Catching MPC Cheaters: Identification and Openability*

Robert Cunningham, Benjamin Fuller, and Sophia Yakoubov

MIT Lincoln Laboratory
{rkc, bfuller, sophia.yakoubov}@ll.mit.edu

Abstract. Secure multi-party computation (MPC) protocols do not completely prevent malicious parties from cheating and disrupting the computation. A coalition of malicious parties can repeatedly cause the computation to abort or provide an input that does not correspond to reality.

In this work, we augment MPC with two new properties to discourage cheating. The first of these is a strengthening of *identifiable abort* where all parties who do not follow the protocol will be identified as cheaters by each honest party. The second is *openability*, which means that if a computation output is discovered to be untrue (e.g. by a real-world event contradicting it), a distinguished coalition of parties can recover the MPC inputs.

We provide the first efficient MPC protocol achieving both of those properties. Our scheme extends the SPDZ protocol (Damgard et al., Crypto 2012). SPDZ leverages an offline (computation-independent) pre-processing phase to speed up the online computation. Our protocol is *optimistic*: it has the same communication and computation complexity in the online phase as SPDZ when no parties cheat. If cheating does occur, each honest party can additionally perform a local computation to identify all cheaters.

We achieve identifiable abort by using a new locally identifiable secret sharing scheme (as defined by Ishai, Ostrovsky, and Zikas (TCC 2012)) which we call *commitment enhanced secret sharing*, or CESS. In CESS, each SPDZ input share is augmented with a linearly homomorphic commitment. When cheating occurs, each party can use the linear homomorphism to compute a commitment to the corresponding share of the output value. Parties whose claimed output share does not match their output share commitments are identified as cheaters.

We achieve openability through the use of verifiable encryption and specialized zero-knowledge proofs. Openability relies on the availability of an auditable public transcript of the MPC execution, as introduced by Baum, Damgard, and Orlandi (SCN 2014).

1 Introduction

Secure multi-party computation (MPC) allows multiple parties to evaluate a function of their private inputs while maintaining their privacy. In this work, we focus on identifying malicious behavior that is not prevented by the guarantees of traditional MPC. We describe two such behaviors, and introduce consequences for parties that engage in them.

Completely Identifiable Abort First, in traditional MPC, while a malicious party cannot cause the computation to return an incorrect output, it *can* cheat by deviating from the protocol and causing an abort (a termination with an error). Since the cheater remains anonymous, it does

* Approved for public release: distribution unlimited. This material is based upon work supported under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Air Force.

not face any consequences for its actions. Even if the honest parties want to try and recompute the function, they are stuck because they do not know whom they need to exclude. In order to avoid such stalemates, it is desirable to be able to identify the cheaters. MPC protocols that support cheater identification are referred to as *MPC with identifiable abort* [BS90,WW95,CFN94,IOZ14], and guarantee that all honest parties agree on a subset of cheating parties.[1] We introduce MPC with *completely* identifiable abort, which guarantees that all honest parties agree on *all* cheating parties.

Openability Second, traditional MPC guarantees that each party provides a well-formed input. However, parties are free to provide any well-formed input they want. Many MPC applications require that the inputs to the computation be measurement values from the real world, but since there is no binding between each party's input and the real measurement value, malicious parties can lie about their measurements. These lies may change the output of the MPC to one that clearly does not match reality; if this occurs, it is desirable to be able to identify the party responsible. To achieve this goal, we enable the recovery of MPC inputs by an *opening coalition*. We call this *openability*.

To ensure that openability does not break the security of the underlying MPC, we need opening coalitions to contain at least one distinguished party that did not participate in the MPC. We call this distinguished party a *judge* J. In reality, this judge's role may be shared by more than one party.

1.1 Our construction

We now show how to augment MPC with completely identifiable abort and openability while maintaining efficiency. Our construction achieves security in the presence of an active adversary statically making arbitrarily many corruptions. We extend SPDZ [DPSZ12,DKL$^+$13], which is one of the fastest known MPC protocols; it leverages an inefficient offline pre-processing phase to enable quick online computation. Our protocol is *optimistic*; if cheating does not occur, the online communication and computation are only twice those of the SPDZ protocol, which uses primarily fast information-theoretic tools. In particular, the communication cost remains $O(nm)$, where n is the number of parties in the MPC and m is the number of multiplications evaluated. If cheating does occur, each party must perform an additional local computation whose complexity is $O(n|C|)$ in order to identify the cheaters, where $|C|$ is the size of the circuit being evaluated. (The additional local work uses computionally-secure tools, which are slower.) This is much more efficient than previous MPC constructions with identifiable abort, which require a significant communication overhead - at least $O(n^2|C|)$ [CFN94].

Starting Point: SPDZ SPDZ [DPSZ12,DKL$^+$13] is an MPC protocol (defined in Section 2.2) which is secure against an active and adaptive adversary making $n-1$ corruptions, where n is the number of parties. SPDZ leverages Beaver triples [Bea92], which are pre-computed during the offline phase. Each input is additively secret shared; the computation then proceeds gate by gate. Additions are computed by each party locally. Multiplications use Beaver triples (described in Section 4.2), and require two values to be reconstructed (which requires two broadcast messages from each participant). To prevent malicious parties from providing incorrect shares during these

[1] Cheater identification gained popularity in the areas of secret sharing [BS90,WW95,IOS12] and pay television [CFN94].

reconstructions, SPDZ uses a linear MAC of the form $\mathsf{MAC}(x) = \alpha x$, where α is is the MAC key which is secret shared amongst all the parties. The linear MAC shares follow the computation, and are checked only at the end of the computation to detect whether any cheating took place.

Adding Completely Identifiable Abort We add completely identifiable abort to the SPDZ protocol using a new locally-identifiable secret sharing scheme which we call *commitment-enhanced secret sharing (CESS)*. A locally-identifiable secret sharing scheme is a secret sharing scheme (defined in Section 3) where during reconstruction, all honest parties agree on the set of parties that modified their shares [IOS12]. Each CESS share contains an additive share (as in SPDZ), and additionally includes linearly homomorphic commitments to *every* additive share.[2] Each CESS share also contains the decommitment value for the commitment to the corresponding additive share. In Section 3, we show how these commitments enable local identification during reconstruction. Informally, by the binding property of the commitment scheme, no party should be able to convince any other party of the validity of an altered additive share.

In Sections 4 and 5, we show how CESS enables MPC with completely identifiable abort. Since we use linearly homomorphic commitments, if cheating occurs, each honest party can use the homomorphism of the commitment scheme to transform the input share commitments of each other party into a commitment to that party's output share. All parties whose claimed output shares do not match their output share commitments are identified as cheaters.[3]

Auditability Openability relies on a property called *auditability*, introduced by Baum, Damgård, and Orlandi [BDO14]. They build on top of the SPDZ protocol by adding a public transcript τ (modeled as a public append-only bulletin board) to allow for public auditing of the correctness of a protocol execution even in the case when all parties are malicious. The protocol assumes a single public output out that is used with the transcript τ to check if the protocol executed correctly. The public transcript τ contains Pedersen commitments [Ped92] (described in Appendix C.1) to each precomputed Beaver triple value and input in_i. (We emphasize that our construction includes commitments for each input *share*, while the construction of Baum et. al [BDO14] includes a single commitment for each *input*. This does increase the number of committed values, but does not affect the online communication complexity at all.)

τ also contains all values reconstructed during the computation; namely the Beaver triple differences, described in Section 4.2. Though τ does not contain Pedersen commitments to intermediate computation values or to the output, these commitments can be computed using the linear homomorphism of Pedersen commitments and the posted Beaver triple differences. So, an auditor holding a transcript τ and the evaluation circuit C can derive a Pedersen commitment $\mathsf{c}_{\mathsf{out}}$ to the correct computation output out. The auditor can then check that $\mathsf{c}_{\mathsf{out}}$ is indeed a commitment to the claimed output out'.

We leverage auditability when we add openability: we only require the inputs to be recoverable if an audit of the transcript succeeds. Without an audit check to ensure that the transcript is well-formed, openability is unachievable, as the parties would then be able to omit crucial information from the transcript.

[2] We use Pedersen commitments to enable efficient zero-knowledge proofs.
[3] An alternative approach uses bitcoin to introduce financial repercussions for cheating [KB14,ADMM16].

Adding Openability An openable MPC protocol allows a distinguished *opening coalition* to recover the computation inputs. We add openability by adding four tasks for every party providing an input:

1. additively secret sharing the input for the opening coalition,
2. computing commitments to each additive share,
3. computing verifiable encryptions [FO97,Bou00,DF02,CS03] to each additive share and the corresponding decommitment value, and
4. computing non-interactive zero-knowledge proofs (NIZKPs) of the validity and consistency of these commitments and encryptions.

Each verifiable encryption will use the public key of the relevant coalition party. Pedersen commitments and our verifiable encryption scheme (a variant of the one described by Camenisch and Shoup [CS03]) lend themselves to particularly efficient zero-knowledge proofs. We detail our openable MPC construction in Section 6.

Concurrent Work The recently posted work of Baum et. al [BOS16] augments SPDZ with identifiable abort using an information-theoretic signature scheme. In terms of efficiency, their scheme requires $n + 2nm + O(n^2)$ broadcast messages during the online phase, where n is the number of parties involved in the computation, and m is the number of multiplications evaluated. In contrast, our online phase requires $n + 2nm + O(n)$ broadcast messages. However, in the event of cheating, our scheme relies on computational techniques to identify the misbehaving party, while the scheme of Baum et. al uses information-theoretic techniques.[4] They also do not consider openability.

1.2 A Motivating Example

In this section we present satellite conjunction analysis [HWB14] as a motivating use case for our augmented MPC.[5] Those readers who are convinced of the need to catch MPC cheaters may proceed to Section 1.3.

Multiple government organizations and companies own satellites. The purpose of many satellites is secret, so organizations are not willing to share their positions. However, there is risk to not sharing position information. The active Iridium 33 satellite collided with the inactive Cosmos 2251 satellite in 2009 [Jak10] creating significant debris which endangered other satellites [Wri09]. To avoid such catastrophes, the organizations want to jointly compute whether collisions will occur without revealing satellite positions. As a result of the computation, parties should learn only whether a collision will take place, and who the involved organizations are. Hemenway et al. observed that MPC enables such a joint and private computation [HWB14,HLOW16].

In traditional MPC protocols malicious parties cannot affect the output of the computation (other than by changing their inputs). However, malicious parties can cause the computation to abort - to terminate with an error - without ever being identified as the culprit. Imagine that some malicious organization wants to cause a satellite collision. All it would have to do is aim its satellite at another, prevent the MPC from completing every time it is run, sit back and wait! Because no

[4] We use the Pedersen commitment scheme, which is information-theoretically hiding but only computationally binding. So, computational assumptions are only necessary for the correctness of cheater identification.

[5] Other sensitive applications include economic markets [BCD+09] and elections [BDO14].

culprit in an abort can be identified, the malicious organization would not be caught until it is too late. In order to avoid this, we augment MPC with completely identifiable abort.

Satellites generally reside in one of three bands: low-earth orbit (LEO), medium-earth orbit (MEO), or geonsynchronous orbit (GEO). Collisions between functioning objects at different levels are unlikely; however, if a collision occurs at one level, the resulting debris may collide with objects at other levels. Suppose that the above satellite collision computation is performed by all organizations with objects in medium earth orbit. Organizations with satellites in low earth orbit are also affected by the results of the computation, even though they don't participate, since a collision in medium earth orbit could cause debris to fly into low earth orbit, potentially damaging the satellites there. For convenience we will refer to one of the organizations owning satellites in low earth orbit as Leo. Leo wants to be able to determine whether the medium earth orbit computation was performed correctly even if *all* of the organizations involved in it might have malicious intentions, so as to determine the risk to his own satellites. Given a transcript τ of the MPC, any external organization such as Leo should be able to *audit* the correctness of the computation, as described by Baum et. al [BDO14].

Now, imagine that Leo performed the audit, and determined that the MPC was performed correctly. However, the next day, a collision occurs and debris destroys one of Leo's satellites! This could only have occurred if one of the organizations participating in the MPC provided incorrect inputs to the computation. In such a situation, it would be crucial to be able to determine who is responsible. We achieve this property by adding openability.

1.3 Organization

The rest of the paper is organized as follows. In Section 2 we describe our augmented MPC definitions. In Section 3 we introduce *commitment-enhanced secret sharing*, which is crucial for our construction. In Section 4, we describe how commitment-enhanced secret sharing can be used in MPC. Section 5 adds completely identifiable abort to SPDZ; finally, Section 6 adds openability.

In Appendix A, we present the preprocessing phase of our scheme. In Appendix B we prove security. In Appendix C, we describe the building blocks we use in our construction: commitments, verifiable encryption and zero-knowledge proofs.

2 Definitions

2.1 Preliminaries

Notation Throughout this work we implicitly consider a sequence of protocols parameterized by a security parameter k. For notational clarity we usually omit k (except in the cryptographic building block descriptions in Appendix C), but it is implied that all algorithms take k as input.

All of our MPC protocols consider arithmetic circuits over p-order fields, where p is a large Sophie-Germain prime (that is, $q = 2p + 1$ is also a prime). $\mathbb{Z}_p$ refers to the field $\{0, \ldots, p-1\}$; $\mathbb{Z}_p^*$ refers to $\mathbb{Z}_p \backslash \{0\} = \{1, ..., p-1\}$. $\mathbb{QR}_p$ refers to $\{x^2 \bmod p : x \in \mathbb{Z}_p^*\}$ (quadratic residues modulo p). An element g of a group G is a generator of that group if $\forall x \in G, \exists a$ such that $g^a = x$.

Model We implicitly assume two available functionalities: a broadcast channel, and an append-only bulletin board. Like Damgård et. al [DPSZ12,DKL$^+$13], we assume the availability of a broadcast channel for the same cost as a point-to-point communication. See Appendix A.3 of Damgård et. al for a discussion on implementing broadcast channels.

There are several ways to implement an append-only public bulletin board. One simple way is using a public server against which privacy is desirable (so, this server cannot simply be used to perform the computation in question), but which is trusted to behave semi-honestly. Another way, which does not require trust in an additional third party, is using a blockchain (but without necessarily using proofs of work which rely on an honest majority). Put very simply, every post p to the bulletin board is broadcast together with a signature σ by the posting party on p and a hash of the previous post (or posts, if there were simultaneous posts broadcast). The use of the public bulletin board in our protocol is unusual in that it is public knowledge who needs to be providing a post at which point in the protocol. Thus, omitting a post contributed by a party would not result in a valid bulletin board transcript. Chaining the posts together by signing the posts together with hashes of previous posts ensures that parties' posts cannot be replayed from protocol execution to protocol execution.

2.2 Multi-Party Computation (MPC)

Consider n parties $(\mathsf{P}_1, \ldots, \mathsf{P}_n)$ each of whom has a secret input $(\mathsf{in}_1, \ldots, \mathsf{in}_n)$. *Secure Multi-Party Computation* (MPC) allows them to compute a joint function $C(\mathsf{in}_1, \ldots, \mathsf{in}_n) = \mathsf{out}$ on their values, where C is a circuit representing the function. As a result of this computation, all of the parties learn the output out, but no party learns anything else about others' inputs.

This privacy guarantee should hold even if some parties are adversarially controlled, meaning that they are trying to learn something about other parties' inputs. Different MPC protocols maintain their security in the presence of different numbers and types of adversarially controlled parties. In this paper, we consider security in the presence of arbitrarily many adversarial parties, chosen statically (meaning that the adversarial parties are fixed before the protocol begins, but it could be that all parties participating in the protocol are adversarial). Adversarial parties run in probabilistic polynomial time and can act maliciously, meaning that they can deviate arbitrarily from the specified protocol.

The security requirement of MPC is formally defined with respect to an *ideal functionality*, wherein a trusted third party receives inputs from everyone, performs the computation internally, and then distributes the output. When interacting with this ideal functionality, no party learns more than their own input and the output, since those are the only values it sees. For an MPC protocol to be secure, there must exist an efficient simulator that, given the view of all adversarial parties in an ideal execution (meaning their input and the output), can produce a view that is indistinguishable from a real protocol execution view.[6]

Intuitively, the two most important properties of an MPC protocol (both implied by this definition) are *correctness* and *privacy*. Informally, an MPC protocol π satisfies *correctness* if for all inputs $(\mathsf{in}_1, \ldots, \mathsf{in}_n)$ and circuits C where $C(\mathsf{in}_1, \ldots, \mathsf{in}_n) = \mathsf{out}$, the protocol π returns out when evaluating C on inputs $\mathsf{in}_1, \ldots, \mathsf{in}_n$. An MPC protocol π satisfies *privacy* if no party P_i can learn anything about the inputs of any other party, other than what is revealed by out.

Another desirable property is *fairness*; fairness means that if one party learns the output, so do all parties. In the setting where the majority of parties may be adversarial, fairness is known to be unachievable [Cle86]. So, we instead consider *security with abort*, a weaker notion of security that allows an adversary to violate fairness by causing an abort. The ideal functionality for secure MPC with abort is shown in Figure 1.

[6] We call the list of protocol messages the *view* of the protocol. We use the word transcript or τ to refer to the public information used for auditing (following the notation of [BDO14]).

Functionality $\mathcal{F}$

INIT: On input (INIT, C, p) from all parties (where C is a circuit with n inputs and 1 output, consisting of addition and multiplication gates over $\mathbb{Z}_p$):
1. Store C and p.
2. Wait for $\mathcal{A}$ to provide the set $\mathcal{I}$ of adversarially controlled parties.

INPUT: On input $(\text{INPUT}, \mathsf{P}_i, \mathsf{in}_i)$ from party P_i and $(\text{INPUT}, \mathsf{P}_i)$ from all parties P_j, $j \neq i$:
Store $(\text{INPUT}, \mathsf{P}_i, \mathsf{in}_i)$.

EVAL: On input (EVAL) from all parties:
1. If not all inputs values have been provided, output `REJECT`.
2. Evaluate the circuit C on inputs $(\mathsf{in}_1, \ldots, \mathsf{in}_n)$. When the evaluation is completed, store the resulting value out as $(\text{OUTPUT}, \mathsf{out})$.

OUTPUT: On input (OUTPUT) from all parties:
1. Retrieve $(\text{OUTPUT}, \mathsf{out})$.
2. Send out to all adversarially controlled parties $\mathsf{P} \in \mathcal{I}$; wait for a response.

3a. If all adversarially controlled parties respond `ACCEPT`, output out to all parties.

3b. Otherwise, output $\perp$ to all parties.

Fig. 1. Ideal Functionality for MPC.

2.3 MPC with Identifiable Abort

The ideal MPC functionality given in Figure 1 implies that if any malicious parties attempt to cause the computation to return anything other than the correct output, the protocol aborts (returns $\perp$). The honest parties are left knowing something went wrong - however, they do not learn *what* went wrong, or which of the other parties are to blame. *MPC with identifiable abort*, defined by Ishai, Ostrovsky and Zikas [IOZ14], ensures that when an abort occurs all the honest parties agree on the identity of *at least one* malicious party P_i. We extend the definition of Ishai et. al [IOZ14], defining MPC with *completely* identifiable abort as MPC which ensures that when an abort occurs all honest parties agree on the identities of *all* parties who deviated from the protocol. More formally, Figure 2 describes the ideal functionality $\mathcal{F}_{\texttt{CIDA}}$ for MPC with completely identifiable abort. $\mathcal{F}_{\texttt{CIDA}}$ is simply $\mathcal{F}_{\texttt{CIDA},\texttt{AUDIT},\texttt{OPEN}}$ without the AUDIT and OPEN commands.

2.4 Auditability

Any MPC which supports arbitrarily many adversarially controlled parties enables all honest parties to determine whether the protocol was executed correctly. It is also useful to allow any third party to inspect some evidence of the computation and arrive at the same conclusion. Baum, Damgard and Orlandi [BDO14] introduce *auditability* to MPC; they describe a protocol where, given the circuit C being evaluated, a presumed output out' and a public transcript τ updated throughout the computation, any third party can audit the computation and ascertain that it was performed correctly with output out'. In this setting, we model the MPC as also outputing τ.

More formally, Baum et. al introduce the VERIFY algorithm. VERIFY takes in the public transcript τ which is created during computation, the circuit C which was evaluated, and the computation output out, and returns a 0 or a 1, depending on whether the computation was correct.

Definition 1 (Auditable Correctness, from Baum et. al [BDO14]). *We say that an MPC protocol satisfies* Auditable Correctness *if for all circuits C and for all potential outputs* out*,*

- $\textsc{Verify}(\tau, C, \mathsf{out}) = 1$ *with overwhelming probability if for some inputs* $\mathsf{in}_1, \ldots, \mathsf{in}_n$*,* $C(\mathsf{in}_1, \ldots, \mathsf{in}_n) = \mathsf{out}$ *and τ is a transcript of the MPC evaluation of C on* $\mathsf{in}_1, \ldots, \mathsf{in}_n$*, and*
- $\textsc{Verify}(\tau, C, \mathsf{out}) = 0$ *with overwhelming probability if for all inputs* $\mathsf{in}_1, \ldots, \mathsf{in}_n$*,* $C(\mathsf{in}_1, \ldots, \mathsf{in}_n) \neq \mathsf{out}$ *or τ is not a valid transcript of an MPC evaluation of C on inputs* $\mathsf{in}_1, \ldots, \mathsf{in}_n$*.*

While auditability makes the computation execution more transparent, it does not provide any check on the veracity of the computation inputs. As motivated in the introduction, a correct computation on false inputs can be catastrophic. To address this issue, we define *openability* in Section 2.5.

2.5 Openability

In extreme cases, it may be necessary to open the inputs of an MPC evaluation. Using the example from Section 1.2, let's say a satellite collision occurred despite the fact that an MPC determined that there was no risk of collision. Some party must have lied about their satellite trajectory, but followed the protocol while executing the MPC. It would be useful to be able to recover the inputs to the protocol and identify the lying party (or parties).

Of course, inputs should not be recoverable by any one party; this would violate the privacy guarantees of MPC. However, we can define *allowable coalitions*, or *groups* of parties who we trust not to abuse this privilege. In this context, one might want several additional players that we will call judges $\{\mathsf{J}_i\}$. A judge J_i notionally has the power to determine that an opening is justified. We include multiple judges to compensate in case some of the parties who participated in the MPC do not cooperate. This is something we need to account for, since if party P_i knows that it will be identified as a liar, it will not cooperate with an input opening. Allowable opening coalitions might be all the parties from the MPC together with any judge party ($\{\mathsf{P}_1, \ldots, \mathsf{P}_n, \mathsf{J}_i\}$), or some t of the parties together with two judges ($\{\mathsf{P}_{i_1}, \ldots, \mathsf{P}_{i_t}, \mathsf{J}_i, \mathsf{J}_j\}$).

More formally, we introduce the protocol $\textsc{Open}$ executed jointly by an allowable opening coalition. $\textsc{Open}$ takes in a transcript τ, and returns $(\mathsf{in}_1, \ldots, \mathsf{in}_n)$. We require that the $\textsc{Open}$ protocol be *sound*, as described in Definition 2. Notice that the transcript τ also needs to be *hiding*, meaning that it shouldn't reveal any information about the values being computed on. However, this property is implied by the privacy requirement of MPC, and does not need to be explicitly restated.

Definition 2 (Opening Soundness). *We say that an MPC protocol satisfies* Opening Soundness *if for all circuits C and for all inputs* $\mathsf{in}_1, \ldots, \mathsf{in}_n$*, for all MPC evaluations of C on* $\mathsf{in}_1, \ldots, \mathsf{in}_n$ *resulting in output* out *and transcript τ (where all participants may be malicious), the probability that* $\textsc{Verify}(\tau, C, \mathsf{out}) = 1$ *and* $\textsc{Open}(\tau) \neq (\mathsf{in}_1, \ldots, \mathsf{in}_n)$ *is negligible.*

Figure 2 describes the ideal functionality $\mathcal{F}_{\texttt{CIDA,AUDIT,OPEN}}$ of such a protocol. For $\textsc{Open}$ to work for only allowable coalitions, such coalitions (and their associated cryptographic identity) must be known when $\textsc{Eval}$ is executed.

3 Commitment-Enhanced Secret Sharing

This section describes a new locally identifiable secret sharing scheme which we call *commitment-enhanced secret sharing (CESS)*. CESS is our main building block for MPC with completely identifiable abort, as described in Section 5. We first describe basic secret sharing (Section 3.1), and then

Functionality $\mathcal{F}_{\texttt{CIDA,AUDIT,OPEN}}$

INIT: as in $\mathcal{F}$. Additionally, receive and record the set of allowable coalitions $\{\mathcal{C}_i\}$ from all parties.
INPUT: as in $\mathcal{F}$.
EVAL: as in $\mathcal{F}$.
OUTPUT: On input (OUTPUT) from all parties:
1. Retrieve (OUTPUT, out).
2. Send out to all adversarially controlled parties $\mathsf{P} \in \mathcal{I}$; wait for a response.

3a. If all adversarially controlled parties respond `ACCEPT`, output out to all parties.
3b. Otherwise, create a list L_{cheat} of cheating parties. For each party P_i that responds something other than `ACCEPT`, add P_i to L_{cheat}. Output $(\perp, L_{cheat})$ to all parties (revealing to them the parties responsible for the computation failure).

AUDIT: On input (AUDIT, out′) from a third party auditor:
if EVAL was not executed OR out $= \perp$, then output `NO AUDIT POSSIBLE`.
if out′ = out then output `ACCEPT`.
else output `REJECT`.
OPEN: On input (OPEN) from all of the parties in an allowable coalition (some $\mathcal{C} \in \{\mathcal{C}_i\}$):
if EVAL was executed AND out $\neq \perp$: **return** $\mathsf{in}_1, \ldots, \mathsf{in}_n$.
else: return $\perp$.

Fig. 2. Ideal Functionality for Openable and Auditable MPC with Completely Identifiable Abort

describe locally identifiable secret sharing (LISS, Section 3.2) before proceeding to describe CESS (Section 3.3).

3.1 Secret Sharing

Secret sharing was introduced by Shamir [Sha79]. A t-out-of-n sharing of a secret x is an encoding of the secret into n pieces, or *shares*, such that any t shares together can be used to reconstruct the secret x, but fewer than t shares give no information at all about x. A secret sharing scheme consists of two algorithms: SHARE and REC.

- $\text{SHARE}(x) \to (s_1, \ldots, s_n)$ takes in a secret x and produces the n secret shares.
- $\text{REC}(s_{i_1}, \ldots, s_{i_t}) \to \tilde{x}$ takes in t secret shares and returns the reconstructed secret $\tilde{x}$.

For n-out-of-n secret sharing, a simple scheme called additive secret sharing ($\mathsf{SS}_{\mathsf{Add}}$) can be used. $\mathsf{SS}_{\mathsf{Add}}$.SHARE$(x)$ generates $n-1$ random elements $s_1, \ldots, s_{n-1}$ in some additive group, and computes the nth share as $s_n = x - (s_1 + \cdots + s_{n-1})$. Any $n-1$ shares appear completely random; however, the sum of all n shares gives the secret x. Additive secret sharing has some linear properties: a shared value x can be multiplied by a constant, or added to another shared value x', by separately operating on the individual shares. We use the notation $[x]_{\mathsf{P}_j}$ to denote the additive secret share of element x belonging to party P_j.

Shamir t-out-of-n secret sharing ($\mathsf{SS}_{\mathsf{Shamir}}$) uses degree-$(t-1)$ polynomials over some field. $\mathsf{SS}_{\mathsf{Shamir}}$.SHARE$(x)$ generates a random degree-$(t-1)$ polynomial f with x as its y-intercept; each share s_i is a point $(x_i, f(x_i))$ on the polynomial (with $x_i \neq 0$). For simplicity, we fix $x_i = i$. Any

t shares can be used to interpolate the polynomial, reconstructing x. Any fewer than t shares give no information about x.

Looking ahead, our MPC protocols are presented using additive secret sharing, but can be trivially extended to use Shamir secret sharing if a t-out-of-n sharing (for some $t < n - 1$) is desired.

3.2 Locally-Identifiable Secret Sharing (LISS)

Secret sharing provides confidentiality. However, there are no guarantees that the reconstruction protocol Rec returns the correct secret in the presence of malicious parties. Robust secret sharing guarantees reconstruction correctness in the presence of active adversaries [TW86].[7] It is also useful to identify the parties that provided incorrect shares; this is known as an identifiable secret sharing [KOO95]. Identifiable secret sharing becomes impossible when a majority of parties are adversarial [IOS12, Theorem 3]. However, a slightly weaker task is possible in the presence of an adversarial majority: *honest parties* can agree on the set of parties who provided incorrect shares, but cannot prove it to a third party who did not hold one of the shares. This is known as *locally identifiable secret sharing (LISS)*. We modify the inputs to the reconstruction algorithm Rec of a LISS to also include the index i of the party performing the reconstruction; if that party P_i is honest, it has the additional knowledge that the share s_i has not been tampered with. Definition 3 is taken from [IOS12, Definition 4].

Definition 3 (Locally-Identifiable Secret Sharing). *An n-out-of-n secret sharing scheme is locally identifiable if it satisfies two requirements:* unanimity*, meaning that all honest parties should agree on either a correct reconstruction or on the correct set of cheating parties, and* predictable failure*, meaning that the output of the reconstruct algorithm should be simulatable if it does not return the correct secret. Predictable failure ensures that the output of the reconstruction algorithm does not reveal anything about the secret, unless it correctly returns the secret. We give more rigorous descriptions of unanimity and predictable failure below.*

Unanimity *For any probabilistic polynomial time adversary $\mathcal{A}$ and for any secret x, the probability of $\mathcal{A}$'s success in the following game is negligible:*

1. $(s_1, \ldots, s_n) \leftarrow \textsc{Share}(x)$.
2. *$\mathcal{A}$ outputs a set $\mathcal{I} \subset \{1, \ldots, n\}$ of adversarial party indices. Let $H = \{1, \ldots, n\} \setminus \mathcal{I}$ be the set of honest party indices.*
3. *$\mathcal{A}$ receives s_i for $i \in \mathcal{I}$.*
4. *$\mathcal{A}$ selects some $B \subseteq \mathcal{I}$, and outputs s'_i for $i \in B$, where $s'_i \neq s_i$.*
5. *Let $\tilde{x}_i$ be the value reconstructed by each party P_i, for $i \in H$,* with the assumption that s_i is correct*. That is, each party P_i runs $\tilde{x}_i \leftarrow \textsc{Rec}(i, t_1, \ldots, t_n)$ (where $t_j = s'_j$ if $j \in B$ and $t_j = s_j$ otherwise).*

The adversary $\mathcal{A}$ succeeds unless:

1. *All honest parties reconstruct the correct secret ($\tilde{x}_i = x$ for all $i \in H$), or*
2. *All honest parties agree on the set of cheating players ($\tilde{x}_i = (\texttt{REJECT}, L_{cheat} = B)$ for all $i \in H$).*

[7] Robust secret sharing does not require security in the presence of a malicious dealer. This is in contrast to verifiable secret sharing [RBO89]. Looking ahead, the reason we do not require security against a malicious dealer is that dealing is done via MPC in the preprocessing phase.

Predictable Failure *There exists an algorithm* SimRec *such that for any probabilistic polynomial time adversary $\mathcal{A}$ and for any secret x, the probability of $\mathcal{A}$'s success in the following game is negligible:*

1. $(s_1, \ldots, s_n) \leftarrow \textsc{Share}(x)$.
2. *$\mathcal{A}$ outputs a set $\mathcal{I} \subset \{1, \ldots, n\}$ of adversarial party indices. Let $H = \{1, \ldots, n\} \setminus \mathcal{I}$ be the set of honest party indices.*
3. *$\mathcal{A}$ receives s_i for $i \in \mathcal{I}$.*
4. *$\mathcal{A}$ selects some $B \subseteq \mathcal{I}$, and outputs s'_i for $i \in B$, where $s'_i \neq s_i$.*
5. $\mathsf{simout} \leftarrow \textsc{SimRec}(\mathcal{I}, B, \{s_i\}_{i\in\mathcal{I}}, \{s'_i\}_{i\in B})$.
6. $\tilde{x}_i \leftarrow \textsc{Rec}(i, t_1, \ldots, t_n)$ *for* $i \in \{1, \ldots, n\}$, *where* $t_j = s'_j$ *if* $j \in B$ *and* $t_j = s_j$ *otherwise.*

The adversary $\mathcal{A}$ succeeds unless:

1. $\mathsf{simout} = \mathsf{success}$ *and* $\tilde{x}_i = x$ *for all* $i \in \{1, \ldots, n\}$, *or*
2. $\mathsf{simout} = \{\tilde{x}_i\}_{i\in\mathcal{I}}$.

3.3 Our LISS Construction

In order to support cheater identification, we introduce the *commitment-enhanced secret sharing (CESS)* scheme. A CESS of a secret x is based on an additive secret sharing of x. The ith CESS share additionally includes a Pedersen commitment (described in Appendix C.1) to *each* additive share, as well as the decommitment value for the ith commitment. The decommitment values contained in the CESS shares can be viewed as an additive secret sharing of one global decommitment value r_x. The product of the commitments will itself be a valid commitment c_x to the secret x, and the sum of the individual decommitments will be the corresponding decommitment value. We use the following notation to denote a CESS share of x belonging to party P_i:

$$\langle x \rangle_{\mathsf{P}_i} \stackrel{def}{=} ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n})),$$

where

- $\mathsf{c}_{x,i}$ is the Pedersen commitment $\mathsf{pc}([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i})$ to value $[x]_{\mathsf{P}_i}$ with decommitment value $[r_x]_{\mathsf{P}_i}$ (as described in Appendix C.1),
- $[x]_{\mathsf{P}_i}$ is the additive secret share of x belonging to P_i (as described in Section 3.1), and
- $[r_x]_{\mathsf{P}_i}$ is the decommitment value for $\mathsf{c}_{x,i}$ (equivalently, the additive secret share of the decommitment value r_x for c_x) belonging to P_i.

We informally refer to a CESS share as an $\langle\rangle$-share.

Notice that each $\langle\rangle$-share contains $O(n)$ elements, which makes it large and unwieldy. However, the commitments, which make up the bulk of the $\langle\rangle$-share, do not ever need to be communicated in order to execute reconstruction $\mathsf{CESS}.\textsc{Rec}$, since they are replicated in every share. The reconstruction algorithm $\mathsf{CESS}.\textsc{Rec}$ only receives the additive secret shares, together with one party's local copy of the commitment values $(\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n})$. $\mathsf{CESS}.\textsc{Rec}$ is described in Figure 3.

Theorem 1. *The CESS cheme is a locally identifiable secret sharing scheme (LISS), as described in Definition 3.*

Protocol CESS.REC for the Reconstruction of a CESS $\langle\rangle$-Sharing

PRECONDITIONS: Each party P_i has $\langle x\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}))$.
$\mathsf{CESS}.\text{REC}(i, ([x]_{\mathsf{P}_1}, [r_x]_{\mathsf{P}_1}), \ldots, ([x]_{\mathsf{P}_n}, [r_x]_{\mathsf{P}_n}), (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}))$:
1. Let $L_{cheat} = []$.
2. For $j \in \{1, \ldots, n\}, j \neq i$: check that $\mathsf{pc}([x]_{\mathsf{P}_j}, [r_x]_{\mathsf{P}_j}) = \mathsf{c}_{x,j}$. If this does not hold, add P_j to L_{cheat}.
3. If L_{cheat} is empty: return $x = \sum_{j\in\{1,\ldots,n\}} [x]_{\mathsf{P}_j}$.
4. Else: return $(\texttt{REJECT}, L_{cheat})$.

Fig. 3. Protocol REC for the Reconstruction of a $\langle\rangle$-Sharing

Proof. The CESS scheme achieves unanimity. In order to succeed, the adversary $\mathcal{A}$ would have to provide an incorrect additive share of the secret $[x]'_{\mathsf{P}_i}$ or an incorrect additive share of the decommitment value $[r_x]'_{\mathsf{P}_i}$ for every corrupt party that tampers with their share (P_i, $i \in B$). (Notice that we do not consider the commitments to be a tamperable part of the sharing, since they are never communicated.) In order to avoid having honest parties add P_i to the list of cheaters L_{cheat}, the adversary must supply $[x]'_{\mathsf{P}_i}$ and $[r_x]'_{\mathsf{P}_i}$ such that

$$\mathsf{pc}([x]'_{\mathsf{P}_i}, [r_x]'_{\mathsf{P}_i}) = \mathsf{c}_{x,i} = \mathsf{pc}([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}),$$

which violates the binding property of the Pedersen commitment scheme.

The CESS scheme also achieves predictable failure. The reconstruction simulator SIMREC simply checks the decommitments provided by all of the adversarial parties. If P_i's decommitment does not verify, SIMREC adds P_i to L_{cheat}; it then returns $(\texttt{REJECT}, L_{cheat})$. If all of the decommitments do verify (meaning that none of the shares could have been altered), SIMREC returns success.

4 Adapting Commitment-Enhanced Secret Sharing for use with SPDZ

The CESS scheme as described in Section 3 isn't quite ready to be used in MPC. Firstly, the CESS reconstruction algorithm REC requires each party to compute n commitments to assemble the list of cheaters L_{cheat}, whether cheating occurred at all or not. This is inefficient, and we remedy it in Section 4.1. Secondly, we need to homomorphically compute on CESS shares. We address this in Section 4.2.

4.1 Augmenting CESS with MACs

It would be nice for each party to be able to begin reconstruction by performing an efficient check to determine if cheating occurred, and only proceed with the expensive computation of L_{cheat} when cheating is detected. We can employ the linear MACs from Damgård et. al [DPSZ12] to detect cheating. The linear MACs consist of $\mathsf{MAC}(x) = \alpha x$, where α is an additively secret-shared MAC key. $\mathsf{MAC}(x)$ is then itself additively secret-shared. MACs can be checked without reconstructing the MAC key α, as described in Figure 4.

We use the following notation to denote a MAC-augmented CESS ($\mathsf{CESS}_{\mathsf{MAC}}$) share of x belonging to party P_i:

$$\langle\langle x\rangle\rangle_{\mathsf{P}_i} \stackrel{def}{=} ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}), [\mathsf{MAC}(x)]_{\mathsf{P}_i}),$$

Protocol MACCHECK for Checking a MAC Without Reconstructing the MAC Key
(From Damgård et. al [DKL+13], Figure 10)

PRECONDITIONS: Each party P_i has an additive share $[\alpha]_{\mathsf{P}_i}$ of the MAC key α. All parties have an opened value x, and additionally each party P_i has an additive share $[\mathsf{MAC}(x)]_{\mathsf{P}_i}$ of the MAC value $\mathsf{MAC}(x) = \alpha x$.

MACCHECK:
1. The parties jointly pick a random value $r \in \mathbb{Z}_p$, as described in Figure 4 of Baum et. al.
2. Each party computes the public value $a = rx$.
3. Each party P_i computes $b_i = r[\mathsf{MAC}(x)]_{\mathsf{P}_i}$, and $f_i = b_i - [\alpha]_{\mathsf{P}_i} a$ (which is equal to $r[\mathsf{MAC}(x)]_{\mathsf{P}_i} - [\alpha]_{\mathsf{P}_i} rx$).
4. Each party P_i computes a commitment c_i to f_i and publishes it.
5. Each party P_i, upon the receipt of all other parties' commitments, publishes the decommitment value d_i.
6. Each party P_i, upon the receipt of all other parties' decommitments, checks the decommitments. If any of them are invalid, P_i compiles a list L_{cheat} of parties that gave invalid decommitments, and returns $(\texttt{REJECT}, L_{cheat})$.
7. If $\sum_{i \in \{1,\ldots,n\}} f_i \neq 0$, the parties return REJECT.

Note that such MAC checks can also be *batched* for m values $x_1, \ldots, x_m$, by choosing m random values $r_1, \ldots, r_m$, setting $a = \sum_{j \in \{1,\ldots,m\}} r_j x_j$, and setting $b_i = \sum_{j \in \{1,\ldots,m\}} r_j [\mathsf{MAC}(x_j)]_{\mathsf{P}_i}$.

Fig. 4. Protocol MACCHECK for Checking a MAC Without Reconstructing the MAC Key

where $\mathsf{c}_{x,i} = \mathsf{pc}([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i})$, all of $(\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n})$ is public, and each party P_i is separately assumed to hold an additive share of the secret MAC key α. The reconstruction algorithm $\mathsf{CESS}_{\mathsf{MAC}}$.REC, executed interactively by the parties, is shown in Figure 5.

Protocol $\mathsf{CESS}_{\mathsf{MAC}}$.REC for the Reconstruction of a MAC-augmented CESS $\langle\langle\rangle\rangle$-Sharing

PRECONDITIONS: Each party P_i has $\langle\langle x \rangle\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}), [\mathsf{MAC}(x)]_{\mathsf{P}_i})$.

$\mathsf{CESS}_{\mathsf{MAC}}$.REC:
1. Each party P_i broadcasts $[x]_{\mathsf{P}_i}$ and $[r_x]_{\mathsf{P}_i}$, and computes $x = \sum_{j \in \{1,\ldots,n\}} [x]_{\mathsf{P}_j}$.
2. All parties execute MACCHECK, described in Figure 4, to check the MAC without reconstructing the MAC itself or the MAC key α.
3. If the MAC verifies: return x.
4. Otherwise: all parties P_i execute the CESS reconstruction protocol $\mathsf{CESS}.\mathrm{REC}(i, ([x]_{\mathsf{P}_1}, [r_x]_{\mathsf{P}_1}), \ldots, ([x]_{\mathsf{P}_n}, [r_x]_{\mathsf{P}_n}), (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}))$ as described in Figure 3.

Fig. 5. Protocol REC for the Reconstruction of a $\langle\langle\rangle\rangle$-Sharing

This remains a locally identifiable secret sharing scheme, because a cheating party would have to cause the MAC to verify in order to avoid detection, which they can only do with negligible probability, as shown by Damgård et. al [DKL+13].

In later sections, we will describe how multiple $\langle\langle\rangle\rangle$-sharings are dealt with throughout our MPC protocol. If it is desired, steps 2 through 4 in Figure 5 can be postponed, and then performed in batch form. If the MAC verifies, each party's MAC check communication overhead is independent of the number of sharings being verified.

4.2 Computing on Commitment-Enhanced Secret Shares

Finally, in order to use MAC-augmented CESS ($\mathsf{CESS}_{\mathsf{MAC}}$) in MPC, we need to describe how to compute on shares. Once we can compute $\mathsf{CESS}_{\mathsf{MAC}}$ shares, the locally identifiable property of $\mathsf{CESS}_{\mathsf{MAC}}$ will be used to provide completely identifiable abort.

Linear Computations Linear computations on $\mathsf{CESS}_{\mathsf{MAC}}$ shares can be performed locally, as shown below, since both additive shares and Pedersen commitments are linearly homomorphic.

- To add a constant ϵ to $\langle\langle x\rangle\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}), [\mathsf{MAC}(x)]_{\mathsf{P}_i})$, first compute $[x+\epsilon]_{\mathsf{P}_i}$ as
$$[x+\epsilon]_{\mathsf{P}_1} = [x]_{\mathsf{P}_1} + \epsilon \text{ and } [x+\epsilon]_{\mathsf{P}_i} = [x]_{\mathsf{P}_i} \text{ for } i \neq 1.$$
Then compute
$$\langle\langle x+\epsilon\rangle\rangle_{\mathsf{P}_i} = ([x+\epsilon]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}\mathsf{pc}(\epsilon, 0), \mathsf{c}_{x,2}, \ldots, \mathsf{c}_{x,n})), [\mathsf{MAC}(x)]_{\mathsf{P}_i} + \epsilon[\alpha]_{\mathsf{P}_i}.$$
- To add $\langle\langle x\rangle\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}), [\mathsf{MAC}(x)]_{\mathsf{P}_i})$ and $\langle\langle y\rangle\rangle_{\mathsf{P}_i} = ([y]_{\mathsf{P}_i}, [r_y]_{\mathsf{P}_i}, (\mathsf{c}_{y,1}, \ldots, \mathsf{c}_{y,n}), [\mathsf{MAC}(y)]_{\mathsf{P}_i})$, compute
$$\langle\langle x+y\rangle\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i} + [y]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i} + [r_y]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}\mathsf{c}_{y,1}, \ldots, \mathsf{c}_{x,n}\mathsf{c}_{y,n}), [\mathsf{MAC}(x)]_{\mathsf{P}_i} + [\mathsf{MAC}(y)]_{\mathsf{P}_i}).$$
- To multiply $\langle\langle x\rangle\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}), [\mathsf{MAC}(x)]_{\mathsf{P}_i})$ by a constant ϵ, compute
$$\langle\langle \epsilon x\rangle\rangle_{\mathsf{P}_i} = (\epsilon[x]_{\mathsf{P}_i}, \epsilon[r_x]_{\mathsf{P}_i}, (\mathsf{c}^{\epsilon}_{x,1}, \ldots, \mathsf{c}^{\epsilon}_{x,n}), \epsilon[\mathsf{MAC}(x)]_{\mathsf{P}_i}).$$

Multiplications Using Beaver Triples Beaver triples are a commonly used technique in MPC [Bea92]. A Beaver triple consists of secret sharings (computed during the preprocessing phase) of values a, b and c such that $ab = c$. Each Beaver triple allows a single multiplication to be efficiently computed during the online phase. Beaver triples can be augmented for $\mathsf{CESS}_{\mathsf{MAC}}$. Given a Beaver triple $\langle\langle a\rangle\rangle$, $\langle\langle b\rangle\rangle$ and $\langle\langle c\rangle\rangle$, the multiplication of $\langle\langle x\rangle\rangle$ and $\langle\langle y\rangle\rangle$ can be done as follows:

- To multiply $\langle\langle x\rangle\rangle_{\mathsf{P}_i}$ by $\langle\langle y\rangle\rangle_{\mathsf{P}_i}$:
  1. Open the sharings $\langle\langle \epsilon\rangle\rangle_{\mathsf{P}_i} = \langle\langle x-a\rangle\rangle_{\mathsf{P}_i}$ and $\langle\langle \delta\rangle\rangle_{\mathsf{P}_i} = \langle\langle y-b\rangle\rangle_{\mathsf{P}_i}$ to obtain the difference values ϵ and δ.
  2. Compute the product $\langle\langle xy\rangle\rangle_{\mathsf{P}_i} = \langle\langle c + \delta a + \epsilon b + \epsilon\delta\rangle\rangle_{\mathsf{P}_i}$ by performing the linear operations as described above.

5 Efficient Malicious-Majority MPC with Identifiable Abort

In the previous two sections, we introduced $\mathsf{CESS}_{\mathsf{MAC}}$ (a locally-identifiable secret sharing scheme) and showed how to compute on it. In this section, we build an efficient MPC scheme with completely identifiable abort on top of $\mathsf{CESS}_{\mathsf{MAC}}$. As discussed in the introduction, we augment the SPDZ protocol.

In the setup phase Init of SPDZ, shares of random values and Beaver triples are generated ahead of time (using slower somewhat-homomorphic encryption techniques), and are then used to facilitate fast multiplications throughout the on-line computation. For our construction, we need a

Protocol $\pi_{\mathtt{CIDA}}^{\mathcal{F}_{\mathtt{SETUP}}}$: MPC With Identifiable Abort, with oracle access to the setup functionality $\mathcal{F}_{\mathtt{SETUP}}$

INIT: To initiate the evaluation of circuit C with n inputs and one output, consisting of addition and multiplication gates over $\mathbb{Z}_p$,

1. Parties invoke the functionality $\mathcal{F}_{\mathtt{SETUP}}$. (Appendix A contains an instantiation $\pi_{\mathtt{SETUP}}$ of $\mathcal{F}_{\mathtt{SETUP}}$.) $\mathcal{F}_{\mathtt{SETUP}}$ generates the additively secret-shared MAC key α, as well as $\langle\langle\rangle\rangle$-shared Beaver triples and $\langle\langle\rangle\rangle$-shared random values.

INPUT: To enable P_i to provide input in_i, the parties do the following (using a fresh random $\mathsf{CESS}_{\mathsf{MAC}}$ sharing $\langle\langle s\rangle\rangle$ generated during INIT):

1. $\langle\langle s\rangle\rangle$ is privately opened as s, r_s to P_i (meaning that all parties send their additive shares of s and r_s to P_i, where r_s is the decommitment value, and P_i reconstructs s and r_s as described in Figure 5.
2. P_i broadcasts $\epsilon = \mathsf{in}_i - s$ (to all parties P_j and to the transcript τ if one is used).
3. All parties P_j locally compute $\langle\langle \mathsf{in}_i\rangle\rangle_{\mathsf{P}_j} = \langle\langle s + \epsilon\rangle\rangle_{\mathsf{P}_j}$.

EVAL: If INIT has been executed and inputs for all input wires of C have been assigned, proceed gate by gate as follows:

ADD: For two values $\langle\langle x_1\rangle\rangle_{\mathsf{P}_j}$, $\langle\langle x_2\rangle\rangle_{\mathsf{P}_j}$:

(a) Each party P_j locally computes $\langle\langle x_3\rangle\rangle_{\mathsf{P}_j} = \langle\langle x_1 + x_2\rangle\rangle_{\mathsf{P}_j}$.

MULT: For two values $\langle\langle x_1\rangle\rangle, \langle\langle x_2\rangle\rangle$:

(a) Let $\langle\langle a\rangle\rangle$, $\langle\langle b\rangle\rangle$ and $\langle\langle c\rangle\rangle$ be a fresh Beaver triple generated during $\pi_{\mathtt{SETUP}}$. The parties use this Beaver triple to compute $\langle\langle t\rangle\rangle = \langle\langle x_1 x_2\rangle\rangle$, as described in Section 4.2.

OUTPUT: To recover the output,

1. Each party P_i broadcasts the additive shares contained in $\langle\langle \mathsf{out}\rangle\rangle_{\mathsf{P}_i}$.
2. The parties execute the reconstruction procedure $\mathsf{CESS}_{\mathsf{MAC}}$.REC described in Figure 5, catching any parties providing malicious shares.

Fig. 6. MPC with Completely Identifiable Abort

setup functionality $\mathcal{F}_{\mathtt{SETUP}}$ that generates $\langle\langle\rangle\rangle$ sharings of random numbers and beaver triples.[8] We describe a secure instantiation $\pi_{\mathtt{SETUP}}$ of $\mathcal{F}_{\mathtt{SETUP}}$ in Appendix A.

Figure 6 gives a slightly simplified illustration of our protocol. The simplification comes from our modular usage of the $\langle\langle\rangle\rangle$-share reconstruction protocol REC, so that cheating detection and cheater identification is performed with every reconstruction. During EVAL, the only communication involved is the reconstruction of two values ϵ and δ. Using the reconstruction procedure $\mathsf{CESS}_{\mathsf{MAC}}$.REC described in Figure 5, any parties providing malicious shares will be caught.

Theorem 2. *Assuming that the discrete log problem (DLP) is hard in the Pedersen commitment group $\mathbb{QR}_q$, the protocol $\pi_{\mathtt{CIDA}}^{\mathcal{F}_{\mathtt{SETUP}}}$ with oracle access to the functionality $\mathcal{F}_{\mathtt{SETUP}}$ is a UC-secure implementation of the functionality $\mathcal{F}_{\mathtt{CIDA}}$.*

Informally, Theorem 2 holds because after running INIT, the only messages sent are (1) a single value broadcast during INPUT, and (2) reconstructions of $\langle\langle\rangle\rangle$-sharings. Since the $\langle\langle\rangle\rangle$-sharing scheme ($\mathsf{CESS}_{\mathsf{MAC}}$) is a locally identifiable secret sharing scheme, adversarially controlled parties are not be able to change any shared values without the honest parties identifying their malicious behavior. The value broadcast during INPUT defines the input in question, and inconsistencies with that value will also be detected during reconstructions. A formal proof of Theorem 2 appears in Appendix B.2.

By the universal composition theorem [Can00], this implies that a UC-secure implementation $\pi_{\mathtt{SETUP}}$ of $\mathcal{F}_{\mathtt{SETUP}}$ gives a UC-secure implementation $\pi_{\mathtt{CIDA}}^{\pi_{\mathtt{SETUP}}}$ of $\mathcal{F}_{\mathtt{CIDA}}$, simply by replacing the call to $\mathcal{F}_{\mathtt{SETUP}}$ with a call to $\pi_{\mathtt{SETUP}}$.

[8] The SPDZ protocol generates the same number of shared values. However, their sharings only contain an additive secret sharing and a linear MAC. The size of $\langle\langle\rangle\rangle$-shares grows linearly with the number of players, while SPDZ shares have a constant size for a fixed security parameter.

Optimistic Protocol The cheater detection and identification inherent in the $\mathsf{CESS}_{\mathsf{MAC}}$ openings of Eval can be safely postponed to Output. That way, cheating detection (MACCheck) is batched in such a way that the communication required is independent of the number of multiplications performed, as described in Figure 4. If MACCheck reveals that cheating occurred, the parties will finally perform all of the relevant computations on the $\mathsf{CESS}_{\mathsf{MAC}}$ commitments, as described in Section 4.2.

To make the protocol optimistic, parties must save all of the difference values ϵ and δ from Beaver triple multiplications performed throughout the computation.[9] This adds an $O(nm)$ storage overhead for each party, where n is the number of parties and m is the number of multiplications in the computation. However, this does not asymptotically increase the storage requirements, because each party must store $O(nm)$ secret-shared Beaver triples anyway, which are generated during $\pi_{\texttt{SETUP}}$.

6 Openable Auditable MPC

In this section, we augment our construction from Section 5 with auditability and openability. Auditability is achieved by logging all public values (including commitments from setup and publicly opened difference values) from the construction in Section 5 to the public transcript τ. As in Baum et. al [BDO14], the input commitments together with the public values can be used to obtain a commitment to the output, and that commitment can then be checked against the claimed output and output decommitment values.

Intuitively, openability should be achievable by doing the following:

1. Parties in the opening coalition $\mathcal{C}$ generate a verifiable encryption key pair $(pk_{\mathcal{C}}, sk_{\mathcal{C}})$, in such a way that
   - the encryption key $pk_{\mathcal{C}}$ is publicly available, and
   - the decryption key $sk_{\mathcal{C}}$ is secret-shared among the opening coalition $\mathcal{C}$.[10]
2. Each party P_i publishes the following:
   - a verifiable encryption $e_i = \textsc{Enc}_{\mathsf{ver}}(pk_{\mathcal{C}}, \mathsf{in}_i)$ of their input in_i, and
   - a non-interactive zero-knowledge proof that e_i is an encryption of the input.

This approach doesn't quite work. For each honest party, the simulator needs to produce an encryption e_i of their input in_i without knowing in_i. Since most encryption schemes are perfectly binding, the simulator would be unable to produce encryptions that open to the correct inputs.

To overcome this problem, we add a layer of secret sharing and commitments to secret shares. The augmented construction is as follows (formal description in Figure 7):

1. Each party P_i additively shares in_i among the opening coalition $\mathsf{P}_j \in \mathcal{C}$.
2. The parties in $\mathcal{C}$ might not be online at the time of the sharing, so P_i encrypts the shares to the parties in $\mathcal{C}$ and posts these encryptions to the transcript τ. More specifically, P_i publishes the following values to τ for each $\mathsf{P}_j \in \mathcal{C}$:
   - $\mathsf{c}_{(i,j)}$: a commitment to P_j's share of in_i;
   - $e_{(i,j)}$: a verifiable encryption (under P_j's key) of the same value,
   - $e_{(i,j,r)}$: a verifiable encryption of the decommitment value $r_{(i,j)}$ for $\mathsf{c}_{(i,j)}$.

[9] Note that if a public transcript τ is maintained, it contains all of these difference values.
[10] For convenience we assume a single coalition $\mathcal{C}$.

3. P_i computes and posts the following non-interactive zero-knowledge proofs:
   (a) That $e_{(i,j)}$ encrypts the value in $\mathsf{c}_{(i,j)}$ for each $\mathsf{P}_j \in \mathcal{C}$.
   (b) That $e_{(i,j,r)}$ encrypts the decommitment value $r_{(i,j)}$ for $\mathsf{c}_{(i,j)}$.
   (c) That the sum of the committed shares in $\mathsf{c}_{(i,j)}$ for all $\mathsf{P}_j \in \mathcal{C}$ is equal to the sum of the committed shares in the input $\langle\langle\rangle\rangle$-sharing.
4. During OPEN, members P_j of the opening coalition $\mathcal{C}$ reveal $[\mathsf{in}_i]_{\mathsf{P}_j}$ and $r_{(i,j)}$ for $\mathsf{c}_{(i,j)}$ to one another. The commitments are checked, and the output reconstructed.

A simulator for this protocol only needs to open a commitment to a secret share of in_i, not an encryption. Since Pedersen commitments are only computationally binding the simulator can break computational binding.[11]

Protocol $\pi^{\mathcal{F}_{\text{SETUP}}}_{\text{CIDA,AUDIT,OPEN}}$: Openable MPC with Identifiable Abort, with oracle access to the setup functionality $\mathcal{F}_{\text{SETUP}}$

INIT: Same as in π_{CIDA} (Figure 6). Additionally,
1. All parties P_i in the opening coalition $\mathcal{C}$ generate a public-secret verifiable encryption key pair (pk_i, sk_i), and publish pk_i. For parties in the opening coalition $\mathcal{C}$ not participating in the computation, we assume that verifiable encryption key pairs already exist.

INPUT: Same as in π_{CIDA} (Figure 6). Additionally, party P_i does the following:
1. Let c_i be the commitment to the input in_i in the $\langle\langle\rangle\rangle$-sharing of in_i.
2. P_i computes an additive secret sharing $[\mathsf{in}_i]$ of in_i for the opening coalition $\mathcal{C}$. For each $\mathsf{P}_j \in \mathcal{C}$:
   (a) P_i chooses a fresh random $r_{(i,j)}$, and computes and publishes a commitment $\mathsf{c}_{(i,j)} = \mathsf{pc}([\mathsf{in}_i]_{\mathsf{P}_j}, r_{(i,j)})$.
   (b) P_i computes and publishes a verifiable encryption $e_{(i,j)} = \text{ENC}_{\mathsf{ver}}([\mathsf{in}_i]_{\mathsf{P}_j}, pk_j)$ where pk_j is the public verifiable encryption key of P_j.
   (c) P_i computes and publishes a verifiable encryption $e_{(i,j,r)} = \text{ENC}_{\mathsf{ver}}(r_{(i,j)}, pk_j)$.
   (d) P_i computes and publishes a NIZKP (as in Appendix C.3) that $\mathsf{c}_{(i,j)}$ and $e_{(i,j)}$ are to the same thing.
   (e) P_i computes and publishes a NIZKP (as in Appendix C.3) that $e_{(i,j,r)}$ encrypts the decommitment value of $\mathsf{c}_{(i,j)}$.
3. P_i computes and publishes a NIZKP that the sum of the committed values in $\mathsf{c}_{(i,j)}$ is the same as the committed value in c_i. This can be done by computing $\mathsf{c}'_i = \prod_{\mathsf{P}_j \in \mathcal{C}} \mathsf{c}_{(i,j)}$, and computing a NIZKP (as in Appendix C.3) that c_i and c'_i are to the same thing.

EVAL: Same as in π_{CIDA} (Figure 6).

OUTPUT: Same as in π_{CIDA} (Figure 6).

AUDIT: The auditor receives the transcript τ, together with the output value out and decommitment value r_{out}. They use the commitments in the transcript τ, together with the opened values, to compute commitments to the additive shares of the output belonging to each party. (Informally, this can be done using exactly as described in Section 4.2, but ignoring all secret share values except the commitments.) They then compute the product of these additive share commitments; this value is $\mathsf{c}_{\mathsf{out}}$. If $\mathsf{c}_{\mathsf{out}} = \mathsf{pc}(\mathsf{out}, r_{\mathsf{out}})$, the auditor outputs `ACCEPT`. Otherwise, they output `REJECT`. (Similarly, if the auditor receives each party P_i's additive secret shares of the output $[\mathsf{out}]_{\mathsf{P}_i}$ and of the decommitment value $[r_{\mathsf{out}}]_{\mathsf{P}_i}$, they can audit each individual party's behavior.)

OPEN: For each $i \in \{1, \ldots, n\}$:
1. All parties P_j in the opening coalition $\mathcal{C}$ retrieve their encrypted shares $e_{(i,j)}$ and decommitment values $e_{(i,j,r)}$, decrypt them to obtain $[\mathsf{in}_i]_{\mathsf{P}_j}$ and $r_{(i,j)}$, and broadcast them.
2. All parties check that $\mathsf{c}_{(i,j)} = \mathsf{pc}([\mathsf{in}_i]_{\mathsf{P}_j}, r_{(i,j)})$.
3. All parties reconstruct in_i.

Fig. 7. Openable MPC with Identifiable Abort

Theorem 3. *Assuming (a) that the discrete log problem (DLP) is hard in the Pedersen commitment group $\mathbb{QR}_q$, (b) a secure NIZKP scheme, and (c) that* $(\text{KEYGEN}, \text{ENC}_{\mathsf{ver}}, \text{DEC}_{\mathsf{ver}})$ *is a semantically*

[11] The simulator chooses the generators used in the Pedersen commitment scheme when selecting the CRS.

secure verifiable encryption scheme, the protocol $\pi^{\mathcal{F}_{\texttt{SETUP}}}_{\texttt{CIDA},\texttt{AUDIT},\texttt{OPEN}}$ *with oracle access to the functionality* $\mathcal{F}_{\texttt{SETUP}}$ *is a UC-secure implementation of the functionality* $\mathcal{F}_{\texttt{CIDA},\texttt{AUDIT},\texttt{OPEN}}$.

Informally, Theorem 3 holds because the zero-knowledge proofs in INPUT prove that encryptions to valid shares of the input values are decryptable by the opening coalition $\mathcal{C}$. A formal proof of Theorem 3 appears in Appendix B.3.

Efficiency To achieve auditability, no additional values need to be computed at all. As stated above, public values are posted to the bulletin board. Openability requires one additional additive secret-sharing of each input (to the opening coalition $\mathcal{C}$), and a verifiable encryption (described in Appendix C.2) and commitment (described in Appendix C.1) to each share. Additionally, $2|\mathcal{C}| + 1$ non-interactive zero-knowledge proofs (described in Appendix C.3) are required, where $|\mathcal{C}|$ is the size of opening coalition. This cost is small, and independent of the computation.

Open Problems

One interesting open problem is achieving our communication complexity ($O(nm)$, as opposed to $O(n^2 + nm)$) using only efficient information-theoretically secure techniques.

Another open problem is to integrate fairness into our protocol. Our protocol makes no attempt to provide fairness when the majority of players act honestly (when less than $n/2$ players are corrupt). It may be possible to construct an MPC protocol that provides fairness when the majority of parties act honestly and completely identifiable abort when they do not.

Acknowledgements

The authors would like to thank Samuel Yeom, Mayank Varia and Arkady Yerukhimovich for helpful discussion.

References

[ADMM16] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on Bitcoin. *Commun. ACM*, 59(4):76–84, March 2016.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.

[BCD+09] Peter Bogetoft, DanLund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krigaard, JanusDam Nielsen, JesperBuus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer Berlin Heidelberg, 2009.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 175–196, 2014.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology–CRYPTO91*, pages 420–432. Springer, 1992.

[BOS16] Carsten Baum, Emmanuela Orsini, and Peter Scholl. Efficient secure multiparty computation with identifiable abort. Cryptology ePrint Archive, Report 2016/187, 2016. `http://eprint.iacr.org/`.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology-EUROCRYPT 2000*, pages 431–444. Springer, 2000.

[BS90] Ernest F Brickell and Douglas R Stinson. The detection of cheaters in threshold schemes. In *Proceedings on Advances in Cryptology*, pages 564–577. Springer-Verlag New York, Inc., 1990.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/`.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *Advances in Cryptology–CRYPTO–94*, pages 257–270. Springer, 1994.

[Cle86] R Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 364–369, New York, NY, USA, 1986. ACM.

[CM99] Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In Michael Wiener, editor, *Advances in Cryptology, CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 413–430. Springer Berlin Heidelberg, 1999.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology-CRYPTO'97*, pages 410–424. Springer, 1997.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer Berlin Heidelberg, 2003.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *Advances in Cryptology-ASIACRYPT 2002*, pages 125–142. Springer, 2002.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure MPC for dishonest majority–or: Breaking the SPDZ limits. In *Computer Security–ESORICS 2013*, pages 1–18. Springer, 2013.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology-CRYPTO'97*, pages 16–30. Springer, 1997.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology-CRYPTO'86*, pages 186–194. Springer, 1986.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, February 1989.

[HLOW16] Bret Hemenway, Steve Lu, Rafail Ostrovsky, and William IV Welser. High-precision secure computation of satellite collision probabilities. Cryptology ePrint Archive, Report 2016/319, 2016. `http://eprint.iacr.org/`.

[HWB14] Brett Hemenway, William IV Welser, and Dave Baiocchi. Achieving higher-fidelity conjunction analyses using cryptography to improve information sharing. Technical Report, 2014. `http://www.rand.org/pubs/research_reports/RR344.html`.

[IOS12] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 21–38, 2012.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *Advances in Cryptology – CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 369–386, 2014.

[Jak10] Ram S Jakhu. Iridium-cosmos collision and its implications for space operations. In *Yearbook on Space Policy 2008/2009*, pages 254–275. Springer, 2010.

[KB14] Ranjit Kumaresan and Iddo Bentov. How to use Bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 30–41, New York, NY, USA, 2014. ACM.

[KOO95] Kaoru Kurosawa, Satoshi Obana, and Wakaha Ogata. t-cheater identifiable (k, n) threshold secret sharing schemes. In *Advances in Cryptology–CRYPT0–95*, pages 410–423. Springer, 1995.

[Lys02] Anna Lysyanskaya. Signature schemes and applications to cryptographic protocol design. MIT PhD Dissertation, 2002. `http://groups.csail.mit.edu/cis/theses/anna-phd.pdf`.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology-CRYPTO91*, pages 129–140. Springer, 1992.

[RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[TW86] Martin Tompa and Heather Woll. How to share a secret with cheaters. In *Advances in Cryptology–CRYPTO–86*, pages 261–265. Springer, 1986.

[Wri09] David Wright. Colliding satellites: Consequences and implications. *Union of Concerned Scientists*, 26, 2009.

[WW95] T-C Wu and T-S Wu. Cheating detection and cheater identification in secret sharing schemes. In *IEE Proceedings - Computers and Digital Techniques*, volume 142, pages 367–369. IET, 1995.

A Preprocessing

In describing our MPC schemes, we assumed a secure, auditable setup implementation $\pi_{\texttt{SETUP}}$ which generates commitment-enhanced secret sharings of Beaver triples. In Figure 14, we describe such a $\pi_{\texttt{SETUP}}$. It is similar to the setup phases of Damgård et. al [DPSZ12] and Baum et. al [BDO14], but it supports $\langle\langle\cdot\rangle\rangle$-sharings. $\pi_{\texttt{SETUP}}$ must do the following:

1. Establish an additively secret-shared MAC key α,
2. Generate random $\langle\langle\cdot\rangle\rangle$-shared values to be used during Input, and
3. Generate $\langle\langle\cdot\rangle\rangle$-shared Beaver triples for every multiplication to be performed in the online phase.

To achieve these goals, like Damgård et. al [DPSZ12] and Baum et. al [BDO14], we use somewhat homomorphic encryption. We also assume a protocol PickRandom for choosing randomness. Such a protocol is described in Figure 4 of Baum et. al.

Somewhat Homomorphic Encryption $\pi_{\texttt{SETUP}}$ requires the use of a somewhat homomorphic encryption scheme; let $(\text{HKeyGen}, \text{HEnc}, \text{HDec})$ be such a scheme. We refer to Baum et. al [BDO14] for an extensive discussion of appropriate encryption schemes; here, we will only describe their basic functionality requirements. $(\text{HKeyGen}, \text{HEnc}, \text{HDec})$ must support homomorphic additions and one homomorphic multiplication. It must also support *distributed key generation* (denoted DistKeyGen), wherein each participating party ends up holding a share of the secret decryption key, and all parties hold the public encryption key. The secret-shared decryption key can then be used for *distributed decryption* (denoted DistDec), which takes in a ciphertext and results in each party holding the plaintext. We assume that DistKeyGen and DistDec are both implemented with completely identifiable abort; this can easily be achieved by having each party give a non-interactive zero-knowledge proof (NIZKP) of the validity of each message it sends. Given the simplicity of both protocols, this is fairly efficient. Posting all public values and NIZKPs to the transcript τ also makes the protocols publicly auditable.

Building $\pi_{\texttt{SETUP}}$, The Preprocessing Protocol Using DistDec, we can implement a resharing protocol AdditiveReshare. AdditiveReshare takes in a ciphertext, and returns an additive []-sharing of the underlying plaintext to the parties. AdditiveReshare is shown in Figure 8. The sharing we truly want, however, is a $\langle\langle\rangle\rangle$-sharing, and in order to acquire such a sharing, we first need to generate a secret-shared MAC key α. DistMACKeyGen, shown in Figure 9, is a protocol for distributed MAC key generation (which returns additive shares of the MAC key together with its encryption e_α). Reshare, shown in Figure 10, uses DistMACKeyGen; it takes in a ciphertext and returns a $\langle\langle\rangle\rangle$-sharing of the underlying plaintext. We also introduce PickSecretSharedRandom (described in Figure 11), which generates a $\langle\langle\rangle\rangle$-shared random value.

Next, we use these protocols to construct MultSecretSharedValues, a protocol that enable parties to multiply $\langle\langle\rangle\rangle$-sharings *without* using Beaver triples. This is used to generate the Beaver triples themselves (as described in Figure 13) and is less efficient than multiplication *using* Beaver triples.

MultSecretSharedValues is described in Figure 12. Finally, $\pi_{\texttt{SETUP}}$ simply runs DistKeyGen, DistMACKeyGen, PickSecretSharedRandom and PickSecretSharedBeaverTriple.

We do not show the simulator for preprocessing here. It is similar to the one shown by Baum et. al [BDO14].

Completely Identifiable Abort and Auditability Notice that all communication in each protocol described in this section consists of (a) calls to sub-protocols, and (b) messages the correctness of which is proved in non-interactive zero-knowledge (NIZK). Our base sub-protocols DistKeyGen and DistDec have completely identifiable abort; so, by also checking each provided NIZK proof, it can be shown that we get completely identifiable abort in the composite protocol. By logging all public values and NIZK proofs to the transcript τ, it can be shown that we also get auditability.

AdditiveReshare: Protocol for Additively Resharing an Element x Given its Ciphertext e_x

Preconditions: The parties are assumed to have already run DistKeyGen, so each party knows the public encryption key pk_{hom}, and an additive secret share of the secret decryption key sk_{hom}.

AdditiveReshare(e_x): In this protocol, the parties take a ciphertext e_x of x, and compute an additive []-secret sharing of the underlying plaintext x.

1. Each party P_i does the following:
   (a) Picks a random value $f_i \in \mathbb{Z}_p$. Notionally, these are additive secret shares of a random value $f \in \mathbb{Z}_p$.
   (b) Computes the ciphertext $e_{f_i} = \text{HEnc}(pk_{\mathsf{hom}}, f_i)$, and a non-interactive zero-knowledge proof (NIZKP) of its validity.
   (c) Publishes e_{f_i} together with the NIZKP (either broadcasting them to all parties or posting them to the transcript τ if one is being used).
   (d) Upon the receipt of ciphertexts and NIZKPs from all parties, checks all the NIZKPs. If any of them do not verify, compiles a list L_{cheat} of parties whose NIZKP did not verify, and returns ($\texttt{REJECT}, L_{cheat}$).
   (e) Using the homomorphic properties of the encryption scheme, computes the encryption e_f of $f = \sum_{i\in\{1,\ldots,n\}} f_i$, and the encryption e_{x+f} of $x + f$.
2. The parties invoke DistDec to decrypt e_{x+f}. All parties learn the value $x + f$.
3. The parties can now compute the additive secret sharing: party P_1 sets $[x]_{\mathsf{P}_1} = x + f - f_1$, and every other party P_i ($i \in \{2, \ldots, n\}$) sets $[x]_{\mathsf{P}_i} = -f_i$.

Fig. 8. AdditiveReshare: Additively Resharing an Element x Given its Ciphertext e_x

DistMACKeyGen: Protocol for the Distributed Generation of an Additively Secret-Shared MAC key α

Preconditions: The parties are assumed to have already run DistKeyGen, so each party knows the public encryption key pk_{hom}, and an additive secret share of the secret decryption key sk_{hom}.

DistMACKeyGen: In this protocol, the parties compute an additive []-sharing of a random MAC key α, together with an encryption e_α of α. Each party P_i does the following:

1. Picks a random value $\alpha_i \in \mathbb{Z}_p$, and sets $[\alpha]_{\mathsf{P}_i} = \alpha_i$.
2. Computes the ciphertext $e_{\alpha_i} = \text{HEnc}(pk_{\mathsf{hom}}, \alpha_i)$, and a non-interactive zero-knowledge proof (NIZKP) of its validity.
3. Publishes e_{α_i} together with the NIZKP (either broadcasting them to all parties or posting them to the transcript τ if one is being used).
4. Upon the receipt of ciphertexts and NIZKPs from all other parties, checks all the NIZKPs. If any of them do not verify, compiles a list L_{cheat} of parties whose NIZKP did not verify, and returns $(\texttt{REJECT}, L_{cheat})$.
5. Using the homomorphic properties of the encryption scheme, computes the encryption e_α of $\alpha = \sum_{i \in \{1,\ldots,n\}} \alpha_i$.

Fig. 9. DistMACKeyGen: Distributed Generation of an Additively Secret-Shared MAC key α

B Proofs of Security

B.1 Background: Universal Composability (UC)

Universal composability [Can00] is defined in the context of two worlds - the real world and the ideal world. In both worlds, the environment $\mathcal{Z}$ sets all participating parties' inputs, and receives all of their outputs. Additionally, it participates in the protocol π on behalf of the corrupt parties, sending arbitrary messages and observing all messages received. In the ideal world, unbeknownst to the environment, the execution of the protocol π is replaced with a simulated execution courtesy of a simulator $\mathcal{S}$, who observes corrupt parties' inputs and outputs from the functionality $\mathcal{F}$, but sees nothing else. Afterwards, the environment $\mathcal{Z}$ outputs a guess as to which world it was in - "real" or "ideal". Protocol π UC-securely implements a functionality $\mathcal{F}$ if there exists an probabilistic polynomial time (PPT) simulator $\mathcal{S}$ such that for all PPT environments $\mathcal{Z}$, the outputs of $\mathcal{Z}$ in the real and ideal worlds are computationally indistinguishable.

B.2 Simulator for MPC with Identifiable Abort

Proof (Theorem 2). We can build a simulator $\mathcal{S}_{\texttt{CIDA}}$ for $\pi_{\texttt{CIDA}}$. $\mathcal{S}_{\texttt{CIDA}}$ gains an advantage by providing the CRS from which the Pedersen commitment generators g and h are derived. $\mathcal{S}_{\texttt{CIDA}}$ can provide a CRS such that it will know the discrete log relationship between the two generators, thus allowing $\mathcal{S}_{\texttt{CIDA}}$, given a commitment c and any element $x \in \mathbb{Z}_p$, to compute randomness r_x such that $\mathsf{pc}(x, r_x) = \mathsf{c}$.

In order to generate a simulated view, $\mathcal{S}_{\texttt{CIDA}}$ follows the protocol $\pi_{\texttt{CIDA}}$ closely, with some exceptions, as shown in Figure 15. One such exception is using fixed inputs $\mathsf{in}_i = 0$ for honest parties P_i. Fewer than n $\langle\langle\rangle\rangle$-shares are information-theoretically hiding, and so reveal nothing about shared values, making it impossible for an adversary to detect whether $\mathsf{in}_i = 0$. Thus, the simulated view is indistinguishable from the real view.

RESHARE: Protocol for $\langle\langle\cdot\rangle\rangle$-Resharing an Element x Given its Ciphertext e_x

PRECONDITIONS: – A modulus and generators for the Pedersen commitment scheme are known.
– The parties are assumed to have already run DISTKEYGEN, so each party holds the public encryption key pk_{hom}, and an additive secret share of the secret decryption key sk_{hom}.
– The parties are assumed to have already run DISTMACKEYGEN (described in Figure 9), so each party holds an additive secret share of the MAC key α, and an encryption e_α of α.

RESHARE(e_x): In this protocol, the parties take a ciphertext e_x of x, and compute a $\langle\langle\cdot\rangle\rangle$-secret sharing of the underlying plaintext x.

1. Each party P_i does the following:
   (a) Picks random values $f_i, r_i \in \mathbb{Z}_p$. The values f_i are additive secret shares of a random value $f \in \mathbb{Z}_p$, and the values r_i are additive secret shares of a random value $r_f \in \mathbb{Z}_p$.
   (b) Computes the ciphertext $e_{f_i} = \text{HENC}(pk_{\mathsf{hom}}, f_i)$, and a non-interactive zero-knowledge proof (NIZKP) of its validity.
   (c) Computes the commitment $\mathsf{c}_{f_i} = \mathsf{pc}(f_i, r_i)$, and a NIZKP that it is to the same thing as $e_{f,i}$.
   (d) Publishes e_{f_i} and c_{f_i} together with the NIZKPs (either broadcasting them to all parties or posting them to the transcript τ if one is being used).
   (e) Upon the receipt of values from all other parties, checks all the NIZKPs. If any of them do not verify, compiles a list L_{cheat} of parties whose NIZKP did not verify, and returns $(\texttt{REJECT}, L_{cheat})$.
   (f) Using the homomorphic properties of the encryption scheme, computes the encryption e_f of $f = \sum_{i\in\{1,\ldots,n\}} f_i$, and the encryption e_{x+f} of $x + f$.
   (g) The parties invoke DISTDEC to decrypt e_{x+f}. All parties learn the value $x + f$.
2. The parties invoke PICKRANDOM to jointly choose a random value $r_{x+f} \in \mathbb{Z}_p$.
3. Each party locally computes the commitment $\mathsf{c}_{x+f} = \mathsf{pc}(x + f, r_{x+f})$.
4. The parties can now compute additive secret shares of x and of the decommitment value r_x:
   – Party P_1 sets $[x]_{\mathsf{P}_1} = x + f - f_1$, and $[r_x]_{\mathsf{P}_1} = r_{x+f} - r_1$.
   – Party P_i ($i \in \{2, \ldots, n\}$) sets $[x]_{\mathsf{P}_i} = -f_i$, and $[r_x]_{\mathsf{P}_i} = -r_i$
5. Each party also locally computes commitments to *every party's* additive share of x, as follows:
   – $\mathsf{c}_1 = \mathsf{c}_{x+f}\mathsf{c}_{f_1}^{-1}$.
   – For $i \in \{2, \ldots, n\}$, $\mathsf{c}_i = \mathsf{c}_{f_i}^{-1}$.

   Note that $[r_x]_{\mathsf{P}_i}$ is the decommitment value for the commitment c_i.
6. Using the homomorphic properties of the encryption scheme and the ciphertexts e_α (encrypting the MAC key α) and e_x (encrypting the element x), each party locally computes the encryption $e_{\mathsf{MAC}(x)}$ of the MAC value $\mathsf{MAC}(x) = \alpha x$.
7. The parties run ADDITIVERESHARE on $e_{\mathsf{MAC}(x)}$ as described in Figure 8, such that each party P_i ends up with $[\mathsf{MAC}(x)]_{\mathsf{P}_i}$.
8. Each party P_i now holds $\langle\langle x\rangle\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i}, r_i, (\mathsf{c}_1, \ldots, \mathsf{c}_n), [\mathsf{MAC}(x)]_{\mathsf{P}_i})$.

Fig. 10. RESHARE: Resharing an Element x Given its Ciphertext e_x (A Sub-Protocol of $\pi_{\texttt{SETUP}}$, described in Figure 14)

PickSecretSharedRandom: Protocol for Generating Secret-Shared Randomness

Preconditions: – A modulus and generators for the Pedersen commitment scheme are known.
– The parties are assumed to have already run DistKeyGen, so each party holds the public encryption key pk_{hom}, and an additive secret share of the secret decryption key sk_{hom}.
– The parties are assumed to have already run DistMACKeyGen (described in Figure 9), so each party holds an additive secret share of the MAC key α, and an encryption e_α of α.

PickSecretSharedRandom: In this protocol, the parties jointly choose a $\langle\langle\cdot\rangle\rangle$-shared random value $r \in \mathbb{Z}_p$.

1. Each party P_i does the following:
   (a) Picks random $s_i \in \mathbb{Z}_p$. Notionally, s_i is an additive secret share of random values $s \in \mathbb{Z}_p$.
   (b) Computes ciphertext $e_i = \text{HEnc}(pk_{\mathsf{hom}}, s_i)$, together with a non-interactive zero-knowledge proof (NIZKP) of its validity.
   (c) Publishes e_i together with the NIZKP (either broadcasting them to all parties or posting them to the transcript τ if one is being used).
   (d) Upon receipt of ciphertexts and NIZKPs from all parties, checks all NIZKPs. If any NIZKP is incorrect, compiles a list L_{cheat} of parties whose NIZKPs are incorrect, and returns $(\texttt{REJECT}, L_{cheat})$.
   (e) Using the homomorphic properties of the encryption scheme, computes an encryption e of $s = \sum_{i\in\{1,\ldots,n\}} s_i$.
2. The parties run Reshare on e, resulting each party P_i holding $\langle\langle s\rangle\rangle_{\mathsf{P}_i}$.

Fig. 11. PickSecretSharedRandom: Generating Secret-Shared Randomness

MultSecretSharedValues: Protocol for Multiplying Secret Shared Values Without Using Beaver Triples

Preconditions: – A modulus and generators for the Pedersen commitment scheme are known.
– The parties are assumed to have already run DistKeyGen, so each party holds the public encryption key pk_{hom}, and an additive secret share of the secret decryption key sk_{hom}.
– The parties are assumed to have already run DistMACKeyGen (described in Figure 9), so each party holds an additive secret share of the MAC key α, and an encryption e_α of α.

MultSecretSharedValues($\langle\langle a\rangle\rangle, \langle\langle b\rangle\rangle$): Each party P_i holds $\langle\langle a\rangle\rangle_{\mathsf{P}_i} = ([a]_{\mathsf{P}_i}, [r_a]_{\mathsf{P}_i}, (\mathsf{c}_{a,1}, \ldots, \mathsf{c}_{a,n}), [\mathsf{MAC}(a)]_{\mathsf{P}_i})$ and $\langle\langle b\rangle\rangle_{\mathsf{P}_i} = ([b]_{\mathsf{P}_i}, [r_b]_{\mathsf{P}_i}, (\mathsf{c}_{b,1}, \ldots, \mathsf{c}_{b,n}), [\mathsf{MAC}(b)]_{\mathsf{P}_i})$. In this protocol, the parties compute $\langle\langle c\rangle\rangle$, where $c = a \times b$. Each party P_i should end up holding $\langle\langle c\rangle\rangle_{\mathsf{P}_i}$.

1. Each party P_i does the following:
   (a) Computes ciphertexts $e_{a,i} = \text{HEnc}(pk_{\mathsf{hom}}, [a]_{\mathsf{P}_i})$ and $e_{b,i} = \text{HEnc}(pk_{\mathsf{hom}}, [b]_{\mathsf{P}_i})$.
   (b) Computes non-interactive zero-knowledge proofs (NIZKPs) of:
      – the validity of $e_{a,i}$ and $e_{b,i}$,
      – the fact that $\mathsf{c}_{a,i}$ is to the same thing as $e_{a,i}$, and
      – the fact that $\mathsf{c}_{b,i}$ is to the same thing as $e_{b,i}$.
   (c) Publishes $e_{a,i}$, $e_{b,i}$ and the NIZKPs (either broadcasting them to all parties or posting them to the transcript τ if one is being used).
   (d) Upon receipt of ciphertexts and NIZKPs from all parties, checks all NIZKPs. If any NIZKP is incorrect, compiles a list L_{cheat} of parties whose NIZKPs are incorrect, and returns $(\texttt{REJECT}, L_{cheat})$.
2. Using the homomorphic properties of the encryption scheme, all parties compute:
   – an encryption e_a of $a = \sum_{i\in\{1,\ldots,n\}} a_i$,
   – an encryption e_b of $b = \sum_{i\in\{1,\ldots,n\}} b_i$, and
   – an encryption $e_{a\times b}$ of the product of a and b.
3. The parties run Reshare on $e_{a\times b}$, as described in Figure 10. Each party P_i ends up with a secret share $\langle\langle c\rangle\rangle_{\mathsf{P}_i}$ of $c = a \times b$.

Fig. 12. MultSecretSharedValues: Multiplying Secret Shared Values Without Using Beaver Triples

PickSecretSharedBeaverTriple: Protocol for Generating a Secret-Shared Beaver Triple

Preconditions:
- A modulus and generators for the Pedersen commitment scheme are known.
- The parties are assumed to have already run DistKeyGen, so each party holds the public encryption key pk_{hom}, and an additive secret share of the secret decryption key sk_{hom}.
- The parties are assumed to have already run DistMACKeyGen (described in Figure 9), so each party holds an additive secret share of the MAC key α, and an encryption e_α of α.

PickSecretSharedBeaverTriple: In this protocol, the parties compute three $\langle\langle\cdot\rangle\rangle$-shared values $\langle\langle a\rangle\rangle$, $\langle\langle b\rangle\rangle$ and $\langle\langle c\rangle\rangle$, where a and b are random elements in $\mathbb{Z}_p$, and $c = a \times b$.

1. The parties invoke PickSecretSharedRandom (described in Figure 11) to create sharings $\langle\langle a\rangle\rangle$ and $\langle\langle b\rangle\rangle$ of random values $a, b \in \mathbb{Z}_p$.
2. The parties invoke MultSecretSharedValues$(\langle\langle a\rangle\rangle, \langle\langle b\rangle\rangle)$ (described in Figure 12) to multiply the sharings $\langle\langle a\rangle\rangle$ and $\langle\langle b\rangle\rangle$, resulting in a new sharing $\langle\langle c\rangle\rangle$, where $c = a \times b$.

Fig. 13. PickSecretSharedBeaverTriple: Generating Secret-Shared Beaver Triples

Protocol $\pi_{\mathtt{SETUP}}$ (Setup for Auditable MPC)

Preconditions:
- A modulus and generators for the Pedersen commitment scheme are known.

Init: On input (Init, p) from all parties:

1. The parties invoke DistKeyGen to obtain a shared homomorphic encryption key pair $(sk_{\mathsf{hom}}, pk_{\mathsf{hom}})$, where pk_{hom} is public and sk_{hom} is additively secret-shared among the parties.
2. The parties invoke DistMACKeyGen, shown in Figure 9, to obtain a $\langle\langle\cdot\rangle\rangle$-shared secret-shared MAC key α (together with its encryption e_α).
3. The parties invoke PickSecretSharedRandom, shown in Figure 11, to generate a $\langle\langle\cdot\rangle\rangle$-sharing of a random number for each input.
4. The parties invoke PickSecretSharedBeaverTriple, shown in Figure 13, to generate $\langle\langle\cdot\rangle\rangle$-sharings of a Beaver triple for each multiplication they will need to perform.

Fig. 14. Preprocessing: a Circuit Independent Protocol to be Executed Prior to Circuit Evaluation. This processing stage is used for all of our constructions.

Simulator $\mathcal{S}_{\texttt{CIDA}}$ for Protocol $\pi_{\texttt{CIDA}}$ (Private-Output Auditable MPC)

INIT: 1. $\mathcal{S}_{\texttt{CIDA}}$ provides a CRS such that it knows the discrete log relationship of the two generators g and h of the Pedersen commitment scheme. This is done in such a way that the CRS is indistinguishable from random.
2. $\mathcal{S}_{\texttt{CIDA}}$ simulates $\pi_{\texttt{SETUP}}$ (in a way similar to that described by Baum et. al [BDO14]). During this simulation, $\mathcal{S}_{\texttt{CIDA}}$ learns the private decryption key sk_{hom}, recovering the MAC key α.

INPUT: For each party P_i,
1. If P_i is adversarial, $\mathcal{S}_{\texttt{CIDA}}$ learns the input in_i and executes INPUT with it as described in $\pi_{\texttt{CIDA}}$.
2. If P_i is honest, $\mathcal{S}_{\texttt{CIDA}}$ executes INPUT (interacting with the adversarial parties as needed) as described in $\pi_{\texttt{CIDA}}$ with a fixed input of 0.

EVAL: $\mathcal{S}_{\texttt{CIDA}}$ evaluates C gate by gate (interacting with the adversarial parties as needed), as described in $\pi_{\texttt{CIDA}}$.

OUTPUT: $\mathcal{S}_{\texttt{CIDA}}$ gets the output out from the functionality $\mathcal{F}_{\texttt{CIDA}}$.
1. $\mathcal{S}_{\texttt{CIDA}}$ changes the output additive share $[\mathsf{out}]_{\mathsf{P}_i}$ of one of the honest parties P_i to force the shares to reconstruct to out.
2. $\mathcal{S}_{\texttt{CIDA}}$ finds a randomness $r_{\mathsf{out},i}$ such that $\mathsf{pc}([\mathsf{out}]_{\mathsf{P}_i}, r_{\mathsf{out},i}) = \mathsf{c}_{\mathsf{out},i}$. (It can do this since it has the trapdoor information - the discrete log relationship between the two Pedersen commitment generators.) $\mathcal{S}_{\texttt{CIDA}}$ sets $[r_{\mathsf{out}}]_{\mathsf{P}_i} = r_{\mathsf{out},i}$.
3. Similarly, $\mathcal{S}_{\texttt{CIDA}}$ changes the output MAC additive share $[\mathsf{MAC}(\mathsf{out})]_{\mathsf{P}_i}$ of one of the honest parties P_i to force the shares to reconstruct to MAC value $\alpha \cdot \mathsf{out}$.
4. $\mathcal{S}_{\texttt{CIDA}}$ then runs OUTPUT as described in $\pi_{\texttt{CIDA}}$ (interacting with the adversarial parties as needed).

Fig. 15. Simulator for MPC with Identifiable Abort

B.3 Simulator for Openable MPC

Proof (Theorem 3). We build a simulator $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ for $\pi_{\texttt{CIDA,AUDIT,OPEN}}$. Much like $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ described in Appendix B.3, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ gains an advantage by providing the CRS from which the Pedersen commitment generators g and h are derived. Like $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$, in order to generate a simulated view, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ follows the protocol $\pi_{\texttt{CIDA,AUDIT,OPEN}}$ closely, with some exceptions, as shown in Figure 16. One such exception is using fixed inputs $\mathsf{in}_i = 0$ for honest parties P_i.

During OPEN, there is assumed to be at least one honest party in the opening coalition. For that honest party P_j, the simulator computes a new share $[\mathsf{in}_i]'_{\mathsf{P}_j}$ and decommitment $r'_{(i,j)}$ such that $[\mathsf{in}_i]'$ reconstructs to in_i and $\mathsf{pc}([\mathsf{in}_i]'_{\mathsf{P}_j}, r'_{(i,j)}) = \mathsf{c}_{(i,j)}$. The simulator then runs OPEN with those values.

C Primitives and Components

In this section we describe the technical tools we use in our constructions (Figures 6 and 7). These are Pedersen commitments (Appendix C.1), verifiable encryption (Appendix C.2) and non-interactive zero-knowledge proofs (Appendix C.3).

C.1 Commitments

Let $\mathcal{M}$ be a message space. A *commitment scheme* [BCC88] consists of three algorithms: SETUP, COMMIT and OPEN.

- $\text{SETUP}(1^k) \to \mathsf{ck}$ generates the public commitment key.
- For any $x \in \mathcal{M}$, $\text{COMMIT}(\mathsf{ck}, x) \to (\mathsf{c}, \mathsf{d})$ produces a commitment/decommitment pair for x.
- $\text{OPEN}(\mathsf{ck}, \mathsf{c}, \mathsf{d}) \to \tilde{x} \in \mathcal{M} \cup \{\perp\}$ opens the commitment, where $\perp$ is returned if c is not a valid commitment.

Simulator $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ for Protocol $\pi_{\texttt{CIDA,AUDIT,OPEN}}$ (Openable MPC)

INIT: 1. $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ provides a CRS such that it knows the discrete log relationship of the two generators g and h of the Pedersen commitment scheme. This is done in such a way that the CRS is indistinguishable from random.

2. $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ simulates $\pi_{\texttt{SETUP}}$ (in a way similar to that described by Baum et. al [BDO14]). During this simulation, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ learns the private decryption key sk_{hom}, recovering the MAC key α.

INPUT: For each party P_i,

1. If P_i is adversarial, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ learns the input in_i and executes INPUT with it as described in $\pi_{\texttt{CIDA,AUDIT,OPEN}}$.
2. If P_i is honest, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ executes INPUT (interacting with the adversarial parties as needed) as described in $\pi_{\texttt{CIDA,AUDIT,OPEN}}$ with a fixed input of 0.

EVAL: $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ evaluates C gate by gate (interacting with the adversarial parties as needed), as described in $\pi_{\texttt{CIDA,AUDIT,OPEN}}$.

OUTPUT: $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ gets the output out from the functionality $\mathcal{F}_{\texttt{CIDA,AUDIT,OPEN}}$.

1. $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ changes the output additive share $[\mathsf{out}]_{\mathsf{P}_i}$ of one of the honest parties P_i to force the shares to reconstruct to out.
2. $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ finds a randomness $r_{\mathsf{out},i}$ such that $\mathsf{pc}([\mathsf{out}]_{\mathsf{P}_i}, r_{\mathsf{out},i}) = \mathsf{c}_{\mathsf{out},i}$. (It can do this since it has the trapdoor information - the discrete log relationship between the two Pedersen commitment generators.) $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ sets $[r_{\mathsf{out}}]_{\mathsf{P}_i} = r_{\mathsf{out},i}$.
3. Similarly, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ changes the output MAC additive share $[\mathsf{MAC}(\mathsf{out})]_{\mathsf{P}_i}$ of one of the honest parties P_i to force the shares to reconstruct to the MAC value $\alpha \cdot \mathsf{out}$.
4. $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ then runs OUTPUT as described in $\pi_{\texttt{CIDA,AUDIT,OPEN}}$ (interacting with the adversarial parties as needed).

AUDIT: $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ runs AUDIT as described in $\pi_{\texttt{CIDA,AUDIT,OPEN}}$.

OPEN: $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ gets the inputs $(\mathsf{in}_1, \ldots, \mathsf{in}_n)$ from the functionality $\mathcal{F}_{\texttt{CIDA,AUDIT,OPEN}}$. The allowable opening coalition $\mathcal{C}$ is assumed to contain at least one honest party P_j. For each input in_i,

1. $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ computes an additive share $[\mathsf{in}_i]'_{\mathsf{P}_j}$ and decommitment $r'_{(i,j)}$ such that $[\mathsf{in}_i]'$ reconstructs to in_i and $\mathsf{pc}([\mathsf{in}_i]'_{\mathsf{P}_j}, r'_{(i,j)}) = \mathsf{c}_{(i,j)}$. (It can do this since it has the trapdoor information - the discrete log relationship between the two Pedersen commitment generators.)

$\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ then runs OPEN as described in $\pi_{\texttt{CIDA,AUDIT,OPEN}}$ (interacting with the adversarial parties as needed).

Fig. 16. Simulator for Openable MPC

A commitment scheme should be *correct*, meaning that for any message $x \in \mathcal{M}$,

$$\textsc{Open}(\mathtt{ck}, \textsc{Commit}(\mathtt{ck}, x)) = x.$$

It should also be *hiding* (meaning that the commitment value $\mathtt{c}$ should reveal nothing about the message x), and *binding* (meaning that for a given commitment value $\mathtt{c}$, it should be hard or impossible to produce two decommitment values $\mathtt{d}_1$, $\mathtt{d}_2$ such that $\perp \neq \textsc{Open}(\mathtt{ck}, \mathtt{c}, \mathtt{d}_1) \neq \textsc{Open}(\mathtt{ck}, \mathtt{c}, \mathtt{d}_2) \neq \perp$.

Our MPC protocols use Pedersen commitments [Ped92] in $\mathbb{QR}_q$, the group of quadratic residues modulo $q = 2p+1$. (This choice of group ensures that the commitment messages space $\mathcal{M}$ is $\mathbb{Z}_p$, just like the message space supported by our secret sharing scheme described in Section 3.) Pedersen commitments work as follows:

- $\textsc{Setup}(1^k)$: Let $q = 2p+1$, and choose two random generators g and h of $\mathbb{QR}_q$. Let $\mathtt{ck} = (q, g, h)$.
- $\mathtt{pc}(\mathtt{ck} = (q, g, h), x)$: Choose a random value $r \leftarrow \mathbb{Z}_p^*$, and set $\mathtt{c} = g^x h^r \bmod q, \mathtt{d} = (x, r)$.
- $\textsc{Open}_p(\mathtt{ck} = (q, g, h), \mathtt{c}, \mathtt{d} = (x, r))$: Return x if $\mathtt{c} = g^x h^r \bmod q$, and return $\perp$ otherwise.

For simplicity, in this paper we often refer to r (instead of (x, r)) as the decommitment value.

C.2 Verifiable Encryption

To achieve openability, we leverage *verifiable encryption*. Verifiable encryption schemes support efficient zero knowledge proofs on ciphertexts. We introduce a slightly modified version of the verifiable encryption scheme described by Camenisch and Shoup [CS03].[12] Our modifications consist solely of removing elements from the ciphertext, so the modified scheme naturally inherits CPA security of the original (but not its CCA security).

- $\textsc{KeyGen}(k)$:
  - Let $n = pq$ where $p = 2p' + 1$ and $q = 2q' + 1$, and p' and q' are k-bit primes.
  - Let $h = 1 + n$.
  - Choose a random generator $g \in Z_{n^2}^*$.
  - Choose a random secret key $sk \in \{1, \ldots, \lfloor (n^2)/4 \rfloor\}$.
  - Let $pk = g^{sk} \bmod n^2$.
- $\textsc{Enc}_{\mathsf{ver}}(pk, x)$:
  - Choose a random $r \in [n/4]$.
  - $e = (g^r \bmod n^2, pk^r h^x \bmod n^2)$.
  - Return the ciphertext e.
- $\textsc{Dec}_{\mathsf{ver}}(sk, e = (u, v))$:
  - $h^x = v/(u^{sk}) \bmod n^2$.
  - Compute x (this is possible for $h = 1 + n$).
  - Return the plaintext x.

This encryption scheme is verifiable, because statements about the underlying plaintext can be proven by executing the zero knowledge proofs described below.

[12] Their scheme is designed it to be secure against chosen ciphertext attacks, which is unnecessary for our purposes.

C.3 (Non Interactive) Zero Knowledge Proofs

To achieve the auditability and openability properties (described in Sections 2.4 and 2.5), we leverage *zero knowledge proofs* [GMR89]. A zero knowledge proof (ZKP) is a two-party protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where $\mathcal{P}$ convinces $\mathcal{V}$ that some claim is true without the verifier learning anything beyond the claim statement.

This section describes two interactive zero knowledge proofs. The proofs can be made non-interactive by means of the Fiat-Shamir heuristic [FS86]. We suggest this concrete non-interactive zero knowledge proof only for the sake of efficiency. It can be replaced with any other non-interactive zero knowledge proof of the same statement.

Figure 17 describes a concrete efficient zero knowledge proof used in our protocol, wherein the prover $\mathcal{P}$ proves knowledge of a committed value. More precisely, $\mathcal{P}$ proves knowledge of an element x and decommitment value r such that $\mathsf{c} = g^x h^r \bmod n$. In Camenisch-Stadler notation [CS97], this can be expressed as:

$$ZKP[(x, r) : \mathsf{c} = g^x h^r \bmod n](n, g, h, \mathsf{c})$$

	(n, g, h, c)	
$\mathcal{P}(x, r)$		$\mathcal{V}$
pick $t_1, t_2 \in \mathbb{Z}_n$		
$T = g^{t_1} h^{t_2}$	$\xrightarrow{T}$	
	$\xleftarrow{c}$	pick $c \in \mathbb{Z}_n$
$s_1 = t_1 + cx \bmod \phi(n)$		
$s_2 = t_2 + cr \bmod \phi(n)$	$\xrightarrow{s_1, s_2}$	check $T\mathsf{c}^c = g^{s_1} h^{s_2} \bmod n$

Fig. 17. Proof of Knowledge of Committed/Encrypted Value

Figure 18 describes another concrete efficient zero knowledge proof used in our construction, wherein the prover $\mathcal{P}$ proves that two commitments are to the same value. (Note that these two commitments do not need to have the same parameters - they can use different moduli and generators.) More precisely, $\mathcal{P}$ proves that $\mathsf{c}_1 = g_1^x h_1^{r_1} \bmod n_1$ and $\mathsf{c}_2 = g_2^x h_2^{r_2} \bmod n_2$ for some x [CM99,Lys02]. In Camenisch-Stadler notation [CS97], this can be expressed as:

$$ZKP[(x, r_1, r_2) : \mathsf{c}_1 = g_1^x h_1^{r_1} \bmod n_1 \wedge \mathsf{c}_2 = g_2^x h_2^{r_2} \bmod n_2] \\ (n_1, n_2, g_1, g_2, h_1, h_2, \mathsf{c}_1, \mathsf{c}_2).$$

$(n_1, n_2, g_1, g_2, h_1, h_2, \mathtt{c}_1, \mathtt{c}_2)$		
$\mathcal{P}(x, r_1, r_2)$		$\mathcal{V}$
pick random l-bit numbers t_1, t_2, t_3		
$T_1 = g_1^{t_1} h_1^{t_2} \bmod n_1$		
$T_2 = g_2^{t_1} h_2^{t_3} \bmod n_2$	$\xrightarrow{T_1, T_2}$	
		pick l-bit[1] random
	$\xleftarrow{c}$	challenge c
$s_1 = t_1 + cx$		
$s_2 = t_2 + cr_1$		
$s_3 = t_3 + cr_2$	$\xrightarrow{s_1, s_2, s_3}$	check that $T_1 \mathtt{c}_1{}^c \equiv g_1^{s_1} h_1^{s_2} \bmod n_1$ and $T_2 \mathtt{c}_2{}^c \equiv g_2^{s_1} h_2^{s_3} \bmod n_2$.

[1] l is a security parameter much larger than the size of x.

Fig. 18. Proof of Equality of Committed/Encrypted Values

Both of the above proofs can be used to prove statements about Pedersen commitments (described in Section C.1) *and* verifiable encryptions (described in Section C.2), since they have the same structure; both consist of a product of two exponentials.