Catching MPC Cheaters: Identification and Openability[*]

Robert Cunningham[1], Benjamin Fuller[2], and Sophia Yakoubov[1]

[1] MIT Lincoln Laboratory
[2] University of Connecticut
rkc@ll.mit.edu, benjamin.fuller@uconn.edu, sophia.yakoubov@ll.mit.edu

Abstract. Secure multi-party computation (MPC) protocols do not completely prevent malicious parties from cheating or disrupting the computation. We augment MPC with three new properties to discourage cheating. First is a strengthening of *identifiable abort*, called *completely* identifiable abort, where all parties who do not follow the protocol will be identified as cheaters by each honest party. The second is *completely identifiable auditability*, which means that a third party can determine whether the computation was performed correctly (and who cheated if it was not). The third is *openability*, which means that a distinguished coalition of parties can recover the MPC inputs.

We construct the first (efficient) MPC protocol achieving these properties. Our scheme is built on top of the SPDZ protocol (Damgard et al., Crypto 2012), which leverages an offline (computation-independent) pre-processing phase to speed up the online computation. Our protocol is *optimistic*, retaining online SPDZ efficiency when no one cheats. If cheating does occur, each honest party performs only local computation to identify cheaters.

Our main technical tool is 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.

The work of Baum, Damgård, and Orlandi (SCN 2014) introduces the concept of auditability, which allows a third party to verify that the computation was executed correctly, but not to identify the cheaters if it was not. We enable the third party to identify the cheaters by augmenting the scheme with CESS. We add openability through the use of verifiable encryption and specialized zero-knowledge proofs.

1 Introduction

Secure multi-party computation (MPC) allows multiple parties to evaluate a function of their secret inputs while maintaining their privacy. In this work, we focus on preventing malicious behavior that is not prevented by the guarantees of traditional MPC.

Completely Identifiable Abort In traditional MPC, a malicious party cannot cause the computation to return an incorrect output, but it *can* cheat by deviating from the protocol and causing a termination with an error, known as an abort.[1] Since the cheater remains anonymous, it does not face any consequences for its actions. There is no point in recomputing the function, as

[*] 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.

[1] In this work we consider an arbitrary number of malicious parties. In this setting, it is impossible to guarantee termination without error [Cle86].

the honest parties do not know who to exclude. In order to avoid such stalemates, it is desirable to be able to identify the cheaters. An *MPC with identifiable abort* [BS90,WW95,CFN94,IOZ14] guarantees that all honest parties agree on a subset of cheating parties.[2] We introduce MPC with *completely* identifiable abort, which guarantees that all honest parties agree on *all* cheating parties.

Completely Identifiable Auditability Baum et. al [BDO14] define *auditability*, which enables any third party to verify the correctness of an MPC given a public transcript of the computation, and the output it produced. We introduce *completely identifiable* auditability, which allows the third party auditor to identify all the cheating parties if the computation was not carried out correctly.

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

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

1.1 Our construction

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. The online portion of SPDZ is very efficient, using fast information-theoretic tools. For n users, the online communication cost is $O(n^2m)$ messages where m is the number of multiplications evaluated.

Like SPDZ, our construction is secure in the presence of a malicious adversary statically making arbitrarily many corruptions. Our protocol is *optimistic*; if cheating does not occur, the online communication and computation are very efficient. Our online protocol has only twice the online complexity of SPDZ. In particular, the communication complexity remains $O(n^2m)$. If cheating does occur, each party must perform an additional local computation whose complexity is $O(nm)$ in order to identify the cheaters. The additional local work uses computationally secure tools, which are slower.

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

Adding Completely Identifiable Abort We design a new locally identifiable secret sharing scheme (as defined by [IOS12]) which we call *commitment-enhanced secret sharing (CESS).* A locally identifiable secret sharing scheme is a secret sharing scheme where, after reconstruction, all honest parties agree on the set of parties that modified their shares [IOS12].

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

Each CESS share contains an additive share (as in SPDZ), and additionally includes linearly homomorphic commitments to *every* additive share.[3] Each CESS share also contains the decommitment value for the commitment to the corresponding additive share. This conceptually simple change of giving each party a commitment to every additive share allows identification of cheaters. When CESS shares are used for computation, 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.

On the Use of Broadcast Unlike SPDZ, our protocol requires a fully secure broadcast channel. Secure (or *authenticated*) broadcast is a very expensive primitive; constructions typically require $O(n^2)$ messages, and use public key primitives. Secure broadcast is used by our protocol *only* in the Input operation, a total of n times.

The reason we can use broadcast so sparingly is that, because our CESS shares are checkable, there is no need to securely broadcast them. Instead, we implement a specific, *optimistic* share broadcast protocol. Since the validity of each decommitment can be checked, the parties send one another their respective additive shares and decommitments. Any honest party P_i receiving an incorrect additive share and decommitment from P_j can send help requests to all other parties asking for the correct values. If any honest parties received correct values from P_j, they will forward those to P_i, solving the problem. Otherwise, all honest parties will agree that P_j cheated, and abort the protocol. If cheating does not occur, each CESS reconstruction takes only n^2 messages. (If cheating does occur, up to an additional $2n(n-1)t$ messages may be required, where t is the number of corrupt parties.)

Related Work on SPDZ with identifiable abort Recent works by Baum et. al [BOS16] and Spini and Fehr [SF16] add identifiable abort to the SPDZ protocol.[4] In Table 1, we compare our efficiency to theirs, both in the case that cheating does not occur and in the case that it does. We improve on the number of broadcast messages for both schemes in all cases.

Baum et. al augment SPDZ with identifiable abort using a homomorphic information-theoretic signature scheme. 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.[5] Our use of computational techniques necessitates a larger field (not necessary in Baum et. al [BOS16]). This makes the length of each message depend logarithmically on the security parameter. Baum et. al [BOS16] (and Spini and Fehr [SF16]) assume a broadcast channel, which may mitigate the efficiency advantage of the smaller field. Implementing broadcast often uses signatures, which already adds a logarithmic dependence in the security parameter to all messages.

Spini and Fehr [SF16] take a different approach. Their approach is based on dispute control. If no cheating occurs, they retain the exact online efficiency of SPDZ (while our protocol is slower by a factor of 2). If cheating does occur, they have two forms of identification. Their protocol can ensure that each honest party knows the identity of some cheating party by doubling the cost of SPDZ. At an additional cost, they can ensure that all honest parties *agree* on the identity of some cheating

[3] We use Pedersen commitments [Ped92] to enable efficient zero-knowledge proofs.

[4] An alternative approach uses bitcoin to introduce financial repercussions for cheating [KB14,ADMM16].

[5] 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.

		cheating?	our scheme	Baum et. al	Spini and Fehr
# messages	broadcast (bc)	no	n	$n(n+2m+2)$	$O(n^2)$
	point-to-point		$2(m+1)n(n-1)$	$2mn(n-1)$	$O(mn)$
	broadcast (bc)	yes (worst case)	n	$n(n+2m+2)$	$O(n^2)$
	point-to-point		$2(m+1)n(n-1)(2t+1)$	$2mn(n-1)$	$O(mn)$
public key operations/party		no	none, except in broadcast	none, except in broadcast	none, except in broadcast
		yes (worst case)	$O(nm)$	none, except in broadcast	$O(nm)$

Table 1. Online Complexity of Protocols with Identifiable Abort. If cheating does occur we consider the worst case complexity of the protocol. Spini and Fehr move through multiple different protocols of decreasing efficiency, we assume if cheating occurs they are forced to use the most expensive protocol.

party. Spini and Fehr require multiple rounds of blame assignment, in contrast to our conceptually simple approach.

If the prospect of being identified is likely to discourage all cheating in the first place, the Spini and Fehr protocol achieves better efficiency. Our protocol is more prudent if cheating may still occur, for example in the setting of multiple independent malicious actors. Also, as we discuss next, our protocol allows for an outside observer to identify cheaters. Public identification is not possible using the protocol of Spini and Fehr.

Auditability Openability relies on a property called *auditability*, introduced by Baum, Damgård, and Orlandi [BDO14]. They add a public transcript τ to SPDZ (modeled as a public append-only bulletin board) to allow external parties to check protocol correctness even if all participants are malicious. The public transcript τ contains Pedersen commitments [Ped92] to each precomputed Beaver triple value and input in_i.

The transcript τ contains all values reconstructed during the computation; namely the Beaver triple differences, described in Section 4. Though τ does not contain Pedersen commitments to intermediate computation values or to the output, these commitments can be derived using the linear homomorphism of Pedersen commitments and the posted Beaver triple differences. 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'.

Completely Identifiable Auditability Our construction includes commitments for each input *share*, while the construction of Baum et. al [BDO14] includes a single commitment for each *input*. This increases the number of committed values, but does not affect the online communication complexity. The additional commitments enable *completely identifiable* auditability, meaning that in addition to public auditing of the protocol correctness, we can support public identification of cheaters. We also rely on auditability to add openability; openability is unachievable without an audit check to ensure that the transcript is well formed.

Adding Openability An openable MPC protocol allows a distinguished *opening coalition* to recover the computation inputs. This is useful in case there is cause to doubt the truthfulness of these inputs. One might think that having commitments to the inputs would be enough — each party can the "open" by providing the decommitment, and anyone who does not cooperate is identified as a cheater. However, we cannot rely on the input owners to cooperate with the opening. While it is tempting to simply identify parties who do not cooperate as cheaters, they might actually be honest. The adversary might be blocking their messages, or their opening values might have been destroyed by the event that caused doubts about the input veracity. (An example of such a situation is discussed in more detail in Section 1.2; in the case of a satellite collision, the collision may have destroyed the data stored on the satellites involved.)

Our approach to building openability is to encrypt each MPC input and prove consistency with the public transcript τ of the MPC protocol. This encryption scheme must have two properties: (1) the message must only be readable by an allowed coalition, and (2) the proof of consistency must be efficient. A threshold encryption scheme splits the secret key between a coalition of parties and only the entire coalition (or an allowed subset) can jointly decrypt a ciphertext. A verifiable encryption scheme allows efficient proofs about the underlying plaintext [FO97,Bou00,DF02,CS03].

We design a new threshold verifiable encryption scheme which, to the best of our knowledge, is the first *universally composable* [Can00] threshold verifiable encryption scheme. Our scheme uses a variant of the scheme described by Camenisch and Shoup [CS03]. The scheme of Camenisch and Shoup cannot be universally composable, since a ciphertext commits to the underlying plaintext in a perfectly binding way. Simulating decryption would involve breaking this commitment, which even a powerful simulator cannot do. We avoid this problem using a layer of secret sharing and commitments that are only *computationally* binding.

Note that, in our construction, it is possible to open some inputs while maintaining the privacy of others. This is very useful in the case when there is cause to suspect some parties of lying, but not others.

1.2 A Motivating Example

In this section we present satellite conjunction analysis [HWB14] as a motivating use case for our augmented MPC.[6] Those readers who are convinced of the need to catch MPC cheaters may proceed to Section 1.2.

Multiple government organizations and companies own satellites. The purpose of many satellites is secret, so organizations are not willing to share their trajectories. However, there is risk to not sharing trajectory 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 trajectories. 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,HLOI16].

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 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

[6] Other sensitive applications include economic markets [BCD$^+$09] and elections [BDO14].

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. Once inputs are opened, of the organizations whose satellites were involved in the collision, whoever's claimed input trajectory does not intersect witht he collision location is the one to blame.

Complete identifiable abort, completely identifiable auditability and openability make MPC much more appealing, especially for high-risk applications such as determining the likelihood of satellite collisions. The increased accountability dis-incentivizes cheating, and increases all parties' trust in the computation output.

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. In Section 5 we construct our MPC protocol with completely identifiable abort. In Section 6 we introduce a universally composable threshold verifiable encryption scheme. Finally, in Section 7 we add openability using theshold verifiable encryption. In Appendix A, we present the preprocessing phase of our scheme. In Appendix Bwe 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. We assume the availability of a secure broadcast channel for unit cost. If a broadcast channel is not naturally available, it can be implemented using digital signatures. We make very sparing use of the broadcast channel; in fact, values need only be broadcast once per input. This is because all other values that may need to be broadcast are secret shares, and we do not require the full power of a secure broadcast channels for those, as discussed in Section 1.1.

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.[7]

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

Functionality $\mathcal{F}$

INIT: On input $(\textsc{Init}, C, p)$ 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 party indices.
3. Store $\mathsf{out} = \perp$.

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

EVAL: On input $(\textsc{Eval})$ 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 as out.

OUTPUT: On input $(\textsc{Output})$ from all parties:
1. Send out to all adversarially controlled parties P_i for $i \in \mathcal{I}$.
2. Run ABORT, waiting for each adversarially controlled party to provide an input.
3. Output out to all parties, where out may now be $\perp$.

ABORT: On input (x_i) from each adversarial party P_i, $i \in \mathcal{I}$:
1. If $x_i \neq \mathsf{ACCEPT}$ for some $i \in \mathcal{I}$: set $\mathsf{out} = \perp$.

Fig. 1. Ideal Functionality for MPC.

allows an adversary to violate fairness by causing an abort. The ideal functionality for secure MPC with abort is shown in Figure 1.

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 $\mathcal{F}_{\texttt{CIDA,AUDIT,OPEN}}$ 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 outputting τ.

More formally, Baum et. al introduce the AUDIT algorithm. AUDIT 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.

[7] 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 $\mathcal{F}_{\texttt{CIDA,AUDIT,OPEN}}$
INIT: as in $\mathcal{F}$. Additionally, receive and record the set of allowable coalitions $\{\mathcal{C}_i\}$ from all parties. Set $L_{cheat} = \emptyset$.
INPUT: as in $\mathcal{F}$. Additionally, run ABORT, waiting for each adversarially controlled party to provide an input. If $L_{cheat} \neq \emptyset$, output $(\perp, L_{cheat})$ to all parties.
EVAL: as in $\mathcal{F}$. Additionally, run ABORT, waiting for each adversarially controlled party to provide an input. If $L_{cheat} \neq \emptyset$, output $(\perp, L_{cheat})$ to all parties.
OUTPUT: as in $\mathcal{F}$.
AUDIT: On input $(\text{AUDIT}, \mathsf{out}')$ from a third party auditor:
    if EVAL was not executed, output NO AUDIT POSSIBLE.
    else if $\mathsf{out} = (\perp, L_{cheat})$, output $(\mathsf{REJECT}, L_{cheat})$.
    else if $\mathsf{out}' = \mathsf{out}$, output ACCEPT.
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 $\mathsf{out} \neq \perp$: return $\mathsf{in}_1, \ldots, \mathsf{in}_n$.
    else: return $\perp$.
ABORT: On input $(x_i)$ from each adversarial party $\mathsf{P}_i$, $i \in \mathcal{I}$:
    1. For each $i \in \mathcal{I}$ such that $x_i \neq \mathsf{ACCEPT}$, add $\mathsf{P}_i$ to $L_{cheat}$.
    2. Set $\mathsf{out} = (\perp, L_{cheat})$.
```

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

Definition 1 (Auditable Correctness [BDO14]). *An MPC protocol satisfies* Auditable Correctness *if for all circuits C and for all potential outputs* out*,*

- $\text{AUDIT}(\tau, C, \mathsf{out}) = 1$ *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*
- $\text{AUDIT}(\tau, C, \mathsf{out}) = 0$ *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* $\mathsf{in}_1, \ldots, \mathsf{in}_n$*.*

Completely Identifiable Auditability We add *completely identifiable* auditability, which allows a third party to identify all cheaters if the protocol was not executed correctly. Informally, we say that an MPC protocol satisfies *Completely Identifiable Auditable Correctness* if it satisfies auditable correctness and, when AUDIT outputs 0, it additionally outputs L_{cheat} (a list of all cheaters). Figure 2 describes this enhanced AUDIT protocol.

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* next.

2.5 Openability

In extreme cases, it may be necessary to open the inputs of an MPC evaluation (see Section 1.2 for a motivating example). 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. Two reasonable examples of 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 $\textsc{Audit}(\tau, C, \mathsf{out}) = 1$ and $\textsc{Open}(\tau) \neq (\mathsf{in}_1, \ldots, \mathsf{in}_n)$ is negligible.*

Figure 2 describes the ideal functionality $\mathcal{F}_{\mathtt{CIDA,AUDIT,OPEN}}$ 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 to add completely identifiable abort to MPC. We first describe basic secret sharing (Section 3.1), and then 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: $\textsc{Share}$ and $\textsc{Rec}$.

- $\textsc{Share}(x) \to (s_1, \ldots, s_n)$ takes in a secret x and produces the n secret shares.
- $\textsc{Rec}(s_{i_1}, \ldots, s_{i_t}) \to \tilde{x}$ 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. $\mathsf{SS}_{\mathsf{Add}}.\textsc{Share}(x)$ 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. $\mathsf{SS}_{\mathsf{Shamir}}.\textsc{Share}(x)$ 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 [TW87].[8] 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 (L_{cheat}), 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 $\mathcal{I} \subsetneq \{1, \ldots, n\}$ 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 ($\tilde{x}_i = (\mathsf{REJECT}, L_{cheat} = B)$ for all $i \in H$).*

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. $(s_1, \ldots, s_n) \leftarrow \textsc{Share}(x)$.

[8] 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.

2. *$\mathcal{A}$ outputs a set $\mathcal{I} \subsetneq \{1, \ldots, n\}$ 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. $\mathsf{simout} = \{\tilde{x}_i\}_{i \in \mathcal{I}}$ *(where $\tilde{x}_i$ is either a reconstructed value, or $(\mathsf{REJECT}, L_{cheat})$).*

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:

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

where
- $[x]_{\mathsf{P}_i}$ is the additive secret share of x belonging to P_i (as described in Section 3.1),
- $[r_x]_{\mathsf{P}_i}$ 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, and
- $\mathsf{c}_{x,i}$ 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).

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.

We additionally describe *private reconstruction* $\mathsf{CESS}.\textsc{PrivRec}$ (Figure 4), which describes how a value can be reconstructed by only one party, while everyone still agrees on the cheaters' identities.[9] The sole object of all but one parties performing $\mathsf{CESS}.\textsc{PrivRec}$ is to compile the correct L_{cheat}, not to reconstruct the value in question. In order to support private reconstruction $\mathsf{CESS}.\textsc{PrivRec}$, the shares can't be used for reconstruction directly; rather, related values must be derived from the shares. We call this process $\mathsf{CESS}.\textsc{PrivOpen}$.

[9] We do not extend the definition of a locally identifiable secret sharing scheme to support private opening; rather, we just describe the functionality. We leave a formal definition to future work.

```
Protocol $\mathsf{CESS}.\textsc{Rec}$ for the Reconstruction of a CESS $\langle\rangle$-Sharing

PRECONDITIONS: Each party $\mathsf{P}_i$ has $\langle x\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}))$.
$\mathsf{CESS}.\textsc{Rec}(i, ([x]_{\mathsf{P}_1}, [r_x]_{\mathsf{P}_1}), \ldots, ([x]_{\mathsf{P}_n}, [r_x]_{\mathsf{P}_n}), (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}))$:
    1. Let $L_{cheat} = \emptyset$.
    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 $\mathsf{P}_j$ to $L_{cheat}$.
    3. If $L_{cheat}$ is empty: return $x = \sum_{j\in\{1,\ldots,n\}} [x]_{\mathsf{P}_j}$.
    4. Else: return $(\mathsf{REJECT}, L_{cheat})$.
```

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

```
Protocol $\mathsf{CESS}.\textsc{PrivRec}$ for the Private Reconstruction of a CESS $\langle\rangle$-Sharing by Party $\mathsf{P}_j$.

PRECONDITIONS:
    1. Each party $\mathsf{P}_i$ has $\langle x\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}))$.
    2. Each party has a private decryption key $sk_i$, and knows all parties' public encryption keys $\{pk_j\}_{j\in[1,\ldots,n]}$.
       The encryption and decryption keys should use a verifiable encryption scheme, as described in Section 6.
$\mathsf{CESS}.\textsc{PrivOpen}(i, j, \langle x\rangle_{\mathsf{P}_i} = ([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i}, (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n})))$: Party $\mathsf{P}_i$ does the following to enable only $\mathsf{P}_j$ to recover
  the input, but all parties to compile $L_{cheat}$:
    1. Computes a verifiable encryption $e_{x,i}$ of $[x]_{\mathsf{P}_i}$ using $\mathsf{P}_j$'s public key.
    2. Computes a verifiable encryption $e_{x,i,r}$ of $[r_x]_{\mathsf{P}_i}$ using $\mathsf{P}_j$'s public key.
    3. Computes a proof $\pi_i$ that $e_{x,i}$ encrypts the value committed in $\mathsf{c}_{x,i}$, and that $e_{x,i,r}$ encrypts the decommit-
       ment value of $\mathsf{c}_{x,i}$.
    4. Returns $(e_{x,i}, e_{x,i,r}, \pi_i)$.

$\mathsf{CESS}.\textsc{PrivRec}(i, j, ((e_{x,1}, e_{x,1,r}, \pi_1), \ldots, (e_{x,n}, e_{x,n,r}, \pi_n)), (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}))$:   1. Let $L_{cheat} = []$.
    2. For $k \in \{1, \ldots, n\}$: Check the proof $\pi_k$. If the proof does not verify, add $\mathsf{P}_k$ to $L_{cheat}$.
    3. If $L_{cheat}$ is empty:
       (a) If $i = j$:
             i. For $k \in \{1, \ldots, n\}$: Decrypt $e_{x,k}$ to learn $[x]_{\mathsf{P}_k}$.
            ii. Return $x = \sum_{k\in\{1,\ldots,n\}} [x]_{\mathsf{P}_k}$.
       (b) Else: return $\perp$.
    4. Else: return $(\mathsf{REJECT}, L_{cheat})$.
```

Fig. 4. Protocol PrivRec for the Private Reconstruction of a $\langle\rangle$-Sharing

Theorem 1. *Assuming that the commitment scheme* $\mathbf{p}$ *is secure, the CESS scheme is a locally identifiable secret sharing scheme (LISS) as described in Definition 3.*

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

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 $\mathtt{d}_i$.
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 $(\mathsf{REJECT}, L_{cheat})$.
7. If $\sum_{i \in \{1,\dots,n\}} f_i \neq 0$, the parties return REJECT.

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

Fig. 5. Protocol MACCheck for Checking a MAC Without Reconstructing the Key

not verify, SimRec adds P_i to L_{cheat}; it then returns $(\mathsf{REJECT}, L_{cheat})$. If all of the decommitments do verify (meaning that none of the shares could have been altered), SimRec returns $\mathsf{success}$.

4 Adapting CESS 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 5.

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

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

where $\mathsf{c}_{x,i} = \mathsf{pc}([x]_{\mathsf{P}_i}, [r_x]_{\mathsf{P}_i})$, all of $(\mathsf{c}_{x,1}, \dots, \mathsf{c}_{x,n})$ 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 6.

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

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

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 5, 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 $\mathsf{CESS}.\textsc{Rec}(i, ([x]_{\mathsf{P}_1}, [r_x]_{\mathsf{P}_1}), \ldots, ([x]_{\mathsf{P}_n}, [r_x]_{\mathsf{P}_n}), (\mathsf{c}_{x,1}, \ldots, \mathsf{c}_{x,n}))$ as described in Figure 3.

Fig. 6. 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].

Note that a party *can* cause MACCheck to fail, while being honest about all other values, without being identified. If a party does this, $\mathsf{CESS}_{\mathsf{MAC}}.\textsc{Rec}$ will still produce the correct output, but the parties will be forced to execute the more expensive $\mathsf{CESS}.\textsc{Rec}$. In this situation, our cheating party forces the participants to waste computational resources; but, since the reconstruction still succeeds, we do not require that they be identified.

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 6 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 Malicious-Majority MPC with Identifiable Abort

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 Acontains 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* reconstructed as s by P_i using CESS.PrivRec (Figure 4). (All parties will catch anyone providing malicious shares.)
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 $\langle\langle x_1\rangle\rangle_{\mathsf{P}_j}$, $\langle\langle x_2\rangle\rangle_{\mathsf{P}_j}$: 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 $\langle\langle x_1\rangle\rangle, \langle\langle x_2\rangle\rangle$: 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, The parties execute the reconstruction procedure $\mathsf{CESS}_{\mathsf{MAC}}$.Rec described in Figure 6, catching any parties providing malicious shares.

Fig. 7. MPC with Completely 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 setup functionality $\mathcal{F}_{\mathtt{SETUP}}$ that generates $\langle\langle\rangle\rangle$ sharings of random numbers and Beaver triples.[10] We describe a secure instantiation $\pi_{\mathtt{SETUP}}$ of $\mathcal{F}_{\mathtt{SETUP}}$ in Appendix A.

[10] 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.

Figure 7 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 6, 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 $\pi_{\mathtt{SETUP}}$.

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 5. 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 received shares of the difference values ϵ and δ from Beaver triple multiplications performed throughout the computation.[11] 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 $\pi_{\mathtt{SETUP}}$.

6 UC Threshold Verifiable Encryption

To achieve openability, we leverage *threshold verifiable encryption.* Verifiable encryption schemes support efficient zero-knowledge proofs on ciphertexts; *threshold* verifiable encryption schemes additionally require (at least a threshold number of parties in) a coalition $\mathcal{C}$ in order to preform decryption.

We leverage a 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). The modified version of their scheme consists of the following algorithms:

[11] Note that if a public transcript τ is maintained, it contains all of these difference values.
[12] Their scheme is secure against chosen ciphertext attacks, which is unnecessary for our purposes.

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

This encryption scheme is verifiable, because statements about the underlying plaintext can be proven using efficient zero-knowledge proofs (Appendix C.2). It can also be instantiated as a threshold verifiable encryption scheme, by secret-sharing the secret key among a coalition $\mathcal{C}$, and performing decryption in a distributed way using that secret shared key.

However, this candidate threshold verifiable encryption scheme is not universally composable. Informally, for each honest party, the simulator needs to produce an encryption e_i of their input in_i without knowing in_i. Since the encryption scheme is perfectly binding, the simulator would be unable to produce encryptions that decrypt 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:

```
$\textsc{ThreshVer.KeyGen}(1^k, i)$:
    Run $(pk_i, sk_i) \leftarrow \textsc{Ver.KeyGen}(1^k)$.
$\textsc{ThreshVer.Enc}(\{pk_i\}_{i \in \mathcal{C}}, x)$:
    1. Additively share $x$ into $|\mathcal{C}|$ values, denoted $[x]$.
    2. For each $i \in \mathcal{C}$:
        (a) Choose a random value $r_i$ and compute the commitment $\mathsf{c}_i = \mathsf{pc}([x]_{\mathsf{P}_i}, r_i)$.
        (b) Encrypt $[x]_{\mathsf{P}_i}$ as $e_i \leftarrow \textsc{Ver.Enc}(pk_i, [x]_{\mathsf{P}_i})$.
        (c) Encrypt $r_i$ as $e_{i,r} \leftarrow \textsc{Ver.Enc}(pk_i, r_i)$.
        (d) Compute the following non-interactive zero-knowledge proofs (Appendix C.2):
            i. Proof $\pi_i$ that $e_i$ encrypts the value in $\mathsf{c}_i$.
            ii. Proof $\pi_{i,r}$ that $e_{i,r}$ encrypts the decommitment value $r_i$ for $\mathsf{c}_i$.
    3. Return $e = \{\mathsf{c}_i, e_i, e_{i,r}, \pi_i, \pi_{i,r}\}_{i \in \mathcal{C}}$.
$\textsc{ThreshVer.Dec}(e)$:
    1. Each party $\mathsf{P}_i$ $(i \in \mathcal{C})$:
        (a) Decrypts $e_i$ and $e_{i,r}$: $[x]_{\mathsf{P}_i} = \textsc{Ver.Dec}(sk_i, e_i)$, $r_i = \textsc{Ver.Dec}(sk_i, e_{i,r})$.
        (b) Sends $[x]_{\mathsf{P}_i}$ and $r_i$ to all other parties.
    2. All parties check that $\mathsf{c}_i = \mathsf{pc}([x]_{\mathsf{P}_i}, r_i)$. If not, REJECT.
    3. All parties reconstruct $x$ using $[x]$.
```

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

INIT: Same as in $\pi_{\texttt{CIDA}}$ (Figure 7). Additionally,

1. All parties P_i in the opening coalition $\mathcal{C}$ generate a 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 $\pi_{\texttt{CIDA}}$ (Figure 7). Additionally, let c_i be the product of the commitments to the additive shares of input in_i in the $\langle\langle\rangle\rangle$-sharing of in_i. (So, c_i is a commitment to in_i.) Party P_i does the following:

1. Computes a threshold verifiable encryption $e_i \leftarrow \text{ThreshVer.Enc}(\{pk_j\}_{j\in\mathcal{C}}, \mathsf{in}_i)$, and posts it to the transcript τ.
2. Computes π_i, a non-interactive zero-knowledge proof (Appendix C.2) that the encrypted value in e_i is the same as the committed value in c_i, and publishes it to the transcript τ.

EVAL: Same as in $\pi_{\texttt{CIDA}}$ (Figure 7); however, all broadcast values are logged to the transcript τ.

OUTPUT: Same as in $\pi_{\texttt{CIDA}}$ (Figure 7); however, all broadcast values are logged to the transcript τ.

AUDIT: The auditor receives the transcript τ, together with the additive secret shares $[\mathsf{out}]$ and $[r_{\mathsf{out}}]$. They then:

1. Check the non-interactive zero-knowledge proofs in the threshold verifiable encryptions e_i.
2. Check the non-interactive zero-knowledge proofs π_i.
3. 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. (This can be done exactly as described in Section 4.2, but ignoring all secret share values except the commitments.)The auditor sets $L_{cheat} = \emptyset$. For each party P_i, if the output share commitment $\mathsf{c}_{\mathsf{out},i} \neq \mathsf{pc}([\mathsf{out}]_{\mathsf{P}_i}, [r_{\mathsf{out}}]_{\mathsf{P}_i})$, the auditor adds P_i to L_{cheat}. If at the end $L_{cheat} = \emptyset$, the auditor outputs ACCEPT. Otherwise, they output $(\mathsf{REJECT}, L_{cheat})$.

OPEN: For each input in_i, the parties in $\mathcal{C}$ retrieve e_i from τ, and run $\text{ThreshVer.Dec}(e_i)$.

Fig. 8. Openable MPC with Identifiable Abort

Informally, no party can cause incorrect decryption without a REJECT because that would involve breaking the binding property of the Pedersen commitment scheme.

However, a UC simulator can force decryption to return a desired value. The simulator only needs to open a commitment to an appropriate share of x, not an encryption. Since Pedersen commitments are only computationally binding, the simulator can commit to an arbitrary value, and break the binding property (using a trapdoor) to open to the share of x.[13]

Recall that a verifiable encryption scheme supports proofs about the underlying plaintext. Because multiple encryption public keys are now involved, we compute proofs relative to the commitments c_i, not the ciphertexts e_i. A commitment to the value x can be computed by taking the product of the additive share commitments: $\mathsf{c} = \prod_{i\in\mathcal{C}} \mathsf{c}_i$. Proofs like those described in Appendix C.2 can then be done with respect to c.

Efficiency Notice that this encryption scheme is very inefficient: a ciphertext consists of $O(|\mathcal{C}|)$ elements, as opposed to $O(1)$ elements as in the scheme of Camenisch and Shoup [CS03]. However, as we show in Section 7, we only use this scheme to encrypt computation inputs. This additional work is independent of the size of the computed circuit f.

7 Openable Auditable MPC

In this section, we augment our construction from Section 5 with completely identifiable auditability and openability. Completely identifiable auditability is achieved by logging all public values (including commitments from setup and publicly opened difference values) from the construction

[13] The simulator chooses the generators used in the Pedersen commitment scheme when selecting the CRS; he does so in such a way that he knows their discrete log relationship, which serves as his trapdoor.

in Section 5 to the public transcript τ. As in Baum et. al [BDO14], the input share commitments together with the public values can be used to obtain a commitment to the output shares, and those commitments can be checked against the claimed output share and decommitment values.

In order to add openability, we leverage threshold verifiable encryption (Section 6). The augmented construction is shown in Figure 8. Informally, each party encrypts their input in such a way that the opening coalition can decrypt it, publishes the resulting ciphertext to the transcript τ, and proves that this ciphertext encrypts the input used in the computation.

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* ($\textsc{ThreshVer.KeyGen}$, $\textsc{ThreshVer.Enc}$, $\textsc{ThreshVer.Dec}$) *is a semantically secure verifiable encryption scheme, the protocol $\pi^{\mathcal{F}_{\mathtt{SETUP}}}_{\mathtt{CIDA,AUDIT,OPEN}}$ with oracle access to the functionality $\mathcal{F}_{\mathtt{SETUP}}$ is a UC-secure implementation of the functionality $\mathcal{F}_{\mathtt{CIDA,AUDIT,OPEN}}$.*

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

Efficiency To achieve completely identifiable auditability, no additional values need to be computed at all. As stated above, values that were previously broadcast are now additionally posted to the transcript.

Openability requires one additional threshold verifiable encryption (Section 6) to each input. Each threshold verifiable encryption entails one additive secret-sharing (to the opening coalition $\mathcal{C}$), and two verifiable encryptions and a commitment (described in Appendix C.1) for each share. Additionally, $2|\mathcal{C}| + 1$ non-interactive zero-knowledge proofs (described in Appendix C.2) 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 work of Benjamin Fuller was done in part at MIT Lincoln Laboratory.

The authors would like to thank Carsten Baum, Mayank Varia, Samuel Yeom, 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, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009: 13th International Conference on Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343, Accra Beach, Barbados, February 23–26, 2009. Springer, Heidelberg, Germany.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14: 9th International Conference on Security in Communication Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196, Amalfi, Italy, September 3–5, 2014. Springer, Heidelberg, Germany.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[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/2016/187`.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

[BS90] Ernest F. Brickell and Douglas R. Stinson. The detection of cheaters in threshold schemes (rump session). In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 564–577, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Heidelberg, Germany.

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

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 257–270, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Heidelberg, Germany.

[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 J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 413–430, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany.

[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, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[DF02] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany.

[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 Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013: 18th European Symposium on Research in Computer Security*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18, Egham, UK, September 9–13, 2013. Springer, Heidelberg, Germany.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[HLOI16] Brett Hemenway, Steve Lu, Rafail Ostrovsky, and William Welser IV. High-precision secure computation of satellite collision probabilities. Cryptology ePrint Archive, Report 2016/319, 2016. `http://eprint.iacr.org/2016/319`.

[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 Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 21–38, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 369–386, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[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 Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14: 21st Conference on Computer and Communications Security*, pages 30–41, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[KOO95] Kaoru Kurosawa, Satoshi Obana, and Wakaha Ogata. t-Cheater identifiable (k, n) threshold secret sharing schemes. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 410–423, Santa Barbara, CA, USA, August 27–31, 1995. Springer, Heidelberg, Germany.

[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 P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture*

Notes in Computer Science, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st Annual ACM Symposium on Theory of Computing*, pages 73–85, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[SF16] Gabriele Spini and Serge Fehr. Cheater detection in spdz multiparty computation. In *Information Theoretic Security: 9th International Conference, ICITS 2016, Tacoma, WA, USA, August 9-12, 2016, Revised Selected Papers 9*, pages 151–176. Springer, 2016.

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

[TW87] Martin Tompa and Heather Woll. How to share a secret with cheaters. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 261–265, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[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 15, 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 $\langle\langle\rangle\rangle$-sharings. $\pi_{\texttt{SETUP}}$ must do the following:

1. Establish an additively secret-shared MAC key α,
2. Generate random $\langle\langle\rangle\rangle$-shared values to be used during Input, and
3. Generate $\langle\langle\rangle\rangle$-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 (HKeyGen, HEnc, 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. (HKeyGen, HEnc, 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 9. 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 10, 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 11, 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 12), which generates a $\langle\langle\rangle\rangle$-shared random value.

Next, we use these protocols to construct MultSecretSharedValues, a protocol that enables 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 14) and is less efficient than multiplication *using* Beaver triples.

MultSecretSharedValues is described in Figure 13. Finally, π_{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 (REJECT, L_{cheat}).
   (e) Using the homomorphic properties of the encryption scheme, computes the encryption e_f of $f = \sum_{i\in\{1,\dots,n\}} f_i$, 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 ($i \in \{2, \dots, n\}$) sets $[x]_{\mathsf{P}_i} = -f_i$.

Fig. 9. 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 $(\mathsf{REJECT}, L_{cheat})$.
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. 10. 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 $\mathcal{S}_{\mathtt{CIDA}}$ for $\pi_{\mathtt{CIDA}}$. $\mathcal{S}_{\mathtt{CIDA}}$ gains an advantage by providing the CRS from which the Pedersen commitment generators g and h are derived. $\mathcal{S}_{\mathtt{CIDA}}$ can provide a CRS such that it will know the discrete log relationship between the two generators, thus allowing $\mathcal{S}_{\mathtt{CIDA}}$, 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, $\mathcal{S}_{\mathtt{CIDA}}$ follows the protocol $\pi_{\mathtt{CIDA}}$ 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. 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.

B.3 Simulator for Openable MPC

Proof (Theorem 3). We build a simulator $\mathcal{S}_{\mathtt{CIDA,AUDIT,OPEN}}$ for $\pi_{\mathtt{CIDA,AUDIT,OPEN}}$. Much like $\mathcal{S}_{\mathtt{CIDA,AUDIT,OPEN}}$ described in Appendix B.3, $\mathcal{S}_{\mathtt{CIDA,AUDIT,OPEN}}$ gains an advantage by providing the CRS from which

RESHARE: Protocol for $\langle\langle\rangle\rangle$-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 10), 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 $\langle\langle\rangle\rangle$-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 $(\mathsf{REJECT}, L_{cheat})$.
   (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 9, 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. 11. RESHARE: Resharing an Element x Given its Ciphertext e_x (A Sub-Protocol of π_{SETUP}, described in Figure 15)

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 10), 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 $\langle\langle\rangle\rangle$-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 $(\mathsf{REJECT}, L_{cheat})$.
   (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. 12. 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 10), 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 $(\mathsf{REJECT}, L_{cheat})$.
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 11. Each party P_i ends up with a secret share $\langle\langle c\rangle\rangle_{\mathsf{P}_i}$ of $c = a \times b$.

Fig. 13. 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 10), 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 $\langle\langle\rangle\rangle$-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 12) 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 13) 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. 14. PickSecretSharedBeaverTriple: Generating Secret-Shared Beaver Triples

Protocol π_{SETUP} (Setup for Auditable MPC)

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

Init: On input $(\textsc{Init}, p)$ 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 10, to obtain a $\langle\langle\rangle\rangle$-shared secret-shared MAC key α (together with its encryption e_α).
3. The parties invoke PickSecretSharedRandom, shown in Figure 12, to generate a $\langle\langle\rangle\rangle$-sharing of a random number for each input.
4. The parties invoke PickSecretSharedBeaverTriple, shown in Figure 14, to generate $\langle\langle\rangle\rangle$-sharings of a Beaver triple for each multiplication they will need to perform.

Fig. 15. 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}, thus learning the MAC key α, and all of the random values and Beaver triples generated during $\pi_{\texttt{SETUP}}$.

Input: For each party P_i,
1. 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.
2. If P_i is adversarially controlled, $\mathcal{S}_{\texttt{CIDA}}$ executes Input (interacting with adversarial parties as needed) as described in $\pi_{\texttt{CIDA}}$. As a result of this step, $\mathcal{S}_{\texttt{CIDA}}$ learns the input in_i, since it sees the broadcast value ϵ, and it knows the random value s.

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. (It can do this because it knows the value out' which is the result of the function evaluated on 0's for honest party inputs, and the values chosen by $\mathcal{Z}$ for the adversarial parties' inputs. It then only has to add $\mathsf{out} - \mathsf{out}'$ to $[\mathsf{out}]_{\mathsf{P}_I}$.)
2. $\mathcal{S}_{\texttt{CIDA}}$ finds a randomness $r_{\mathsf{out},i}$ such that $\mathtt{pc}([\mathsf{out}]_{\mathsf{P}_i}, r_{\mathsf{out},i}) = \mathsf{c}_{\mathsf{out},i}$. (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 $\alpha \cdot \mathsf{out}$. (It can do this because it knows α.)
4. $\mathcal{S}_{\texttt{CIDA}}$ then runs Output as described in $\pi_{\texttt{CIDA}}$ (interacting with the adversarial parties as needed).

Fig. 16. Simulator for MPC with Identifiable Abort

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 17. 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 $[\mathsf{in}_i]_{\mathsf{P}_j} = [0]_{\mathsf{P}_j} + \mathsf{in}_i$ and decommitment $r'_{i,j}$ such that $[\mathsf{in}_i]$ reconstructs to in_i and $\mathtt{pc}([\mathsf{in}_i]_{\mathsf{P}_j}, r'_{i,j}) = \mathsf{c}_{i,j}$. 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 7 and 8). These are Pedersen commitments (Appendix C.1) and non-interactive zero-knowledge proofs (Appendix C.2).

C.1 Commitments

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

- Setup$(1^k) \to \mathtt{ck}$ generates the public commitment key.
- For any $x \in \mathcal{M}$, Commit$(\mathtt{ck}, x) \to (\mathtt{c}, \mathtt{d})$ produces a commitment/decommitment pair for x.
- Open$(\mathtt{ck}, \mathtt{c}, \mathtt{d}) \to \tilde{x} \in \mathcal{M} \cup \{\perp\}$ opens the commitment, where $\perp$ is returned if $\mathtt{c}$ 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}, thus learning the MAC key α, and all of the random values and Beaver triples generated during $\pi_{\texttt{SETUP}}$.

INPUT: For each party P_i,
1. If P_i is honest, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ executes INPUT (interacting with the adversarial parties as needed) as described in $\pi_{\texttt{CIDA}}$ with a fixed input of 0.
2. If P_i is adversarially controlled, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ executes INPUT (interacting with adversarial parties as needed) as described in $\pi_{\texttt{CIDA}}$. As a result of this step, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ learns the input in_i, since it sees the broadcast value ϵ, and it knows the random value s.

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. (It can do this because it knows the value out' which is the result of the function evaluated on 0's for honest party inputs, and the values chosen by $\mathcal{Z}$ for the adversarial parties' inputs. It then only has to add $\mathsf{out} - \mathsf{out}'$ to $[\mathsf{out}]_{\mathsf{P}_i}$.)
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 $\alpha \cdot \mathsf{out}$. (It can do this because it knows α.)
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, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ simulates a threshold verifiable opening of $e_i = \{\mathsf{c}_{i,j}, e_{i,j}, e_{i,j,r}, \pi_{i,j}, \pi_{i,j,r}\}_{j \in \mathcal{C}}$ to in_i. It does so as follows:
1. If P_i (the owner of input in_i) is adversarially controlled, $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ runs OPEN as described in $\pi_{\texttt{CIDA,AUDIT,OPEN}}$ (interacting with the adversarial parties as needed).
2. If P_i is honest:
   (a) We know that e_i is an encryption of 0. $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ can decrypt $e_{i,j}$ to obtain $[0]_{\mathsf{P}_j}$, which is also the committed value in $\mathsf{c}_{i,j}$.
   (b) $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ then computes a decommitment $r'_{i,j}$ such that $\mathsf{pc}([0]_{\mathsf{P}_j} + \mathsf{in}_i, r'_{i,j}) = \mathsf{c}_{i,j}$. (It can do this since it has the trapdoor information - the discrete log relationship between the two Pedersen commitment generators.)
   (c) $\mathcal{S}_{\texttt{CIDA,AUDIT,OPEN}}$ then runs OPEN as described in $\pi_{\texttt{CIDA,AUDIT,OPEN}}$ (interacting with the adversarial parties as needed), using $[0]_{\mathsf{P}_j} + \mathsf{in}_i$ and $r'_{i,j}$ as the additive share and decommitment value of party P_j.

Fig. 17. Simulator for Openable MPC

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

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

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

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 $\texttt{ck} = (q, g, h)$.
- $\texttt{pc}(\texttt{ck} = (q, g, h), x)$: Choose a random value $r \leftarrow \mathbb{Z}_p^*$, and set $\texttt{c} = g^x h^r \bmod q, \texttt{d} = (x, r)$.
- $\textsc{Open}_p(\texttt{ck} = (q, g, h), \texttt{c}, \texttt{d} = (x, r))$: Return x if $\texttt{c} = g^x h^r \bmod q$, and return $\perp$ otherwise.

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

C.2 (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 [FS87]. 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 18 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 $\texttt{c} = g^x h^r \bmod n$. In Camenisch-Stadler notation [CS97], this can be expressed as:

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

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

Fig. 18. Proof of Knowledge of Committed/Encrypted Value

Figure 19 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:

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

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

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

Fig. 19. Proof of Equality of Committed/Encrypted Values

Both of the above proofs can be used to prove statements about Pedersen commitments (described in Appendix C.1) *and* verifiable encryptions (described in Section 6), since they have the same structure; both consist of a product of two exponentials.