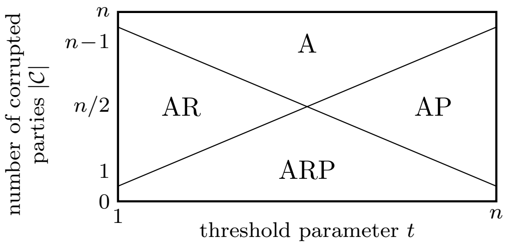


Fig. 1. Security guarantees of our protocol. Public accountability is indicated with "A" (recall that this also implies public verifiability), robustness/guaranteed output delivery with "R", and full privacy with "P".

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 Appendix B. 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

[6] Our construction aims for simple setup on the client side and to allow synergy effects in settings like e-voting (see Appendix G-D). Focusing on other goals might lead to a different protocol design.

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) [62]–[67] 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.Note that one can reduce our setup assumptions to only requiring $\mathcal{F}_{\text{RO}}$ and $\mathcal{F}_{\text{BB}}$ as one can construct the other functionalities from these. $\mathcal{F}_{\text{CRS}}$ can be implemented with a call to $\mathcal{F}_{\text{RO}}$ and $\mathcal{F}_{\text{PKI}}$ can be implemented by a simple protocol that lets all parties publish their public keys on the (authenticated) bulletin board.

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 $|\mathcal{M}| > n - t$.*

```
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 $\mathcal{A}$ (overriding the input
    $x_j$). 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 $\beta$).
Compute (once): On input $(\texttt{comp}, f)$ by each $\mathcal{P}_i \in \mathcal{P}$:
    if $\beta = \texttt{ok}$ then
        Compute the result $\boldsymbol{y} := f(\boldsymbol{x})$.
        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 $\beta$).
        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 $\mathcal{F}_{\text{online}}$.

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 in Figs. 13 and 14. The set of possible verdicts

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

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, 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.

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

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

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

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

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 [59]. A *view* of a shared value x is

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

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 $\boldsymbol{m} := \boldsymbol{x} - \boldsymbol{r}$ 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 $\langle \boldsymbol{x}\rangle_i := \langle \boldsymbol{r}\rangle_i + \boldsymbol{m}$. 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:

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

for values x and y that should be multiplied. $u := x - b, v := y - a$ 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 $\big(u$ and v above are results of the openings for $\langle x - b\rangle_i$ and $\langle y - a\rangle_i\big)$ 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 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. [11] or Cunningham et al. [15], 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 [11], [15]. There, equivocation is necessary for the simulation-

[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.

```
1  *Prepare* (once): On input $(\mathtt{prep})$ by each $\mathcal{P}_i \in \mathcal{P}$:
2      Parties setup commitment parameters *par* via $\mathcal{F}_{\text{CRS}}$.
3      Parties send $(\mathtt{prep}, \mathit{par})$ 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 $(\mathtt{input}, x_j)$ by each $\mathcal{I}_j \in \mathcal{I}$ and input $(\mathtt{input})$ 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 $(\mathtt{comp}, f)$ by each $\mathcal{P}_i \in \mathcal{P}$:
10     Assign the identifiers $\mathrm{id}_x$ 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 $(\mathtt{mul}, \mathrm{id}_x, \mathrm{id}_y, \mathrm{id}_z)$ 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 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 $(\mathtt{output}, l, \mathrm{id}_x)$ do
20             $x_l := \mathtt{OpenOutput}(\langle x \rangle_i)$ (cf. Fig. 5).
21     Pack all outputs $x_l$ of output-gates into $\boldsymbol{y}$.
22     reply $\mathcal{M}, \boldsymbol{y}$.
23 *Audit*: On input $(\mathtt{audit}, \mathtt{comp}, f)$ by $\mathcal{J}$:
24     Obtain *par* via $\mathcal{F}_{\text{CRS}}$.
25     Send $(\mathtt{audit}, \mathtt{prep}, \mathit{par})$ to $\mathcal{F}_{\text{offline}}$ to obtain $\mathcal{M}, \boldsymbol{t}$ and reply $\mathcal{M}, \perp$ if $\boldsymbol{t} = \perp$.
26     Send $(\mathtt{audit}, \mathtt{input})$ to $\mathcal{F}_{\text{offline}}$ to obtain $\mathcal{M}', \boldsymbol{m}$, add $\mathcal{M}'$ to $\mathcal{M}$, and reply $\mathcal{M}, \perp$ if $\boldsymbol{m} = \perp$.
27     Unpack $\boldsymbol{t}$ to $\langle \boldsymbol{r} \rangle_{\text{audit}}, \langle \boldsymbol{a} \rangle_{\text{audit}}, \langle \boldsymbol{b} \rangle_{\text{audit}}, \langle \boldsymbol{c} \rangle_{\text{audit}}$.
28     Define input views as $\langle \boldsymbol{x} \rangle_{\text{audit}} := \langle \boldsymbol{r} \rangle_{\text{audit}} + \boldsymbol{m}$ and assign the identifiers $\mathrm{id}_x$ for inputs of $f$ to $\langle \boldsymbol{x} \rangle_{\text{audit}}$.
29     foreach gate $g \in f$ in topological order do
30         case $g$ is linear do
31             Compute $g$ locally as in Fig. 3 while only considering operations on commitments.
32         case $(\mathtt{mul}, \mathrm{id}_x, \mathrm{id}_y, \mathrm{id}_z)$ do
33             Get the next triple $\langle a \rangle_{\text{audit}}, \langle b \rangle_{\text{audit}}, \langle c \rangle_{\text{audit}}$.
34             $\langle u \rangle_{\text{audit}} := \langle x \rangle_{\text{audit}} - \langle b \rangle_{\text{audit}}$ and $\langle v \rangle_{\text{audit}} := \langle y \rangle_{\text{audit}} - \langle a \rangle_{\text{audit}}$.
35             Open $\langle u \rangle_{\text{audit}}$ and $\langle v \rangle_{\text{audit}}$ with Open while only retrieving and verifying data. Return $\mathcal{M}, \perp$ where parties in Fig. 5 would abort.
36             $\langle z \rangle_{\text{audit}} := \langle c \rangle_{\text{audit}} + u \cdot \langle a \rangle_{\text{audit}} + v \cdot \langle b \rangle_{\text{audit}} + u \cdot v$.
37         case $(\mathtt{output}, l, \mathrm{id}_x)$ do
38             $x_l := \mathtt{OpenOutput}(\langle x \rangle_i)$ (cf. Fig. 5 with the same changes as described above).
39     Pack all outputs $x_l$ of output-gates into $\boldsymbol{y}$.
40     reply $\mathcal{M}, \boldsymbol{y}$.
```

Fig. 4. Online protocol Π_{online}.

```
1  macro $\mathtt{Open}(\langle v \rangle_i, \mathcal{M}' = \emptyset)$:
2      Send $[v]_i$ and $\mathsf{R}_{\mathsf{C}}([v]_i)$ to $\mathcal{F}_{\text{BB}}$.
3      Retrieve $[v]_j$ and $\mathsf{R}_{\mathsf{C}}([v]_j)$ from other $\mathcal{P}_j \in \mathcal{P}$.
4      Add all $\mathcal{P}_j$ to $\mathcal{M}'$ and to $\mathcal{M}$ for which $\mathsf{Verify}([v]_j, \mathsf{R}_{\mathsf{C}}([v]_j), \mathsf{Com}([v]_j)) = 0$.
5      if $|\mathcal{P} \setminus \mathcal{M}'| < t$ then abort.
6      Reconstruct $v$ while ignoring shares of $\mathcal{P}_j \in \mathcal{M}'$.
7      return $v$.
8  macro $\mathtt{OpenOutput}(\langle v \rangle_i)$:
9      Commit to $[v]_i$ to get $\mathsf{Com}([w]_i)$.
10     Prove in ZK that $\mathsf{Com}([v]_i)$ and $\mathsf{Com}([w]_i)$ contain the same plaintext. Let the NIZKP be $z_i$.
11     Send $z_i$ to $\mathcal{F}_{\text{BB}}$ and retrieve $z_j$ from other $\mathcal{P}_j \in \mathcal{P}$.
12     Add all $\mathcal{P}_j$ to a new set $\mathcal{M}'$ and to $\mathcal{M}$ where verification of $z_j$ failed.
13     return $\mathtt{Open}(\langle w \rangle_i, \mathcal{M}')$.
```

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

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 and Appendix I). 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 Appendix D-A. Note that using ZKPs for all openings would only increase the amount of data that needs to be communicated without being necessary to securely realize the online phase.

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, were we use the programmable random oracle model.*

[8] Π_{equiv} is mostly equivalent to Π_{online} but Open is used instead of 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.

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 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 Appendix D-A. 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 [11], [15]) 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 [11], [15] 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, 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

```
1  macro $\texttt{Decrypt}(\mathsf{Enc}(x))$:
2      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 $\mathcal{M}$.
3      if $x = \bot$ then abort else return $x$.
4  Prepare (once): On input $(\texttt{prep}, par)$ by each $\mathcal{P}_i \in \mathcal{P}$:
5      Parties setup commitment parameters $par'$ via $\mathcal{F}_{\text{CRS}}$.
6      Parties setup a key $k$ via $\mathcal{F}_{\text{PK}}$ and get a set $\mathcal{M}$.
7      if $k = \bot$ then abort.
8      Sample coefficients $\mathbf{W}_i \xleftarrow{\$} U(R_p^{(I+3\cdot M)\times t})$ to (later)
       construct shares and masks $\boldsymbol{y}_i \xleftarrow{\$} U(R_p^{I+3\cdot M})$.
9      Encrypt these value to get $\mathsf{Enc}(\mathbf{W}_i), \mathsf{Enc}(\boldsymbol{y}_i)$.
10     Commit to $\boldsymbol{y}_i$ with $\mathsf{R_C}(\boldsymbol{y}_i)$ to get $\mathsf{Com}(\boldsymbol{y}_i)$.
11     Commit to the decommitments $\boldsymbol{y}_i, \mathsf{R_C}(\boldsymbol{y}_i)$ with $par'$
       to get $\mathsf{Com}'(\boldsymbol{y}_i), \mathsf{Com}'(\mathsf{R_C}(\boldsymbol{y}_i))$.
12     Prove Line 9 to 11 in ZK. Let the NIZKP be $z_i$.
13     Send $z_i$ to $\mathcal{F}_{\text{BB}}$ and retrieve $z_j$ for other $\mathcal{P}_j \in \mathcal{P}$.
14     Add $\mathcal{P}_j$ to a new set $\mathcal{M}'$ and to $\mathcal{M}$ where
       verification of $z_j$ failed.
15     Use $\mathsf{Com}(\boldsymbol{y}_j) = \top$ for $\mathcal{P}_j \in \mathcal{M}'$ and combine
       coefficients to $\mathsf{Enc}(\mathbf{W}) := \sum_{\mathcal{P}_j \in \mathcal{P} \setminus \mathcal{M}'} \mathsf{Enc}(\mathbf{W}_j)$.
16     foreach $\mathcal{P}_j \in \mathcal{P} \setminus \mathcal{M}'$ do
17         Define the encrypted share of $\boldsymbol{v} = \mathbf{W}[\,\cdot\,, 0]$ as
           $\mathsf{Enc}([\boldsymbol{v}]_j) := \sum_{l=0}^{t-1} j^l \cdot \mathsf{Enc}(\mathbf{W})[\,\cdot\,, l]$.
18         $\boldsymbol{m}_j := \texttt{Decrypt}(\mathsf{Enc}(\boldsymbol{y}_j) - \mathsf{Enc}([\boldsymbol{v}]_j))$.
19     Construct the view $\langle \boldsymbol{v} \rangle_i := (\boldsymbol{y}_i - \boldsymbol{m}_i, \mathsf{R_C}(\boldsymbol{y}_i),$
       $\mathsf{Com}(\boldsymbol{y}_1) - \boldsymbol{m}_1, \dots, \mathsf{Com}(\boldsymbol{y}_n) - \boldsymbol{m}_n)$.
20     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 $\boldsymbol{d}$.
21     Compute $\mathsf{Enc}(\boldsymbol{c})$ with SHE or $\texttt{Multiply}$ (Fig. 7).
22     $\langle \boldsymbol{c} \rangle_i := \langle \boldsymbol{d} \rangle_i + \texttt{Decrypt}(\mathsf{Enc}(\boldsymbol{c}) - \mathsf{Enc}(\boldsymbol{d}))$.
23     reply $\mathcal{M}, \langle \boldsymbol{r} \rangle_i, \langle \boldsymbol{a} \rangle_i, \langle \boldsymbol{b} \rangle_i, \langle \boldsymbol{c} \rangle_i$ to $\mathcal{P}_i$.
24 Input (once): On input $(\texttt{input}, x_j)$ by each $\mathcal{I}_j \in \mathcal{I}$
   and input $(\texttt{input})$ by each $\mathcal{P}_i \in \mathcal{P}$:
25     Each $\mathcal{I}_j$ audits $\mathcal{F}_{\text{offline}}$ and $\mathcal{F}_{\text{PK}}$. $\mathcal{I}_j$ also gets key $k$.
26     if $\mathcal{F}_{\text{offline}}$ outputs $\bot$ then abort.
27     Each $\mathcal{I}_j$ sends $\mathsf{Enc}_k(x_j)$ and a proof that the
       encryption is well-formed to $\mathcal{F}_{\text{BB}}$.
28     Each $\mathcal{P}_i$ retrieves $\mathsf{Enc}(x_j)$ from $\mathcal{F}_{\text{BB}}$.
29     Replace malformed inputs with zero-ciphertexts.
30     Pack all $\mathsf{Enc}(x_j)$ into $\mathsf{Enc}(\boldsymbol{x})$.
31     reply $\mathcal{M}, \boldsymbol{m} := \texttt{Decrypt}(\mathsf{Enc}(\boldsymbol{x}) - \mathsf{Enc}(\boldsymbol{r}))$ to $\mathcal{P}_i$.
32 Audit: On input $(\texttt{audit}, \texttt{prep}, par)$ by $\mathcal{J}$:
33     Perform the Prepare phase on public data.
34 Audit: On input $(\texttt{audit}, \texttt{input})$ from $\mathcal{J}$:
35     Perform the Prepare and Input phases on public
       data.
```

Fig. 6. Offline protocol Π_{offline}.

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 $m_j := y_j - [v]_j$ 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 $[v]_j := y_j - m_j$ and $\mathsf{R_C}([v]_j) := \mathsf{R_C}(y_j)$ where $\mathsf{R_C}(y_j)$ was used to compute $\mathsf{Com}(y_j)$. 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 $\mathsf{Enc}(\mathbf{W}) := \sum_i \mathsf{Enc}(\mathbf{W}_i)$ 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: $\mathsf{Enc}([\boldsymbol{v}]_i) := \sum_{l=0}^{t-1} i^l \cdot \mathsf{Enc}(\mathbf{w})[\,\cdot\,, l]$. This is a share of $\mathsf{Enc}(\boldsymbol{v}) := \mathsf{Enc}(\mathbf{W})[\,\cdot\,, 0]$, 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 [60] 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 [61][9] which decreases the slack

[9] This possibility was already mentioned in [68], [69] 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 for our use-case in Appendix F.

by approximately a factor of 2^η. While this does not solve the 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 F 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 [57]). *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 reuse 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 Appendix G-D. 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] *were we use the programmable random oracle model.*

The proof of this theorem can be found in Appendix D-B.

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 [16] (or BDOZ [10]): 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 $r = (v, \boldsymbol{e})^{\mathrm{T}}$). 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 [57], [70] 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 [10], 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 [10], [16] 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.

[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.

```
1  macro Multiply:          // to multiply BGV ciphertexts $\mathsf{Enc}(\boldsymbol{a})$ and $\mathsf{Enc}(\boldsymbol{b})$ using only LHE
2      Let $\boldsymbol{y}'_j, \boldsymbol{m}'_j$ be the subset of $\boldsymbol{y}_j, \boldsymbol{m}_j$ used to construct $\langle \boldsymbol{a} \rangle_j$.
3      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 $\mathcal{P}_j \in \mathcal{P} \setminus \mathcal{M}'$.
4      Generate re-encryption randomness $(\boldsymbol{v}_i, \boldsymbol{e}_i) \xleftarrow{\$} D_{\sigma_v}(R_q^M) \times D_{\sigma_e}(R_q^M)^2$.
5      Compute re-encrypted share of product
       $\mathsf{Enc}([\boldsymbol{c}]_i) := [\boldsymbol{a}]_i \cdot \mathsf{Enc}(\boldsymbol{b}) + \mathsf{Enc}_k(0, (\boldsymbol{v}_i, \boldsymbol{e}_i)^\mathrm{T}) = \begin{pmatrix} [\boldsymbol{a}]_i \cdot \mathsf{Enc}(\boldsymbol{b})[0] + k[1] \cdot \boldsymbol{v}_i + p \cdot \boldsymbol{e}_i[0] \\ [\boldsymbol{a}]_i \cdot \mathsf{Enc}(\boldsymbol{b})[1] + k[0] \cdot \boldsymbol{v}_i + p \cdot \boldsymbol{e}_i[1] \end{pmatrix}$.
6      Commit to $\boldsymbol{v}_i, \boldsymbol{e}_i, \mathsf{Enc}([\boldsymbol{c}]_i)$ with $par'$.
7      Let $\mathsf{Com}'(\boldsymbol{v}_i), \mathsf{Com}'(\boldsymbol{e}_i), \mathsf{Com}'(\mathsf{Enc}([\boldsymbol{c}]_i))$ be the resulting commitments.
8      Prove Line 5 to 7 in ZK. Let the NIZKP be $z_i$.
9      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 $\mathcal{P}_j \in \mathcal{P}$.
10     Add $\mathcal{P}_j$ to $\mathcal{M}$ and $\mathcal{M}'$ if verification of $z_j$ failed or $\mathsf{Verify}_{par'}(\mathsf{Enc}([\boldsymbol{c}]_j), \mathsf{R}'_\mathsf{C}(\mathsf{Enc}([\boldsymbol{c}]_j)), \mathsf{Com}'(\mathsf{Enc}([\boldsymbol{c}]_j)) = 0$.
11     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 $\mathcal{P}_i \in \mathcal{P}$.

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 [11], [15] does not combine very well with BGV encryption, i.e., it 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., [57], [71]–[74]. The overall best suited scheme appears to be the BDLOP commitment scheme [57], 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 [58], [75]. The (public) commitment parameters consist of two matrices $\mathbf{A}_0 := \begin{pmatrix} \mathbf{I}_{d_1} & \mathbf{A}'_0 \end{pmatrix}$ and $\mathbf{A}_1 := \begin{pmatrix} \mathbf{0}_{1 \times d_1} & \mathbf{I}_1 & \mathbf{A}'_1 \end{pmatrix}$ 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

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

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 Appendix E.

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. [57] for more details on the norm estimates).[12] 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. We explain this issue in more detail in Appendix E.

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.[13] For the scheme with the above modifications, we get the following theorem.

[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 $r[i]$ with overwhelming probability in N.

[12] Depending on the norm one is considering, additional factors of, for example, $\sqrt{N}$ (for the L^2-norm) or N (for the L^∞-norm) appear in front of p.

[13] This idea is similar to [70], but [70] considers the case of two primes with $p' < p$. In contrast, we consider and argue security of the case $p \mid p'$. We further note that our scheme, by supporting non-primes p', comes with the added benefit of enabling more efficient implementations by splitting p' into multiple smaller primes [76].

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 Appendix E, 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. 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. [11], [15], and compare the two approaches in Section IX. We add such trapdoors by following and adapting the construction of Damgård et al. [77] to the generalized variant of BDLOP (cf. Appendix E for full 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 [77], [78]). We denote the public commitment parameters of this second instance by *par*$'$.

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 I.

BGV Parameters: The main parameter that determines practicality of the BGV scheme is the ciphertext modulus q.[14] 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 [16] and TopGear [17] (two recent very efficient protocols without identifiable abort) in addition to classical SPDZ [8], [9] 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 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.

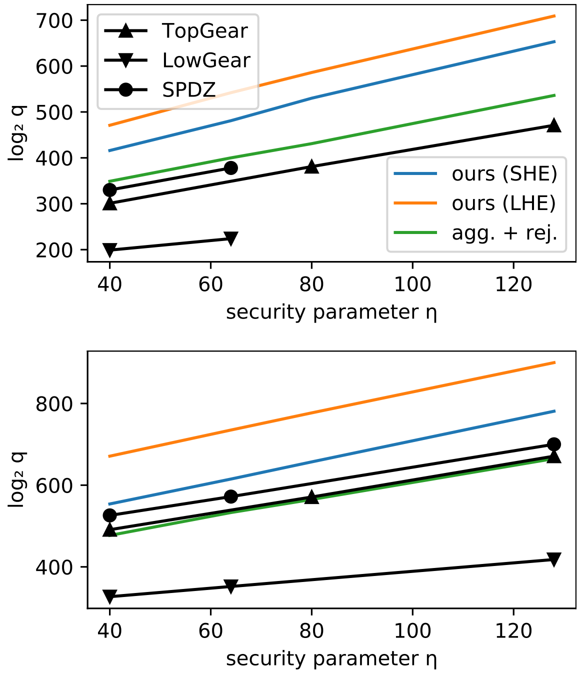


Fig. 8. Comparison of BGV parameters against TopGear, LowGear, and SPDZ for $\log p = 64$ (top) and $\log p = 128$ (bottom). Parameters are essentially independent of the choice of t and n, with the above values being for $n = t = 2$. E.g., for $\eta = 80$ and $\log p = 64$, increasing $n = t$ from 2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

Recall that our offline phase combines zero-knowledge proof aggregation [60] with rejection sampling [61]. 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

[14] While N also depends on the security parameter η, changing N does not affect the combined size of ciphertexts (an increase of N allows for encrypting more plaintexts within each ciphertext). We thus concentrate on q.

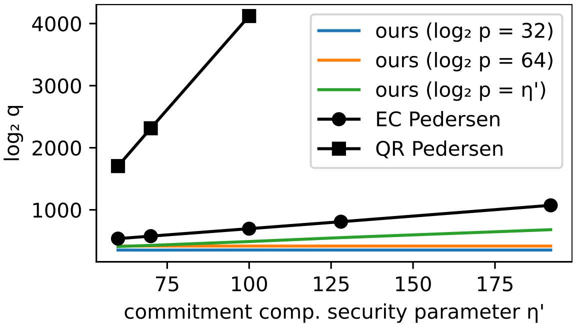


Fig. 9. Comparison of BGV parameters against Cunningham et al. [15] with elliptic curve (EC) and quadratic residue (QR) Pedersen commitments.

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. [15]) 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 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 [79] 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; additionally, in Appendix I, we show that changing N can have negative effects on the efficiency of BDLOP). 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 [76]. Also note that for commitments with *par*$'$, we do not need to increase the modulus of the commitment scheme and can use the BGV modulus q. We also computed parameters for the generalized commitment scheme but with equivocation. The results can be found in Appendix I 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 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

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 [15], which is a SPDZ-like protocol that offers strong publicly identifiable abort, with a best-of-both-worlds protocol [51], [54] that provides robustness. While not formally proven,[15] this combination, which we denote by BoBW$_{[15]}$, likely provides strong publicly identifiable abort (and also strong public accountability) while additionally being robust.

Let us first discuss why simply restarting the protocol is not enough (and constructions as in [51], [54] are needed). The usual notion of security with abort involves an ideal functionality that models the real protocol. There, in almost all cases, the adversary learns the output of the protocol in case of an abort (or even gets to know the output and has then the chance to cause an abort based on this knowledge). With this knowledge, the subsequent protocol runs could be influenced. An example where this gets very apparent is in auctions. Knowing the auction result (possible only the winner; other settings might include the highest bid as well), an adversary can easily abort the auction and use a higher bid in the next run. As already mentioned, restarting the protocol is also simply undesirable in the client-server context, e.g., in e-voting, where it implies re-running the election. With this in mind, we continue with the comparison of our protocol and BoBW$_{[15]}$.

Table III provides an overview of the properties of our protocol, BoBW protocols [51], [54], and Cunningham et al.'s protocol [15], where the properties of BoBW$_{[15]}$ can be derived from the combination of [51], [54] and [15]. The table shows the number of corrupted parties that are needed to break

[15] [51], [54] consider and give proofs for the traditional non-client-server setting where at least one participant is honest.

TABLE III
COMPARISON OF MPC PROTOCOLS

	ours	[51], [54][a]	[15]	[12]	[32]
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 [54], 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 [15].

[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; $|f|$ denotes the number of multiplication resp. AND gates in a circuit f.

[d] Operations *per party* in $\mathcal{O}$-notation; $|f|$ denotes the number of addition and multiplication resp. XOR and AND gates.

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 $\text{BoBW}_{[15]}$ in detail.

Comparison of Properties: The security properties and thresholds of our protocol and $\text{BoBW}_{[15]}$ are mostly identical, except that the privacy property of $\text{BoBW}_{[15]}$ 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 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 $\text{BoBW}_{[15]}$ comes at the cost of using protocol iteration, which not only negatively impacts performance (see below) but also makes $\text{BoBW}_{[15]}$ 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 $\text{BoBW}_{[15]}$ (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 [51], [54]. Hence, in the optimal situation for $\text{BoBW}_{[15]}$ where no malicious parties cause an abort and thus the online phase of $\text{BoBW}_{[15]}$ requires just a single iteration of Cunningham et al.'s protocol, both $\text{BoBW}_{[15]}$ and our protocol have identical online complexity. However, the performance of the online phase of $\text{BoBW}_{[15]}$ progressively deteriorates with every abort. Notably, each rerun of [15] due to an abort requires first rerunning the entire (expensive) offline phase within the online phase of $\text{BoBW}_{[15]}$, which is impractical.[16] 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.

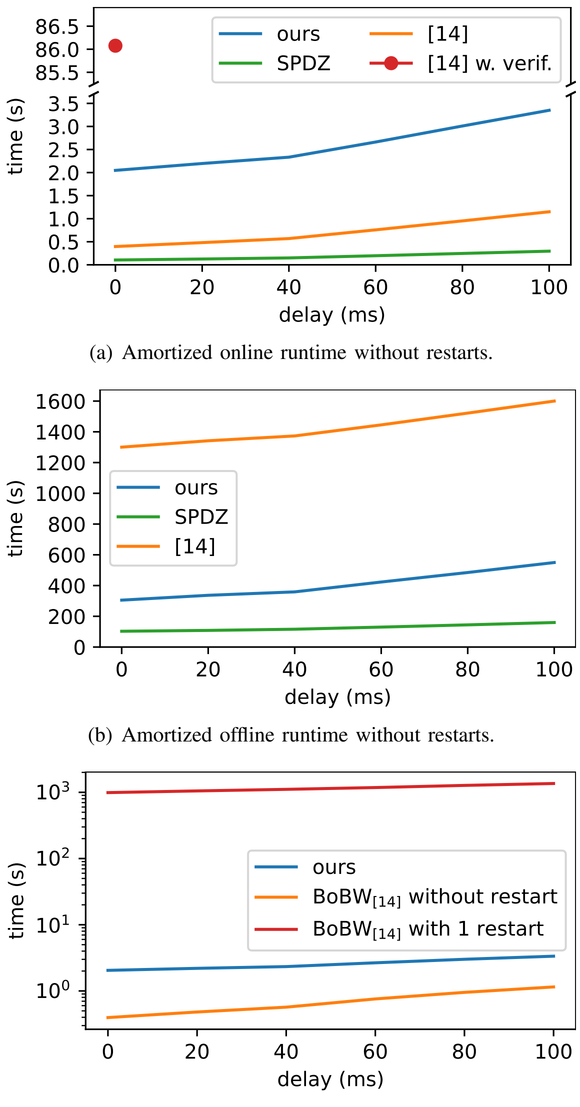
Table III provides a comparison of our protocol with three efficient two-phase protocols that offer public accountability but no robustness [12], [15], [32]. Compared to these protocols, we achieve identical or better asymptotic complexity, except for the online round complexity of Baum et al.'s protocol [32] and, as mentioned, the offline communication complexity of Cunningham et al.'s protocol [15], which are better. Note that [12], [32] do not achieve strong publicly identifiable abort (cf. Table I) and are based on primitives whose security breaks down entirely if all servers are corrupted (e.g., information theoretic signatures). So they cannot be adapted to the fully corrupted case without redesigning the entire protocols.

More generally, even when compared to highly-efficient SPDZ-like protocols without public accountability such as [7]–[10], [13], [16], [17], [20], [21], 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 in practice. We discuss the concrete performance of our protocol next.

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 [15]. The comparison with [15] serves as an approximation of the so far theoretical $\text{BoBW}_{[15]}$, discussed in Section VIII: The online phase of [15] (without performing lazy verification of commitments) is essentially the online phase of $\text{BoBW}_{[15]}$ in an ideal case, i.e., without restarts. The combination of several

[16] It might be possible to precompute the offline phase of [15] for all subsets of parties, but this introduces an additional factor of $\mathcal{O}(2^n)$ to the offline phase of $\text{BoBW}_{[15]}$.



(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” [2], [80] (118016 addition and 118272 multiplication gates, with batch size $b = N = 32768$; see Appendix J) 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 $40\,\mathrm{bit}$ and $128\,\mathrm{bit}$, respectively.

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

			ours		Π_{equiv} [a]		
$\log p$	η	SPDZ	unopt.	opt. [b]	unopt.	opt. [b]	[15] [c]
64	40	128	1040	722	3608	3020	1008
64	64	128	1124	722	3860	3128	1008
64	80	– [c]	1184	728	4100	3260	1008
64	128	– [c]	1340	728	4604	3476	1008
128	40	256	1564	1240	5092	4504	1008
128	64	256	1648	1240	5344	4600	1008
128	80	256	1696	1240	5512	4672	1008
128	128	256	1852	1240	6016	4888	1008

[a] with trapdoor for equivocation (cf. Section V-A)
[b] considering that one can avoid sending all bits of bounded randomness
[c] Using Curve25519 to get a computational security level of 128 bit restricts our choice of p. Therefore, $\log p = 252$ in all cases.
[d] not secure with statistical security η as MAC error is $2/p$

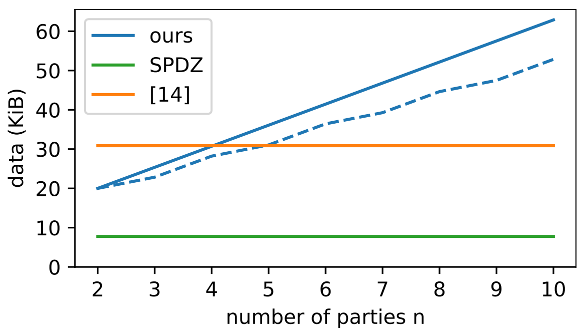


Fig. 11. Offline communication cost per party and multiplication. The solid line for our protocol is for $t = n$, the dashed line is for $t = \lfloor n/2 \rfloor + 1$.

offline and online phases of [15] (including verification for aborted online phases) corresponds to BoBW$_{[15]}$ 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 [80], [81], introduced by Mohassel and Zhang [2]) 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 Appendix K. The precise setting and resulting benchmarks for all protocols are given in Fig. 10 with more details available in Appendix J. 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 [15]. 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 Appendix J.

To summarize key findings, the overhead of the additional security properties offered by our protocol compared to basic SPDZ is a factor between 20 ($0\,\mathrm{ms}$ network delay) to 11 ($100\,\mathrm{ms}$ network delay) in the online phase. While the online phase of [15] (i.e., BoBW$_{[15]}$ without restarts) is faster than our protocol as long as no error occurs and hence no lazy verification is performed, the online phase of [15] becomes slower than ours if a misbehaving party needs to be identified (cf. red dot in Fig. 10(a)). If [15] is then restarted after such an error (i.e., BoBW$_{[15]}$), 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 [15] 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 of 17.48 MiB for our protocol, 3.61 MiB for SPDZ, and 14.21 MiB for [15]. 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 [15], 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., [8], [16], [17]).

Altogether, our protocol performs significantly better than (so-far theoretical) BoBW$_{[15]}$ 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 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 Deutsche Forschungsgemeinschaft (DFG) through grant KU 1434/11-1 and by the CRYPTECS project. The CRYPTECS project has received funding from the German Federal Ministry of Education and Research under Grant Agreement No. 16KIS1441 and from the French National Research Agency under Grant Agreement No. ANR-20-CYAL-0006.

References

[1] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, “Publicly Accountable Robust Multi-Party Computation,” in *SP 2022*. IEEE Computer Society, 2022, pp. 2430–2449.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

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

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

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

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

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

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

[17] 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.

[18] 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.

[19] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof, “Fast Large-Scale Honest-Majority MPC for Malicious Adversaries,” in *CRYPTO 2018*. Springer, 2018, pp. 34–64.

[20] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, “SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority,” in *CRYPTO 2018*. Springer, 2018, pp. 769–798.

[21] E. Orsini, N. P. Smart, and F. Vercauteren, “Overdrive2k: Efficient Secure MPC over $\mathbb{Z}_{2^k}$ from Somewhat Homomorphic Encryption,” in *CT-RSA 2020*. Springer, 2020, pp. 254–283.

[22] 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.

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

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

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

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

[35] 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.

[36] 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.

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

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

[39] 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.

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

[41] 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.

[42] 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.

[43] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)," in *STOC 1988*. ACM, 1988, pp. 1–10.

[44] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty Unconditionally Secure Protocols (Extended Abstract)," in *STOC 1988*. ACM, 1988, pp. 11–19.

[45] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty Computation from Threshold Homomorphic Encryption," in *EUROCRYPT 2001*. Springer, 2001, pp. 280–299.

[46] M. Hirt and U. M. Maurer, "Robustness for Free in Unconditional Multi-party Computation," in *CRYPTO 2001*. Springer, 2001, pp. 101–118.

[47] Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank, "On Combining Privacy with Guaranteed Output Delivery in Secure Multiparty Computation," in *CRYPTO 2006*. Springer, 2006, pp. 483–500.

[48] Z. Beerliová-Trubíniová and M. Hirt, "Efficient Multi-party Computation with Dispute Control," in *TCC 2006*. Springer, 2006, pp. 305–328.

[49] I. Damgård and J. B. Nielsen, "Scalable and Unconditionally Secure Multiparty Computation," in *CRYPTO 2007*. Springer, 2007, pp. 572–590.

[50] I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. D. Smith, "Scalable Multiparty Computation with Nearly Optimal Work and Resilience," in *CRYPTO 2008*. Springer, 2008, pp. 241–261.

[51] 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.

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

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

[54] 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.

[55] 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.

[56] 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.

[57] 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.

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

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

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

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

[62] R. Küsters, M. Tuengerthal, and D. Rausch, "The IITM Model: A Simple and Expressive Model for Universal Composability," *J. Cryptol.*, vol. 33, no. 4, pp. 1461–1584, 2020.

[63] J. Camenisch, S. Krenn, R. Küsters, and D. Rausch, "iUC: Flexible Universal Composability Made Simple," in *ASIACRYPT 2019*. Springer, 2019, pp. 191–221.

[64] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," Cryptology ePrint Archive, Tech. Rep. 2000/67, 2000.

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

[66] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally Composable Security with Global Setup," in *TCC 2007*. Springer, 2007, pp. 61–85.

[67] D. Hofheinz and V. Shoup, "GNUC: A New Universal Composability Framework," *J. Cryptology*, vol. 28, no. 3, pp. 423–508, 2015.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] X. Xie, R. Xue, and M. Wang, "Zero Knowledge Proofs from Ring-LWE," in *CANS 2013*. Springer, 2013, pp. 57–73.

[73] 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.

[74] C. Boschini, J. Camenisch, and G. Neven, "Relaxed Lattice-Based Signatures with Short Zero-Knowledge Proofs," in *ISC 2018*. Springer, 2018, pp. 3–22.

[75] 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.

[76] V. Lyubashevsky, C. Peikert, and O. Regev, "A Toolkit for Ring-LWE Cryptography," in *EUROCRYPT 2013*. Springer, 2013, pp. 35–54.

[77] 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.

[78] D. Micciancio and C. Peikert, "Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller," in *EUROCRYPT 2012*. Springer, 2012, pp. 700–718.

[79] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," in *ECCV 2016*. Springer, 2016, pp. 630–645.

[80] CSIRO Data61 Engineering & Design, "MP-SPDZ," https://github.com/data61/MP-SPDZ, 2022.

[81] M. Keller, "MP-SPDZ: A Versatile Framework for Multi-Party Computation," in *CCS 2020*. ACM, 2020, pp. 1575–1590.

[82] E. Orsini, "Efficient, Actively Secure MPC with a Dishonest Majority: A Survey," in *WAIFI 2020*. Springer, 2020, pp. 42–71.

[83] 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.

[84] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic Evaluation of the AES Circuit," in *CRYPTO 2012*. Springer, 2012, pp. 850–867.

[85] 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.

[86] T. Attema, V. Lyubashevsky, and G. Seiler, "Practical Product Proofs for Lattice Commitments," in *CRYPTO 2020*. Springer, 2020, pp. 470–499.

[87] 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.

[88] D. Unruh, "Post-quantum Security of Fiat-Shamir," in *ASIACRYPT 2017*. Springer, 2017, pp. 65–95.

[89] ——, "Post-Quantum Security of Fiat-Shamir," Cryptology ePrint Archive, Tech. Rep. 2017/398, 2017.

[90] W. Banaszczyk, "New bounds in some transference theorems in the geometry of numbers," *Math. Ann.*, vol. 296, no. 1, pp. 625–635, 1993.

[91] M. Ajtai, "Generating Hard Instances of Lattice Problems (Extended Abstract)," in *STOC 1996*. ACM, 1996, pp. 99–108.

[92] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, pp. 34:1–34:40, 2009.

[93] D. Micciancio, "Generalized Compact Knapsacks, Cyclic Lattices, and Efficient One-Way Functions from Worst-Case Complexity Assumptions," in *FOCS 2002*. IEEE Computer Society, 2002, pp. 356–365.

[94] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings," in *EUROCRYPT 2010*. Springer, 2010, pp. 1–23.

[95] 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.

[96] R. del Pino and V. Lyubashevsky, "Amortization with Fewer Equations for Proving Knowledge of Small Secrets," in *CRYPTO 2017*. Springer, 2017, pp. 365–394.

[97] Z. Beerliová-Trubíniová and M. Hirt, "Perfectly-Secure MPC with Linear Communication Complexity," in *TCC 2008*. Springer, 2008, pp. 213–230.

[98] I. Damgård and S. Zakarias, "Constant-Overhead Secure Computation of Boolean Circuits using Preprocessing," in *TCC 2013*. Springer, 2013, pp. 621–641.

[99] Federal Office for Information Security, "Cryptographic mechanisms: Recommendations and key lengths," Federal Office for Information Security (BSI), P.O.B. 20 03 63, 53133 Bonn, Germany, Technical Guideline BSI TR-02102-1, March 2020, version: 2020-01. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=6

[100] 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.

[101] M. Albrecht, "lwe-estimator," https://bitbucket.org/malb/lwe-estimator, 2020, commit fb7deba98e599df10b665eeb6a26332e43fb-5004.

[102] 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.

[103] F. Virdia and M. R. Albrecht, "estimate-all-the-lwe-ntru-schemes.github.io," https://github.com/estimate-all-the-lwe-ntru-schemes/estimate-all-the-lwe-ntru-schemes.github.io, 2018, commit 2bf02ccd8d488050c7e398ce5bc2530abff24c4f.

[104] D. Micciancio and O. Regev, *Lattice-based Cryptography*. Springer, 2009, pp. 147–191.

[105] M. Rückert and M. Schneider, "Estimating the Security of Lattice-based Cryptosystems," Cryptology ePrint Archive, Tech. Rep. 2010/137, 2010.

[106] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR 2016*. IEEE Computer Society, 2016, pp. 770–778.

[107] Open Neural Network Exchange, "ResNet," https://github.com/onnx/models/tree/master/vision/classification/resnet, 2021, commit 41cc-f18ba5a815dab714899ac234e9b1e4293c20.

[108] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Des. Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.

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. [8] has given rise a to a line of efficient MPC protocols in the dishonest majority setting (see [82] 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 [59] (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 [9].

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 [13] uses an OT-based offline phase, Overdrive [16] and TopGear [17] compute triples with the linear homomorphic BGV encryption scheme (improving on original use of BGV in SPDZ [8], [9]).

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 [58] here. We use the version of BGV without modulus switching. This is easier to present and analyze. Additionally, Keller et al. [16] show little difference when comparing parameter sizes for SPDZ without modulus switching [8] and SPDZ with modulus switching [9]. 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

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

with encryption randomness $(v, \boldsymbol{e}) \in R_q \times R_q^2$ sampled from D_{σ_v} and D_{σ_e}, respectively.[17] 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

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

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 [11], [15]). For public key cryptography, we present a robust and accountable protocol based on Asharov et al.'s work [83]—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.

To describe the protocol, we first describe the standard (non-threshold) BGV key generation in more detail. The key generation for the version of BGV we use needs to pick a small private key $s \in R_q$ and additional randomness $e \in R_q$ from discrete Gaussian distributions $D_{\sigma_s}, D_{\sigma_\epsilon}$.[18] With these (and a uniformly random $a \in R_q$), the public key $k := (a, a \cdot s + p \cdot \epsilon)$ is constructed, while s can be used to decrypt. For SHE BGV we have $\mathsf{Dec}_s((c[0], c[1], c[2])^{\mathrm{T}}) := c[0] - s \cdot c[1] - s^2 \cdot c[2] \bmod p$ – the last component can be ignored for LHE applications. We briefly describe our protocol for *distributed* key generation and *threshold* decryption. The protocol Π_{PK} itself with the respective functionality can be found in Appendix C.

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 [8], [9], [16], [17] 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 [85]. 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 [83], 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.

In the key generation protocol, we first fix a part of the public key (for simplicity, a is sampled from the CRS) and then, every party constructs their own private key s_i and public key $k_i := (a, b_i) = (a, a \cdot s_i + p \cdot \epsilon_i)$. The s_i will then be a full-threshold secret-sharing of the private key that corresponds to $k := (a, \sum_i b_i)$. Letting the parties prove in ZK that s_i, ϵ_i are short and that b_i was defined correctly, we are guaranteed to get a valid BGV key. Observe that this requires that only correctly constructed b_i are used in the definition of k. In particular, this includes proving shortness of s_i, ϵ_i. We can do this by extending recent range proof constructions [86] from $\mathbb{Z}_q$ to R_q (see Appendix F for details). This new alternative construction could be of independent interest.

By now we have no robustness yet as *all* private key shares s_i are needed for decryption. However, we can add robustness by requiring each party to additionally share its secret key s_i with Shamir secret-sharing, i.e., letting party $\mathcal{P}_i$ produce $[s_i]_j$ for all $\mathcal{P}_j$, solves this. Now, the parties can publicly reconstruct s_i of a malicious or uncooperative party,[19] or abort if that is no longer possible. For threshold decryption, the parties then have s_i as a private key share (their own share and shares for corrupted parties that are publicly reconstructed as mentioned above). A decryption share d_i is defined as $p \cdot r_i - s_i \cdot c[1]$ and

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

[18] In other works, s is picked from a slightly different sub-Gaussian distribution along the lines of [84]: Each coefficient of s is sampled randomly from $\{-1, 0, 1\}$ under the constraint that s has a Hamming weight of a fixed value, for example, 128. Also the encryption randomness (cf. Section V-B1) can be chosen differently. Setting all distributions to be the same is easier to present and analyze – also the original formulation of BGV uses the same discrete Gaussian distribution for $s, \epsilon, v, \boldsymbol{e}$ [58]. The parameters we get are not influenced by this. For the exposition, we use different standard deviations ($\sigma_s, \sigma_\epsilon$, etc.).

[19] If a party $\mathcal{P}_i$ does not provide sharings for s_i, obviously b_i should not be part of k.

```
1 Store: On input $(\texttt{store}, t, x)$ by party $\mathcal{P}_i \in \mathcal{P} \cup \mathcal{I}$:
2     if $(\mathcal{P}_i, tag, \_)$ is stored then reply $\bot$.
3     Send $(\texttt{store}, \mathcal{P}_i, x)$ to adversary $\mathcal{A}$.
4     Wait to receive $\texttt{ok}$ from $\mathcal{A}$.
5     Store $(\mathcal{P}_i, tag, x)$.
6     reply $\texttt{ok}$.
7 Load: On input $(\texttt{load}, \mathcal{P}_j, tag)$ by any party:
8     Wait until $(\mathcal{P}_j, tag, x)$ is stored (as in Line 5).
9     reply $x$.
```

Fig. 12. Bulletin board functionality $\mathcal{F}_{\text{BB}}$.

the decryption result is $x = c[0] + \sum_i d_i \bmod p$, as in, e.g., [8], [9], [16], [17]. Here $p \cdot r_i$ is the usual noise to hide s_i (and also s). Also, the proof that $b_i, [s_i]_j, d_i$ are correct boils down to efficient proofs of linear relations on commitments, while commitments for $[s_i]_j$ also make sure that publicly reconstructing missing shares is done correctly.

Until now, we only talked about key generation for LHE BGV. For SHE BGV, we need additional terms that depend on s^2 (either as described for the decryption above or for modulus switching, e.g., in [8], [58]). For the former, this would also include commitments and decommitments. A natural way to do 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. [87]. 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 [12], [15], [32], do not provide strong public accountability and/or use SHE BGV themselves. After having a key generation like this, doing decryption is simply a matter of adding additional terms to the decryption shares (or for BGV with modulus switching, nothing has to be done here). 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, were we use the programmable random oracle model.*

The proof of this theorem can be found in Appendix D-C.

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 bulletin board,
- Fig. 13 for the functionality of the offline phase, and
- Figs. 14 and 15 for protocol and functionality of the key generation and distributed decryption.

```
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 $\boldsymbol{c} := \boldsymbol{a} \cdot \boldsymbol{b}$.
        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 $\boldsymbol{c} := \boldsymbol{a} \cdot \boldsymbol{b}$.
        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 $\beta$), 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 $\mathcal{A}$ (overriding the input
    $x_i$).
    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 $(\texttt{mask}, \boldsymbol{m} := \boldsymbol{x} - \boldsymbol{r})$ to $\mathcal{A}$.
        Receive $(\mathcal{M}'', \beta'') \in \mathcal{V}$ with $\mathcal{M}'' \supseteq \mathcal{M}$
        (overriding the previous values of $\mathcal{M}$ and $\beta$).
        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. 13. Offline functionality $\mathcal{F}_{\text{offline}}$.

Appendix D
Security Proofs

The games in Figs. 16 to 18 are a vital part of the security proofs for our protocols. More precisely, we can use the fact that the advantage of an adversary (suitably defined as the absolute difference of the probabilities between winning in case of $b = 0$ and $b = 1$, i.e., the advantage compared to guessing b) is negligible for CPA-secure encryption schemes and computationally hiding commitment schemes.

Theorem 5 (Random Sum Decryption Indistinguishability)**.** *The the* real world *($b = 1$) and the* random world *($b = 0$) of Fig. 16 are indistinguishable for a CPA-secure encryption scheme.*

```
Generation (once): On input $(\texttt{gen})$ by each party $\mathcal{P}_i \in \mathcal{P}$:
    Sample part of the public key $a \xleftarrow{\$} U(R_q)$.
    Pick $s_i, \epsilon_i$, and $b_i := a \cdot s_i + p \cdot \epsilon_i$ accordingly for honest $\mathcal{P}_i \in \mathcal{H}$.
    if $|\mathcal{C}| < t$ then
        Send $a, b_i$ for honest parties $\mathcal{P}_i$ to the adversary $\mathcal{A}$ and receive $(\mathcal{M}, \beta) \in \mathcal{V}$ and $s_j, \epsilon_j$ for corrupted parties $\mathcal{P}_j \in \mathcal{C}$.
    else
        Send $a, s_i, \epsilon_i, b_i$ for honest parties $\mathcal{P}_i$ to $\mathcal{A}$ and receive $(\mathcal{M}, \beta) \in \mathcal{V}$ and $s_j, \epsilon_j$ for corrupted parties $\mathcal{P}_j \in \mathcal{C}$.
    Add $\mathcal{P}_j \in \mathcal{C}$ to $\mathcal{M}$ if $s_j$ or $\epsilon_j$ have larger norms than expected.
    Set private key $s := \sum_{\mathcal{P}_i \in \mathcal{P} \setminus \mathcal{M}} s_i$ and set $\epsilon := \sum_{\mathcal{P}_i \in \mathcal{P} \setminus \mathcal{M}} \epsilon_i$.
    if $\beta = \texttt{ok}$ then Set public key $k := (a, b) = (a, a \cdot s + p \cdot \epsilon)$ else $k := \bot$.
    reply $\mathcal{M}, k$ to $\mathcal{P}_i \in \mathcal{P}$.
Decryption: On input $(\texttt{dec}, \mathsf{Enc}(\boldsymbol{x}))$ by each party $\mathcal{P}_i \in \mathcal{P}$:
    if $\beta = \texttt{ok}$ then
        Send $\boldsymbol{x} := \mathsf{Dec}_s(\mathsf{Enc}(\boldsymbol{x}))$ to $\mathcal{A}$.
        Receive $(\mathcal{M}_{\mathsf{Enc}(\boldsymbol{x})}, \beta_{\mathsf{Enc}(\boldsymbol{x})}) \in \mathcal{V}$ with $\mathcal{M}_{\mathsf{Enc}(\boldsymbol{x})} \supseteq \mathcal{M}$ (overriding the previous values of $\mathcal{M}$ and $\beta$).
        if $\beta = \texttt{ok}$ then reply $\mathcal{M}, \boldsymbol{x}$ to $\mathcal{P}_i$.
    reply $\mathcal{M}, \bot$ to $\mathcal{P}_i$.
Audit: On input $(\texttt{audit}, \texttt{gen})$ by $\mathcal{J}$:
    Let $\mathcal{M}'$ be $\mathcal{M}$ at the end of the Generation phase.
    reply $\mathcal{M}', k$.
Audit: On input $(\texttt{audit}, \texttt{dec}, \mathsf{Enc}(\boldsymbol{x}))$ by $\mathcal{J}$:
    if $\mathcal{M}_{\mathsf{Enc}(\boldsymbol{x})}$ was sent by $\mathcal{A}$ then
        Decrypt $\boldsymbol{x} := \mathsf{Dec}_s(\mathsf{Enc}(\boldsymbol{x}))$.
        if $\beta_{\mathsf{Enc}(\boldsymbol{x})} = \texttt{ok}$ then reply $\mathcal{M}_{\mathsf{Enc}(\boldsymbol{x})}, \boldsymbol{x}$ else reply $\mathcal{M}_{\mathsf{Enc}(\boldsymbol{x})}, \bot$.
    else reply $\mathcal{M}, \bot$.
```

Fig. 14. Public key functionality $\mathcal{F}_{\text{PK}}$.

Theorem 6 (Random Sum Decommitment Indistinguishability)**.** *The the* real world *$(b = 1)$ and the* random world *$(b = 0)$ of Fig. 17 are indistinguishable for a computationally hiding commitment scheme and a zero-knowledge proof of sum.*

Theorem 7 (Random Zero-Knowledge Decommitment Indistinguishability)**.** *The the* real world *$(b = 1)$ and the* random world *$(b = 0)$ of Fig. 18 are indistinguishable for a computationally hiding commitment scheme and a zero-knowledge proof of equal plaintexts.*

Proof of Theorem 6. For this, we can transform the game for the *real world* to the *random world*. If we start with the *real world*, we can rewrite the expressions of y and z to picking z randomly and setting $y := z - x$. This is only syntactical. Next, we can replace the zero-knowledge proof by a simulated proof. This is (statistically) indistinguishable by the zero-knowledge property of the proof. In the last step, we replace y again by picking it uniformly at random and and we also replace the zero-knowledge proof simulator by the algorithm that constructs fake proofs. The latter one is no change for the zero-knowledge proofs we consider (the simulator and proof faking algorithm are the same). The other change is indistinguishable as replacing a commitment for a fixed value with one for a random value is exactly the "real-or-random" formulation of the hiding property. In total, the difference of the games is exactly the advantage for the hiding game and the difference between a real and a simulated zero-knowledge proof. Both are negligible. □

Remark 3. *We did the proof for Theorem 6 (Fig. 17) but a similar proof is possible for Theorem 5 (Fig. 16). Also, changing the games for other linear functions than a plain sum is straightforward. The distribution for y (and consequently z) does not have to be a uniform distribution as well. This is used in some security proofs in this paper.*

Similarly, indistinguishability in Theorem 7 (Fig. 18) can be proven by replacing the ZKP for $b = 1$ with its simulation. Again, the simulation of the proof and the algorithm for generating a fake proof are identical, leaving us with a game that is equivalent to the real-or-random formulation of the hiding property.

Remark 4. *In our protocols, we only continue to use the equivalent of z and x (not $\mathsf{Enc}(y), \mathsf{Com}(y), \mathsf{Enc}(z), \mathsf{Com}(z)$). This ensures that we can actually use this security game. If, for example, $\mathsf{Enc}(y)$ was decrypted later, the one could distinguish between the real and the random world. Also, strong simulation soundness is guaranteed with Fiat-Shamir (even in the post-quantum setting) [88]. This makes sure than adversaries cannot forge fake proofs, even after seeing fake proofs as in Fig. 17.*

Remark 5. *Note that in the games, the adversary has to pick their decommitment first, whereas in the protocols and functionalities, the adversary can wait until seeing data from other parties. We can switch this because of the NIZKPs are proofs of knowledge in the classical setting and simulation-sound extractable [89] in the quantum setting. This means,*

```
1  Generation (once): On input $(\mathtt{gen})$ by each $\mathcal{P}_i \in \mathcal{P}$, each $\mathcal{P}_i$ does:
2      Use $\mathcal{F}_{\text{CRS}}$ to get $a \in R_q$ and a commitment key $par'$.
3      Pick $s_i, \epsilon_i$ as in the BGV key generation; Set $b_i := a \cdot s_i + p \cdot \epsilon_i$.
4      Pick sufficiently many decryption masks $\boldsymbol{r}_i$ of (canonical) $L^\infty$-norm larger by a factor of $2^\eta/p$ than the noise of
       decryption without masks.
5      Pick shares $[s_i]_j$ for all parties $\mathcal{P}_j \in \mathcal{P}$.
6      Commit to $s_i, \epsilon_i, b_i, [s_i]_j, \boldsymbol{r}_i$ with $par'$.
7      Encrypt $[s_i]_j$ and the respective randomness under $\mathcal{P}_j$'s public key (for other parties $\mathcal{P}_j \in \mathcal{P}$).
8      Prove in ZK that
       • $s_i, \epsilon_i, \boldsymbol{r}_i$ are in the right range,
       • $b_i$ is computed from $s_i, \epsilon_i$ in the right way,
       • $[s_i]_j$ are shares for $s_i$, and
       • the encryptions are correct.
9      Send the proof, $b_i$, and its randomness to $\mathcal{F}_{\text{BB}}$ and receive data from all other parties $\mathcal{P}_j \in \mathcal{P}$.
10     Add $\mathcal{P}_j$ to a set $\mathcal{M}$ if the proof failed or their decommitment is not valid.
11     Initialize a second set $\mathcal{M}' := \emptyset$.
12     Decrypt own shares $[s_j]_i$ and the respective randomness.
13     Set $b := \sum_{\mathcal{P}_j \in \mathcal{P} \setminus \mathcal{M}} b_j$.
14     if $|\mathcal{P} \setminus \mathcal{M}| \geq t$ then reply $\mathcal{M}, (a, b)$ else reply $\mathcal{M}, \perp$ to $\mathcal{P}_i$.
15 Decryption: On input $(\mathtt{dec}, \mathsf{Enc}(\boldsymbol{x}))$ by each $\mathcal{P}_i \in \mathcal{P}$, each $\mathcal{P}_i$ does:
16     Compute $\boldsymbol{d}_i := p \cdot \boldsymbol{r}_i - s_i \cdot \mathsf{Enc}(\boldsymbol{x})[1]$ for the next unused subset of $\boldsymbol{r}_i$.
17     Commit to $\boldsymbol{d}_i$ and prove in ZK that it is correctly computed from $s_i, \boldsymbol{r}_i$.
18     Send the proof, $\boldsymbol{d}_i$, and its randomness to the bulletin board and receive data from all other parties $\mathcal{P}_j \in \mathcal{P}$.
19     Add $\mathcal{P}_j$ to a set $\mathcal{M}''$ if the proof failed or their decommitment is not valid.
20     while $\mathcal{M}'' \setminus (\mathcal{M}' \cup \mathcal{M}) \neq \emptyset$ do                    // There are new malicious parties
21         $\mathcal{M}'' \leftarrow \mathcal{M}'' \setminus (\mathcal{M}' \cup \mathcal{M})$.
22         Send $[s_j]_i$ and its randomness to $\mathcal{F}_{\text{BB}}$ for all $\mathcal{P}_j \in \mathcal{M}''$ and receive data from all other parties.
23         Add $\mathcal{M}''$ to $\mathcal{M}'$.
24         Override the set $\mathcal{M}''$ with the parties that provided invalid decommitments in Line 22 (or did not respond).
25         if $|\mathcal{P} \setminus (\mathcal{M} \cup \mathcal{M}' \cup \mathcal{M}'')| < t$ then abort.
26     foreach $\mathcal{P}_j \in \mathcal{M}'$ do
27         Reconstruct $s_j := \mathsf{Rec}([s_j]_1, \ldots, [s_j]_n)$ while ignoring shares that are not available.
28         Set default values $\boldsymbol{d}_j := -s_j \cdot \mathsf{Enc}(\boldsymbol{x})[1]$.
29     Compute $\boldsymbol{x} := \mathsf{Enc}(\boldsymbol{x})[0] + \sum_{\mathcal{P}_j \in \mathcal{P} \setminus \mathcal{M}} \boldsymbol{d}_j \bmod p$.
30     reply $\mathcal{M} \cup \mathcal{M}', \boldsymbol{x}$ to $\mathcal{P}_i$.
31 Audit: On input $(\mathtt{audit}, \mathtt{gen})$ by $\mathcal{J}$:
32     Perform the Generation phase on public data.
33 Audit: On input $(\mathtt{audit}, \mathtt{dec}, \mathsf{Enc}(\boldsymbol{x}))$ by $\mathcal{J}$:
34     Perform the Generation and Decryption phases on public data.
```

Fig. 15. Public key protocol Π_{PK}.

if the adversary's ciphertext or commitment would depend on the ones of other parties, it would break the security of the scheme (the CPA-security or the hiding property). Conversely, it does not depend on them and the order can be switched.

A. Proof of Theorem 1

Proof. We consider two separate cases for the proof: $|\mathcal{C}| < t$ and $|\mathcal{C}| \geq t$. In the first case, the simulator $\mathcal{S}$ does the following.

In the ***Prepare*** phase:

1) The commitment parameters *par* are chosen. The simulated $\mathcal{F}_{\text{CRS}}$ is programmed to output this.
2) The ***Prepare*** phase of $\mathcal{F}_{\text{offline}}$ is simulated.
3) $\mathcal{S}$ keeps track of $\langle \boldsymbol{r} \rangle_i, \langle \boldsymbol{a} \rangle_i, \langle \boldsymbol{b} \rangle_i, \langle \boldsymbol{c} \rangle_i$ for each $\mathcal{P}_i \in \mathcal{P}$.
4) Malicious parties of $\mathcal{F}_{\text{offline}}$ are forwarded to $\mathcal{F}_{\text{online}}$ (as well as $\beta = \mathtt{abort}$ if the offline phase aborted).

In the ***Input*** phase:

1) The ***Input*** phase of $\mathcal{F}_{\text{offline}}$ is simulated with dummy inputs (e.g., zero or random values) for honest input parties.
2) The inputs x_j of corrupted input parties $\mathcal{P}_j$ are revealed to $\mathcal{S}$ by the simulated instance $\mathcal{F}_{\text{offline}}$ as $\mathcal{S}$ is in full control of it.
3) The corrupted inputs are forwarded to $\mathcal{F}_{\text{online}}$.
4) $\boldsymbol{m}[i]$ is chosen uniformly at random for inputs of honest parties $\mathcal{P}_i$ and as $\boldsymbol{m}[j] := x_j - \boldsymbol{r}[j]$ for corrupted input parties $\mathcal{P}_j$.

```
$(\_, k) \xleftarrow{\$} \mathsf{KeyGen}(1^\eta)$.                // for encryption
$(x, r, state) \xleftarrow{\$} F(1^\eta, k)$.
$b \xleftarrow{\$} U(\{0,1\})$.
$y \xleftarrow{\$} U(R_p)$.
Pick valid encryption randomness $s$.
$\mathsf{Enc}(y) := \mathsf{Enc}_k(y, s)$.
if $b = 1$ then
    $z := x + y$.
else
    $z \xleftarrow{\$} U(R_p)$.
$b' \xleftarrow{\$} G(1^\eta, state, \mathsf{Enc}(y), z)$.
if $b = b'$ and $(x, r)$ is in the valid range for encryption
then
    return 1.                                  // $\mathcal{A} = (F, G)$ wins
else
    return 0.                                  // $\mathcal{A} = (F, G)$ looses
```

Fig. 16. Random sum decryption.

```
$par \xleftarrow{\$} \mathsf{KeyGen}(1^\eta)$.                    // for committing
$(x, r, state) \xleftarrow{\$} F(1^\eta, par)$.
$b \xleftarrow{\$} U(\{0,1\})$.
$y \xleftarrow{\$} U(R_p)$.
Pick valid commitment randomness $s, t$.
$\mathsf{Com}(y) := \mathsf{Com}_{par}(y, s)$.
if $b = 1$ then
    $z := x + y$.
    $\mathsf{Com}(z) := \mathsf{Com}_{par}(z, t)$.
    Generate a commitment proof $zk$ for $z = x + y$.
else
    $z \xleftarrow{\$} U(R_p)$.
    $\mathsf{Com}(z) := \mathsf{Com}_{par}(z, t)$.
    Generate a fake commitment proof $zk$ for $z = x + y$.
$b' \xleftarrow{\$} G(1^\eta, state, zk, \mathsf{Com}(y), \mathsf{Com}(z), z, t)$.
if $b = b'$ and $(x, r)$ is in the valid range for committing
then
    return 1.                                  // $\mathcal{A} = (F, G)$ wins
else
    return 0.                                  // $\mathcal{A} = (F, G)$ looses
```

Fig. 17. Random sum decommitment.

```
$par \xleftarrow{\$} \mathsf{KeyGen}(1^\eta)$.                    // for committing
$(x, state) \xleftarrow{\$} F(1^\eta, par)$.
$b \xleftarrow{\$} U(\{0,1\})$.
Pick valid commitment randomness $r, r'$.
$c' := \mathsf{Com}_{par}(x, r')$.
if $b = 1$ then
    $c := \mathsf{Com}_{par}(x, r)$.
    Generate a commitment proof $zk$ for $c = c'$
    (w.r.t. plaintexts).
else
    $z \xleftarrow{\$} U(R_p)$.
    $c := \mathsf{Com}_{par}(z, r)$.
    Generate a fake commitment proof $zk$ for $c = c'$
    (w.r.t. plaintexts).
$b' \xleftarrow{\$} G(1^\eta, state, zk, c, c', r')$.
if $b = b'$ then
    return 1.                                  // $\mathcal{A} = (F, G)$ wins
else
    return 0.                                  // $\mathcal{A} = (F, G)$ looses
```

Fig. 18. Random zero-knowledge decommitment.

5) Malicious parties of the ***Input*** phase of $\mathcal{F}_{\text{offline}}$ are forwarded to $\mathcal{F}_{\text{online}}$.

In the ***Compute*** phase:

1) $\mathcal{F}_{\text{online}}$ sends the result $\boldsymbol{y}$ of the computation to $\mathcal{S}$.
2) The circuit evaluation is simulated with the available data except for output-gates. Recall that these correspond to outputs of the protocol. For the circuit evaluation, the "honest" views of corrupted parties (what they would compute if they were honest) are computed as well.
3) At an gate $(\texttt{output}, l, \text{id}_x)$, we have access to $\langle x \rangle_j$ for all corrupted parties $\mathcal{P}_j$ as if they were honest. Pick the shares $[x]_i$ of honest $\mathcal{P}_i$ in a way that the shares reconstruct to $\boldsymbol{y}[l]$. Commit to the shares of honest parties. Fake the ZKP for honest parties to prove that the newly generated commitment for these shares corresponds to the commitments of honest output shares (computed in the simulation with the wrong inputs, available to all parties). Use these shares and the new commitment randomness for honest parties in the simulation of the final Open-call.
4) Collect all parties that revealed invalid commitments or NIZKPs in a set $\mathcal{M}$ and send it to $\mathcal{F}_{\text{online}}$.

Aborts and the ***Audit*** phase are handled accordingly (the value β sent to $\mathcal{F}_{\text{online}}$ is abort is the protocol aborts in the simulation and ok otherwise):

1) If an honest party $\mathcal{P}_i$ aborts during ***Prepare***, the ***Input*** and ***Compute*** phases are skipped and in the ***Audit*** phase we get the output $\perp$ and the malicious parties of ***Prepare*** are returned.
2) If there is an abort in the ***Input*** phase, the ***Compute*** phase is skipped and ***Audit*** returns $\perp$ and the malicious parties of the ***Prepare*** and ***Input*** phases.
3) If there is an abort in the ***Compute*** phase, ***Audit*** returns $\perp$ and all malicious parties up to the abort.
4) If no abort happened, the result of the ***Compute*** and malicious parties of all phases are returned in ***Audit***.

Note that honest parties are never added to a set of malicious parties as they do not misbehave in $\mathcal{F}_{\text{online}}$ (except for faking ZKPs which cannot be detected by Theorem 7; more on that later) and can always reveal the right decommitments (as long as they did not abort; then they would also not be required to give these). On the other hand, misbehaving parties are always detected with overwhelming probability. This is by-design for $\mathcal{F}_{\text{offline}}$ and follows from the binding property of the commitment scheme, similarly to [11], [15]. Additionally, we use the soundness of the used NIZKP.

The simulation and a real protocol execution are indistinguishable (along the lines of [11], [15]) by taking into account that $\mathcal{F}_{\text{offline}}$ produces shares for a value $\boldsymbol{r}$ that is uniformly distributed and unknown to the adversary. This means the value $\boldsymbol{m}$ provided by $\mathcal{S}$ is distributed exactly as the one in a real protocol execution. Additionally, no information of the shares $[\boldsymbol{x}]_i$ or shared values $\boldsymbol{x}$ can be gained from the commitments that are part of $\langle \boldsymbol{x} \rangle_j$ (because of the hiding property of the commitment scheme) or the opened auxillary values during multiplication (as the triples are random and unknown to the adversary by $\mathcal{F}_{\text{offline}}$'s design). For the latter, the plaintexts in the decommitments are distributed uniformly at random (the shares reconstruct to a uniformly random value) and the randomness is distributed the same in the protocol and the simulation (as only different plaintext-related values are used for the inputs). Only outputs of the protocol might reveal that not the right inputs were used for honest parties, but $\mathcal{S}$ can construct ZKPs and final decommitments that are consistent with commitments and the circuit evaluation on real inputs. By Theorem 7, an adversary cannot distinguish that one of the commitments in the ZKP has a different plaintext than the other (otherwise, they could win the security game which is equivalent to breaking the hiding property of the commitment scheme).

Simulating the high corruption ($t \leq |\mathcal{C}| < n$) or full corruption ($|\mathcal{C}| = n$) case is rather trivial. The functionality gives the simulator the necessary information for honest input parties, so the protocol can be fully simulated. Additionally, the simulator can extract data for corrupted parties to give to $\mathcal{F}_{\text{online}}$ just as before. Malicious parties can still be detected in the simulation (even with full corruption) and this can be forwarded to $\mathcal{F}_{\text{online}}$ at the required points in the protocol execution. □

Remark 6. *For the version of the protocol with computationally binding and statistically hiding commitments (Π_{equiv}), the security proofs also works in the lines of [11], [15] but different properties of the lattice-based commitments have to be considered. The added difficulty is that the distribution of the noise in the decommitments might reveal that a trapdoor was used.*

To circumvent this, the simulator acts as follows. It considers the linearized arithmetic circuit (by replacing all multiplication gates by the linear operations and openings corresponding to the use of Beaver triples). In this circuit, the outputs are results of linear functions in the inputs and Beaver triples. Let $y = f(x_1, \ldots, x_l)$ be one such function. The simulator can solve this for the x_j (a solution exists as an honest protocol instance would have computed y from such x_j). Now, it can use the "honest" shares of corrupted parties (shares that are computed as if the party was honest) to find shares for honest parties $[x_j]_i$ that would reconstruct to the x_j. It can also use the trapdoor to get randomness for the corresponding share commitments. Computing the decommitments for the honest parties is now simply a matter of evaluating the linear function on these intermediate decommitments. The distribution is now as required because the intermediate values can be decommitted with the trapdoor (by construction in an indistinguishable way) and the final value is now computed as a linear function as in the protocol. Finally, the commitment key with trapdoor and a real commitment key are indistinguishable as in [77] (see also Appendix E).

Remark 7. *Conversely to [11], [15], we can directly base the fact that the adversary cannot decommit to wrong shares on the M-SIS problem (with our used commitment scheme; see Appendix E). This allows us to further reduce the commitment parameters and is necessary for proving security of the final ZKPs of the online phase and the corresponding decommitments. By chosing the bounds for valid decommitments minimally small, i.e., to the maximum bounds for honest parties, we can do a proof similar to the one of Theorem 8. As we deal with exact decommitments, we set the approximation factors to one and consider two types of decommitments. Firstly, the decommitment the adversary "knows" (extracted by the simulator and proven with an aggregated ZKP) and, secondly, the decommitment revealed in the protocol. Let the first be (x, r) with $\|r\|_\infty \leq h \cdot B_{agg}$ and the second one be (x', r') with $\|r'\|_\infty \leq h \cdot B_r$.*[20] *The resulting bound relevant for the M-SIS problem is then $h \cdot (B_{agg} + B_r)$ instead of $2 \cdot h \cdot B_{agg}$ that we would get for the binding property. Note that $B_r = 1$ can be used (as we do) and B_{agg} is exponentially larger than B_r. Also, note that we use the L^∞-norm here instead of the L^2-norm used in Theorem 8.*

Similarly, to prove that an adversary cannot cheat in the final zero-knowledge proof of commitments with equal plaintexts, we again reduce this to the M-SIS problem. Let the NIZKP relation be for knowing approximate decommitments (x, r, e) and (x, r', e) for commitments c, c' with $\mathsf{Com}_{par}(e \cdot x, r) = e \cdot c$ and $\mathsf{Com}_{par}(e \cdot x, r') = e \cdot c'$ with $\|r\|_\infty \leq B_{eq}$, $\|r'\|_\infty \leq B'_{eq}$, and $e \in \bar{C}$. Also, let B'_{agg} be a bound for the L^2-norm *of the above mentioned aggregated proofs. Then, cheating would break M-SIS with bound $2\sqrt{\kappa} \cdot h \cdot B'_{agg} + B_{eq}$.*

The final M-SIS instance of relevance is used to prove that the decommitments after the above ZKP are not broken. For this, M-SIS with bounds $B'_{eq} + 2\kappa \cdot B_r$ has to be a hard problem.

B. Proof of Theorem 2

Here, we present the proof of Theorem 2 for Π_{offline} realizing $\mathcal{F}_{\text{offline}}$.

Proof. Again (as for Theorem 1), we construct a simulator $\mathcal{S}$ in cases dependent on the number of corrupted parties. If $|\mathcal{C}| < t$, we define $\mathcal{S}$ to do the following.

In the ***Prepare*** phase:

1) Receive $(\texttt{prep}, par)$ from $\mathcal{F}_{\text{offline}}$.
2) Simulate key generation in the ***Prepare*** phase, while also keeping track of the private key for decryption. Let the simulated $\mathcal{F}_{\text{PK}}$ output this key.
3) Generate a commitment key par' with an extractable trapdoor and let the simulated $\mathcal{F}_{\text{CRS}}$ return this key.
4) Send the set $\mathcal{M}$ that is returned by the simulated $\mathcal{F}_{\text{PK}}$ to $\mathcal{F}_{\text{offline}}$.
5) Receive shares $[\boldsymbol{r}]_j, [\boldsymbol{a}]_j, [\boldsymbol{b}]_j, [\boldsymbol{c}]_j$ for corrupted parties $\mathcal{P}_j$ and $\mathsf{Com}([\boldsymbol{r}]_i), \mathsf{Com}([\boldsymbol{a}]_i), \mathsf{Com}([\boldsymbol{b}]_i), \mathsf{Com}([\boldsymbol{c}]_i)$ for honest $\mathcal{P}_i$ from $\mathcal{F}_{\text{offline}}$.
6) Pick $\boldsymbol{m}$ uniformly at random. This will be used instead of the decryption of $\mathsf{Enc}(\boldsymbol{c}) - \mathsf{Enc}(\boldsymbol{d})$.
7) For honest $\mathcal{P}_i$ and $\boldsymbol{v} \in \{\boldsymbol{r}, \boldsymbol{a}, \boldsymbol{b}\}$, pick $\boldsymbol{m}_i$ uniformly at random. Set $\mathsf{Com}(\boldsymbol{y}_i) := \mathsf{Com}([\boldsymbol{v}]_i) + \boldsymbol{m}_i$. For the mask $\boldsymbol{y}_i$ that is used for $\boldsymbol{d}$, set $\mathsf{Com}(\boldsymbol{y}_i) := \mathsf{Com}([\boldsymbol{c}]_i) + \boldsymbol{m}_i -$

[20] h is a factor depending on the number of linear homomorphic operations.

$\boldsymbol{m}$. This makes sure that $\mathsf{Com}([v]_i) = \mathsf{Com}(\boldsymbol{y}_i) - \boldsymbol{m}_i$ and $\mathsf{Com}([\boldsymbol{c}]_i) = \mathsf{Com}([\boldsymbol{d}]_i) + \boldsymbol{m} = \mathsf{Com}(\boldsymbol{y}_i) - \boldsymbol{m}_i + \boldsymbol{m}$.

8) Simulate the preprocessing further until all masks $\boldsymbol{y}_j$ have published proofs. Use the trapdoor for *par′* to get the shares and commitment randomness (for the masks $\boldsymbol{y}_j$) for corrupted $\mathcal{P}_j$. Corrupted parties with invalid proofs can be simply ignored in the following steps as we do not compute anything for them in the protocol (except setting their commitments to $\top$). For honest parties $\mathcal{P}_i$, use the ZK simulator to produce accepted proofs for the commitments $\mathsf{Com}(\boldsymbol{y}_i)$. For the latter, also pick ciphertexts $\mathsf{Enc}(\boldsymbol{y}_i)$ for uniformly random plaintexts. These are also used in the simulated proof of committing ciphertexts (for a wrong statement as the simulator does not know the decommitment of $\mathsf{Com}(\boldsymbol{y}_i)$ and thus is unlikely to have a matching plaintext for $\mathsf{Enc}(\boldsymbol{y}_i)$).
9) For corrupted $\mathcal{P}_j$ and $\boldsymbol{v} \in \{\boldsymbol{r}, \boldsymbol{a}, \boldsymbol{b}\}$, set $\boldsymbol{m}_j := \boldsymbol{y}_j - [\boldsymbol{v}]_j$. For the mask $\boldsymbol{y}_i$ that is used for $\boldsymbol{d}$, set $\boldsymbol{m}_j := \boldsymbol{y}_j - [\boldsymbol{v}]_j + \boldsymbol{m}$. Compute the decommitments for the shares of corrupted parties from these values and the extracted decommitments of $\mathsf{Com}(\boldsymbol{y}_j)$.
10) Continue the simulation and use $\boldsymbol{m}_i, \boldsymbol{m}_j, \boldsymbol{m}$ for the respective decryptions. With this, the simulator sets the shares in the simulation to the share that it got from the functionality.
11) If SHE is not used, pick $[\boldsymbol{a}]_i$ for honest parties $\mathcal{P}_i$ randomly and compute the rest of the protocol in Fig. 7 normally. Use the ZK simulator to produce accepted proofs. Otherwise, multiply the ciphertexts with SHE.
12) Continue with the last step and do the decryption of $\mathsf{Enc}(\boldsymbol{c}) - \mathsf{Enc}(\boldsymbol{d})$ as mentioned above (use $\boldsymbol{m}$).
13) Collect all parties that failed to produce some ZK proof or were revealed by $\mathcal{F}_{\text{PK}}$ in $\mathcal{M}$ and send this set of parties to $\mathcal{F}_{\text{offline}}$ (together with the commitments for corrupted parties and their extracted decommitments).

In the ***Input*** phase:

1) Simulate the ***Input*** by picking dummy values for honest input parties.
2) Extract the inputs of corrupted input parties with the private key of the simulated functionality $\mathcal{F}_{\text{PK}}$.
3) If corrupted input parties fail to prove that their input is well-formed, override their (extracted) inputs with 0.
4) Send the inputs of corrupted parties to $\mathcal{F}_{\text{offline}}$.
5) Receive $\boldsymbol{m}$ from $\mathcal{F}_{\text{offline}}$ and use it as output of the simulated $\mathcal{F}_{\text{PK}}$ (for the decryption of $\mathsf{Enc}(\boldsymbol{x}) - \mathsf{Enc}(\boldsymbol{r})$). Note, this is a different value than the ones used in the ***Prepare*** phase (also called "$\boldsymbol{m}$").
6) Forward the malicious parties of this decryption to $\mathcal{F}_{\text{offline}}$.

Again, aborts during the protocol are handled accordingly. Once an honest party aborts in the simulation, the set of malicious parties is handed over to the functionality, which triggers the same abort-behavior (i.e., if the ***Input*** phase will be skipped if it did not already happen and $\perp$ is returned instead of computed values).

Proving indistinguishability for the offline phase will be more elaborate than for the online phase. Firstly, the key generation (using $\mathcal{F}_{\text{PK}}$ and $\mathcal{F}_{\text{CRS}}$) can be perfectly simulated while giving no indication that the simulator knows the respective trapdoors (private key of the encryption and commitment trapdoor). Next, no information can be obtained from the commitments and ciphertexts before any decryption takes place. This is due to the zero-knowledge property of the proofs and the security of the schemes. Additionally, the fact that the NIZKPs are proofs of knowledge guarantees that ciphertexts and commitments for corrupted parties do not depend on the ones of honest parties. Furthermore, the corrupted parties cannot prove wrong statements (even if there were simulated proofs of wrong statements) due to simulation soundness. Thus, the combined ciphertexts $\mathsf{Enc}(\mathbf{W})$ are distributed uniformly at random. This makes the values $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$ in the simulation indistinguishable from the real decryptions (using Theorem 5; if some could distinguish these cases, one would win in the security game and thus break CPA-security).

The faked proofs of multiplication cannot be detected by an adversary (using Theorem 6; similarly to using Theorem 5 above) as re-encryptions of $[\boldsymbol{a}]_i \cdot \mathsf{Enc}(\boldsymbol{b})$ and $\mathsf{Enc}(\boldsymbol{u})$ for arbitrary $\boldsymbol{u}$ are indistinguishable (here we use an adapted version of Theorem 6 that uses *multiplication and re-encryption* instead of a plain sum and *re-encryption indistinguishability* (implied by by CPA-security) instead of indistinguishability of uniformly random values). Again, simulation soundness makes sure that the adversary does not prove wrong statements (even in the presence of fake proofs).

The final decryption of the ***Prepare*** phase gives no indication that the ciphertexts in the simulation are inconsistent with the plaintexts and commitments (again, using Theorem 5). The same can be said for the decryption in the ***Input*** phase, too. Here, we also use the fact that the ciphertexts of dummy values (for honest parties in the simulation) give no information about the contained plaintexts and the decryption result is distributed uniformly at random.

In the high (or full) corruption case, we can again argue that the simulator lets the adversary control the simulation fully (except for the functionalities which still have some constraints). The values expected by the functionality $\mathcal{F}_{\text{offline}}$ are then either known (for honest parties) or can be extracted (for corrupted parties). Finally, we note that the adversary cannot break the commitment scheme or the guarantees of the ZK proofs, even if all parties are corrupted. In the full corruption case, we have to make sure that the simulator only identifies malicious parties until an honest party (or an auditor) would abort and send this set $\mathcal{M}$ to $\mathcal{F}_{\text{offline}}$ as this is the behavior of auditors in the real protocol. □

In above proof, we used Theorem 5 and Theorem 6 in the following way. Firstly, Theorem 5 was used for every decryption. This has several reasons.

- We picked ciphertexts for honest parties but the ciphertexts for shares $\mathsf{Enc}([v]_i)$ in the protocol do not correspond to the shares picked by the functionality (as these are not all known). Luckily, the shares of the corrupted parties are distributed uniformly at random.[21] This allows us to use the theorem, as also in the real or simulated protocol execution some parts of the sum (the share can be interpreted as a sum of values) are picked uniformly at random – the parts of honest parties. There are enough distinct terms in the sum picked by honest parties to play the role of "y" in the theorem.

- For honest parties, we can use a similar reasoning but this time, the maskings of the honest parties play the role of "y" in the theorem.
- The decryption of $\mathsf{Enc}(\boldsymbol{c}) - \mathsf{Enc}(\boldsymbol{d})$ can also be replaced (without giving a hint that not the actual $\mathsf{Enc}(\boldsymbol{c})$ is decrypted) as $\boldsymbol{d}$ is a ciphertext for a still unknown random value.
- The same can be said for the final decryption in the ***Input*** phase as $\boldsymbol{r}$ is unknown and random, and not the real inputs are used for $\mathsf{Enc}(\boldsymbol{x})$ (but $\boldsymbol{x}$ is part of the value $\boldsymbol{m}$ that we get from the functionality).

Theorem 6 is only used if we do ciphertext multiplication without SHE. For faking that we do not know $\mathsf{Enc}(\boldsymbol{y}_i)$ for honest $\mathcal{P}_i$, we can use the simulator to construct zero-knowledge proofs for wrong statements. In the multiplication, we have to compute $[a]_i \cdot \mathsf{Enc}(b) + \mathsf{Enc}_k(0, (v, \boldsymbol{e})^{\mathrm{T}})$. As we use commitments to prove that the values are correctly multiplied and summed up, we can fake the proof. The decommitment (of the multiplication and re-encryption) does not reveal information about the fact that we did not use the real share as re-encryptions of two values (the one we actually decommitted to and the expected one) are indistinguishable by CPA-security. Finally, we do not care that the resulting ciphertext (after applying Rec) is not for $a \cdot b$ as the above mentioned Theorem 5 is used and we do not give the real decryption of the ciphertexts computed in Fig. 7.

Note that using r and d in decryptions might seem like a contradiction to Remark 4 where we state that we do not use the "random" part (i.e., y in Fig. 16) in the rest of our protocol. To see that this is not the case, recall that we are looking at the case of less than t corruptions in the security proof when we use Theorem 5. There are then also less than t shares for corrupted parties where we use the theorem for (for honest parties, there will always be a fresh random masking, so they do not have to be considered further). Each share uses exactly t coefficients, each containing a summand of at least one honest parties. This means that we can "use up" one of the non-constant coefficients[22] for the shares and the constant coefficient (either r or d) is still random and unknown and can be used for another application of Theorem 5.

C. *Proof of Theorem 4*

Here, we present the proof of Theorem 4 for Π_{PK} realizing $\mathcal{F}_{\text{PK}}$.

Proof. As in our other proofs, we distinguish between high and low corruption for our simulator $\mathcal{S}$. For $|\mathcal{C}| < t$, we have the following.

In the ***Generation*** phase:

1) Use a from $\mathcal{F}_{\text{PK}}$ as part of the CRS and also program a commitment key *par*$'$ for which $\mathcal{S}$ knows an extraction trapdoor.
2) Let each honest party pick their s_i, ϵ_i as in the protocol. Pick random values $[s_i]_j$ from honest parties $\mathcal{P}_i$ for corrupted parties $\mathcal{P}_j$. Also pick the decryption masks. Create a commitment for b_i from $\mathcal{F}_{\text{PK}}$. Fake the proof of sum to decommit to the b_i from the functionality.
3) Extract the values for corrupted parties with the trapdoor.
4) Send the required values to the functionality. Note that the public key is then the same in the simulation and the functionality. Additionally, the privatekey s (in the functionality) is the sum of the s_j by corrupted parties (known to the simulator) and the s_i by honest parties (unknown to the simulator). Let s' be the private key in the simulation (derived from the corrupted s_j and the random values picked for simulated honest parties). The simulator knows shares of s' for all parties, as well as shares for the decryption masks $\boldsymbol{r}$.

In the ***Decryption*** phase:

1) Get the decryption $\boldsymbol{x}$ from $\mathcal{F}_{\text{PK}}$.
2) Given a ciphertext $c = (c[0], c[1])$, the simulator can compute the values $\boldsymbol{y}_j := \boldsymbol{r}_j - s_j \cdot c[1]$ for corrupted parties $\mathcal{P}_j$.
3) Set $\boldsymbol{y}_i$ for honest $\mathcal{P}_i$ to satisfy $x - c[0] = \sum_{\mathcal{P}_j \in \mathcal{C}} \boldsymbol{y}_j + \sum_{\mathcal{P}_i \in \mathcal{H}} \boldsymbol{y}_i - p \cdot \boldsymbol{r}$. Here, each $\boldsymbol{y}_i$ is a randomly chosen such that its norm is a factor of 2^η larger than the standard decryption noise. $\boldsymbol{r}$ can then be arbitrary but is smaller than n times this bound. Note that the shares $\boldsymbol{y}_i$ are picked to be consistent with the shares corrupted parties – even if the values $\boldsymbol{r}_j$ is replaced with the default value of zero. For honest parties, the zero-knowledge proofs can be faked in a way that these $\boldsymbol{y}_i$ are accepted without knowing the shares s_i picked by the functionality.

The simulation and real protocol differ for s_i and y_i for honest $\mathcal{P}_i$. For the first, we can use Theorem 6 to show that it is unnoticeable that the honest b_i are not actually computed from the s_i in the simulation. Here, we use that b_i is indistinguishable from random by the hardness of the R-LWE problem, i.e., it is also indistinguishable from another R-LWE sample.

The decryption obviously gives the right result (modulo p) but we also have to argue indistinguishability. Again, the adversary cannot see that the simulator gave a value that is not the sum as in the protocol. For this, we use Theorem 6 while noticing that the noise bound in the decryption is indistinguishable in the simulation and the protocol. In the protocol, this bound B is determined by $\boldsymbol{r}_i$ *and* the noise bound without it, i.e., the noise bound of the ciphertext to be decrypted, B_c. In other words, $B = (2^\eta + 1) \cdot B_c$. In the simulation, the bound on $\boldsymbol{y}_i$ is exactly the bound on $\boldsymbol{r}_i$ in the protocol, i.e., $B = 2^\eta \cdot B_c$. As the distributions are statistically indistinguishable by the choice of $\boldsymbol{r}$ (the bound

[21] Instead of picking coefficients of a polynomial, the shares for corrupted parties could have been picked uniformly at random and other shares are constructed by interpolating the polynomial (or one random polynomial if it is not uniquely defined by the shares) that intersects these shares.

[22] i.e., one that is multiplied with j^l with $l \neq 0$ for party $\mathcal{P}_j$

on $\boldsymbol{r}$ is exponentially larger than B_c), this cannot be detected when we apply Theorem 6 again.

Finally, one can rather trivially show again that the malicious parties are always detected and aborts are handled correctly. Also, for high and full corruption the proof can be done without any large difficulties. □

Remark 8. *Note that we only use $\mathcal{F}_{\text{PKI}}$ (for a public key infrastructure) to provide public keys for all parties to everyone. We do not need to extract anything from ciphertexts that use these keys as the values we are interested in are also extractable with the commitment scheme. Additionally, we know that the encryptions are consistent with the commitments because of the used zero-knowledge proofs.*

Remark 9. *If we consider the SHE BGV key generation we get the required security (in particular, accountability and robustness) from the properties of Π_{online} that we use in this case to generate the necessary keys (and commitments needed for accountable BGV decryption with these SHE keys).*

Appendix E
Lattice-Based Commitments (Continued)

Let us start with giving the definition of the discrete Gaussian distribution. It is defined by

$$D_\sigma(S) : \boldsymbol{x} \mapsto \frac{g_\sigma(\boldsymbol{x})}{\sum_{\boldsymbol{y} \in S} g_\sigma(\boldsymbol{y})}$$

with $g_\sigma : \boldsymbol{x} \mapsto \exp(-\|\boldsymbol{x}\|_2^2/2\sigma^2)$ and standard deviation σ. With the fact that

$$\Pr\left[\|\boldsymbol{x}\|_2 > \sigma \cdot \sqrt{2 \cdot d} \;\middle|\; \boldsymbol{x} \xleftarrow{\$} D_\sigma(\mathbb{Z}^d)\right] \tag{6}$$

is negligible in d (e.g., from [61], [90]) and by identifying $\mathbb{Z}^N$ with R, we get a bound B_σ for sampling elements of R (and R_p) with Gaussians (the right-hand-side of the inequality in (6)). This will be used later to define relevant bounds for the commitment scheme. We now proceed with details the problems that we base the security of commitments on and on our generalization of Baum et al.'s [57] BDLOP commitment scheme.

A. Lattice Problems

Many recent cryptographic primitives are based on the hardness of lattice problems. Most notably, many primitives are based on the Learning With Errors (LWE) and Short Integer Solution (SIS) problems which are at least as hard to solve as certain worst-case lattice problems [91], [92]. Ring [93], [94] and module [58], [75] variants of LWE and SIS (R-LWE, R-SIS, M-LWE, and M-SIS, respectively) were proposed and analyzed – enabling more efficient cryptographic primitives.

We present problems equivalent to M-LWE and M-SIS from Baum et al. [57]. The security of the commitment scheme is based on these.

Definition 3 (Module-Learning With Errors)**.** *The advantage of an adversary $\mathcal{A}$ for (decision) $\mathsf{M\text{-}LWE}(R_p, n, m, dist)$ is defined as*

$$\left|\Pr\left[b = 1 \;\middle|\; \begin{array}{l} \mathbf{A} \xleftarrow{\$} U(R_p^{n\times m}); \boldsymbol{y} \xleftarrow{\$} dist(R_p^{n+m}); \\ \boldsymbol{u} \leftarrow \begin{pmatrix}\mathbf{I}_n & \mathbf{A}\end{pmatrix} \cdot \boldsymbol{y}; b \xleftarrow{\$} \mathcal{A}(\mathbf{A}, \boldsymbol{u}) \end{array}\right] - \right.$$
$$\left. \Pr\left[b = 1 \;\middle|\; \mathbf{A} \xleftarrow{\$} U(R_p^{n\times m}); \boldsymbol{u} \xleftarrow{\$} U(R_p^n); b \xleftarrow{\$} \mathcal{A}(\mathbf{A}, \boldsymbol{u})\right]\right|,$$

i.e., $\mathcal{A}$ has to distinguish a random $\boldsymbol{u}$ from $\begin{pmatrix}\mathbf{I}_n & \mathbf{A}\end{pmatrix} \cdot \boldsymbol{y}$ where $\boldsymbol{y}$ is picked according to the distribution dist.

Definition 4 (Module-Short Integer Solution)**.** *The advantage of an adversary $\mathcal{A}$ for (search) $\mathsf{M\text{-}SIS}(R_p, n, m, norm, B)$ is defined as*

$$\Pr\left[\begin{array}{c} \begin{pmatrix}\mathbf{I}_n & \mathbf{A}\end{pmatrix} \cdot \boldsymbol{y} = \mathbf{0}; \\ \bigwedge_{i=0}^{n+m-1} norm(\boldsymbol{y}[i]) \leq B \end{array} \;\middle|\; \begin{array}{l} \mathbf{A} \xleftarrow{\$} U(R_p^{n\times m}); \\ \boldsymbol{y} \xleftarrow{\$} \mathcal{A}(\mathbf{A}); \\ \boldsymbol{y} \neq \mathbf{0} \end{array}\right],$$

i.e., $\mathcal{A}$ has to find a norm-short non-zero vector $\boldsymbol{y} \in R_p^{n+m}$ that is a root of the linear function $f : \boldsymbol{x} \mapsto \begin{pmatrix}\mathbf{I}_n & \mathbf{A}\end{pmatrix} \cdot \boldsymbol{x}$.

By now there is a wide variety of lattice-based commitment schemes, for example the already mentioned paper by Baum et al. [57], the schemes of [71]–[74], or lattice-based encryption schemes that could be used as commitments (as done by Spini and Fehr [14] with BGV). However, finding a suitable commitment scheme is again not as simple as using just any arbitrary lattice-based commitment scheme. Some schemes cannot be used since they are perfectly binding (and thus not suitable for a variant of our protocol with equivocation; this includes lattice-based encryption schemes), only support a small number of homomorphic operations, and/or have constraints on the plaintexts (e.g., supporting only short messages) [74]. Another point is that efficient zero-knowledge proofs for commitments are indispensable for an efficient offline phase, ruling out [71], [72]. Finally, there is a need for two different types of security for the online and offline phases (in particular, computationally secure in the online phase and perfectly/statistically binding in the offline phase; or even perfectly/statistically hiding *and* perfectly/statistically binding, respectively, if we would use equivocation in the online phase), as we already described. Overall best suited seems to be the scheme of Baum et al. [57] which we described in Section VI. Now, we proceed to give more details on our generalization, mentioned there as well.

B. Details on Our Generalization of BDLOP

Recall the BDLOP commitment scheme of Baum et al. [57] and our generalization from Section VI. One detail we left out was approximate decommitments. Such a decommitment would use the adapted verification $\mathsf{Verify}_{par}(x, r, e, c)$ that checks $\mathsf{Com}_{par}(e \cdot x, r) = e \cdot c$ (and also the already mentioned range checks on r). The element e can be from $\bar{C} := \{a - b \mid a, b \in C; a \neq b\}$ with $C := \{a \in R \mid \|a\|_\infty = 1; \|a\|_1 = \kappa\}$ and is used to get a relaxed decommitment with more efficient zero-knowledge proofs. κ is chosen so $|C| \geq 2^\eta$ for a statistical security of η and one has to take care that elements of $\bar{C}$ are invertible in R_p (by setting N and p accordingly [57]). An honest party can always use $e = 1$ with x and r (used to create a commitment) in a decommitment.

Our adaption of the BDLOP commitment scheme to support multiple moduli is straightforward. As we consider a larger modulus p' for the parts where del Pino et al. [70] use a smaller one, we cannot reuse their proofs. Nevertheless, we can simply redo the security proofs of Baum et al. with two moduli and use the fact that p' is a multiple of p, i.e., a uniform distribution w.r.t. p' is also uniformly random for p. We get that the binding and hiding properties depend on the hardness of M-SIS and M-LWE in the larger modulus p' – the $\mathsf{M\text{-}SIS}(R_{p'}, d_1, d_2 + d_1 + 1, \|\cdot\|_2, 4 \cdot \sqrt{N \cdot \kappa} \cdot B_\varsigma)$ and $\mathsf{M\text{-}LWE}(R_{p'}, 1 + d_1, d_2 + d_1 + 1, D_{\sigma_r})$ problems to be exact. ς is the Gaussian parameter obtained from Theorem 10 when considering the bound of $c \cdot r$ for $c \in C$ and r sampled from $D_{\sigma_r}(R_{p'}^{d_2+d_1+1})$. This is for fresh commitments. For commitments after homomorphic operations, one has to increase B_ς accordingly. Additionally, we require the challenges to be invertible in R_p. In more detail, we get the following.

Theorem 8. *The above commitment scheme is binding. The strength of the binding property is based on the hardness of* $\mathsf{M\text{-}SIS}(R_{p'}, d_1, d_2 + d_1 + 1, \|\cdot\|_2, 4 \cdot \sqrt{N \cdot \kappa} \cdot B_\varsigma)$.

Proof. Given an instance $\mathbf{A}_0$ of the M-SIS problem and an adversary $\mathcal{A}_{\text{binding}}$ for the binding property, we show how to construct an adversary $\mathcal{A}_{\text{SIS}}$ on the SIS problem. We proceed just as in [57].

We do this by letting $\mathcal{A}_{\text{SIS}}$ pick $\mathbf{A}_1$ just as in the parameter generation and gives $(\mathbf{A}_0, \mathbf{A}_1)$ to $\mathcal{A}_{\text{binding}}$. $\mathcal{A}_{\text{binding}}$ answers with $((c[0], c[1])^{\mathrm{T}}, x, r, e, x', r', e')$. If both (x, r, e) and (x', r', e') are valid decommitments for c with $x \neq x'$, we have

$$e \cdot c[0] = \mathbf{A}_0 \cdot r \tag{7a}$$
$$e \cdot c[1] = \mathbf{A}_1 \cdot r + e \cdot x \tag{7b}$$
$$e' \cdot c[0] = \mathbf{A}_0 \cdot r' \tag{7c}$$
$$e' \cdot c[1] = \mathbf{A}_1 \cdot r' + e' \cdot x' \tag{7d}$$

Equations (7b) and (7d) lead to

$$\mathbf{0} \equiv \mathbf{A}_1 \cdot (e \cdot r' - e' \cdot r) + e \cdot e' \cdot (x' - x) \pmod{p}$$

As $x' \neq x$, we have that $x' - x \not\equiv \mathbf{0} \pmod{p}$. Additionally, e and e' are invertible in R_p. This implies $e \cdot r' - e' \cdot r \not\equiv \mathbf{0} \pmod{p}$. The same is true modulo p'. Equations (7a) and (7c) imply

$$\mathbf{0} \equiv \mathbf{A}_0 \cdot (e \cdot r' - e' \cdot r) \pmod{p'}$$

With the above observation that the difference of the commitment randomness is non-zero, we get a solution for the $\mathsf{M\text{-}SIS}(R_{p'}, d_1, d_2 + d_1 + 1, \|\cdot\|_2, 4 \cdot \sqrt{N \cdot \kappa} \cdot B_\varsigma)$ problem as we know that $\|r[i]\|, \|r'[i]\| \leq B_\varsigma$ and $\|e\|, \|e'\| \leq 2 \cdot \sqrt{\kappa}$. □

Theorem 9. *The above commitment scheme is hiding. The strength of the hiding property is based on the hardness of* $\mathsf{M\text{-}LWE}(R_{p'}, 1 + d_1, d_2 + d_1 + 1, D_{\sigma_r})$.

Proof. Given an instance $(\mathbf{B}, t)$ of the problem and an adversary $\mathcal{A}_{\text{hiding}}$ for the hiding property, we show how to construct an adversary $\mathcal{A}_{\text{LWE}}$ on the M-LWE problem. We proceed just as in [57].

Fist, we pick $\mathbf{R}$ uniformly at random from $R_{p'}^{d_1 \times 1}$ and send

$$\mathbf{A}_0 = \begin{pmatrix} \mathbf{I}_{d_1} & \mathbf{R} \end{pmatrix} \cdot \mathbf{B}$$
$$\mathbf{A}_1 = \begin{pmatrix} \mathbf{0}_{1 \times d_1} & \mathbf{I}_1 \end{pmatrix} \cdot \mathbf{B}$$

to $\mathcal{A}_{\text{hiding}}$. As an answer, we get x_0 and x_1. $\mathcal{A}_{\text{LWE}}$ picks a bit b uniformly at random and computes

$$c[0] = \begin{pmatrix} \mathbf{I}_{d_1} & \mathbf{R} \end{pmatrix} \cdot t$$
$$c[1] = \begin{pmatrix} \mathbf{0}_{1 \times d_1} & \mathbf{I}_1 \end{pmatrix} \cdot t + x_b$$

This is sent to $\mathcal{A}_{\text{hiding}}$, who answers with b'. $\mathcal{A}_{\text{LWE}}$ outputs 1 if $b = b'$ and 0 otherwise.

Now, we show that the construction of Baum et al. still works. First, write

$$\mathbf{B} = \begin{pmatrix} \mathbf{I}_{d_1} & \mathbf{0}_{d_1 \times 1} & \mathbf{B}'_0 \\ \mathbf{0}_{1 \times d_1} & \mathbf{I}_1 & \mathbf{B}'_1 \end{pmatrix}$$

where the entries of $\mathbf{B}'_i$ are picked uniformly at random from $R_{p'}$. This leads to

$$\mathbf{A}_0 = \begin{pmatrix} \mathbf{I}_{d_1} & \mathbf{R} & \mathbf{B}'_0 + \mathbf{R} \cdot \mathbf{B}'_1 \end{pmatrix}$$
$$\mathbf{A}_1 = \begin{pmatrix} \mathbf{0}_{1 \times d_1} & \mathbf{I}_1 & \mathbf{B}'_1 \end{pmatrix}$$

For $\mathbf{A}_0$ to be distributed as in the parameter generation, we need $\mathbf{R}$ and $\mathbf{B}'_0 + \mathbf{R} \cdot \mathbf{B}_1$ to be distributed uniformly in $R_{p'}$. Both are random as $\mathbf{R}$ and $\mathbf{B}'_0$ are uniformly distributed. For $\mathbf{A}_1$, we need that $\mathbf{B}'_1$ is randomly distributed over R_p. We get this, because $\mathbf{B}'_1$ is random over $R_{p'}$ and because p' is a multiple of p.

The rest of the proof is exactly the same as in [57]: If t is uniformly random, c is random as well. This implies that the value of b' is independent of x, i.e., $\Pr[b = b'] = 1/2$. If $t = \mathbf{B} \cdot r$ for a short commitment randomness r, c is a valid commitment for x. Now, $\Pr[b = b'] = 1/2 + \epsilon$ for the advantage ϵ of $\mathcal{A}_{\text{hiding}}$. The advantage of $\mathcal{A}_{\text{LWE}}$ (for the M-LWE problem) is then at least the advantage of $\mathcal{A}_{\text{hiding}}$ (for the hiding game). □

Similarly, one can show that the ZKPs for knowing decommitments and for linear relations of Baum et al. [57] still work.

For the security proofs of our protocols, we also need trapdoors for the BDLOP commitment scheme. We need two flavors of trapdoors: extractable and equivocal ones. Equivocal trapdoors for this commitment scheme were recently used by Damgård et al. [77] and we use the same approach. Same as Damgård et al., we use the results of Micciancio and Peikert [78] – but we need to construct both types of trapdoors. The extractable trapdoors are for R_q commitments (i.e., without our adjustments to the commitment scheme) and we can adapt the results of [78, Section 3.3] to do the following. A trapdoor as described in [78, Section 3.2] is constructed for $\mathbf{A}'_0$. With this, r can be extracted from $\mathbf{A}_0 \cdot r$ (the first part of a commitment). This value of r can then also be used to compute x from the second part of a commitment. For the extractable trapdoor, it is important that

- $\mathbf{A}'_0$ is distributed uniformly at random (as in a normal key generation),

- M-SIS is statistically hard (so there will be only one message that can be extracted for each commitment), and
- M-LWE is computationally hard (so commitments do not leak information about the committed messages).

We can use the same approach as Damgård et al. [77] to achieve the first point: We use [76, Corollary 4.2] to determine the parameter for the Gaussian. The second point is covered in the original BDLOP paper by Baum et al.. We consider all points in our parameter search in Section VII.

We want to use equivocal trapdoors for our adapted version of the commitment scheme. For this, we adapt the results of [78, Section 3.4] and Damgård et al. to construct a trapdoor for all of *par*. For the trapdoor, we consider the commitment key completely modulo p' ($\mathbf{A}_1$ can be reduced modulo p later). This means, we can use the technique of Damgård et al. "as is". We need to consider the following.

- *par* is distributed as in a normal key generation but modulo p' (using [76, Corollary 4.2]). Then, *par* will be distributed correctly in $R_{p'}^{d_1} \times R_p$ as well.
- The scheme is statistically hiding. For this, we also make sure that multiplying Gaussian commitment randomness with *par* is uniformly distributed modulo p' (again, using [76, Corollary 4.2]).
- M-SIS is computationally hard (so the commitment scheme is binding).

In our search in Section VII, we do not use this kind of trapdoor. Instead, we use a modified protocol to open the outputs of the protocol. Nevertheless, we compare the results of our presented protocol with the alternative version utilizing equivocal commitments in Section IX. Additionally, the alternative set of parameters can be found in Appendix I.

C. Problems with Beaver Multiplication

As hinted in Section VI, Beaver multiplication is problematic with the original version of BDLOP. We show how one can break the binding property for a commitments after the multiplication. Recall that we computed a bound of at least $p \cdot B_r$ for the randomness of commitments in Section VI. For the L^2-norm, the bound is larger than $\sqrt{N} \cdot p$. In particular, all elements of R_p trivially fulfill this. If we consider the original version of the commitment scheme with modulus p, this means that the range check on the randomness can be factually omitted when verifying a decommitment.

Let

$$c = \begin{pmatrix} \mathbf{I}_{d_1} & & \mathbf{A}'_0 \\ \mathbf{0}_{1\times d_1} & \mathbf{I}_1 & \mathbf{A}'_1 \end{pmatrix} \cdot r + \begin{pmatrix} 0 \\ x \end{pmatrix} = \mathbf{A} \cdot r + \begin{pmatrix} 0 \\ x \end{pmatrix}$$

be a commitment for an arbitrary plaintext x. A second decommitment (x', r') can be obtained by solving the following system of linear equations for r'

$$c - \begin{pmatrix} 0 \\ x' \end{pmatrix} = \mathbf{A} \cdot r'$$

This might not be possible for all choices of x, x', r, r' but one can find such values with non-negligible probability, winning the binding game.

Appendix F
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 [88] 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 [61] and the aggregation technique of [60].

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 [57], as well as products [86] and linear relations using commitments [57], [70]. 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 [57]). 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 [57], [70], [86] 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 [86] is used to prove $\boldsymbol{b} \cdot \bar{\boldsymbol{b}} = \mathbf{0}$.
3) The generalized sum proof [70] (as a generalization of the proof by Baum et al. [57]) is used to prove $\boldsymbol{b} + (\bar{\boldsymbol{b}} - 1) = \mathbf{0} \Leftrightarrow \bar{\boldsymbol{b}} = 1 - \boldsymbol{b}$. 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 [57], [70] 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 [86]. Instead, we want to prove a bound on $\|x\|_\infty$. As in [86], 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 [61] and classical aggregation [60].

```
function RejectionSample($\mathbf{Z}, \mathbf{B}, \sigma, \rho$):
    $u \xleftarrow{\$} U([0,1))$.
    if $u > \frac{1}{\rho} \cdot \exp\left(\frac{-2\langle \mathbf{Z},\mathbf{B}\rangle + \|\mathbf{B}\|^2}{2\cdot\sigma^2}\right)$ then
        return $0$.
    else
        return $1$.
function Sample($\rho$):
    $u \xleftarrow{\$} U([0,1))$.
    if $u > \frac{1}{\rho}$ then
        return $0$.
    else
        return $1$.
macro $P(A, \sigma, \rho)$:
    $\mathbf{B} \xleftarrow{\$} A()$.
    $\mathbf{Y} \xleftarrow{\$} D_\sigma(\mathbb{Z}^{n\times m})$.
    $\mathbf{Z} := \mathbf{B} + \mathbf{Y}$.
    if RejectionSample($\mathbf{Z}, \mathbf{B}, \sigma, \rho$) then
        return $\mathbf{B}, \mathbf{Z}$.
    else
        abort.
macro $S(A, \sigma, \rho)$:
    $\mathbf{B} \xleftarrow{\$} A()$.
    $\mathbf{Z} \xleftarrow{\$} D_\sigma(\mathbb{Z}^{n\times m})$.
    if Sample($\rho$) then
        return $\mathbf{B}, \mathbf{Z}$.
    else
        abort.
```

Fig. 19. Rejection sampling.

Theorem 10 (Rejection Sampling [61][23]). *The distribution of subprotocols P and S in Fig. 19 is within statistical distance of 2^{-100}. This requires $\sigma \geq \frac{12}{\ln \rho} \cdot \|\mathbf{B}\|$ for all $\mathbf{B} \xleftarrow{\$} A()$.*

We do not get proofs that have a slack as small as the one obtained by Baum et al. [95], 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 [68] uses a cut-and-choose technique which we want to avoid as this was identified to be impractical by Keller et al. [16]. Cramer et al. [69] need to amortize over many instances (η^2 or $\eta^{3/2}$) to be efficient. The work by del Pino and Lyubashevsky [96] builds upon [68], [69] to reduce the large number of instances required by Cramer et al. [69] but it is still a cut-and-choose proof. Another reason why we opted for the classical aggregation technique [60] 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 [68], [69] 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 and Appendix I).

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 [8] has a probability of repeating the proof: With probability $1/32$ the proof has to be discarded.

To present this technique, we show it for *committing ciphertexts*, i.e., the cases where a party has to prove that a ciphertext and a commitment were produced for the same plaintext. The same can be used for only proving that ciphertexts were correctly computed by the parts that are not needed. Similarly, we use it to prove that the re-encryption randomness in Fig. 7 and the decryption masks in Fig. 15 are for plaintexts with the appropriately sampled distribution.

The matrix $\mathbf{M}_{\boldsymbol{c}}$ is constructed as in [8], [60]. $\mathbf{M}_{\boldsymbol{c}} \in \{0,1\}^{\mu\times\eta}$ is defined as $\boldsymbol{c}[i-j]$ at position (i,j) for $0 \leq i-j < \eta$ and zero otherwise. This allows us to solve for the secrets in the knowledge extractor. The parameters ς are obtained from Theorem 10, the construction of $\mathbf{M}_{\boldsymbol{c}}$, and bounds on the witnesses (the exact relation is given in the following proof).

Theorem 11. *The protocol of Fig. 20 is a Σ-protocol as in [88, Definition 6].*

Proof. Completeness: This is quite trivial and follows from the linearity of the commitment and encryption schemes.

Unpredictable commitments: Obviously, picking commitments (in the sense of Σ-protocols) as in the protocol does not produce collisions, except with low probability.

Honest-verifier zero-knowledge: The simulator of the Σ-protocol can pick the $\boldsymbol{z}$ values according to the Gaussian distributions of the respective $\boldsymbol{y}$s. Also $\boldsymbol{c}$ can be picked as in the protocol. The commitments (in the sense of Σ-protocols) can be solved for with these values. The distance between this simulation and a real protocol execution is the same as for P and S in Theorem 10.

Statistical soundness: Here, we show statistical special soundness (which implies statistical soundness). Given two accepting transcripts with the same commitment (in the sense of Σ-protocols) and different challenges, an extractor can regain witnesses (messages and randomness) for the inputs of the Σ-protocol. Let $\boldsymbol{z}$ and $\boldsymbol{z}'$ be the responses of the transcripts (this is done for all $\boldsymbol{z}$-value pairs, e.g., $\boldsymbol{z}_x$ and $\boldsymbol{z}'_x$) and $\boldsymbol{c}, \boldsymbol{c}'$ the challenges. As in [8], we can get a witness from conceptually computing $(\mathbf{M}_{\boldsymbol{c}'} - \mathbf{M}_{\boldsymbol{c}})^{-1} \cdot (\boldsymbol{z}' - \boldsymbol{z})$. The special structure of the matrices allows one to do the multiplication with the inverse in two steps. Two triangle matrices in $\mathbf{M}_{\boldsymbol{c}'} - \mathbf{M}_{\boldsymbol{c}}$ can be identified to solve for the first and second half of the witness, respectively. In the worst case, $2^{\eta/2}$ additions of entries of $\boldsymbol{z}' - \boldsymbol{z}$ are used to compute an entry of the witness. This means, the norm of the witness can be bound by $2^{\eta/2} \cdot 2 \cdot B_\varsigma$ for the respective bound B_ς on $\boldsymbol{z}$ and $\boldsymbol{z}'$. We get ς from Theorem 10 and set $\varsigma := \frac{12}{\ln \rho} \cdot \eta \cdot B_\sigma$, i.e., it is the parameter of the Gaussian obtained from the rejection sampling theorem where the bound in the theorem is $\eta \cdot B_\sigma$. The latter is the bound on the terms $\boldsymbol{z} - \boldsymbol{y}$ in the Σ-protocol. B_σ is the bound on the respective witness and the factor of η comes from the

[23] with the notation of [95]

```
input for $\mathcal{P}_{\text{verifier}}$: $k, par, par'$,
                         $\mathsf{Enc}(\boldsymbol{x}), \mathsf{Com}(\boldsymbol{x}), \mathsf{Com}'(\boldsymbol{x}), \mathsf{Com}'(\boldsymbol{r})$
input for $\mathcal{P}_{\text{prover}}$ : (additionally) $\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{e}, \boldsymbol{r}, \boldsymbol{r}'_x, \boldsymbol{r}'_r$ s.t.
                         $\mathsf{Enc}_k(\boldsymbol{x}, (\boldsymbol{v}, \boldsymbol{e})^{\mathrm{T}}) = \mathsf{Enc}(\boldsymbol{x})$,
                         $\mathsf{Com}_{par}(\boldsymbol{x}, \boldsymbol{r}) = \mathsf{Com}(\boldsymbol{x})$,
                         $\mathsf{Com}_{par'}(\boldsymbol{x}, \boldsymbol{r}'_x) = \mathsf{Com}'(\boldsymbol{x})$,
                         $\mathsf{Com}_{par'}(\boldsymbol{r}, \boldsymbol{r}'_r) = \mathsf{Com}'(\boldsymbol{r})$
Commitment: The prover $\mathcal{P}_{\text{prover}}$ does:
    Let $\mu := 2 \cdot \eta - 1$.
    $\boldsymbol{y}_x \xleftarrow{\$} D_{\varsigma_x}(R^\mu)$.
    $\boldsymbol{y}_r \xleftarrow{\$} D_{\varsigma_r}(R^\mu)^{d_2+d_1+1}$.
    $\boldsymbol{y}_{v,e} \xleftarrow{\$} D_{\varsigma_v}(R^\mu) \times D_{\varsigma_e}(R^\mu) \times D_{\varsigma_e}(R^\mu)$.
    $\boldsymbol{y}_{r',x} \xleftarrow{\$} D_{\varsigma_{r'}}(R^\mu)^{d_2+d_1+1}$.
    $\boldsymbol{y}_{r',r} \xleftarrow{\$} D_{\varsigma_{r'}}(R^\mu)^{d_2+d_1+1}$.
    $\boldsymbol{a}_{x,\mathsf{Enc}} := \mathsf{Enc}_k(\boldsymbol{y}_x, \boldsymbol{y}_{v,e})$.
    $\boldsymbol{a}_{x,\mathsf{Com}} := \mathsf{Com}_{par}(\boldsymbol{y}_x, \boldsymbol{y}_r)$.
    $\boldsymbol{a}_{x,\mathsf{Com}'} := \mathsf{Com}_{par'}(\boldsymbol{y}_x, \boldsymbol{y}_{r',x})$.
    $\boldsymbol{a}_{r,\mathsf{Com}'} := \mathsf{Com}_{par'}(\boldsymbol{y}_r, \boldsymbol{y}_{r',r})$.
    send $\boldsymbol{a}_{x,\mathit{Enc}}, \boldsymbol{a}_{x,\mathit{Com}}, \boldsymbol{a}_{x,\mathit{Com}'}, \boldsymbol{a}_{r,\mathit{Com}'}$
Challenge: The verifier $\mathcal{P}_{\text{verifier}}$ does:
    $\boldsymbol{c} \xleftarrow{\$} U(\{0,1\}^\eta)$.
    send $\boldsymbol{c}$.
Response: The prover $\mathcal{P}_{\text{prover}}$ does:
    $\boldsymbol{z}_x := \boldsymbol{y}_x + \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{x}$.
    $\boldsymbol{z}_r := \boldsymbol{y}_r + \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{r}$.
    $\boldsymbol{z}_{v,e} := \boldsymbol{y}_{v,e} + \mathbf{M}_{\boldsymbol{c}} \cdot (\boldsymbol{v}, \boldsymbol{e})^{\mathrm{T}}$.
    $\boldsymbol{z}_{r',x} := \boldsymbol{y}_{r'} + \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{r}'_x$.
    $\boldsymbol{z}_{r',r} := \boldsymbol{y}_{r'} + \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{r}'_r$.
    Do rejection sampling:
    if $\neg\texttt{RejectionSample}(\boldsymbol{z}_x, \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{x}, \varsigma_x, \rho) \vee \cdots \vee$
      $\neg\texttt{RejectionSample}(\boldsymbol{z}_{r',r}, \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{r}'_r, \varsigma_{r'}, \rho)$ then
        abort.          // and repeat the proof
    send $\boldsymbol{z}_x, \boldsymbol{z}_r, \boldsymbol{z}_{v,e}, \boldsymbol{z}_{r',x}, \boldsymbol{z}_{r',r}$.
Verification: The Verifier $\mathcal{P}_{\text{verifier}}$ does:
    Check that $\mathsf{Enc}_k(\boldsymbol{z}_x, \boldsymbol{z}_{v,e}) = \mathsf{Enc}(\boldsymbol{x}) + \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{a}_{x,\mathsf{Enc}}$,
    $\mathsf{Com}_{par}(\boldsymbol{z}_x, \boldsymbol{z}_r) = \mathsf{Com}(\boldsymbol{x}) + \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{a}_{x,\mathsf{Com}}$,
    $\mathsf{Com}_{par'}(\boldsymbol{z}_x, \boldsymbol{z}_{r',x}) = \mathsf{Com}'(\boldsymbol{x}) + \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{a}_{x,\mathsf{Com}'}$,
    $\mathsf{Com}_{par'}(\boldsymbol{z}_r, \boldsymbol{z}_{r',r}) = \mathsf{Com}'(\boldsymbol{r}) + \mathbf{M}_{\boldsymbol{c}} \cdot \boldsymbol{a}_{r,\mathsf{Com}'}$.
    Check that $\|\boldsymbol{z}_x\| \le B_{\varsigma_x}$, $\|\boldsymbol{z}_r[i]\| \le B_{\varsigma_r}$,
    $\|\boldsymbol{z}_{v,e}[0]\| \le B_{\varsigma_v}$, $\|\boldsymbol{z}_{v,e}[1]\| \le B_{\varsigma_e}$, $\|\boldsymbol{z}_{v,e}[2]\| \le B_{\varsigma_e}$,
    $\|\boldsymbol{z}_{r',x}[i]\| \le B_{\varsigma_{r'}}$, and $\|\boldsymbol{z}_{r',r}[i]\| \le B_{\varsigma_{r'}}$ for
    $0 \le i < d_2 + d_1 + 1$.
    if any check failed then return $0$ else return $1$.
```

Fig. 20. Σ-protocol for committing ciphertexts.

multiplication with $\mathbf{M}_{\boldsymbol{c}}$. Instead of B_{σ_x}, we convert the L^∞-norm on $\boldsymbol{x}$ (i.e., $(p-1)/2$) into a L^2-norm (we upper-bound it with an L^2-norm of $\sqrt{N} \cdot (p-1)/2$ [57]).

The commitments (in the sense of Σ-protocols) and the linearity of the schemes imply that the extracted values are witnesses for the relation.

Unique responses: The right-hand side of the checks in the verification phase of Fig. 20 depends only the commitments (in the sense of Σ-protocols), challenge and public inputs of the protocol. For the verification to succeed for multiple responses, the prover would have to break the binding property of the commitment schemes. The encryption can be seen as an perfectly binding commitment. □

Remark 10. *Completeness and unpredictable responses imply that the Fiat-Shamir transform of the protocol is complete w.r.t. quantum adversaries [88, Lemma 13]. Adding honest-verifier zero-knowledge implies zero-knowledge for the transformed protocol [88, Theorem 14].*

Statistical soundness implies soundness after the transform [88, Theorem 15]. This and unique responses implies simulation-soundness [88, Theorem 17].

With the extractability we get from letting the protocol simulator know the private key and trapdoor to the commitment scheme with key par′, we get that the transformed Σ-protocol is strongly simulation-sound extractable [89, Theorem 25], i.e., we have soundness, even if the adversaries sees fake proofs from a simulator and also extractability in presence of a simulator [89].

Remark 11. *By the general rejection sampling theorem [61], we get that the overall probability to get an output from the ZKP (no abort) is the probabilities of not aborting in all calls to* `RejectionSample`*, i.e., $(1/\rho)^l$, or statistically close to that, for l rejection samplings.*

Appendix G
Usability Improvements

There a few aspects, one could solve differently in our protocols. Some are outlined here, including a way to support private inputs, larger circuits, and doing less work for verifications (under certain assumptions).

A. Private Outputs

The mentioned online protocol only supports public outputs. Private outputs can be modelled with the existing protocol as well. For this, a party that should receive a private output y picks an additional input r uniformly at random. The circuit is modelled to (secretly) compute and output $z := y + r$. The party can then easily compute y as $z - r$ after the protocol. Correctness of z (and therefore of y) is guaranteed by the existing protocol.

B. Handling Many Homomorphic Commitments

Obviously, using lattice-based commitments does not allow us to do arbitrary many additions (opposed to, for example, the Pedersen commitment scheme in [11], [15]). A straightforward way to support a circuit is to estimate an upper bound on the number of additions needed and use this in the parameter search. This is done in Section VII. One can also inject *no-ops* in the circuit to keep the noise small.

As the noise in commitments *resets* after multiplications, one could *multiply by one*. For this, $\langle x \rangle_i := \langle c \rangle_i + (x - b) \cdot \langle a \rangle_i + (1 - a) \cdot \langle b \rangle_i + (x - b) \cdot (1 - a)$ could be computed by using a multiplication triple. This means, $x - b$ and $1 - a$ are opened in the protocol. Both do not leak any information about x if the preprocessing was correct (and not too many parties are corrupted).

Alternatively, one could *add zero*. With $\langle x \rangle_i := \langle r \rangle_i + (x - r)$, one can *refresh* commitments similarly to adjusting shares for the inputs of the protocol. Here, we only need more random views $\langle r \rangle_i$ instead of more complex triples, only one value $(x - r)$ has to be opened, and less operations are needed to adjust the view.

Additionally, we can assume that many operations will reset the commitment noise if they are implemented like Beaver triples. Examples are matrix multiplications and convolutions [7].

When computing the parameters, one has to consider the slack added from the zero-knowledge proofs. While this might be smaller than the number of additions (cf. Table II), it is still plays a role in the parameter search and cannot be avoided with the above techniques. Instead, one has to lower the slack introduced by the zero-knowledge proofs.

C. Delayed Verification

In the online phase, the parties could first compute the whole circuit without checking the commitments (while still decommitting at every `output`-gate). For his, one could use hyper-invertible Van der Monde matrices to reconstruct correctly if less than t shares are inconsistent [49], [97]. An *error correction* in this way is less efficient in $\mathcal{O}$-notation (dependent on n) than checking each commitment but only field operations are required, which might make this variant more efficient in practice for certain settings (expensive to verify commitments and not too many parties).

Obviously, if too many parties are corrupted, we might not get the correct result. However, at the end (or as external auditor), one can still check the commitments – possibly in an aggregated fashion by checking random linear combinations.

Another possibility to reduce the verification overhead is the following. Consider that only checks of the form

$$\mathsf{Verify}\left(\underbrace{\sum_l a_l \cdot [x_l]_j}_{\substack{:=[z]_j \\ \text{sent by } \mathcal{P}_j}}, \underbrace{\sum_l a_l \cdot \mathsf{R}_\mathsf{C}([x_l]_j)}_{\substack{:=\mathsf{R}_\mathsf{C}([z]_j) \\ \text{sent by } \mathcal{P}_j}}, \underbrace{\sum_l a_l \cdot \mathsf{Com}([x_l]_j)}_{\text{computed by } \mathcal{P}_i}\right)$$

have to be done by $\mathcal{P}_i$ for each $\mathcal{P}_j$. This is because the use of Beaver triples creates a circuit with only linear operations. Instead, one could do an aggregated check

$$\mathsf{Verify}\left(\sum_{j=1}^{n} [z]_j, \sum_{j=1}^{n} \mathsf{R}_\mathsf{C}([z]_j), \sum_l a_l \cdot \sum_{j=1}^{n} \mathsf{Com}([x_l]_j)\right)$$

where $\mathsf{Com}(x'_l) := \sum_{j=1}^{n} \mathsf{Com}([x_l]_j)$ can be computed first, so only one commitment for each x_l has to be kept in memory instead of n commitments. Note that, in general, $x'_l \neq x_l$ as we use Shamir secret-sharing instead of full-threshold secret-sharing. Otherwise, this is similar to the technique of Baum et al. [11] where commitments for the shared secrets (not the shares) are used to get verifiability. We have the commitments for shares and can go back to the individual checks if the aggregated check fails. Unfortunately, one might have to go all the way back and recompute the whole circuit if the aggregated check fails. This makes this technique more suitable if cheating is unlikely (otherwise, one has more verification overhead). It could be used to only get public verifiability as well (as in [11]).

TABLE V
COMPARISON OF BGV PARAMETERS AGAINST [15] WITH ELLIPTIC CURVE (EC) AND QUADRATIC RESIDUE (QR) PEDERSEN COMMITMENTS

	ours[a]	ours[b]	ours[c]	[15] EC		[15] QR	
η'	$\log q$	$\log q$	$\log q$	$\log p$	$\log q$	$\log p$	$\log q$
60	353	417	410	120	537	700	1705
70	352	418	428	140	577	1000	2314
100	354	417	490	200	697	1900	4121
128	353	417	553	256	809	3200	–
192	353	417	681	384	1073	7900	–

[a] $\log p = 32$, [b] $\log p = 64$, [c] $\log p = \eta'$

D. Adaptions for E-Voting

The protocols as presented above are able to handle e-voting by interpreting each voter as an input party. This comes with a high number of decryptions to bring the encrypted votes into the online phase. This seems unnecessary if the votes are summed up anyways. In these cases, one could slightly modify the protocol to reduce the number of decryptions needed for the inputs.

For simplicity, we assume each voter encrypts the vote for each candidate (e.g., a one if a candidate is voted for or a zero otherwise) in the slots of the ciphertext-vote. This means there will be one ciphertext for each voter. The first step of voting is to add up the votes for each candidate. This can now be done by simply adding up all (valid) encrypted votes. Now, only one ciphertext has to be decrypted.

To bring this in a form that is comparable to the presented protocol, we introduce a dummy input party. This party does not give a valid input (i.e., it does not have to exist) and the default ciphertext that is picked if this party does not give a valid input is set to the sum of all (valid) voters' ciphertexts (instead of taking a encryption of zero as in the above protocol description). This can be computed by all parties as the ciphertexts are published on the bulletin board. For the MPC protocol, the voters are not considered further as actual input parties. The function that the MPC protocol computes has to be stripped of the vote aggregation that already happened. The functionality needs only a minor adjustment: While, the voters would directly give their votes to the functionality and the function that the functionality computes still includes the aggregation of votes, it also has to check the inputs for validity. This validity check is added to the NIZKPs the voters have to publish (in the protocol).

Finally, one has to recheck (and potentially adjust) the parameter analysis for the encryption scheme. With a similar approach, other linear function-preprocessing can be done. This is especially useful if the number of inputs to the MPC protocol is reduced in this way (as in the example of e-voting). If the encryption scheme supports (a limited number of) non-linear operations, these could be performed as well.

APPENDIX H
PROBLEMS WITH PEDERSEN COMMITMENTS

The protocol by Cunningham et al. [15] (and Baum et al. [11]) use Pedersen commitments to detect cheating in the online phase. From an efficiency point-of-view, this seems to be suboptimal as the plaintext space (or p) has to be large enough to make the commitments secure. Small values of p already lead to an insecure protocol if a SPDZ-MACs is used for verification but solutions to make small fields still secure were developed [19], [98]. Building Pedersen commitments from quadratic residues as suggested by Cunningham et al. is highly impractical because of the required size of p needed to make the scheme secure. Basing the commitments on elliptic curves will drastically reduce the required size of p but it still depends on the computational security parameter ($\log p \approx 2{\cdot}\eta'$ will be required [99]).

One could argue that the security of the commitment scheme is only important in the online phase and by extension only for a short time. This is because once the protocol is finished, only the hiding property of the commitment scheme is important and Pedersen commitments are perfectly hiding. We argue that having lower security of commitments for this reason is not always acceptable. Firstly, in a high-stakes computation (national elections or certain auctions), a malicious actor might have the incentive to invest in the computational power to influence the result of the computation. If the malicious actor manages to forge decommitments, their outcome would be backed by a publicly verifiable computation and nobody could extract evidence from the computation that they cheated.

Another argument can be made if the time between the offline and online phase is not short. In this case, a malicious actor knows a random share $[r]_i$ that will be adjusted to fit another party's input. They also know a commitment for this share. If they now manage to produce a decommitment for the same commitment but with the decommitment $[r']_i := [r]_i - c$, they could use this (undetectedly) in the protocol and influence another party's input, as the value used in the protocol would be reduced by the same offset c (as full-threshold secret-sharing is used by Cunningham et al.; Shamir secret-sharing could be attacked similarly). In an auction, this could be used to reduce other parties' bids and increase the chances of winning.

The last two arguments made clear, why (at least in certain situations), weakly binding commitments are not acceptable. To achieve high security with Pedersen commitments, the plaintext space has to increase with the security level which leads to inefficiencies in other parts of the protocol. Table V shows our computations for the required BGV parameters for the offline phase of Cunningham et al.'s protocol if one would use Pedersen commitments based on elliptic curves or quadratic residues. We also included numbers for our protocol for two fixed sizes of p (as our protocol security does not depend on the size of p) and also the case where the size of p equals the security parameter η'. Our scheme can always use a smaller ciphertext modulus and this will make the offline phase more efficient. Additionally, the table shows that even for low security levels (e.g., 60 bit), using quadratic residues for Pedersen commitments requires parameters that are beyond practical limits (and even higher than ours for the highest security levels considered).

TABLE VI
COMPARISON OF BGV PARAMETERS ($\log q$) AGAINST TOPGEAR [17], LOWGEAR [16], AND SPDZ [8], [9]

		ours	ours	ours			
η	$\log p$	(SHE)	(LHE)	(SHE)[a]	[17]	[16]	[8], [9]
40	64	416	473	349	301	199	330
64	64	480	544	398	–	224	378
80	64	530	584	432	381	–	–
128	64	652	707	535	471	–	–
40	128	554	672	478	491	327	526
64	128	617	735	532	–	352	572
80	128	658	776	566	571	–	–
128	128	782	898	664	671	418	700

[a] aggregation [60] and rejection sampling [61] with SHE and no zero-knowledge proof of correct decryption

TABLE VII
COMMITMENT PARAMETERS par FOR R_p

$\log p$	η	N	$\log h$	d_2	d_1	$\log p'$	B_r
32	40	16384	59	1	1	120	1
32	64	16384	59	1	1	134	1
32	80	16384	59	1	1	142	1
32	128	32768	60	1	1	170	1
64	40	16384	91	1	1	152	1
64	64	16384	91	1	1	166	1
64	80	32768	92	1	1	176	1
64	128	32768	92	1	1	202	1
128	40	32768	156	1	1	218	1
128	64	32768	156	1	1	232	1
128	80	32768	156	1	1	240	1
128	128	32768	156	1	1	266	1

APPENDIX I
PARAMETERS (CONTINUED)

We use a statistical security of 40 bit and a computational security level of 128 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 [9], [84] but we consider the worst-case bounds for elements that are distributed w.r.t. Gaussian distributions

TABLE VIII
EQUIVOCAL COMMITMENT PARAMETERS par FOR R_p

$\log p$	η	N	$\log h$	d_2	d_1	$\log p'$	$\log \sigma_r$
32	40	16384	59	4	1	236	119
32	64	16384	59	4	1	258	128
32	80	16384	59	4	1	272	134
32	128	32768	60	4	1	320	155
64	40	16384	91	4	1	290	141
64	64	16384	91	4	1	311	149
64	80	32768	92	4	1	331	159
64	128	32768	92	4	1	373	176
128	40	32768	156	4	1	403	188
128	64	32768	156	4	1	424	196
128	80	32768	156	4	1	438	202
128	128	32768	156	4	1	480	219

TABLE IX
COMMITMENT PARAMETERS par' FOR R_q

$\log p$	η	N	d_2	d_1	$\log q$	σ_r
32	40	16384	2	1	352	2^{91}
32	64	16384	2	2	416	2^{78}
32	80	16384	2	2	458	2^{88}
32	128	32768	1	1	589	1
64	40	16384	2	2	418	2^{78}
64	64	16384	2	2	480	2^{93}
64	80	32768	1	1	530	1
64	128	32768	1	1	653	1
128	40	32768	1	1	553	1
128	64	32768	1	1	616	1
128	80	32768	2	2	658	2^{136}
128	128	32768	2	2	780	2^{167}

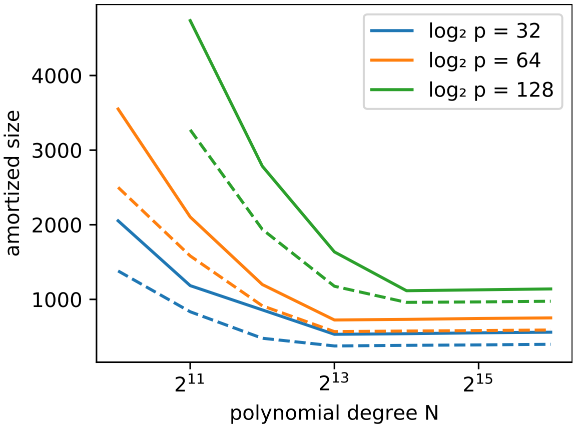


Fig. 21. Amortized total size of commitments for R_p. Dashed lines consider that one can avoid sending/storing all bits of randomness with a bounded norm.

TABLE X
PARAMETERS AND AMORTIZED TOTAL SIZE OF COMMITMENTS FOR R_p

$\log p$	N	$\log h$	d_2	d_1	$\log p'$	size[a]	size[a,b]
32	1024	55	6	5	117	2053	1381
32	2048	56	3	3	112	1184	834
32	4096	57	2	1	159	859	479
32	8192	58	1	1	117	532	376
32	16384	59	1	1	119	540	384
32	32768	60	1	1	122	552	390
32	65536	61	1	1	124	560	398
64	1024	87	8	6	163	3551	2501
64	2048	88	4	4	152	2104	1582
64	4096	89	2	2	153	1199	914
64	8192	90	1	1	149	724	568
64	16384	91	1	1	151	732	576
64	32768	92	1	1	154	744	582
64	65536	93	1	1	156	752	590
128	2048	152	7	4	280	4736	3272
128	4096	153	4	2	281	2785	1938
128	8192	154	2	1	276	1636	1176
128	16384	155	1	1	215	1116	960
128	32768	156	1	1	218	1128	966
128	65536	157	1	1	221	1140	975

[a] the amortized total size, i.e., the size of a $(x, \mathsf{R_C}(x), \mathsf{Com}(x))$-tuple divided by N in bit
[b] considering that one can avoid sending/storing all bits of randomness with a bounded norm

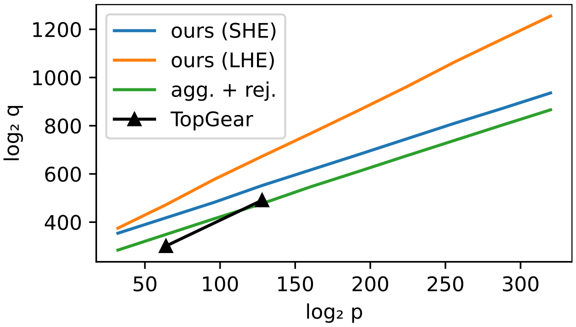


Fig. 22. Comparison of BGV parameters against TopGear [17].

TABLE XI
COMPARISON OF BGV PARAMETERS ($\log q$) AGAINST TOPGEAR [17]

$\log p$	ours (SHE)	ours (LHE)	ours[a] (SHE)	TopGear [17]
32	354	375	284	–
64	418	472	349	301
96	482	577	414	–
128	552	673	477	491
160	616	767	546	–
192	680	863	610	–
224	745	961	675	–
256	810	1064	739	–
288	872	1160	803	–
320	936	1255	866	–

[a] aggregation [60] and rejection sampling [61] with SHE and no zero-knowledge proof of correct decryption

(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) 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 [100], [101][24] with cost models from [102], [103] to estimate the hardness of M-LWE. Additionally, we used results from the literature [100], [104], [105] to estimate the hardness of M-SIS. We also made sure that the constraints for the trapdoors are met (see Appendix E).

As mentioned, we use ResNet [79], [106] as an example application to estimate how many homomorphic operations on commitments are needed; more specifically ResNet152 V2 [79], [107], a non-trivial arithmetic circuit for the real-world application of machine learning for image classification.

For rejection sampling, we used the parameter $\rho = 100/99$, i.e., there is a $1\,\%$ chance of an abort in the rejection sampling theorem (the protocol does not abort then, the proof is simply repeated). The values for Fig. 8 can be found in Table VI – the ones for Fig. 9 in Table V. Tables VII and IX are extended

[24] We also used [101] to double-check the security of the BGV parameters (to verify that we achieve the 128 bit security we aimed for).

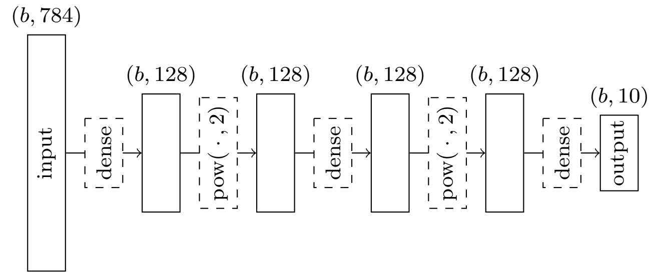


Fig. 23. Architecture of "network A" [2], [80]. The shape of the input and (intermediate) feature tensors is given above each tensor. b represents the batch size. Dashed boxes represent the network layers (dense layers, nonlinearities).

versions of Table II. To compute the values, a computational security level of $128\,\mathrm{bit}$ was used for BGV. All BGV-related standard deviations were set to a value of 3.2. The values in Table V for Cunningham et al. [15] were produced with our techniques as they did not specify details on the zero-knowledge proofs or distributed decryption. Security levels for elliptic curves and quadratic residues were taken from [99, Table 3.2].

Figure 21 (with values in Table X) highlights another reasoning why we want to use larger values of N for the commitment scheme (except for a synergy with BGV). As can be seen, the amortized size (that we optimize the parameters for) is much larger for low values of N. When picking N as for BGV, the size is also (nearly) minimized.

Table VIII show the commitment parameters for our protocol when we utilize statistically hiding commitments with a trapdoor for equivocation. As already mentioned in Section IX, these are much larger than the ones in the computationally secure variant (seen in Table VII) that we use in our protocol.

Figure 22 and Table XI make us believe that our way of combining classical aggregation [60] and rejection sampling [61] or zero-knowledge proofs might lead to smaller BGV parameters than TopGear [17] for larger plaintext sizes. The relation of $\log p$ and $\log q$ seems linear for all protocols. For TopGear, the incline seems to be more steep, leading us to suspect benefits for our approach with even larger sizes of p. The results of our computations for $\eta = 40$ are shown here. For $\eta = 80$ and $\eta = 128$, this behavior is the same (for the values of $\log p$ that we have data from [17]).

Appendix J
Benchmarks

To compare the runtime of our protocol to SPDZ and $\mathrm{BoBW}_{[15]}$, 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. 23; we use $b = N$ as batch size to fully utilize the slots of the BDLOP commitments; packing methods and individual slot manipulation, used for (fully) homomorphic encryption schemes, with the BDLOP scheme can be investigated in future work). The final argmax layer is left out as the operations there are not purely arithmetic. Also, we compute the circuit over integers (modulo p) instead of floating-point or fixed-point numbers.[25] Operations in argmax layers and for fixed-point numbers require similar types of preprocessed data (e.g., (authenticated) shares of bits; besides Beaver triples) and are out of scope for our evaluation.

The runtime for SPDZ and [15] without verification in the online phase is for evaluating the whole arithmetic circuit of "network A", while the other results for the online phase were extrapolated from a smaller circuit (i.e., by computing 4 dot products instead of 128 dot products for the first two dense layers and multiplying the runtime for these layers by 32; note that the size of vectors considered for the dot products is not scaled down). For [15] with verification of commitments, we only ran the protocol once (as seen in Fig. 10(a)) as the benchmarks (even with a smaller circuit and only a single dot product for the first two dense layers) took too long to try it with all delays. Therefore, Fig. 10(c) uses these results for all delays for $\mathrm{BoBW}_{[15]}$ with restarts. Other experiments were run twice and runtime shown here is the average over the runs of all parties. Also, the results shown in Fig. 10(c) for $\mathrm{BoBW}_{[15]}$ with restarts consist of runtime results of an online run with verified commitments (as discussed above) plus the runtime of an online and offline phase with two parties.

Our implementation is based on MP-SPDZ [80], [81] and uses mostly existing building blocks (for arithmetic and communication), i.e., one can expect the results of our micro benchmark to translate to a full implementation of our protocol in MP-SPDZ. Indeed, comparing, for example, the results from Fig. 10(b) to test runs with MP-SPDZ show that the runtime is comparable (our implementation for SPDZ is at most $46\,\%$ slower in the offline phase which we attribute to the optimization effort put into MP-SPDZ's implementation). The online runtime of our implementation for SPDZ is even closer to the one of MP-SPDZ; our amortized runtime is even around $16\,\%$ less than the runtime obtained by running SPDZ in MP-SPDZ (when evaluating "network A" once, i.e., without batching).

We ran our benchmark on a single server (AMD EPYC

[25] Computing the circuit with fixed-point numbers (which is more common than floating-point computation in MPC; e.g., as in [6]) amounts to computing the circuit with integers and normalizing intermediate results. The latter can be done by combining multiplications and fixed-point truncation to a single operation—in the same number of rounds as computations as over integers with only slightly more (local) operations per multiplication.

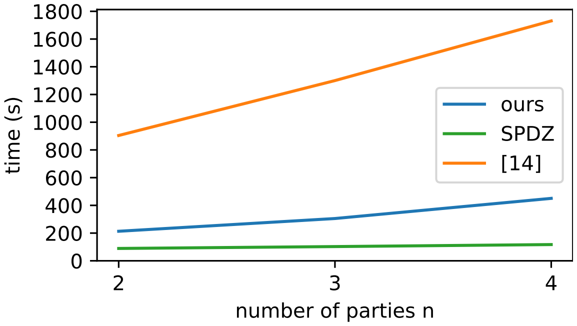
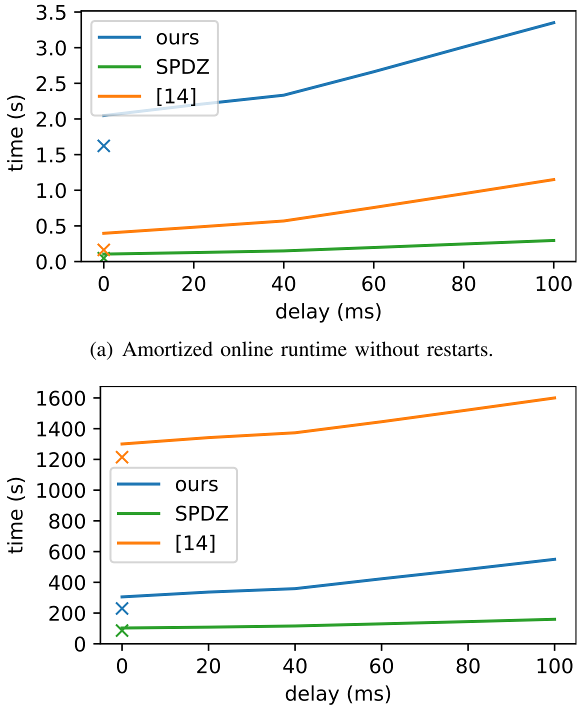


Fig. 24. Amortized offline runtime without restarts in the same settings as in Fig. 10 with 0 ms delay and varying number of parties. $t = \lfloor n/2 \rfloor + 1$.

7443 CPU, 24 cores, 2.85 GHz, 512 GB RAM) to get the presented results. All experiments were done on a single machine running the code for all parties, utilizing a single thread per party for the computations and additional threads for communication. We emulate the network behavior with the netem functionality. The network delay was varied between 0 ms and 100 ms in steps of 20 ms. Additionally, the bandwidth was limited such that each party can send data with at most 1 Gbit (on average). We varied the network delay (i.e., the latency) to simulate different settings where our protocol might be deployed. Low latency with only a few milliseconds delay corresponds to a LAN setting where parties are located closely together, while a delay of 50 ms corresponds to the WAN setting evaluated, for example, in other works [13], [16]. A higher latency of around 100 ms can be observed when communication happens at a global scale.

Discussion: As can be seen in Figs. 10(a) and 10(b), our protocol is significantly faster than [15] when verifying commitments in the online phase and in the offline phase. This can be attributed (for the most part) to the efficiency of the BDLOP commitment scheme compared to (elliptic curve) Pedersen commitments.[26]

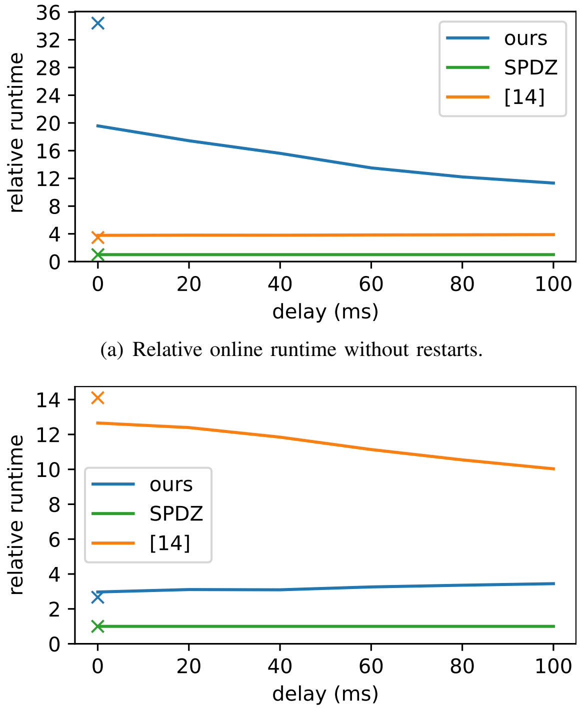
For the online phase, one can see that the overhead of Cunningham et al. to SPDZ in communication corresponds almost exactly to the runtime overhead. When one has to consider additional verification (either for our protocol or for Cunningham et al. when there is an abort), the cost of verifying commitments can be clearly seen to affect the runtime. Therefore, communication volume cannot be used as the sole indicator for (online) performance. For the offline phase, our communication cost (per party) increases linearly with the number of parties as discussed in Section VIII where we analyze the asymptotic performance of our protocol. In concrete settings however, the offline communication cost of our protocol can be even lower than the one for Cunningham et al., as can be seen in Fig. 11, as we do not have to compute SPDZ-like MACs and benefit from smaller BGV parameters (as discussed in Section VII). Figure 24 shows the offline runtime with varying number of parties. Due to the lower overhead of BDLOP commitments, our protocol seems to



(a) Amortized online runtime without restarts.

(b) Amortized offline runtime without restarts.

Fig. 25. Runtime for evaluating "network A" in the same setting as in Fig. 10. Crosses indicate the runtime in a setting without network restrictions (i.e., no delay and no bandwidth limitation to $1\,\mathrm{Gbit\,s^{-1}}$).



(a) Relative online runtime without restarts.

(b) Relative offline runtime without restarts.

Fig. 26. Relative runtime for evaluating "network A" in the same setting as in Fig. 10. Time is given as a factor relative to SPDZ. Crosses indicate the relative runtime in a setting without network restrictions (i.e., no delay and no bandwidth limitation to $1\,\mathrm{Gbit\,s^{-1}}$).

[26] As shown in Section VII, using Pedersen commitments based on quadratic residues would increase the BGV parameters for the offline phase significantly and probably only slow [15] down even more.

TABLE XII
AMORTIZED ONLINE RUNTIME (IN SECONDS)

bandwidth	delay	ours	SPDZ	[15]	[15] w. verif.
unlimited	0 ms	1.62259	0.04717	0.16373	–
1 Gbit s^{-1}	0 ms	2.04747	0.10459	0.39622	86.07595
1 Gbit s^{-1}	20 ms	2.19651	0.12601	0.48117	–
1 Gbit s^{-1}	40 ms	2.33357	0.14945	0.56918	–
1 Gbit s^{-1}	60 ms	2.66247	0.19699	0.75792	–
1 Gbit s^{-1}	80 ms	3.00954	0.24645	0.95213	–
1 Gbit s^{-1}	100 ms	3.35027	0.29578	1.14991	–

TABLE XIII
AMORTIZED OFFLINE RUNTIME (IN SECONDS)

bandwidth	delay	ours	SPDZ	[15]
unlimited	0 ms	229.77221	86.11039	1214.40580
1 Gbit s^{-1}	0 ms	305.12318	102.72793	1300.30141
1 Gbit s^{-1}	20 ms	336.59046	108.25358	1341.79869
1 Gbit s^{-1}	40 ms	358.84045	115.92891	1373.21379
1 Gbit s^{-1}	60 ms	423.00867	129.77403	1445.12607
1 Gbit s^{-1}	80 ms	484.82885	144.34507	1521.79973
1 Gbit s^{-1}	100 ms	549.74075	159.50490	1599.97578

scale better than [15] and we expect that we outperform [15] even beyond the point where Fig. 11 indicates that we have to communicate more than [15] (obviously, not for arbitrary values of n as our asymptotic offline performance is worse than for [15]). We did not simulate the protocols for more than 4 parties as this would exceed the resources available to us (on the machine that the benchmarks were performed on; when running all parties on the same machine).

Figures 25 and 26 give additional context for the results presented in Fig. 10. Both contain results for unconstrained network settings to judge the purely computational overhead of our protocol. Figure 26 shows runtime relative to SPDZ to make the overhead of our protocol and [15] more visible. Note that in a multi-threaded real-world application one would compute and verify commitments of all other parties in parallel and the purely computational overhead should stay closer to the overhead with only a single other party (i.e., $n = 2$). Finally, Tables XII to XV contain the data used to generate Figs. 10, 24 and 25.

APPENDIX K
AMORTIZING COMMITMENT COSTS

As done in our benchmarks (cf. Section IX and Appendix J), one can easily amortize the overhead of BDLOP commitment (i.e., computational cost and communication cost) by batch processing multiple instances of the same circuit (on different inputs). This is done in our evaluation and seems possible in the client-server setting where the evaluation of the same circuit is offered to many clients (e.g., "machine learning as a service"). If it is unreasonable for some applications to amortize N instances of a computation, there are several approaches that can be taken. Note that SPDZ or protocols using, for example, Pedersen commitments do not have to consider the problems discussed here as they use MACs and/or commitments for individual values (not as with BDLOP where there is one commitment for multiple values, one in each "slot"). In the unlikely case of not being able to perform any amortization of the commitment cost, naively using our approach would lead to an additional overhead of factor N, as most of the slots are then unused but have to be computed/verified to get the security guarantees that are backed by BDLOP commitments. Instead, we advise to use one or both of the following two optimization.

TABLE XIV
AMORTIZED OFFLINE RUNTIME (IN SECONDS) WITH VARYING NUMBER OF PARTIES

bandwidth	delay	n	ours	SPDZ	[15]
1 Gbit s^{-1}	0 ms	2	213.15502	89.17087	904.13490
1 Gbit s^{-1}	0 ms	3	305.12318	102.72793	1300.30141
1 Gbit s^{-1}	0 ms	4	450.49692	117.22009	1730.67614

A simple first improvement is to reduce N for BDLOP while keeping it untouched for BGV. The influence of changing N for BDLOP, while keeping the target (computational and statistical) security level constant, can be seen in Fig. 21. Lower values of N usually imply larger parameters for BDLOP and thus the computational cost and communication cost per slot (of the commitment scheme) increases. However, the overhead of unused slots is generally greater than the increased cost from lower values of N (for the values of N that we examined).

Another optimization is to combine multiple operations by packing techniques—widely used for FHE schemes (e.g., [7], [84], [108]). Consider the following example based on "network A" as arithmetic circuit. The second dense layer of the network has 128 inputs and 128 outputs. The computation in this layer can be performed as 128 dot products of vectors with 128 elements each. The inputs can be packed in a single commitment with at least $128 \cdot 128$ slots; the same can be done for the weights that the inputs should be multiplied with. These two commitments can then be multiplied (with Beaver multiplication in the online phase) to get all pair-wise products needed for the dot products. Now, one can use masking and rotating of slots (local operations on commitments that do not need interaction) to add up the right slots for the final computation of the dot product. Like this, one can combine many independent operations in as few commitments as possible—with only added local communication overhead and without increasing the number of communication rounds. The concrete overhead depends on the arithmetic circuit and the choice of N. Also one has to examine which types of operations are compatible as-is with BDLOP and judge their concrete influence on the noise (and therefore parameters), as well as the concrete computational complexity. Combining this technique with the previously mentioned approach, for "network A", we could lower N from 2^{15} to $2^{14} = 128 \cdot 128$ to fully utilize all slots in the mentioned dense layer. In this case, one would even reduce the cost per slot slightly (cf. Table X).

TABLE XV
Amortized Online Runtime (in Seconds) with Restarts

		BoBW$_{[15]}$ without restart	BoBW$_{[15]}$ with 1 restart			
delay	ours[a]	[15] without verif.[a]	[15] w. verif.[a]	[15] offline[b]	[15] without verif.[b]	total
0 ms	2.04747	0.39622	86.07595	904.13490	0.25188	990.46273
20 ms	2.19651	0.48117	–	963.88134	0.34963	1050.30692
40 ms	2.33357	0.56918	–	1022.92169	0.53913	1109.53677
60 ms	2.66247	0.75792	–	1094.17075	0.73880	1180.98550
80 ms	3.00954	0.95213	–	1185.46989	0.93789	1272.48373
100 ms	3.35027	1.14991	–	1272.00691	1.13415	1359.21701

[a] $n = 3$, [b] $n = 2$