Cheater Identification on a Budget: MPC with Identifiable Abort from Pairwise MACs

Carsten Baum[1], Nikolas Melissaris[2], Rahul Rachuri[3]*, Peter Scholl[2]

[1] Technical University of Denmark, cabau@dtu.dk
[2] Aarhus University, {nikolas, peter.scholl}@cs.au.dk
[3] Visa Research, srachuri@visa.com

Abstract. Cheater identification in secure multi-party computation (MPC) allows the honest parties to agree upon the identity of a cheating party, in case the protocol aborts. In the context of a dishonest majority, this becomes especially critical, as it serves to thwart denial-of-service attacks and mitigate known impossibility results on ensuring fairness and guaranteed output delivery.

In this work, we present a new, lightweight approach to achieving identifiable abort in dishonest majority MPC. We avoid all of the heavy machinery used in previous works, instead relying on a careful combination of lightweight detection mechanisms and techniques from state-of-the-art protocols secure with (non-identifiable) abort.

At the core of our construction is a homomorphic, multi-receiver commitment scheme secure with identifiable abort. This commitment scheme can be constructed from cheap vector oblivious linear evaluation protocols based on learning parity with noise. To support cheater identification, we design a general compilation technique, similar to a compiler of Ishai et al. (Crypto 2014), but avoid its requirement for adaptive security of the underlying protocol. Instead, we rely on a different (and seemingly easier to achieve) property we call online extractability, which may be of independent interest. Our MPC protocol can be viewed as a version of the BDOZ MPC scheme (Bendlin et al., Eurocrypt 2011) based on pairwise information-theoretic MACs, enhanced to support cheater identification and a highly efficient preprocessing phase, essentially as efficient as the non-identifiable protocol of Le Mans (Rachuri & Scholl, Crypto 2022).

1 Introduction

Secure multiparty computation (MPC) is a class of cryptographic protocols allowing a group of distrusting parties to jointly compute a function over their private inputs, without revealing anything beyond the output of the computation. While many different factors impact the usability of MPC protocols, one of the most important security-wise is the corruption threshold. It provides a limit on how many of the participants can collude and share their information, without losing the privacy guarantees of MPC. Many popular protocols, such as SPDZ [DPSZ12], BDOZ [BDOZ11] and their follow-up works ensure privacy,

* Work done while at Aarhus University.

even if $n - 1$ out of the n participants are corrupted, and even when attackers may actively deviate from the protocol.

Nevertheless, privacy is not the only security guarantee that an MPC protocol may have to achieve. Fairness requires that if the corrupted parties obtain the output, then so do the honest parties. It is known [Cle86] that in the dishonest majority setting (i.e. when $\geq n/2$ parties are corrupted) we cannot achieve fairness, or the even stronger notion of guaranteed output delivery. Therefore, current highly efficient protocols settle for a weaker notion of security: security with abort. This typically means that a corrupt party can force the protocol to abort, so that some (or all) of the honest parties will abort instead of learning the correct output.[4]

Identifiable Abort for MPC. Since fairness is impossible in the dishonest majority setting, the *next best* property would be if, in the case that the protocol aborts, the honest parties agree that the protocol aborted and also agree on the identity of at least one corrupt party. This can work as a deterrent since honest parties can exclude said corrupt party if they restart the computation. This property is called *identifiable abort*.

Cheater identification in dishonest-majority MPC (ID-MPC) was first formally studied by Ishai, Ostrovsky and Seyalioglu [IOS12], who showed that it is impossible to build unconditionally secure ID-MPC in a model with a broadcast channel and any pairwise ideal functionality, such as oblivious transfer (OT). This is in contrast to the secure-with-abort model, where pairwise OT suffices. Later, Ishai, Ostrovsky and Zikas [IOZ14] constructed a compiler that takes any semi-honest protocol that uses a source of correlated randomness, and transforms it into a protocol with security against malicious parties and with identifiable abort (in the correlated randomness model). The compiler can be seen as an information-theoretic version of the GMW compiler [GMW87]: each party commits to its input and randomness that they intend to use for the semi-honest protocol and then runs the semi-honest protocol by broadcasting their messages in each round and using zero-knowledge to prove that their messages are correct. To generate the correlated randomness needed for this protocol, [IOZ14] also described a compiler that transforms any cryptographic preprocessing phase that is secure-with-abort into one that has identifiable abort.[5] Overall, this yields the first construction with identifiable abort that makes only black-box use of cryptographic primitives, namely an adaptively secure oblivious transfer protocol and a broadcast channel. The main downsides of this construction are the need for adaptively secure OT in the preprocessing phase, and the overall complexity of proving that each protocol step was executed correctly in the online phase.

To resolve this, multiple works [BOS16, SF16, CFY17, BOSS20] have given more "practical" constructions of ID-MPC. Baum et al [BOS16] construct an

[4] This is called *selective abort*, in contrast to *unanimous abort*, where the honest parties must all agree that the protocol aborted.

[5] This result bypasses the impossibility of [IOS12] by relying on black-box use of an *OT protocol* rather than an ideal OT functionality.

identifiable abort protocol for arithmetic circuits in the preprocessing model where the online phase is a variant of BDOZ [BDOZ11] that permits cheater identification. While avoiding adaptively secure OTs, their preprocessing phase needs to perform at least n times as much computation as non-identifiable protocols, and also relies on cheater identification for lattice-based cryptography which is far from being practically efficient. [SF16] modify the SPDZ protocol to identify cheating by ensuring that correct shares are opened. Their preprocessing would, in order to be identifiable, have to rely on the same expensive mechanisms as [BOS16] (such as verifiable decryption). Cunningham et al. [CFY17] used Pedersen commitments to identify cheaters in the online phase, which limits the finite field over which the computation can happen and makes preprocessing costly as all these commitments have to be generated during preprocessing. Finally, Baum et al.[BOSS20] construct an ID-MPC protocol for boolean circuits which runs in a constant number of rounds and uses cryptographic primitives in a black box away. While in their work, public key operations after the setup phase and zero knowledge (ZK) machinery (as well as adaptive OTs) are avoided, their construction is limited to the binary setting and their use of multiparty BMR [BMR90] has a substantial overhead from reconstructing a large garbled circuit.

Challenge of adaptive security and identifiable abort. When considering solely a preprocessing protocol, the [IOZ14] compiler offers a simple and attractive approach to obtaining identifiable abort. At a high level, their idea is to have every party first commit to a random tape, and then run a standard, secure-with-abort protocol; if the protocol aborts, every party will open the commitments to their random tapes. This allows all other parties to detect which party cheated by re-running a local copy of the protocol. Furthermore, intuitively, opening random tapes in case of abort does not pose a privacy issue, since the preprocessing phase is independent of all parties' inputs. What is needed for this to work is that any deviation from the honest protocol can be consistently detected by every party using the randomness that they committed to (called *$\mathcal{P}$-verifiability* in [IOZ14]).

The challenge with this approach lies in simulating the view of the corrupted parties. If the protocol aborts, the simulator needs to be able to open the honest parties' random tapes to the adversary, in a way that is consistent with the previous (simulated) transcript. One way to do this is if the preprocessing protocol is adaptively secure, so that honest random tapes can be 'explained' by the simulator as if that party had just been adaptively corrupted. This is where the reliance of [IOZ14] on adaptive OT comes from, and it seems inherent to this commit-and-open paradigm[6]. While some works have attempted to circumvent

[6] [BOSS20] manages to avoid the use of adaptively secure primitives by making use of a homomorphic commitment scheme and redefinitions of the offline ideal functionality. Specifically, in case of an abort of the offline phase their ideal functionality at this point did not yet output any values to the environment, so the original random tapes can safely be opened. Moreover, their preprocessing protocol uses homomorphic commitments for shares and require that all parties commit to the values they used

the adaptivity problem [BDD20, BOSS20], no efficient UC-secure solution for ID-MPC over arbitrary finite fields is known.

1.1 Our Contribution

In this work, we construct an efficient MPC protocol with identifiable abort for arithmetic circuits over large fields, with UC security [Can01]. A key feature of our protocol is an online phase based on simple, pairwise information-theoretic MACs, just as in the (secure-with-abort) BDOZ protocol [BDOZ11]. Thanks to this simple online phase, the correlated randomness that must be produced by the preprocessing phase is just standard, authenticated multiplication triples, the same as in secure-with-abort protocols. To allow identifiable abort in the online phase, our main tool is a new compiler that transforms certain classes of *sender-receiver protocols*, where one party has private input, into ones that support cheater identification. Our compiler overcomes some limitations of the related compiler from [IOZ14], which only works for preprocessing protocols, and also requires adaptive security of the original protocol.

1.2 Technical Overview

Online Phase. A natural approach to achieving identifiable abort in MPC is to use a form of linear secret sharing where the parties are committed to their shares via linearly homomorphic commitments. If the commitments support *multiple receivers* and *identifiable abort*, then secret-shared values can be reliably opened, by checking commitments on the shares. Given a preprocessing phase that generates random multiplication triples, where each share is authenticated to all other parties using the homomorphic commitments, one can construct a standard MPC protocol by exploiting linearity of the commitments and using Beaver multiplication. This was done, for instance, in [BOS16], using an information-theoretic identifiable commitment scheme; however, the structure of the commitments is more complex than information-theoretic MACs used in secure-with-abort MPC [BDOZ11, DPSZ12], which led to a much more costly preprocessing protocol in [BOS16].

In this work, our online phase follows the same general approach, using preprocessed triples and identifiable linear commitments. The key differences compared with prior work are how we instantiate the preprocessing to generate multiplication triples with identifiable abort, and how we instantiate the identifiable, linearly homomorphic commitments.

Preprocessing Phase. The goal of our preprocessing phase is to create additive secret shares of random multiplication triples over a large field, which are

in the preprocessing. The consistency is then ensured by opening random linear combinations of the commitments, and in the online phase these commitments can be used for cheater detection.

committed to using linearly homomorphic commitments. To do this, the parties will first run a secure-with-abort protocol, Π_{Trip}, to create unauthenticated triples (for instance, using pairwise OLE), and then commit to their shares with the homomorphic commitments. To guarantee that parties have committed to the correct shares, we then run a sacrificing-based correctness check, where one triple is sacrificed to check another, similarly to [DPSZ12].

To identify cheaters in this approach, we must make two important changes. First, following the IOZ compiler [IOZ14] and other similar approaches [BOSS20, SSS22], we have the parties commit to their random tapes of the secure-with-abort protocol Π_{Trip} before running it. If Π_{Trip} aborts, the parties then open their random tapes and reconstruct the protocol transcript to identify who cheated. This stage requires the original protocol to have a form of verifiable transcripts, meaning that it is always possible to identify who cheated, when given the (alleged) views of all parties together with their random tapes. We formalize this property, which we call *identifiable cheating*, and show that it can be cheaply added to any secure-with-abort protocol by adding digital signatures to all pairwise communication. Signatures guarantee that if some party cheats, there is a signed record of the messages it sent that can later be used to help prove this.

It remains to discuss how we can still simulate the execution of Π_{Trip}, without running into the aforementioned adaptive security issue. Following [BOSS20], we have the simulator run an honest copy of Π_{Trip}, to generate the honest parties' messages seen by the adversary. Now, it is easy to simulate the opening of random tapes in case of abort, the only problem is that the simulator no longer has any power to extract the corrupt parties' inputs. In this case, however, the only inputs that need to be extracted are the corrupted parties' shares of multiplication triples, which were already committed to via the homomorphic commitment scheme. By relying on a UC secure homomorphic commitment functionality, these shares can easily be extracted without having to use the Π_{Trip} simulator[7].

The last issue is that even if the triple protocol Π_{Trip} runs correctly, the overall protocol may still abort if the triple sacrifice fails, due to a corrupted party committing to the wrong share. To recover from this, we again have the parties open their random tapes from Π_{Trip}, so that all parties' shares can be recovered, and then compare these with the shares that were committed to in the homomorphic commitment scheme by opening all the committed shares.

Building Identifiable, Homomorphic Commitments. To instantiate the homomorphic commitment scheme, we design a scheme based on pairwise information-theoretic MACs, where the committed value is authenticated to every other party with a MAC, as in BDOZ. An advantage of such MACs is that they can be generated very efficiently using vector oblivious linear evaluation (VOLE) protocols based on variants of the learning parity with noise assumption, such as [BCGI18, WYKW21, BCG$^+$19]. However, the problem is that MACs do not provide a way for parties to agree upon who cheated in the event that

[7] We still rely on the existence of the Π_{Trip} simulator in the proof, to argue that the simulated view is indistinguishable from the real protocol

an opening fails. Indeed, the impossibility result of [IOS12] implies that this is impossible to do with black-box use of pairwise information-theoretic MACs.

At a high level, our approach is to follow the same commit-and-open approach as for the triple generation: if an opening fails, the parties will open their random tapes for the VOLE protocols used to generate the MACs, and use the reconstructed VOLE outputs to help identify who cheated. However, we are now met with two new challenges. Firstly, the commit-and-open paradigm only works for preprocessing protocols, since all parties need to open their random tapes in case of an abort — when using homomorphic commitments in the online phase, this would leak any private, committed inputs. Secondly, we again have to deal with the adaptivity problem, which was essentially deferred in the preprocessing stage, by relying on the security of the homomorphic commitment scheme.

Compiling Sender-Receiver Protocols to Identifiable Abort. We present a new identifiable abort compiler that works for a general class of *sender-receiver protocols*, where only one party (the sender) has private input. Like IOZ, our compiler makes black-box use of the underlying secure-with-abort protocol. Unlike IOZ, however, we are not restricted to preprocessing protocols where no party has private input — this allows us to apply our compiler to an arbitrary, linearly homomorphic commitment scheme with multiple receivers, which we instantiate with a VOLE-based protocol for setting up information-theoretic MACs.

At a high level, our compiler follows the same strategy as our preprocessing phase, except that to prevent leakage of the sender's private inputs, we only require the *receivers* to commit to and open their random tapes, and not the sender. Under a mild assumption on the communication pattern in the sender-receiver protocol, we show that the receivers will still be able to identify a cheating sender, since in this case there will always be at least one honest receiver, who can prove that they followed the protocol and aborted due to the cheating sender. There is one issue with this approach, however. Even though receivers do not have private *inputs*, they may have private *outputs* that can't be revealed. To remedy this, we use two different types of recovery mechanisms, depending on whether a sender or receiver is claiming an abort. In the first case, the receivers will all *privately* send their evidence to the sender, who will select and publish a proof. In the second case, the aborting receiver must instead immediately open its view for all parties to inspect and confirm that it aborted; because of the restricted communication pattern of sender-receiver protocols, this would imply that the sender has cheated (and so it is not a problem to leak the receiver's output, which only depends on the corrupt sender's input).

Avoiding Adaptive Security via Online Extractability. The final challenge in our compiler is ensuring that all of the identification stages, where honest receivers open their random tapes, can be simulated. Instead of relying on adaptive security, we observe that a weaker property suffices, which we call *online extractability.* In the UC security proof, there are two cases, depending on whether the adversary corrupts the sender, or only (a subset of) the receivers.

In the first case, the main job of the simulator is to simulate messages from the honest receivers in such a way that it can extract the inputs of the corrupted sender.[8] If the simulator later has to open the honest receivers' random tapes, due to the malicious sender causing an abort, the natural approach relying on adaptive security is for the simulator to adaptively corrupt the honest receivers, so that it learns randomness that explains the previously simulated messages. Online extractability instead defines a special type of simulation, where the normal protocol execution suffices to extract adversarial inputs, if one does only imperceptible changes to the CRS or other hybrid functionalities. While this is already how many UC protocol simulators work, we define this property formally and show that it is composable. Having online extractability, the task of the simulator in our IA compiler is now much easier: it can simulate messages of the honest receivers by simply running an honest copy of the protocol, except for the interaction with a setup functionality like a CRS. This makes it trivial to open the random tape to identify a cheater, since the simulator has followed the protocol honestly. [9]

Efficiency. We now briefly discuss our efficiency gains, as summarized in Table 1. Due to space constraints, we give a more detailed analysis in Appendix B.

To investigate the overhead of obtaining identifiable abort, we compare our protocol with two versions of the preprocessing and online phases from Le Mans [RS22], which is secure with abort. Le Mans v1 (called Dynamic SPDZ in [RS22]) has lower preprocessing requirements in exchange for a slower online phase, while Le Mans v2 costs more in the preprocessing phase, but gets the fastest online phase. Both versions of Le Mans, as well as our preprocessing, require $\approx n^2$ OLE and VOLE correlations for each multiplication gate, to build authenticated multiplication triples. Asymptotically, if state-of-the-art PCG techniques are used, producing the OLE/VOLEs can be done with a total of $O(n^2 \log(|C|))$ communication, where $|C|$ is the circuit size, and $O(n^2|C|)$ computation. Our preprocessing has the same base cost as Le Mans, plus sending an additional $2(n-1)|C|$ field elements per party to authenticate and check triples.

When it comes to the online phase, we use the standard BDOZ online phase with authenticated triples and signatures added to the messages, which increases the cost by $O(n)$. Overall, our communication cost per party is dominated by $2(n-1)|C|$ field elements for an honest execution. Dynamic SPDZ has an online cost of only $12|C|$ field elements per party, achieving only security with abort.

Compared to other ID-MPC protocols like [BOS16, BOSS20], in the preprocessing phase, [BOS16] requires $O(n^3)$ broadcast messages per multiplication

[8] In this case, note that the simulator does not need to equivocate the corrupted receivers' outputs to match those of the ideal functionality, because of the structure of the sender-receiver protocol: these outputs only depend on the corrupted sender's input, so there is nothing to simulate.

[9] Of course, one still has to prove that a protocol is online-extractable, but this is seemingly simpler than a security proof for adaptive security. Indeed, we observe that many protocols in the literature are already online-extractable.

Protocol	Building blocks	IA	Preprocessing cost	Online cost
Le Mans, v1	(V)OLE	✗	$n^2 \times$ OLE*	$12n$
Le Mans, v2	(V)OLE	✗	$n^2 \times$ OLE*$+ O(n)$	$4n$
[BOS16]	depth-1 HE	✓	$O(n^3)^\dagger$	$O(n^2)^\ddagger$
Ours	(V)OLE	✓	$n^2 \times$ OLE*$+ O(n^2)$ ‡	$O(n^2)^\ddagger$

* Random, pairwise OLE and VOLE correlations. Can be generated with $O(n \log |C|)$ communication per party using variants of LPN [BCG$^+$19, BCG$^+$20].

† Must be broadcast

‡ Corrupted party can force to be broadcast

Table 1: Comparing efficient MPC protocols with and without identifiable abort. Preprocessing cost reflects the cost per multiplication in the preprocessing phase, in terms of building blocks (OLE/VOLE) plus total communication in field elements.

gate, whereas we only need $O(n^2)$ broadcasts even in the worst-case scenario. In the online phase, both [BOS16] and our protocol have $O(n^2)$ complexity.

Our protocol is expected to perform faster than [BOS16], primarily due to the use of pseudorandom correlation generator (PCG) techniques, which are estimated in [BCG$^+$20] to have much lower communication overhead compared to homomorphic encryption-based (HE) approach of [BOS16].

The protocol of [BOSS20] is incomparable to ours as it is a garbled circuit-based (GC) construction that works for Boolean circuits. In comparison, our construction allows the evaluation of circuits over $\mathbb{F}_p$ for large p with a round complexity that depends on the circuit depth.

1.3 Related work

Interest in the area of MPC with Identifiable Abort has increased recently, leading to many exciting research directions.

Brandt et al. [BMMM20] and independently Simkin et al. [SSY22] investigated how to realize dishonest-majority MPC with identifiable abort from correlations among less than all n parties.

Cohen et al. [CGZ20] investigated the two-round MPC setting with dishonest majority and broadcast. They showed in which cases identifiable abort is achievable, depending on the broadcast use. This was extended to the honest majority setting by Damgård et al. [DMR$^+$21]. In follow-up work, [DRSY23] investigated which setup is necessary for the two-round setting to achieve identifiable abort. In the plain model, Ciampi et al.[CRSW22] showed how to construct ID-MPC in the optimal 4 rounds.

When considering covert instead of malicious security, Faust et al. [FHKS21] as well as Scholl et al. [SSS22] constructed compilers from passively secure MPC to covertly [AL07] secure MPC with security against $n-1$ corruptions using time-lock puzzles. Later, Attema et al. [ADEL22] showed how to realize this without time-lock puzzles, although requiring an honest majority. All these constructions actually achieve a stronger property called publicly verifiable MPC which implies identifiable abort.

Hazay et al. [HVW22] used framing-free designated-verifier Zero-Knowledge proofs to construct an alternative to the IOZ14 compiler. Their construction only works for honest majority protocols.

More concretely, Chen et al [CHI$^+$21] constructed a dedicated RSA key generation protocol with Identifiable Abort and security against a dishonest majority. The efficiency of their construction comes from a communication model that uses a centralized "coordinator" which realizes broadcast.

Concurrent Work. Recently, Cohen et al. [CDKs23] presented another approach to identifiable abort, which also manages to avoid the need for adaptively secure OT in the [IOZ14] compiler. Their method is based on revealing committed input values in case of cheating; in contrast to our approach of revealing random tapes to verify protocol messages, [CDKs23] do not make use of the underlying protocol messages in this way, instead relying on a special form of committed OT functionality.

We believe that our work has a couple of advantages over [CDKs23]:

1. We directly support MPC for computing arithmetic circuits on private inputs, instead of just correlated randomness functionalities. While [CDKs23] could be used to instantiate the correlated randomness for, say, the [IOZ14] online phase, the amount of correlated randomness needed would be at least n times more than what's used by our protocol, due to the use of $O(n)$ sized information-theoretic signatures used in [IOZ14] to authenticate the correlated randomness.
2. Even for computing correlated randomness, our protocol seems to be more efficient. [CDKs23] gives an instantiation of MASCOT-like preprocessing, with roughly a 50% overhead on top of the secure-with-abort protocol. Our protocol can be instantiated using VOLE based on SoftSpokenOT and triple generation using OT, to obtain a similar result but with essentially no extra communication cost for achieving identifiable abort.[10]

Roadmap. In contrast to the "top-down" presentation in the technical overview in Section 1.2, the remainder of the paper proceeds in a "bottom-up" fashion. We start with our notion of online extractability in Section 3, followed by a construction of homomorphic commitments in Section 4, which are compiled to support identifiable abort in Section 5. Section 6 then describes our triple generation protocol, which uses the previous building blocks. In the Supplementary Material, we describe the online phase, as well as various additional technical details.

[10] The main overhead incurred in our protocol is that, in the worst case, an adversary can force all all point-to-point messages to be broadcast. This broadcast is done by default in [CDKs23]

2 Preliminaries and Notation

We use κ as the security parameter and ρ as the statistical security parameter. Bold letters such as $\boldsymbol{a}$ are used to indicate vectors, and $\boldsymbol{a}[i]$ refers to the i-th element of the vector. We write $[a, b]$ to denote the set of natural numbers $\{a, \ldots, b\}$ and $[a, b) = \{a, \ldots, b-1\}$. We use $\boldsymbol{a} \odot \boldsymbol{b}$ to indicate the component-wise product of vectors.

2.1 Modeling Security

We work in the universal composability (UC) framework [Can01] for analyzing security and assume some familiarity with this. In UC, protocols are run by interactive Turing Machines (iTMs) called *parties*. We make the simplifying assumption that any protocol π runs between a fixed set of parties, typically denoted $\mathcal{P} = \{P_1, \ldots, P_n\}$. The *adversary* $\mathcal{A}$, which is also an iTM, can actively corrupt a subset $\mathcal{P}_{\mathcal{A}} \subset P$ and gains control over these parties. We denote the set of honest parties by $\mathcal{P}_H = P \setminus \mathcal{P}_{\mathcal{A}}$. We focus on static corruptions, so these sets are fixed from the beginning. The parties can exchange messages via resources, called *ideal functionalities* (which themselves are iTMs) and which are denoted by $\mathcal{F}$. We assume that all parties can communicate via authenticated channels, and sometimes also use secure point-to-point channels and a reliable broadcast channel. These are all modelled as ideal functionalities that can be realized on top of authenticated channels using standard methods. In protocol descriptions, instead of referring to the specific functionalities, we typically write e.g. *P_i privately sends x to P_j* or *P_i broadcasts x*. Moreover, we work in the synchronous model, where protocols proceed in a sequence of rounds, such that every message sent in one round is guaranteed to be delivered before the start of the next round. Such synchronous communication channels can also be modelled in UC [KMTZ13].

As usual, we define security with respect to an iTM $\mathcal{Z}$ called the *environment*. The environment provides inputs to and receives outputs from the parties in $\mathcal{P}$. To define security, let $\pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ be the distribution of the output of an arbitrary $\mathcal{Z}$ when interacting with $\mathcal{A}$ in a real protocol instance π using resources $\mathcal{F}_1, \ldots$. Furthermore, let $\mathcal{S}$ denote an *ideal world adversary* and $\mathcal{F} \circ \mathcal{S}$ be the distribution of the output of $\mathcal{Z}$ when interacting with parties which run with $\mathcal{F}$ instead of π and where $\mathcal{S}$ takes care of adversarial behavior.

Definition 1. *We say that π UC-securely implements $\mathcal{F}$ if for every iTM $\mathcal{A}$ there exists an iTM $\mathcal{S}$ (with black-box access to $\mathcal{A}$) such that for every environment $\mathcal{Z}$, the outputs of $\mathcal{Z} \circ \pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ and $\mathcal{Z} \circ \mathcal{F} \circ \mathcal{S}$ are identical except with negligible probability.*

In the security experiment $\mathcal{Z}$ may arbitrarily activate parties or $\mathcal{A}$, though *only one iTM (including $\mathcal{Z}$) is active at each point of time.*

Definition 2 (Identifiable Abort). *Let $\mathcal{F}$ be a functionality running with a set of parties $\mathcal{P}$. We define $[\mathcal{F}]^{\mathsf{IA}}$ to be the corresponding functionality with identifiable abort, where at any time, if $\mathcal{A}$ sends a message* $(\mathsf{Abort}, \mathcal{J})$ *for some* *non-empty set $\mathcal{J} \subset \mathcal{P}_{\mathcal{A}}$, $[\mathcal{F}]^{\mathsf{IA}}$ sends $(\mathsf{Abort}, \mathcal{J})$ to all parties and terminates. Additionally, if $\mathcal{F}$ would at any point send a message Abort to $\mathcal{P}$, it first waits to receive a non-empty $\mathcal{J} \subset \mathcal{P}_{\mathcal{A}}$ from $\mathcal{A}$, and then sends $(\mathsf{Abort}, \mathcal{J})$ instead.*

2.2 VOLE and Information-Theoretic MACs

Functionality $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$

Parameters: Finite field $\mathbb{F}_{p^r}$, and expansion function $\mathsf{Expand} : S \to \mathbb{F}_p^m$ with seed space S and output length m.
The functionality runs between parties P_A and P_B.

Initialize: On receiving Init from P_A and P_B, sample $\Delta^B \leftarrow \mathbb{F}_{p^r}$ for P_B, and ignore all subsequent Init commands. Store Δ^B and send it to P_B.

Extend: On receiving Extend from P_B and $(\mathsf{Extend}, \mathsf{seed})$ from P_A, where $\mathsf{seed} \in S$:

1. Compute $\boldsymbol{u} = \mathsf{Expand}(\mathsf{seed})$.
2. Sample $\boldsymbol{v} \leftarrow \mathbb{F}_{p^r}^m$ and compute $\boldsymbol{w} = \boldsymbol{u} \cdot \Delta^B + \boldsymbol{v}$.
3. Send $\boldsymbol{w}$ to P_A and $\boldsymbol{v}$ to P_B.

Corrupt Parties: If P_B is corrupt, Δ^B and $\boldsymbol{v}$ may be chosen by $\mathcal{A}$. For a corrupt P_A, $\mathcal{A}$ can choose $\boldsymbol{w}$ (and then $\boldsymbol{v}$ is recomputed accordingly).

Key Query: If P_A is corrupted, $\mathcal{A}$ may send a message $(\mathsf{guess}, \Delta')$ with $\Delta' \in \mathbb{F}_{p^r}$. If $\Delta' = \Delta$, send $\mathsf{success}$ to P_A. Else, send abort to both parties and abort.

Fig. 1: Functionality for Programmable VOLE

The VOLE[11] functionality $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ (Fig. 1) generates a batch of m random VOLE correlations between two parties P_A and P_B, usually called *sender* and *receiver*.

A VOLE correlation consists of two vectors $\boldsymbol{u} \in \mathbb{F}_p^m, \boldsymbol{w} \in \mathbb{F}_{p^r}^m$ held by P_A, and the element $\Delta^B \in \mathbb{F}_{p^r}$ and a vector $\boldsymbol{v} \in \mathbb{F}_{p^r}^m$ held by P_B, such that $\boldsymbol{w} = \boldsymbol{u} \cdot \Delta^B + \boldsymbol{v}$. Δ^B and $\boldsymbol{v}$ are chosen uniformly at random, while the $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ functionality allows the sender to program its share $\boldsymbol{u}$ of the correlation by providing a seed. Hence, when running two instances of $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ with different receivers P_B, P_B' but the same seed, the sender P_A ends up with the same values $\boldsymbol{u}$ as part of its VOLE correlations. The Expand function in $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ should be a pseudorandom generator, whose precise implementation depends on how the protocol is instantiated (for instance, when using LPN-based VOLE [BCGI18, WYKW21], Expand is an LPN-based PRG).

[11] More precisely, this is actually a so-called subfield VOLE.

One can view a VOLE correlation as an information-theoretic MAC on the vector $\boldsymbol{u}$ of P_A. This is because: $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ does not reveal Δ^B to the sender. Assume that P_A could produce a pair of vectors $\boldsymbol{u}' \in \mathbb{F}_p^m, \boldsymbol{w}' \in \mathbb{F}_{p^r}^m$ with $\boldsymbol{u}' \neq \boldsymbol{u}$ such that the VOLE correlation holds, along with the original $\boldsymbol{u}, \boldsymbol{w}$. Therefore,

$$\boldsymbol{w} = \boldsymbol{u} \cdot \Delta^B + \boldsymbol{v}, \qquad \boldsymbol{w}' = \boldsymbol{u}' \cdot \Delta^B + \boldsymbol{v} \tag{1}$$

That means for the pairs, it needs to hold that $(\boldsymbol{w}[i] - \boldsymbol{w}'[i])/(\boldsymbol{u}[i] - \boldsymbol{u}'[i]) = \Delta^B$ (where the index $i \in [1, m]$ is such that $\boldsymbol{u}[i] \neq \boldsymbol{u}'[i]$). In order for this to hold, P_A needs to know the secret Δ^B if it can forge a MAC on a different value $\boldsymbol{u}'$. However, Δ^B is chosen randomly from a set of size p^r so the probability of a correct guess is negligible if p^r is exponential in the security parameter.

In $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$, P_B only obtains $\Delta^B, \boldsymbol{v}$ which are chosen uniformly at random and independent of $\boldsymbol{u}$. Therefore, P_B learns no information about $\boldsymbol{u}$ from its share of the correlation. The MAC scheme implied by VOLE is also linearly homomorphic. Details can be found e.g. in [BDSW23].

2.3 Signatures

In this work, we crucially rely on digital signatures and we use the standard security notion for digital signature schemes, namely, existential unforgeability under adaptive chosen-message attacks (EUF-CMA). The definitions can be found in Appendix A.1.

2.4 Basic Functionalities

We additionally use a one-to-many commitment functionality $\mathcal{F}_{\mathsf{Commit}}$, shown in Fig. 15, as well as a coin-tossing functionality, $\mathcal{F}_{\mathsf{Rand}}$, in Fig. 16. These functionalities should have identifiable abort. UC commitments (with identifiable abort) can be easily realized with a random oracle or CRS and a broadcast channel, while coin-tossing can be realized using $\mathcal{F}_{\mathsf{Commit}}$ with a standard commit-and-open approach.

3 Online-Extractable Protocols

We now define a new subclass of UC-secure protocols, which we call *online-extractable*. For such a protocol π we define a special experiment called the *extractor execution*, that runs with a PPT iTM called the *extractor*, $\mathcal{E}$, which must extract the inputs of the adversary during a real execution of the protocol. The extractor is allowed to manipulate any CRS-like functionalities used in π, or observe any random oracle queries, as well as see all communication between the adversary and any hybrid functionalities or honest parties. Otherwise, the protocol π is run as in the *real world experiment*. This manipulation done by $\mathcal{E}$ should not be noticeable to the environment, while the inputs extracted by $\mathcal{E}$ should be indistinguishable from those that the simulator $\mathcal{S}$ obtains in the ideal world.

The definition is inspired by many security proofs of UC protocols, where the simulator $\mathcal{S}$ in the ideal setting simulates by running the actual protocol π with dummy inputs for honest parties. At the same time, $\mathcal{S}$ can extract the actual inputs of the dishonest parties that are controlled by $\mathcal{A}$ without actually deviating from the protocol[12]. This means that many protocols have this extractability property already, and constructing $\mathcal{E}$ for them will be simple by manipulating $\mathcal{S}$ (removing equivocation etc). We will later see that such $\mathcal{E}$ comes in handy when simulating protocols that have identifiable abort without relying on adaptive security.

Defining Online Extractability. Towards formalizing online extractability, let π be a protocol that UC-implements a functionality $\mathcal{F}$, possibly using some other hybrid functionalities. For simplicity, we assume that parties in π either communicate directly or access hybrid functionalities. We call certain hybrid functionalities "CRS-like functionalities" if their input/output behavior towards parties is independent of which party called it:

Definition 3 (CRS-like Functionality). *A functionality $\mathcal{F}$ is a* CRS-like functionality *if, on receiving an input (sid, x) from party P_i, the functionality will give the same response that it would also give (i) upon later query (sid, x) by P_i; and (ii) upon query (sid, x) by any other party P_j at any point.*

Some examples of *CRS-like functionalities* are the standard CRS functionality $\mathcal{F}_{\mathsf{CRS}}$ (where $x = \perp$) or a random oracle. Definition 3 essentially rules out that $\mathcal{F}$ has an updatable memory that would change query results over time, unless the update is provably undetectable to protocol parties.

Let us denote by $\hat{\mathcal{F}}$ a version of the ideal functionality $\mathcal{F}$ which immediately outputs any inputs from $\mathcal{A}$ to $\mathcal{F}$ onto a special tape. We call this special tape the *ideal input tape*. It can neither be seen by parties nor $\mathcal{A}$ or the environment in any security experiment.

We also denote by $[\pi]_{\mathcal{E}}$ the *extractor execution* of protocol π, which is a modified execution of a protocol π with the extractor $\mathcal{E}$, where:

1. $\mathcal{E}$ is allowed to observe the inputs sent to any CRS-like functionality and freely program the output which is given in response to any input;
2. Every message sent between two parties P_i, P_j, where P_i is honest and P_j corrupt, is first given to $\mathcal{E}$ and then forwarded to the receiver;
3. Every (non-CRS-like) hybrid functionality $\mathcal{F}_H$ in π is modified to $\hat{\mathcal{F}}_H$, with an ideal input tape only accessible to $\mathcal{E}$, on which the inputs from $\mathcal{A}$ to $\mathcal{F}_H$ are placed.
4. $\mathcal{E}$ continuously outputs certain values, whenever they are available, on a special tape of its own, called the *extractor tape*.

As with the ideal input tape of $\hat{\mathcal{F}}$, we assume that in the extractor execution $[\pi]_{\mathcal{E}}$, the extractor tape cannot be accessed by any party, functionality, adversary or environment unless mentioned otherwise. The only difference between $[\pi]_{\mathcal{E}}$ and π that *may* be noticeable to the environment is the change to the CRS-like functionalities.

We now formally define online extractability.

[12] A similar idea, although in the context of public verifiability, was used previously, e.g. in [BDD20].

Definition 4. *Let π be a protocol that UC-securely implements an ideal functionality $\mathcal{F}$ with a fixed set of parties $\mathcal{P}$ and corrupted parties $\mathcal{P}_{\mathcal{A}} \subset \mathcal{P}$. Then, we say that π is* online-extractable for corrupt $\mathcal{P}_{\mathcal{A}}$ *if there exists a PPT iTM $\mathcal{E}$, and a UC simulator $\mathcal{S}$ for π, such that, for all environments $\mathcal{Z}$ and adversaries $\mathcal{A}$ who statically corrupt the parties in $\mathcal{P}_{\mathcal{A}}$:*

1. *$\mathcal{Z}$ cannot distinguish $\pi \circ \mathcal{A}$ from $[\pi]_{\mathcal{E}} \circ \mathcal{A}$ (where the extractor tape of $\mathcal{E}$ is not available to $\mathcal{Z}$ or $\mathcal{A}$).*
2. *The distribution of the ideal input tape of $\hat{\mathcal{F}}$ in $\hat{\mathcal{F}} \circ \mathcal{S}$ is indistinguishable from that of the extractor tape of $\mathcal{E}$ in $[\pi]_{\mathcal{E}} \circ \mathcal{A}$.*

UC Security vs. Online Extractability. The motivation for Definition 4 is that many existing UC-secure protocols can easily have such an online extractor $\mathcal{E}$. On the other hand, we highlight that this does not imply that the extractor $\mathcal{E}$ is a good UC simulator for π. Indeed, while $\mathcal{E}$ is required to successfully extract the corrupted parties' inputs, it does not (and cannot) equivocate by simulating protocol messages to be consistent with outputs of an ideal functionality. Later, when we use the definition, we will restrict ourselves to a class of protocols where such equivocation is not needed.

We also observe that online extractability is *not implied* by UC security. Consider a UC-secure protocol π that uses a trapdoor in the CRS for extraction, such as the oblivious transfer protocol of [PVW08]. Moreover, let $\mathcal{F}_{\mathsf{MPC}}$ be a general MPC functionality. We construct a protocol π' as follows:

1. Generate a CRS crs' for π, using $\mathcal{F}_{\mathsf{MPC}}$, by running a sampling algorithm for the CRS distribution inside $\mathcal{F}_{\mathsf{MPC}}$, seeded using secret randomness.
2. After crs' is output by $\mathcal{F}_{\mathsf{MPC}}$, run π using crs'.

Assuming that $\mathcal{F}_{\mathsf{MPC}}$ is UC-secure, π' is also UC-secure. To construct a simulator for π', one uses the simulator of π to generate a CRS crs that can be used for equivocation. Then, the simulator for π' programs $\mathcal{F}_{\mathsf{MPC}}$'s output to be crs and otherwise runs the simulator of π as before. Indistinguishability of the new simulator follows because of the indistinguishability of the simulator of π, as crs must be indistinguishable from crs' by assumption.

At the same time, π' is clearly not online-extractable because no $\mathcal{E}$ can change the outputs of $\mathcal{F}_{\mathsf{MPC}}$, which would be necessary in order to extract inputs of the adversary as in π.

Remark 1. Clearly, if $\mathcal{F}_{\mathsf{MPC}}$ was replaced with a normal CRS functionality then its output could be programmed. Hence, this makes the above counter-example somewhat contrived, and it might be that all "natural" UC protocols are indeed online-extractable. Unfortunately, a broader definition which may be implied by UC security[13] seems tricky to formalize, since there are always more convoluted counter-examples that can evade Definition 3 or similar requirements: for example, a protocol might use $\mathcal{F}_{\mathsf{MPC}}$ both to perform some useful computation, and simultaneously to generate a CRS used in a subsequent protocol.

[13] We believe that this is unsurprising: while our definition captures an observation of many known UC simulators, given the nature of the UC framework it seems hard to prove that all simulators must follow this simulation strategy.

Composition of Online Extractability. We now provide a lemma which shows under which conditions the online extractability property composes. For this, for protocols ρ, π and a functionality $\mathcal{F}$ we define $\rho^{\mathcal{F} \to \pi}$ to be the protocol that replaces the functionality $\mathcal{F}$ in ρ with an instance of π as usual in UC composition. We show that in such a case the composed protocol is online-extractable if ρ as well as π are online-extractable. The proof is provided in Appendix C.1 and simply embeds online extractability into the standard universal composition argument.

Lemma 1. *Let ρ be a protocol that UC-securely implements $\mathcal{F}_\rho$ in the $\mathcal{F}$-hybrid model and is online-extractable for a corrupt set $\mathcal{P}_\mathcal{A} \subset \mathcal{P}$. Let π be a protocol that UC-securely implements $\mathcal{F}$ and is online-extractable for corrupt $\mathcal{P}_\mathcal{A}$. Then $\rho^{\mathcal{F} \to \pi}$ is online-extractable with the same $\mathcal{F}_\rho$ and $\mathcal{P}_\mathcal{A}$.*

The computational overhead from this composition is additive for each functionality that gets replaced as we only run the new $\mathcal{E}^\pi$ parallel to π. We can therefore apply the lemma a polynomial number of times.

2-Message OT is Online-Extractable. To give an example of an online-extractable protocol, we observe that any 2-message OT protocol in the CRS model, such as [PVW08], is online-extractable against a corrupted receiver. We will later build on this for showing that VOLE can be realised with online-extractable protocols, in Appendix C.2. We assume a standard OT functionality $\mathcal{F}_{\mathsf{OT}}$, for instance, as in [PVW08].

Lemma 2. *Let Π be any 2-message protocol that securely realizes the $\mathcal{F}_{\mathsf{OT}}$ functionality in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model. Then, Π is online-extractable for a corrupted receiver.*

Proof. Recall that in 2-message OT, the receiver must always send the first message. Without loss of generality, any simulator for a corrupted receiver can then be defined in terms of the randomized algorithms:

- crsSim: On input the security parameter, it outputs a crs together with a trapdoor τ.
- Ext_A: On input crs, τ and a receiver message msg_A, it outputs an extracted input $\sigma \in \{0, 1\}$.
- Sim_A: On input crs, τ, a message msg_A and the receiver's output x_σ, it outputs a simulated sender message msg_B.

The UC simulator uses crsSim to emulate $\mathcal{F}_{\mathsf{CRS}}$, and then, on receiving the adversary's message msg_A, extracts its input σ using Ext_A and the trapdoor, before receiving x_σ from the ideal functionality and simulating the sender's response using Sim_A.

We define the extractor $\mathcal{E}$, which starts by emulating $\mathcal{F}_{\mathsf{CRS}}$ using crsSim, and then uses the trapdoor τ and the intercepted msg_A from the honest receiver to extract an input σ with Ext_A, which it writes to the special extractor tape. The only difference between the two executions $[\pi]_\mathcal{E} \circ \mathcal{A}$ and $\pi \circ \mathcal{A}$ to any $\mathcal{Z}$ is the way the CRS is sampled; but since Π is a secure protocol, these must be indistinguishable. Furthermore, since the extractor tape is defined using Ext_A, it is distributed identically to the extracted input in the UC simulation, as required.

4 Homomorphic Commitments Based on VOLE

Functionality $\mathcal{F}_{\mathsf{HCom}}$

Parameters: Finite field $\mathbb{F}_p$. The functionality runs between a sender P_S and a set of receiver parties $\mathcal{P}_R = \{P_1, \dots, P_n\}$. We assume all parties have agreed upon public identifiers id_x, for each variable x used in the computation. For a vector $\boldsymbol{x} = (x_1, \dots, x_m)$, we write $\mathsf{id}_{\boldsymbol{x}} = (\mathsf{id}_{x_1}, \dots, \mathsf{id}_{x_m})$.

Input: On receiving $(\mathsf{Input}, \mathsf{id}_{\boldsymbol{x}}, \boldsymbol{x})$ from P_S, where $\boldsymbol{x} \in \mathbb{F}_p^l$, where l is the length of the vector, and $(\mathsf{Input}, \mathsf{id}_{\boldsymbol{x}})$ from all the other parties, store the pair $(\mathsf{id}_{\boldsymbol{x}}, \boldsymbol{x})$, and send $\mathsf{InputReceived}$ to $\mathcal{A}$.

Linear Operation: On receiving $(\mathsf{LinComb}, \mathsf{id}_{\boldsymbol{z}}, \mathsf{id}_{\boldsymbol{x}}, \mathsf{id}_{\boldsymbol{y}}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ from every P_i, compute $\boldsymbol{z} = \boldsymbol{\alpha} \odot \boldsymbol{x} + \boldsymbol{\beta} \odot \boldsymbol{y} + \boldsymbol{\gamma}$ and store $(\mathsf{id}_{\boldsymbol{z}}, \boldsymbol{z})$.

Random: On receiving $(\mathsf{Random}, \mathsf{id}_{\boldsymbol{r}}, m)$ from all parties:

1. Sample $\boldsymbol{r} \leftarrow \mathbb{F}_p^m$. If $P_S \in \mathcal{P}_{\mathcal{A}}$, instead receive $\boldsymbol{r}$ from $\mathcal{A}$.
2. Store $(\mathsf{id}_{\boldsymbol{r}}, \boldsymbol{r})$ and send $\boldsymbol{r}$ to P_S.

Private Opening: On receiving $(\mathsf{PrivOpen}, \mathsf{id}_{\boldsymbol{x}}, P_j)$ from P_S and if $(\mathsf{id}_{\boldsymbol{x}}, \boldsymbol{x})$ is stored, send $\boldsymbol{x}$ to P_j.

Output: On receiving $(\mathsf{Output}, \mathsf{id}_{\boldsymbol{z}})$ from every P_i, where $\mathsf{id}_{\boldsymbol{z}}$ has been stored previously, if $P_S \in \mathcal{P}_{\mathcal{A}}$, send Abort to the parties $\mathcal{A}$ chooses, and deliver $\boldsymbol{z}$ to the rest. If $P_S \in \mathcal{P}_H$, deliver $\boldsymbol{z}$ to all parties.

Fig. 2: Functionality for a Homomorphic Commitment

In this section, we first define a functionality for homomorphic commitment, $\mathcal{F}_{\mathsf{HCom}}$, which will be used as a building block in our preprocessing phase. We then show how to efficiently instantiate this with a sender-receiver protocol based on VOLE, giving security with abort. Using the compiler of Section 5, this can be directly upgraded to achieve identifiable abort.

The functionality, shown in Fig. 2, allows a sender to input values that will be committed, as well as have random committed values sampled by the functionality. $\mathcal{F}_{\mathsf{HCom}}$ allows linear operations to be performed on the commitments, and for values to be opened privately to any one receiver, as well as publicly to all receivers. Note that $\mathcal{F}_{\mathsf{HCom}}$ only supports selective abort, and not unanimous abort. However, our compiler to identifiable abort only requires a protocol with selective abort.

Protocol Π_{HCom}

Parameters: Extension field $\mathbb{F}_{p^r}$, a sender P_S and receivers $\mathcal{P}_R = \{P_1, \ldots, P_n\}$.

Initialize: Every pair of parties (P_S, P_i), for $P_i \in \mathcal{P}_R$, calls an instance of $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ with Init, so P_i receives $\Delta^i \in \mathbb{F}_{p^r}$.

Input: P_S commits to an input $x \in \mathbb{F}_p$:

1. P_S broadcasts $x - l_j$, where l_j is the next available random value, to all the parties. If no such l_j is available, run the **Random** procedure.
2. Each $P_i \in \mathcal{P}_R$, locally updates its key as $K_S^i[\![x]\!] = K_S^i[\![l_j]\!] - \Delta^i \cdot (x - l_j)$. P_S sets $M_i^S[\![x]\!] = M_i^S[\![l_j]\!]$.
3. $P_i \in \mathcal{P}_R$ sets $\langle x \rangle^i = K_S^i[\![x]\!]$ and P_S sets $\langle x \rangle^S = (x, \{M_i^S[\![x]\!]\}_{i \in [1,n]})$, for $P_i \in \mathcal{P}_R$.

Linear Operation: To compute $z = \alpha \cdot x + \beta \cdot y + \gamma$, for public $\alpha, \beta, \gamma \in \mathbb{F}_{p^r}$, where $\langle x \rangle, \langle y \rangle$ have been committed, the parties locally compute $\langle z \rangle = \alpha \cdot \langle x \rangle + \beta \cdot \langle y \rangle + \gamma$.

Random: To generate m random commitments $\langle l_1 \rangle, \ldots, \langle l_m \rangle$:

1. P_S samples a seed s.
2. Each pair of parties (P_S, P_i), for $P_i \in \mathcal{P}_R$, calls $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$, with P_S sending (Extend, s) and P_i sending Extend. P_S obtains $\{\boldsymbol{u}^S, \boldsymbol{M}_i^S\}$ and P_i receives $\boldsymbol{K}_S^i = M_i^S - \boldsymbol{u}^S \cdot \Delta^i$.
3. Each $P_i \in \mathcal{P}_R$ sets $\langle l_j \rangle^i = K_{S,j}^i$ and P_S sets $\langle l_j \rangle^S = (u_j^S, \{M_{i,j}^S\}_{i \in [1,n]})$, for $j \in [1, m+1]$.
4. The parties do the following to check the consistency of inputs to $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$:
   (a) Call $\mathcal{F}_{\mathsf{Rand}}$ to get random values $\chi_1, \ldots, \chi_m \in \mathbb{F}_{p^r}$.
   (b) Locally compute $\langle C \rangle = \sum_{j=1}^{m} \chi_j \cdot \langle l_j \rangle + \langle l_{m+1} \rangle$
   (c) Write $\langle C \rangle^S = (C, \{M_i^S\}_{i \in [n]})$ and $\langle C \rangle^i = K_S^i$.
   (d) P_S broadcasts C, and privately sends M_i^S to each P_i.
   (e) Each P_i checks that $M_i^S = C \cdot \Delta^i + K_S^i$, for $i \in [1, n]$. If the check fails, abort.

Private Output: To open a value $\langle x \rangle$ to a receiver P_i, P_S privately sends x, $M_i^S[\![x]\!]$ to P_i. P_i checks that $M_i^S[\![x]\!] = \Delta^i \cdot x + K_S^i[\![x]\!]$ and aborts if it fails.

Output: To open a vector of values $\langle \boldsymbol{z} \rangle$, P_S sends $\boldsymbol{z}$, $\boldsymbol{M}_i^S[\![\boldsymbol{z}]\!]$ to P_i. Each P_i checks that $\boldsymbol{M}_i^S[\![\boldsymbol{z}]\!] = \Delta^i \cdot \boldsymbol{z} + \boldsymbol{K}_S^i[\![\boldsymbol{z}]\!]$. If the checks fail, P_i outputs abort. Otherwise, P_i outputs $\boldsymbol{z}$.

Fig. 3: Protocol for a Homomorphic Commitment

Information-theoretic MACs. We now introduce the notation for information-theoretic MACs as we use it in the protocol, building on Section 2.2. In the protocol, the sender P_S will be committed to values in $\mathbb{F}_p$ by holding a MAC on $x \in \mathbb{F}_p$ for each receiver, under keys known only to the receivers. The linear MAC with a receiver P_i is defined as,

$$M_i^S[\![x]\!] = x \cdot \Delta^i + K_S^i[\![x]\!]$$

where $\Delta^i \in \mathbb{F}_{p^r}$ is a long-term or global key and $K_S^i[\![x]\!] \in \mathbb{F}_{p^r}$ is a local key, used only for the MAC on x. Both keys are held by receiver P_i, while P_S holds x and $M_i^S[\![x]\!]$ (for each $i \in [n]$).[14] We occasionally write M_i^S, K_S^i when it is clear from context which value is being MACed.

When x is MACed with every other receiver, we use the notation

$$\langle x \rangle = \{(x, \{M_i^S[\![x]\!]\}_{i\in[n]}), K_S^1[\![x]\!], \ldots, K_S^n[\![x]\!]\}$$

We write $\langle x \rangle^i$ to denote the parts of $\langle x \rangle$ known to P_i, that is, $\langle x \rangle^i = K_S^i[\![x]\!]$ if $i \in [n]$ and $\langle x \rangle^S = (x, \{M_i^S[\![x]\!]\}_{i\in[n]})$ for the sender P_S.

We write $\langle x \rangle + \langle y \rangle$ to denote addition of each party's respective components, which gives a valid set of MACs $\langle x + y \rangle$, thanks to the linearity of the MACs. We also write $\langle x \rangle + \gamma$ to denote adding a public constant γ to $\langle x \rangle$, which is done by having P_S add γ to x, while each receiver P_i subtracts $\gamma \cdot \Delta^i$ from $K_S^i[\![x]\!]$, giving $\langle x + \gamma \rangle$.

4.1 Protocol with Abort

Our protocol (Fig. 3) is based on a similar MAC generation protocol from [RS22], with the difference that we only have a single sender instead of n senders, which allows us to simplify the protocol. MACs are set up using the VOLE functionality $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ (Fig. 1) introduced in Section 2.2, which generates a batch of random MACed values between two parties. Importantly, even though the authenticated values are random, the $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ functionality allows the sender to program these by providing a seed, such that when running two instances of $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ among different receivers, it ends up committed to the same set of random values.

Consistency Check. Since $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ does not guarantee that in each pair (P_S, P_i) for $i \in \mathcal{P}_R$, P_S inputs the same seed s, we use a consistency check in Π_{HCom}. In the check, the receivers challenge the sender to open a linear combination of all the committed values, with an extra random mask (l_{m+1}) to ensure privacy of the opened combination. We formalise the security of the check by modelling the errors introduced by a corrupt P_S as follows.

Suppose that P_S used inconsistent seeds with two receivers P_1 and P_2. Since the seeds are used to compute the $\boldsymbol{u}$ values using the Expand function, this will result in two different $\boldsymbol{u}$ values, say, $\boldsymbol{u}^1$ and $\boldsymbol{u}^2$, and hence, different commitments $\langle l_j^1 \rangle, \langle l_j^2 \rangle$. Without loss of generality, define the seed used with party P_1 to be the correct seed. In the security proof, the simulator can extract all seeds and then compute the errors $\delta_j^2 = l_j^2 - l_j^1$, for $j \in [1, m+1]$. If both P_S and the receiver party are corrupt, we set the errors to be 0. We prove that an adversary cannot pass the check with inconsistent seeds except with negligible probability via the following lemma. Proofs for the lemma as well as the theorem are given in Appendix D.

[14] Note that the index in superscript denotes the party who holds a value.

Lemma 3. *Suppose $P_S \in \mathcal{A}$ introduces errors of the form δ_j^i with party P_i, for $j \in [m+1]$, in the* Random *command in Π_{HCom} (Fig. 3). If the consistency check passes, then every pair of parties (P_S, P_i), for $i \in [1, n]$ hold a secret sharing of $l_j \cdot \Delta^i$, for $j \in [m]$. In other words, $\delta_j^i = 0$, for every i and $j \in [m]$, except with probability $1/|\mathbb{F}|$.*

Theorem 1. *Protocol Π_{HCom} UC-securely realises the functionality $\mathcal{F}_{\mathsf{HCom}}$ assuming a broadcast channel in the presence of a malicious adversary that can statically corrupt up to $n-1$ parties, in the $(\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model.*

4.2 Online Extractability of Π_{HCom}

Lemma 4. *Protocol Π_{HCom} in Fig. 3 is online-extractable, for any adversary corrupting the sender and any subset of receivers.*

Proof. Let $\mathcal{S}$ be the simulator for Π_{HCom} given in the proof of Theorem 1, for the case when the sender and a subset of the receivers are corrupted. In Π_{HCom}, there is never any communication from a receiver to the sender, so the only task of $\mathcal{S}$ is to emulate the hybrid functionalities $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ and $\mathcal{F}_{\mathsf{Rand}}$ towards the adversary, while interacting with the $\mathcal{F}_{\mathsf{HCom}}$ functionality. We can therefore use $\mathcal{S}$ to define the extractor $\mathcal{E}$, running in the execution $[\pi]_{\mathcal{E}}$, as follows:

- $\mathcal{E}$ runs an internal copy of $\mathcal{S}$; since $\mathcal{E}$ receives any message sent from the corrupt sender to an honest receiver, it can forward these messages to $\mathcal{S}$, acting as the adversary.
- Whenever $\mathcal{A}$ sends a message to a hybrid functionality $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$, $\mathcal{E}$ forwards the message to $\mathcal{S}$.
- Whenever $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{HCom}}$ with some message msg, $\mathcal{E}$ outputs msg on its special extractor tape. $\mathcal{E}$ responds to $\mathcal{S}$ exactly as $\mathcal{F}_{\mathsf{HCom}}$ would; this is possible because P_S is corrupted, so $\mathcal{E}$ knows all of the committed inputs and can correctly open them as needed. If $\mathcal{F}_{\mathsf{HCom}}$ aborts, then $\mathcal{E}$ aborts.

To show that $\mathcal{E}$ is a good extractor, we first require that the executions $\pi \circ \mathcal{A}$ and $[\pi]_{\mathcal{E}} \circ \mathcal{A}$ are indistinguishable. The only difference between the two executions is that $\mathcal{E}$ is simulating the $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$ instances, and also may abort in case the underlying simulator $\mathcal{S}$ aborts. However, it follows from the proof of Theorem 1 that these differences are negligible.

Secondly, we must show that the special extractor tape in $[\pi]_{\mathcal{E}}$ is indistinguishable from the special functionality tape of $\hat{\mathcal{F}}_{\mathsf{HCom}}$ in $\hat{\mathcal{F}}_{\mathsf{HCom}} \circ \mathcal{S}$ to any environment $\mathcal{Z}$. This is trivially true, because $\mathcal{E}$ is running $\mathcal{S}$ the same way as in an ideal execution, and the extractor tape of $\mathcal{E}$ contains exactly the messages $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{HCom}}$.

Online Extractability of VOLE Protocol. To use Π_{HCom} in our compiler for identifiable abort, it is not enough that Π_{HCom} is online-extractable on its own, since the compiler from Section 5 requires that Π_{HCom} only uses $\mathcal{F}_{\mathsf{Rand}}$ and/or $\mathcal{F}_{\mathsf{CRS}}$ as its hybrid functionalities. In Appendix C.2, we show that $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$

can be replaced with a VOLE protocol in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model, where the sender plays the OT receiver, and this protocol is online-extractable when the sender is corrupted. Since we showed in Lemma 2 how to realize $\mathcal{F}_{\mathsf{OT}}$ in an online-extractable way, by applying composition (Lemma 1), this gives an online-extractable protocol for $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}$ in the $\mathcal{F}_{\mathsf{CRS}}$-hybrid model.

The identifiable abort version of $\mathcal{F}_{\mathsf{HCom}}$, $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ appears in Fig. 17.

5 Compiling to Identifiable Abort

In this section, we show how to compile a protocol with active security with selective abort, and online extractibility, into a protocol that achieves identifiable abort. More specifically, we handle any class of protocols that are in the CRS model and are sender-receiver protocols, where the receivers do not have any private inputs and do not have any communication between them. This is defined formally below.

Definition 5. *Let Π be a protocol realizing a functionality $\mathcal{F}$ in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model. We say that Π is a sender-receiver protocol if (1) No receiver has private inputs and only interacts with the sender, except when communicating with $\mathcal{F}_{\mathsf{CRS}}$ or $\mathcal{F}_{\mathsf{Rand}}$; and (2) Whenever the sender P_S, with random tape ρ_S, has an input* inp *and sends a message to a receiver, this is done with either:*

- *A broadcast channel, using a function* $\mathsf{NextBC}(\rho_S, \mathsf{inp}, \mathsf{state})$*, which outputs an updated* state *and the message* msg *to be broadcast. The* view_S *may contain any outputs from $\mathcal{F}_{\mathsf{CRS}}$ or $\mathcal{F}_{\mathsf{Rand}}$, but is otherwise only used by* NextBC*.*
- *Private communication to receiver P_i, using a function* $\mathsf{NextMsg}(\rho_S, P_i, \mathsf{msgs}_i, \mathsf{state})$*, where* msgs_i *contains the set of messages previously received from P_i.*

In particular, this definition implies that any messages sent from the sender to a receiver, including via the broadcast channel, cannot depend on any previous message sent from another receiver to the sender.

It is straightforward to see that if we take Π_{HCom} (Fig. 3), and replace $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}$ with any secure 2-party protocol in the CRS model, we obtain a sender-receiver protocol. In **Input**, **Private Output**, **Batch Output**, and **Output**, P_S is the only party sending messages to the receivers in Π_{HCom}. In **Random**, it is clear that the messages sent from P_S to the receivers, and as input to $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}$, only depend on P_S's random tape and the messages received from $\mathcal{F}_{\mathsf{Rand}}$. Since $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}$ is a two-party functionality, replacing it with a secure two-party protocol ensures that the protocol messages to P_i still cannot depend on the view of any other receiver P_j.

5.1 The Compiler

In the protocol (Fig. 6), the parties start by picking a public and secret key pair for a signature scheme, and broadcast the public key. We use an EUF-CMA secure signature scheme $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$. The compiler runs the original protocol Π, and in each round, the parties add signatures to every message they are supposed to send. If any signature does not verify, or a message was not received, the receiving party P_i initiates the complaint procedure in Fig. 4, which forces the sending party to broadcast the message to all parties (or be identified as a cheater).

Subroutine $\mathsf{Complain}(P_i, P_j)$

1. P_i broadcasts $(\mathsf{Complain}, P_j)$.
2. Let (m, σ) be the last message sent from P_j to P_i in round r (if P_j was honest). P_j broadcasts (m, σ).
3. All parties check that $\mathsf{Ver}(\mathsf{pk}_j, \sigma, (r \| P_j \| P_i \| m)) = 1$. If not, or if P_j does not broadcast the message, the parties output abort_j.

Fig. 4: Complaint procedure for a missing message from P_j to P_i

Algorithm $\mathsf{VerifyAbort}\,(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, r)$

1. Using pk_i, check all $(m^r_{S,i}, \sigma^r_{S,i})$ pairs in view_i. If any check fails, output fail.[a]
2. Use view_i and ρ_i, and the round number r to reconstruct what the messages of the receiver should have been according to $\mathsf{NextMsg}_\Pi$. If the output of the receiver is abort, output good. Else, output fail.

[a] An honest receiver would not have an invalid signature in their view without having entered the Complaint procedure which would have either identified the cheater or returned a valid signature.

Fig. 5: Algorithm for verifying whether a receiver should have aborted.

The main challenge is to handle the case where Π aborts, and we use different strategies based on the party which aborts. If there was an abort in Π, the aborting party starts the Abort phase of the protocol, where the parties identify the cheater as follows. If a receiver party, say P_i, aborts, then since Π is a sender-receiver protocol, it must be the case that either P_S or P_i is a cheater, so the parties just need to establish which. We therefore have P_i broadcast its view, and open its random tape to all the parties. The rest of the parties can locally check if P_i cheated by running the VerifyAbort algorithm, which verifies the correctness of the messages it sent by recomputing the actual messages using the NextMsg function. VerifyAbort has the guarantee that if run with an honest P_i's view and random tape, it always outputs good, and that it is not possible to frame an honest P_i by making it output fail because that would require forging a signature. Parties abort with either abort_i or with abort_S depending on who cheated.

On the other hand, if P_S was the party that aborted, the natural approach would be to have P_S broadcast its view and random tape as in the earlier case. However, we do not want to reveal the sender's random tape. We do not want the sender broadcasting even its view with all the receiver parties, as this poses a

Compiler $\Pi^{\mathsf{IA}}_{\mathsf{Cmp}}$

Let Π be a sender-receiver protocol that realises $\mathcal{F}$, is actively secure with selective abort and supports online extractability. Π uses a CRS. We assume that Π is a protocol with one sender P_S and a set of receivers $P_j \in [1, n]$. All the calls to functionalities inside of Π are replaced by their corresponding protocols.

1. Before any step of the protocol is executed, each P_i sends $(\mathsf{Commit}, P_i, \rho_i)$ to $\mathcal{F}_{\mathsf{Commit}}$, where ρ_i is the random tape.
2. Each party P_i also samples a $(\mathsf{pk}_i, \mathsf{sk}_i)$ pair, and broadcasts pk_i.
3. Run the Π protocol as follows. In each round r of the protocol,
   (a) Let $m^r_{i,j}$ be the message that P_i should have sent to P_j, according to $\mathsf{NextMsg}_\Pi$. Note that either P_i or P_j must be P_S.
   (b) P_i sends $(m^r_{i,j}, \sigma^k_{i,j})$ to P_j, where $\sigma^r_{i,j} = \mathsf{Sig}(\mathsf{sk}_i, r||P_i||P_j||m^r_{i,j})$.
   (c) P_j checks $\mathsf{Ver}(\mathsf{pk}_i, \sigma^r_{i,j}, (r||P_i||P_j||m^r_{i,j})) = 1$. If not, or if P_i did not send a message at all, P_j calls $\mathsf{Complain}(P_j, P_i)$ (Fig. 4)
   (d) If any party P_i terminates with output abort then it initiates the Abort procedure.

Abort:

1. If a receiver P_i aborted in round r:
   (a) P_i broadcasts $(\mathsf{abort}, \mathsf{view}_i)$ and opens ρ_i publicly using $\mathcal{F}_{\mathsf{Commit}}$.
   (b) All parties run $\mathsf{VerifyAbort}\,(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, r)$ (Fig. 5) to establish if P_i cheated.
   (c) If $\mathsf{VerifyAbort}$ returns fail, the parties output abort_i, else output abort_S.
2. If the sender P_S aborted:
   (a) P_S broadcasts abort.
   (b) All receivers P_i send view_i to P_S and privately open ρ_i to P_S using $\mathcal{F}_{\mathsf{Commit}}$.
      i. If P_S does not receive the view of some P_i, it broadcasts a complaint message for P_i. P_i is forced to broadcast (view_i)[a] and publicly open ρ_i by calling $\mathcal{F}_{\mathsf{Commit}}$ with Open.
      ii. If P_i does not broadcast, then everyone outputs abort_i.
   (c) P_S runs $\mathsf{IA.Identify}\,(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, \mathsf{pk}_S, \mathsf{view}_{S,i})$ (Fig. 7) for all receivers P_i to establish who cheated. $\mathsf{view}_{S,i}$ is the view of the the sender P_S that contains only the messages from one particular party P_i.
   (d) P_S broadcasts $\mathsf{view}_{S,i}$ for the cheating party P_i. P_i broadcasts view_i and publicly opens ρ_i using $\mathcal{F}_{\mathsf{Commit}}$.
   (e) All honest parties run $\mathsf{IA.Identify}\,(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, \mathsf{pk}_S, \mathsf{view}_{S,i})$ and output abort_i. If P_S never broadcast the views, then they identify P_S as the cheater.

[a] This is ok because in this case either the receiver or the sender is corrupt.

Fig. 6: Compiler for identifiable abort

problem for simulation. In this case, we are operating with an honest sender, and a subset of receivers that are corrupt. Because this is a sender-receiver protocol, the honest receivers may have private outputs from the sender. This means the simulator cannot forge a view for the honest sender to give to the adversary.

Instead, we have all the receivers send their views and random tapes to the sender. The sender locally runs $\mathsf{Identify}$ (Fig. 7) on all the views and random tapes, including its own view, to identify the receiver party that cheated. If a receiver P_i does not send its view to the sender then the sender broadcasts a complaint message for P_i who is then forced to broadcast its view and its random tape. $\mathsf{Identify}$ can reconstruct what an honest receiver should have sent, based on the random tape of the receiver and the $\mathsf{NextMsg}$ function. Then it compares these messages to the messages from the sender's view, which allows it to always identify if the receiver cheated. P_S then broadcasts its view *only* with respect to the cheating party, along with that party's view and random tape. The other honest parties can locally run $\mathsf{Identify}$ on these to be convinced that P_i was the cheater. This avoids the problem of the simulator having to send the *full* view of the honest sender. A formal description of the compiler appears in Fig. 6.

Algorithm $\mathsf{IA.Identify}\ (\mathsf{pk}_i, \mathsf{view}_i, \rho_i, \mathsf{pk}_S, \mathsf{view}_S)$

1. Using the random tape ρ_i and view_i, first check if the ρ_i is the one that was committed to and then compute what the messages of P_i in each round should be. Let those be $m^r_{i,S}$.
2. Check if $m^r_{i,S} \neq \hat{m}^r_{i,S}$, where $\hat{m}^r_{i,S}$ are P_i's messages in view_S, for all rounds r of the protocol so far. If any of them are inconsistent, output P_i as the cheater.
3. Else if any of the signatures from view_S fail the check $\mathsf{Ver}\left(pk_i, \sigma_S, (r_S || P_S || P_i || \hat{m}^r_{i,S})\right) = 1$, output P_i as the cheater.
4. Else, verify signatures sent by P_S. If any of them are not valid, output P_S as the cheater.

Fig. 7: Algorithm for identifying a cheater.

Theorem 2. *Let Π be a perfectly correct, sender-receiver protocol that UC-securely realises a functionality $\mathcal{F}$ with active security and dishonest majority, and supports online extractability when the sender and a subset of receivers are corrupt. Let $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a EUF-CMA secure signature scheme. Then the compiled protocol $\Pi^{\mathsf{IA}}_{\mathsf{Cmp}}$ securely realizes $\mathcal{F}$ with active security in the $\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Commit}}$-hybrid model, and achieves identifiable abort.*

Due to space constraints, we defer the full proof of the theorem to Appendix E and provide a proof sketch.

Proof (Sketch). First, whenever $\mathcal{A}$ communicates with $\mathcal{F}_{\mathsf{CRS}}$, $\mathcal{S}$ calls the extractor $\mathcal{E}$ which picks whichever CRS it wants and $\mathcal{S}$ forwards it to $\mathcal{A}$. We have two cases of corruption:

1. The sender and a subset of the receivers are corrupt. In this case, $\mathcal{S}$ can use $\mathcal{E}$ and an honest execution of the protocol to forward $\mathcal{A}$'s inputs to

$\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. The two interesting cases of abort are: a corrupt receiver or a corrupt sender. Those cases are taken care of by running $\mathsf{VerifyAbort}$ and $\mathsf{IA.Identify}$ respectively on the random tapes that $\mathcal{S}$ received.

2. A subset of receivers is corrupt. In this case $\mathcal{S}$ uses the UC simulator $\mathcal{S}_\Pi$ of the original protocol and whenever $\mathcal{S}_\Pi$ communicates with $\mathcal{F}_{\mathsf{HCom}}$, $\mathcal{S}$ forwards all the messages to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. The abort here can happen either by a corrupt receiver or an honest sender. As before, they are taken care of by running $\mathsf{VerifyAbort}$ and $\mathsf{IA.Identify}$ on the random tapes of the corrupt parties that it received.

In both cases, the correctness of the identified cheaters follows from the EUF-CMA property of the signature scheme and the binding property of $\mathcal{F}_{\mathsf{Commit}}$.

5.2 Identifiable Cheating

We now present another transformation, which does not directly yield identifiable abort, but on the other hand, is not restricted to sender-receiver protocols. Similarly to the previous compiler, we use signatures to verify point-to-point communication. This ensures that a protocol transcript is verifiable, in the sense that, in an execution where an honest party aborts, a cheater can be identified given the views of all parties, even if a corrupt party may lie about its view. We will use this transformation as part of our preprocessing protocol in Section 6, to ensure that the triple generation subprotocol can be verified in case of an abort.

Protocol assumptions. We let $\mathsf{NextMsg}$ denote a component of each party's state transition function that, on input the party identifier P_i, random tape ρ_i, the view of P_i, and round r, outputs the next message m that P_i is supposed to send. We assume that the protocol Π realizes a functionality $\mathcal{F}$, and is in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model.

Definition 6 (Dishonest execution). *Consider a non-aborting execution of protocol* Π *between parties* $P_1, \dots, P_n$ *wit h random tapes* $(\rho_1, \dots, \rho_n)$*, and a set* $\mathcal{P}_\mathcal{A}$ *of corrupted parties. We say that the execution was dishonest with respect to* $\mathcal{P}_\mathcal{A}$*, if there exists at least one honest party whose view in the execution is different compared to the view of the same party in an honest execution of* Π *on* $(\rho_1, \dots, \rho_n)$.

The notion is defined via a game and a polynomial time algorithm $\mathsf{Identify}$. The idea of the definition is as follows, we first run the protocol Π with a set of parties $P_1, \dots, P_n$. The protocol generates the set of views $\{\mathsf{view}_i\}_{i\in[1,n]}$. The adversary is allowed to replace up to $n-1$ views with corrupted ones. We show that the identifiable cheating compiler guarantees that, except with negligible probability, given all the views, the random tapes of the parties, and public keys of all the parties, the $\mathsf{Identify}$ algorithm successfully identifies the cheating party. Formally, the definitions are as follows:

Definition 7 (Identifiable cheating). *Let* Π *be an actively secure protocol that UC-securely realises* $\mathcal{F}$ *and in the* $\mathcal{F}_{\mathsf{CRS}}$*-hybrid model. Let* $\mathsf{Identify}$ *be a deterministic polynomial-time algorithm with the syntax:*

Compiler $\Pi^{\mathsf{IC}}_{\mathsf{Cmp}}$

Let Π be a maliciously secure protocol with abort that realizes $\mathcal{F}$ and is in the $(\mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model. Let $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a signature scheme.

```
1. Each $P_i$, for $i \in [1,n]$, samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ and broadcasts $\mathsf{pk}_i$.
2. The parties run $\Pi$. Let $\rho_i$ be the random tape of each party $P_i$. When $P_i$ receives a message, $\mathsf{inp}$, in round $r$:
   (a) If $\mathsf{inp}$ is a message from $P_j$ of the form $(m, \sigma)$, $P_i$ checks that $\mathsf{Ver}(\mathsf{pk}_j, \sigma, (P_j \| P_i \| m \| r-1) = 1$. If the check fails, run $\mathsf{Complain}(P_i, P_j)$.
   (b) If $P_i$ is next instructed to send a message to some party $P_j$:
       i. Let $(m_{i,j}, \mathsf{state}) = \mathsf{NextMsg}(P_i, \rho_i, \mathsf{view}_i, r)$.
       ii. Let $\sigma_{i,j} = \mathsf{Sig}(\mathsf{sk}_i, P_i || P_j || m_{i,j} \| r)$
       iii. Send $(m_{i,j}, \sigma_{i,j})$ to $P_j$
       Otherwise, $P_i$ executes its next instruction as usual.
```

Fig. 8: Compiler for identifiable cheating

Algorithm $\mathsf{Identify}\left((\mathsf{pk}_i, \mathsf{view}_i, \rho_i)_{i \in [1,n]}\right)$

```
1. Emulate an execution of $\Pi$ with virtual parties $P_1, \ldots, P_n$ and random tapes $\rho_1, \ldots, \rho_n$.
2. At each step where $P_i$ sends a message $m_{i,j}$ to $P_j$ in round $r$:
   (a) Retrieve the next message $(\hat{m}_{i,j}, \sigma)$ from $\mathsf{view}_j$.
   (b) Check whether $m_{i,j} = \hat{m}_{i,j}$, where $m_{i,j} = \mathsf{NextMsg}(P_i, \rho_i, \mathsf{view}_i, r)$. If not, output $P_i$ as a cheater. If $m_{i,j} = \hat{m}_{i,j}$, check if there was a $\mathsf{Complain}$ procedure initiated in the list of broadcasted messages. If there was one, output $P_j$ as a cheater.
   (c) Check whether $\mathsf{Ver}(\mathsf{pk}_i, \sigma, (P_i \| P_j \| \hat{m}_{i,j} \| r)) = 1$. If not, output $P_j$ as a cheater.
3. If $\Pi$ ends successfully without identifying a cheater, output $\perp$.
```

Fig. 9: Algorithm for identifying a cheater.

- $\mathsf{Identify}\left((\mathsf{pk}_i, \rho_i, \mathsf{view}_i)_{i \in [n]}\right)$: *On input the public keys, random tapes and views of all the parties,* $\mathsf{Identify}$ *either outputs a corrupt party* P_i *or an honest execution symbol* $\perp$.

A protocol Π *supports identifiable cheating if for any P.P.T adversary* $\mathcal{A}$ *it holds that:*

$$\Pr[\mathsf{Exp}^{\mathsf{ic}}_{\mathcal{A},\Pi}(\lambda) = 1] \leq v(\lambda)$$

where $\lambda \in \mathbb{N}$, v *is a negligible function and* $\mathsf{Exp}^{\mathsf{ic}}_{\mathcal{A},\Pi}(\lambda)$ *is defined as in Fig. 10.*

The idea of our compiler $\Pi^{\mathsf{IC}}_{\mathsf{Cmp}}$ that ensures identifiable cheating is to have parties add signatures to their messages. In order for parties to sign and verify messages, each party chooses a public key and a secret key for a signature scheme, and broadcasts the public key before running the compiled protocol. If a party in

Experiment $\mathsf{Exp}^{\mathsf{ic}}_{\mathcal{A},\Pi}(\lambda)$

1. $\mathcal{A}$ corrupts a set of parties $\mathcal{P}_{\mathcal{A}} \subset \mathcal{P}$. Let $\mathcal{P}_H$ be the set of honest parties.
2. For each $i \in \mathcal{P}_H$, sample a random tape ρ_i.
3. For each $j \in \mathcal{P}_{\mathcal{A}}$, $\mathcal{A}$ chooses a random tape ρ_j.
4. The parties run Π, where P_i uses ρ_i as its random tape. Let view_i denote the list of messages received by P_i.
5. If all honest parties output abort_j for some $j \in \mathcal{P}_{\mathcal{A}}$, then output 0.
6. $\mathcal{A}$ receives $(\rho_i, \mathsf{view}_i)$, for $i \in \mathcal{P}_H$.
7. $\mathcal{A}$ outputs $\{\widetilde{\mathsf{view}}_j\}$ for $j \in \mathcal{P}_{\mathcal{A}}$. Redefine $\mathsf{view}_j := \widetilde{\mathsf{view}}_j$.
8. Output 1 if one of the following holds:
   - The execution of Π is dishonest with respect to $\mathcal{P}_{\mathcal{A}}$, and $\mathsf{Identify}\left((\mathsf{pk}_i, \rho_i, \mathsf{view}_i)_{i=1}^n\right) = \perp$
   - $\mathsf{Identify}\left((\mathsf{pk}_i, \rho_i, \mathsf{view}_i)_{i=1}^n\right) \in \{P_i\}_{i \in \mathcal{P}_H}$.

   Else, output 0.

Fig. 10: Experiment for Identifiable Cheating

the protocol thinks a message or the signature it received is invalid, it broadcasts a complaint message, upon which the sender of the message must broadcast the message and the signature to all the parties. The formal protocol for the identifiable cheating compiler appears in Fig. 8.

We prove that using this compiler with any protocol that is actively secure in the dishonest majority with abort, gives a protocol that has the identifiable cheating property.

Theorem 3. *Let Π be a protocol that UC-securely realises a functionality $\mathcal{F}$ with active security and dishonest majority. Let $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a EUF-CMA secure signature scheme. Then the compiled protocol Π_{Cmp} securely realises $\mathcal{F}$ with active security in the CRS model and using broadcast, and achieves the identifiable cheating property.*

Due to space constraints, we defer the full proof of the theorem to Appendix E.

6 Preprocessing

In this section, we build our preprocessing protocol with identifiable abort, using the homomorphic commitments with identifiable abort from the previous section. The preprocessing protocol allows parties to secret-share random values such that the secret is known to one party only, as well as to create sharings of multiplication triples, that is, two random values together with a sharing of their product. In both cases, all shares are homomorphically committed. Parties can also apply linear operations to these sharings without interaction, or open them with identifiable abort. The preprocessing functionality, abstracting this, is formally described in Fig. 11.

In the preprocessing protocol $\Pi^{\mathsf{IA}}_{\mathsf{Prep}}$, we will use (amongst other functionalities) n sessions of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$, where we denote by $\mathcal{F}^{\mathsf{IA},i}_{\mathsf{HCom}}$ the session where party P_i is sender

Functionality $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$

Parameters: Finite field $\mathbb{F}_p$, parties $P_1, \ldots, P_n$. The adversary is allowed to corrupt a subset of parties, denoted by $\mathcal{I}$.

Random Input: On receiving $(\mathsf{RandInput}, P_i, l)$ for some $i \in [1, n]$ from all parties:

1. Sample $\boldsymbol{r} \leftarrow \mathbb{F}_p^l$.
2. Store $(\mathsf{id}_{\boldsymbol{r}}, \boldsymbol{r})$ for a fresh $\mathsf{id}_{\boldsymbol{r}}$ and send $(\mathsf{id}_{\boldsymbol{r}}, \boldsymbol{r})$ to P_i and $\mathsf{id}_{\boldsymbol{r}}$ to everyone else.

Linear Operation: On receiving $(\mathsf{LinComb}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y, \alpha, \beta, \gamma)$ from every P_i where $\mathsf{id}_x, \mathsf{id}_y$ are assigned, id_z is unassigned and where $\alpha, \beta, \gamma \in \mathbb{F}_p$, compute $z = \alpha \cdot x + \beta \cdot y + \gamma$ and store (id_z, z).

Triple Generation: On receiving $(\mathsf{TripGen}, l)$ from all parties:

1. Sample $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_p$ of length l and let $\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b}$.
2. Store $(\mathsf{id}_{\boldsymbol{a}}, \boldsymbol{a}), (\mathsf{id}_{\boldsymbol{b}}, \boldsymbol{b}), (\mathsf{id}_{\boldsymbol{c}}, \boldsymbol{c})$ for unused $\mathsf{id}_{\boldsymbol{a}}, \mathsf{id}_{\boldsymbol{b}}, \mathsf{id}_{\boldsymbol{c}}$.
3. Send $(\mathsf{id}_{\boldsymbol{a}}, \mathsf{id}_{\boldsymbol{b}}, \mathsf{id}_{\boldsymbol{c}})$ to all parties.

Output: Upon receiving $(\mathsf{Output}, \mathsf{id}_x)$ from all parties and where id_x is assigned:

1. Send x to $\mathcal{A}$.
2. If $\mathcal{A}$ sends $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \subseteq \mathcal{I}, \mathcal{J} \neq \emptyset$, then send $(\mathsf{Abort}, \mathcal{J})$ to all the parties and terminate.
3. If $\mathcal{A}$ sends $\mathsf{Deliver}$ then output x to all parties.

Corrupt Party Behaviour: Whenever the adversary is supposed to send a value, it can choose to not send a value at all, triggering an abort. The functionality receives $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \subseteq \mathcal{I}$ from $\mathcal{A}$ and $\mathcal{J} \neq \emptyset$, sends it to all the parties and terminates.

Fig. 11: Functionality for Preprocessing

and all other parties are receivers, and refer to this as P_i's session of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. We use the notation $\langle \cdot \rangle_C$ to denote values that are additively shared and where each party P_i's share is committed using $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},i}$ where it is the sender. For a value $\langle x \rangle_C$, each party P_i holds $(x^i, \mathsf{id}_1, \ldots, \mathsf{id}_n)$, where id_i is the identifier where $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},i}$ stores the share x^i. Any linear operation performed on $\langle x \rangle$ can also be performed on the commitments, by calling $\mathsf{LinComb}$ with each $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ session. To open $\langle x \rangle_C$, each P_i calls $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},i}$ with $(\mathsf{Output}, \mathsf{id}_i)$, for $i \in [1, n]$, and all receivers now either receive x or $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J}$ indicates the set of cheating parties. $[x]$ denotes that $x \in \mathbb{F}_p$ is additively shared between the parties, that is, $x = x^1 + \ldots + x^n$ where P_i holds x^i.

The preprocessing protocol is described in Fig. 13 and Fig. 14. To generate a random input towards a party, say P_i, each P_j generates a private random value, committed using $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j}$, and then privately opens this to P_i. P_i sets its random input to be the sum of all the random values it receives across the n

Functionality $\mathcal{F}_{\mathsf{Triple}}$

Parameters: Finite field $\mathbb{F}_p$. Parties $P_1, \ldots, P_n$. The adversary is allowed to corrupt a subset of parties, denoted by $\mathcal{I}$. Denote the honest parties as $\mathcal{P}_H$.

Generate Triples: On receiving (Trip, l) from all the parties, sample a fresh set of l triples $(a_j, b_j, c_j) \in \mathbb{F}_p^3$ for $j \in [1, l]$. Output additive shares $[a_j], [b_j], [c_j]$ of the triple to each party.

Corrupt Parties: The adversary is allowed to choose its shares of the triples, as well as additive errors for the triples. If the errors are δ_a^i, δ_b^i for $i \in \mathcal{P}_H$ for a triple, the triple will now be computed as $c = a \cdot b + \Sigma_{i \in \mathcal{P}_H} \left(a_i \cdot \delta_b^i + b_i \cdot \delta_a^i\right)$.

Fig. 12: Functionality for unauthenticated triples

sessions of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$. Since the parties only use $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ functionalities here, cheater identification is trivial.

To generate triples, we rely on a secure-with-abort triple generation protocol, Π_{Trip}, that securely realizes the $\mathcal{F}_{\mathsf{Triple}}$ functionality (Fig. 12); this can be efficiently realized, for instance, using pairwise OLE correlations as in the preprocessing protocol of Le Mans [RS22]. We then compile Π_{Trip} to support identifiable cheating, using our compiler from Fig. 8, to obtain a protocol $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$.

After running $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$, each party gets unauthenticated shares of a batch of triples. These triples are then authenticated, by having parties commit to their shares using the respective sessions of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$, and then checked for correctness, using random challenges and a standard triple sacrifice protocol.

Notice that there can two types of errors when creating triples. Firstly, the triple generation protocol $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ might abort. The other kind of error is when $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ results in consistent shares of triples, but the adversary inputs inconsistent shares into $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$. This would lead to the triple sacrifice failing. If there is an abort in $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$, parties open their random tapes using $\mathcal{F}_{\mathsf{Commit}}$ and broadcast their views from $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$. Since $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ has identifiable cheating, parties can now run the $\mathsf{Identify}$ algorithm locally on input $(\mathsf{pk}_i, \mathsf{view}_i, \rho_i)_{i=1}^n$ to identify a corrupt party. If there is an abort in the sacrifice check, on the other hand, in addition to running $\mathsf{Identify}$, they also need to check consistency of the inputs to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ to outputs of $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$. In order to do this, they call $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ with Output across all sessions of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ and check that the inputs to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ match with the outputs of $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$. Here, any deviation allows to directly identify a cheater.

Theorem 4. *Suppose that protocol Π_{Trip} UC-securely implements $\mathcal{F}_{\mathsf{Triple}}$ against an active adversary corrupting at most $n-1$ parties.*

Then, the protocol $\Pi^{\mathsf{IA}}_{\mathsf{Prep}}$ (using the compiled protocol $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$) UC-securely implements the functionality $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ in the presence of a malicious adversary that statically corrupts up to $n-1$ parties, in the $(\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}, \mathcal{F}_{\mathsf{Commit}})$-hybrid model.

Protocol $\Pi^{\mathsf{IA}}_{\mathsf{Prep}}$ (Part 1)

Parameters: Finite field $\mathbb{F}_p$. Parties $P_1, \ldots, P_n$.

Initialize: Each P_i samples a random tape Rnd_i. P_i sends $(\mathsf{Commit}, P_i, \mathsf{Rnd}_i)$ while every other party receives P_i from $\mathcal{F}_{\mathsf{Commit}}$. Parties repeat this process with every P_i committing to its random tape. Parties also run the first step of the compiled $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ and each party obtains the verification keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$.

Random Input: P_i uses the next available random input sharing id_i. If it has no random inputs left, generate a batch of l as follows:

1. Each P_j calls $\mathcal{F}^{\mathsf{IA},j}_{\mathsf{HCom}}$ with $(\mathsf{Random}, \mathsf{id}_j, l)$, while the other parties call $\mathcal{F}^{\mathsf{IA},j}_{\mathsf{HCom}}$ acting as receivers. P_j receives $\boldsymbol{r}_j$ of length l.
2. Each P_j then calls each instance of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ with $(\mathsf{PrivOpen}, \mathsf{id_j}, P_i)$. P_i receives $\boldsymbol{r}_j$ for $j \in [1, n]$ and sets its random input as $\boldsymbol{x}_i = \Sigma^n_{j=1} \boldsymbol{r}_j$. It considers the value of id_i as the first unused element of $\boldsymbol{x}_i$.

Linear Operation: To compute $z = \alpha \cdot x + \beta \cdot y + \gamma$, parties set $\langle z \rangle_C = \alpha \cdot \langle x \rangle_C + \beta \cdot \langle y \rangle_C + \gamma$.

Output: To output a value $\langle \boldsymbol{x} \rangle_C$, each party calls all instances of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ with $(\mathsf{Output}, \mathsf{id}_{\boldsymbol{x}})$.

Fig. 13: Protocol for preprocessing

In the proof, we construct a simulator $\mathcal{S}$ which simulates honest parties and the ideal functionalities towards the adversary. For **Random Input** $\mathcal{S}$ just runs the protocol, except for a malicious receiver P_i where $\mathcal{S}$ equivocates an honest party's commitment in $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ to open to the output provided $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$. For **Output**, it does exactly the same. **Linear Operation** is entirely local, so simulation is trivial. For **Triple Generation** $\mathcal{S}$ runs the protocol, but will always abort if the $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ sessions contain values that are not consistent multiplication triples.

To show that $\mathcal{S}$'s output can only be distinguished from $\Pi^{\mathsf{IA}}_{\mathsf{Prep}}$ with the given probability, the main difference lies in the occurrence of aborts and the identified cheaters. The $1/(p-1)$ term comes from aborts that also happen in $\mathcal{S}$ if $d = 0$ (while the protocol never aborts). Concerning identified cheaters, we have that $\mathsf{Identify}$ identifies no or an honest party (i.e. the wrong party) with probability at most $\mathsf{negl}(\lambda)$ due to the Identifiable Cheating property of Π_{Trip}, while $\mathcal{S}$ always identifies corrupt dishonest parties.

Due to space constraints, we defer the full proof of the theorem to Appendix F.

Protocol $\Pi^{\mathsf{IA}}_{\mathsf{Prep}}$ (Part 2)

Triple Generation:

1. Parties run $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ using the random tapes Rnd_i for P_i. They receive additive shares of $2l$ triples – $[a_j], [b_j], [c_j]$, for $j \in [1, 2l]$. If any party notices an abort while running $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$, then it broadcasts Abort and all parties go to **Abort 1**.
2. Each P_i calls $\mathcal{F}^{\mathsf{IA},i}_{\mathsf{HCom}}$, where it acts as the sender, with $(\mathsf{Input}, \mathsf{id}_{a\text{-}j}, \mathsf{id}_{b\text{-}j}, \mathsf{id}_{c\text{-}j}, [a_j], [b_j], [c_j])$ for $j \in [1, 2l]$. Using the identifiers, parties form $\langle a\rangle_C, \langle b\rangle_C, \langle c\rangle_C$.
3. Parties call $\mathcal{F}_{\mathsf{Rand}}$ to receive public random values $\boldsymbol{t} \in (\mathbb{F}_p^*)^l$ and a set of random combiners $\chi_1, \ldots, \chi_l \in \mathbb{F}_p$.
4. For $i = 1, \ldots, l$, the parties do the following (in parallel):
   (a) For iteration i, parties select a pair of previously unused triples $(\langle a\rangle_C, \langle b\rangle_C, \langle c\rangle_C), (\langle a'\rangle_C, \langle b'\rangle_C, \langle c'\rangle_C)$.
   (b) Compute $\langle \alpha\rangle_C = \langle t_i \cdot a + a'\rangle_C$ and $\langle \beta\rangle_C = \langle b + b'\rangle_C$. Open these values using the Output command of each instance of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$. If any $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ instance sends $(\mathsf{Abort}, \mathcal{J})$, parties abort with $\mathcal{J}$ being the set of cheating parties.
   (c) Locally compute $\langle d_i\rangle_C = t_i \cdot \langle c\rangle_C - \langle c'\rangle_C + \alpha \cdot \langle b\rangle_C + \beta\langle a'\rangle_C - \alpha \cdot \beta$.
5. Compute $\langle \sigma\rangle_C = \sum_{i=1}^{l} \chi_i \cdot \langle d_i\rangle_C$.
6. Open σ by calling each $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ with Output. If $\sigma = 0$, accept all $\langle a\rangle_C, \langle b\rangle_C, \langle c\rangle_C$ as good triples and discard all $\langle a'\rangle_C, \langle b'\rangle_C, \langle c'\rangle_C$. If $\sigma \neq 0$, parties abort and go to **Abort 2**.

Abort 1: If there is an abort in $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ in Step 1 of the Triple Generation,
1. Each P_i opens its commitment to Rnd_i to everyone, by sending Open to $\mathcal{F}_{\mathsf{Commit}}$.
2. Each P_i broadcasts view_i from $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$. Then each party runs $\mathsf{Identify}((\mathsf{pk}_i, \mathsf{view}_i, \rho_i\}_{i\in[n]})$ and output as cheater whatever the algorithm outputs.

Abort 2: If there is an abort in the triple sacrifice in Step 4, parties first run the same as in **Abort 1**, and in addition, parties call all instances of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ with Output to open their triple shares. Parties check that the inputs of each P_i to $\mathcal{F}^{\mathsf{IA},i}_{\mathsf{HCom}}$ matched the triple shares P_i obtained as outputs from $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$. If not, parties output $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J}$ is the set of parties with inconsistent inputs to that instance of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$.

Abort 3: If there is an abort in $\mathcal{F}_{\mathsf{Rand}}$ or any instance of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$, all parties abort with $(\mathsf{Abort}, \mathcal{J})$ received from the respective functionality.

Fig. 14: Protocol for preprocessing (Triple Generation)

Acknowledgements

Carsten Baum was supported by research grant VIL53029 from VILLUM FONDEN. Nikolas Melissaris is funded by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation program under grant agreement No 803096 (SPEC). Rahul Rachuri was supported by European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation program under grant agreement No 803096 (SPEC) while at Aarhus University. Peter Scholl is funded by the Independent Research Fund Denmark (DFF) under project number 0165-00107B (C3PO), and the Aarhus University Research Foundation.

References

ADEL22. Thomas Attema, Vincent Dunning, Maarten H. Everts, and Peter Langenkamp. Efficient compiler to covert security with public verifiability for honest majority MPC. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 22*, volume 13269 of *LNCS*, pages 663–683. Springer, Heidelberg, June 2022.

AL07. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, Heidelberg, February 2007.

BCG$^+$19. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

BCG$^+$20. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 387–416. Springer, Heidelberg, August 2020.

BCGI18. Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

BDD20. Carsten Baum, Bernardo David, and Rafael Dowsley. A framework for universally composable publicly verifiable cryptographic protocols. Cryptology ePrint Archive, Report 2020/207, 2020. https://eprint.iacr.org/2020/207.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

BDSW23. Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. Sok: vector OLE-based zero-knowledge protocols. *DCC*, 91(11):3527–3561, 2023.

Bea92. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

BMMM20. Nicholas-Philip Brandt, Sven Maier, Tobias Müller, and Jörn Müller-Quade. Constructing secure multi-party computation with identifiable abort. Cryptology ePrint Archive, Report 2020/153, 2020. https://eprint.iacr.org/2020/153.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

BOS16. Carsten Baum, Emmanuela Orsini, and Peter Scholl. Efficient secure multiparty computation with identifiable abort. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 461–490. Springer, Heidelberg, October / November 2016.

BOSS20. Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 562–592. Springer, Heidelberg, August 2020.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CDKs23. Ran Cohen, Jack Doerner, Yashvanth Kondi, and abhi shelat. Secure multiparty computation with identifiable abort from vindicating release. Cryptology ePrint Archive, Paper 2023/1136, 2023. https://eprint.iacr.org/2023/1136.

CFY17. Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. Catching MPC cheaters: Identification and openability. In Junji Shikata, editor, *ICITS 17*, volume 10681 of *LNCS*, pages 110–134. Springer, Heidelberg, November / December 2017.

CGZ20. Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 828–858. Springer, Heidelberg, May 2020.

CHI$^+$21. Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In *2021 IEEE Symposium on Security and Privacy*, pages 590–607. IEEE Computer Society Press, May 2021.

Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

CRSW22. Michele Ciampi, Divya Ravi, Luisa Siniscalchi, and Hendrik Waldner. Round-optimal multi-party computation with identifiable abort. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 335–364. Springer, Heidelberg, May / June 2022.

DMR$^+$21. Ivan Damgård, Bernardo Magri, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Broadcast-optimal two round MPC with an honest majority. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 155–184, Virtual Event, August 2021. Springer, Heidelberg.

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, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

DRSY23. Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Minimizing setup in broadcast-optimal two round MPC. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 129–158. Springer, Heidelberg, April 2023.

FHKS21. Sebastian Faust, Carmit Hazay, David Kretzler, and Benjamin Schlosser. Generic compiler for publicly verifiable covert multi-party computation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 782–811. Springer, Heidelberg, October 2021.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

HVW22. Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. Protecting distributed primitives against leakage: Equivocal secret sharing and more. In *3rd Conference on Information-Theoretic Cryptography (ITC 2022)*, 2022.

IOS12. Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 21–38. Springer, Heidelberg, March 2012.

IOZ14. Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, August 2014.

KMTZ13. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

LN17. Yehuda Lindell and Ariel Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 259–276. ACM Press, October / November 2017.

PVW08. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

Roy22. Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 657–687. Springer, Heidelberg, August 2022.

RS22. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 719–749. Springer, Heidelberg, August 2022.

SF16. Gabriele Spini and Serge Fehr. Cheater detection in SPDZ multiparty computation. In Anderson C. A. Nascimento and Paulo Barreto, editors, *ICITS 16*, volume 10015 of *LNCS*, pages 151–176. Springer, Heidelberg, August 2016.

SSS22. Peter Scholl, Mark Simkin, and Luisa Siniscalchi. Multiparty computation with covert security and public verifiability. In *3rd Conference on Information-Theoretic Cryptography*, 2022.

SSY22. Mark Simkin, Luisa Siniscalchi, and Sophia Yakoubov. On sufficient oracles for secure computation with identifiable abort. In *Security and Cryptography for Networks: 13th International Conference, SCN 2022, Amalfi (SA), Italy, September 12–14, 2022, Proceedings*, pages 494–515. Springer, 2022.

WYKW21. Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy*, pages 1074–1091. IEEE Computer Society Press, May 2021.

Supplementary Material

A Additional Preliminaries - Functionalities

A.1 Additional Preliminaries

Definition 8 (Signature Scheme). *A signature scheme consists of the following three PPT algorithms,*

$\mathsf{Gen}(1^\lambda)$**:** *On input the security parameter λ, outputs a public key* pk *and signing key* sk.

$\mathsf{Sig}(\mathsf{sk}, \mathsf{msg})$**:** *On input the signing key* sk *and message* $\mathsf{msg} \in \{0,1\}^*$ *outputs a string σ.*

$\mathsf{Ver}(\mathsf{pk}, \sigma, \mathsf{msg})$**:** *On input a public key* pk*, signature σ and message* $\mathsf{msg} \in \{0,1\}^*$ *outputs a bit b.*

We require that $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ is correct, namely that

$$\Pr_{\mathsf{msg} \in \{0,1\}^*}\left[\mathsf{Ver}(\mathsf{pk}, \sigma, \mathsf{msg}) = 1 \;\middle|\; \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, \mathsf{msg})\end{array}\right] = 1$$

Definition 9 (EUF-CMA security). *Given a signature scheme* $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *and security parameter λ, we say that* Sig *is* EUF-CMA-secure *if any PPT algorithm $\mathcal{A}$ has negligible advantage in the EUF-CMA game, defined as*

$$\mathbf{Adv}_{\mathcal{A}}^{\text{EUF-CMA}} = \Pr\left[\begin{array}{c}\mathsf{Ver}(\mathsf{pk}, \sigma^*, \mathsf{msg}^*) = 1 \\ \wedge\, \mathsf{msg}^* \notin Q\end{array} \;\middle|\; \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{msg}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sig}(\mathsf{sk}, \cdot)}(\mathsf{pk})\end{array}\right],$$

where $\mathcal{A}^{\mathsf{Sig}(\mathsf{sk},\cdot)}$ denotes $\mathcal{A}$'s access to a signing oracle with private key sk *and Q denotes the set of messages* msg *that were queried to* $\mathsf{Sig}(\mathsf{sk}, \cdot)$ *by $\mathcal{A}$.*

A.2 Additional Functionalities

Functionality $\mathcal{F}_{\mathsf{Commit}}$

The functionality runs between a set of parties $\mathcal{P}$ and an adversary $\mathcal{A}$.

Commit: On receiving $(\mathsf{Commit}, P_i, x)$ from P_i, store (P_i, x) and send P_i to all parties.

Open: On receiving $(\mathsf{Open}, P_i, P_j)$ from P_i, retrieve x and send (x, P_i) to P_j.

Public Open: On receiving (Open, P_i) from P_i, retrieve x and send (x, P_i) to all parties.

Fig. 15: Functionality for a Commitment

Functionality $\mathcal{F}_{\mathsf{Rand}}$

The functionality runs between a set of parties $\mathcal{P}$ and an adversary $\mathcal{A}$. The adversary can corrupt a subset of the parties, denoted by $\mathcal{I}$.

Upon receiving a description of a domain $\mathbb{F}_{p^r}^m$ from every party in $\mathcal{P}$, uniformly sample $(x_1, \ldots, x_m) \leftarrow \mathbb{F}_{p^r}^m$ and send this to $\mathcal{A}$. If $\mathcal{A}$ responds with $\mathsf{Deliver}$, send $x_1, \ldots, x_m$ to all parties and terminate. Otherwise, if $\mathcal{A}$ sends $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \subset \mathcal{P}_{\mathcal{A}}$, send $(\mathsf{Abort}, \mathcal{J})$ to all parties and terminate.

Fig. 16: Functionality for Coin Tossing with Identifiable Abort

B Efficiency Analysis

Efficiency compared with MPC with abort. To investigate the overhead of obtaining identifiable abort, we compare our protocol with the preprocessing and online phases from Le Mans [RS22], which is secure with abort. There are two ways to run the preprocessing in Le Mans. The first way, called Le Mans 1 in Table 1, is to generate what they call "partial triples", and authenticate the triples during the online phase. Asymptotically, the preprocessing cost in this approach can have a total of $O(n^2 \log|C|)$ communication, where $|C|$ is the circuit size, when using pseudorandom correlation generators for OLE and VOLE correlations. The local computation of PCG approaches is still $O(n^2|C|)$, however. If instead, "non-silent" OLE or VOLE protocols are used, such as from homomorphic encryption or OT, the communication would also be $O(n^2|C|)$. The online cost is $12n$ elements per party (by using the king approach). The second version of Le Mans generate the partial triples in the preprocessing, but also authenticates and checks them, costing an additional $O(n^2|C|)$ field elements, but bringing the online cost down from $12n$ to $4n$ elements per party.

Our preprocessing has the same base cost as Le Mans 1, plus an additional $2(n-1)|C|$ field elements per party, sent via point-to-point channels. When it comes to the online phase, we use the standard BDOZ online phase with authenticated triples and signatures added to the messages, which again, increases the cost by $O(n)$. Overall, our online communication cost per party is dominated by $2(n-1)|C|$ field elements, in an honest execution.

Note that an adversary can always increase the cost of our preprocessing by forcing complaint procedures to be run (by sending invalid messages). This increases our round complexity by a factor of 2, and forces the entire transcript to go via a secure broadcast channel instead of point-to-point channels. The adversary could also cause an abort at any point during the protocol, forcing parties to open their views. However, resolving an abort in our protocol is fairly cheap in terms of computation: once the parties receive the view(s), they only need to locally compute the messages that should have been sent, with no need for expensive ZK proofs.

Efficiency compared to other ID-MPC protocols. We now compare our construction to [BOS16] and [BOSS20].

In the preprocessing phase, [BOS16] requires $O(n^3)$ broadcast messages per multiplication gate because the parties need to perform $O(n^2)$ verifiable decryptions of RLWE ciphertexts. In our protocol, even in the worst case when all messages between parties are forced to be broadcast, we only need $O(n^2)$ broadcasts per multiplication. This asymptotic difference is due to the more complex information-theoretic signatures used in [BOS16], which take more work to set-up than our simple pairwise MACs. In the online phase, both [BOS16] and our protocol have $O(n^2)$ complexity.

Concretely, while it is hard to estimate costs without an implementation, we expect that our protocol will perform much faster than [BOS16]. Our protocol is designed to use Pseudo-random Correlation Generator (PCG) techniques for generating Oblivious Linear Evaluation (OLE) and Vector OLE correlations, and prior works estimate [BCG$^+$20] that concretely, these have orders of magnitude less communication than homomorphic encryption-based (HE) approaches. In [BOS16], the complexity of its preprocessing requirements makes it much harder to employ practical PCG techniques instead of HE, and in particular, we do not see an easy way to avoid their asymptotic $O(n^3)$ overhead.

The protocol of [BOSS20] is incomparable to ours as it is a garbled circuit-based construction that works for Boolean circuits (with a constant-round online phase). In comparison, our construction allows the evaluation of circuits over $\mathbb{F}_p$ for large p with a round complexity that depends on the circuit depth. Both their and our construction use homomorphic commitments during the offline phase: [BOSS20] commits each party to its GC keys, while we let each party commit to its shares. To achieve this, [BOSS20] uses a non-interactive vector commitment while we use a VOLE-based construction. Adapting our commitments to their setting might be interesting future work.

C Online Extractability - Composition and Examples

C.1 Proof of Universal Composability

In this section, for notation, we write $\rho \setminus \{\mathcal{F}\}$ to mean "all parts of the protocol ρ that have neither in- nor output to $\mathcal{F}$. We similarly write $\rho \setminus \pi$, if π is a subprotocol used in ρ.

Proof (Proof of Lemma 1). To prove the statement, we have to construct a PPT algorithm $\mathcal{E}$ that fulfills Definition 4 for $\rho^{\mathcal{F}\to\pi}$. We can assume that $\mathcal{E}^{\rho}$ exists for the protocol ρ and $\mathcal{E}^{\pi}$ for π, and we use these to construct $\mathcal{E}$.

Let $[\rho^{\mathcal{F}\to\pi}]_{\mathcal{E}}$ be the $[\cdot]_{\mathcal{E}}$ transformation applied to the protocol but for a so-far unspecified $\mathcal{E}$. We define $\mathcal{E}$ as follows:

- Initially run an instance of $\mathcal{E}^{\rho}$ and $\mathcal{E}^{\pi}$. Let both manipulate the CRS-like functionalities for the respective protocols $\rho \setminus \pi$ and π.
- Any messages sent between one honest party and a dishonest party in $\rho \setminus \pi$ are forwarded to $\mathcal{E}^{\rho}$. If the message is sent in π then we forward it to $\mathcal{E}^{\pi}$.

– Any hybrid functionality in $\rho \setminus (\{\mathcal{F}\} \cup \pi)$ has its ideal input tape be connected to $\mathcal{E}^\rho$. Any wrapped hybrid functionality in π has its ideal input tape be connected to $\mathcal{E}^\pi$.
– Any output written on the extractor tape of $\mathcal{E}^\pi$ is given to $\mathcal{E}^\rho$ as if it was coming from the ideal input tape of the (non-existent) wrapped $\mathcal{F}$.
– The extractor tape of $\mathcal{E}$ will be the extractor tape of $\mathcal{E}^\rho$.

For criterion 1 of Definition 4, we have to show that $\rho^{\mathcal{F}\to\pi}$ and $[\rho^{\mathcal{F}\to\pi}]_\mathcal{E}$ are indistinguishable to any environment that does not have access to the extractor tape of $\mathcal{E}$.

The change due to wrapping functionalities and copying messages is not visible to $\mathcal{Z}$ as the extractor tapes of $\mathcal{E}^\rho, \mathcal{E}^\pi$ are not accessible to it. The only changes are due to the changes to CRS-like functionalities. By first replacing the CRS-like functionalities as done by $\mathcal{E}^\rho$ and then as done by $\mathcal{E}^\pi$ we get exactly the CRS-like functionality behavior of $\mathcal{E}$, and thereby indistinguishability by a hybrid argument.

For the second criterion, observe that by assumption, the extractor tape of $\mathcal{E}^\pi$ is indistinguishable from the ideal input tape of $\hat{\mathcal{F}}$ because π is online-extractable using $\mathcal{E}^\pi$. But this in particular means that the extractor tape of $\mathcal{E}^\rho$ must have the same distribution, both when using $\hat{\mathcal{F}}$ or the output of $\mathcal{E}^\pi$, as we'd otherwise have constructed a distinguisher for $\mathcal{E}^\pi$ (since the only interaction that $\mathcal{E}^\rho$ has with $\mathcal{E}^\pi$ is via its extractor tape). Therefore, $\mathcal{E}$ must have an extractor tape distribution indistinguishable from $\hat{\mathcal{F}}_\rho$. □

C.2 Online Extractability of VOLE

We show that the VOLE protocol from Wolverine [WYKW21] can be used to realise $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}$ (Fig. 1), and satisfies the online extractability property for a corrupt P_A. Although Wolverine was originally shown to realize a non-programmable VOLE functionality $\mathcal{F}_{\mathsf{VOLE}}$ (where the output of an honest P_A is sampled at random by the functionality), as observed in [RS22], it can easily be extended to be programmable. The analysis in this section is focused on the non-programmable variant, but applies equally to the programmable one.

Single-Point VOLE. The main component of the VOLE construction is a protocol for single-point VOLE, where P_A's vector $\boldsymbol{u}$ has a single non-zero entry. This is modelled by a functionality $\mathcal{F}_{\mathsf{spVOLE}}$, which is then used to build $\mathcal{F}_{\mathsf{VOLE}}$ using the LPN assumption. The latter transformation is completely non-interactive, so clearly online extractable in the $\mathcal{F}_{\mathsf{spVOLE}}$-hybrid model. We now analyze the protocol Π_{spVOLE} that realizes $\mathcal{F}_{\mathsf{spVOLE}}$, which is significantly more complex and also uses several setup functionalities.

Setup Functionalities: $\mathcal{F}_{\mathsf{OT}}$, $\mathcal{F}_{\mathsf{EQ}}$ and $\mathcal{F}_{\mathsf{VOLE}}$. Π_{spVOLE} uses three hybrid functionalities: oblivious transfer ($\mathcal{F}_{\mathsf{OT}}$), equality testing and a smaller $\mathcal{F}_{\mathsf{VOLE}}$ functionality (which is essentially bootstrapped to the larger, more efficient one). Using Lemma 2, we can replace $\mathcal{F}_{\mathsf{OT}}$ by a 2-round OT protocol in the $\mathcal{F}_{\mathsf{CRS}}$

model, and preserve online extractability, since P_A is always the OT receiver. $\mathcal{F}_{\mathsf{EQ}}$ is a weak equality test functionality, which leaks P_A's input if the two parties' inputs differ. It can be easily realized using random oracle based commitment, as described in [WYKW21]. In the resulting protocol, after receiving a commitment from P_B, P_A sends its input in the clear to P_B; this makes the protocol trivially online extractable. Finally, the $\mathcal{F}_{\mathsf{VOLE}}$ functionality used as setup can be realised with $\mathcal{F}_{\mathsf{OT}}$ using OT extension techniques [Roy22]. After analyzing Π_{spVOLE}, we will argue that this setup VOLE protocol also satisfies online extractability.

Online Extractability of Single-Point VOLE. In the following, we refer to the protocol and proof of Π_{spVOLE}, which realizes the functionality $\mathcal{F}_{\mathsf{spVOLE}}$, in Fig. 7 and Theorem 3 of [WYKW21].

Proposition 1. *The protocol* Π_{spVOLE} *for single-point VOLE in [WYKW21, Fig. 7] is online-extractable for a corrupt* P_A.

Proof. To show online extractability, we need to show the existence of an extractor $\mathcal{E}$, which can extract all inputs that are sent to $\mathcal{F}_{\mathsf{spVOLE}}$ in a way that is indistinguishable from the inputs sent by the simulator in the ideal world. Before defining $\mathcal{E}$, we briefly recap the simulator from the proof in [WYKW21, Theorem 3] for a corrupt P_A. Briefly, the view of P_A in this protocol consists of the following:

– Interaction with hybrid functionalities $\mathcal{F}_{\mathsf{VOLE}}$, $\mathcal{F}_{\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{EQ}}$
– A value d sent by P_B, used to fix one of P_B's outputs to the correct value

The simulator, $\mathcal{S}$, emulates the hybrid functionalities and produces a value d, while interacting with $\mathcal{F}_{\mathsf{spVOLE}}$. These interactions happen as follows:

– The $\mathcal{F}_{\mathsf{VOLE}}$ functionality does not send any output to the adversary; the simulator simply receives P_A's inputs.
– For the $\mathcal{F}_{\mathsf{OT}}$ invocations, where P_A plays receiver, the simulator receives the choice bits $\bar{\alpha}_i$ and responds by sending random values $K^i_{\bar{\alpha}_i}$ to the adversary (in the protocol, these are instead pseudorandom, derived using a GGM tree).
– The simulated d is sampled uniformly at random.
– In $\mathcal{F}_{\mathsf{EQ}}$, $\mathcal{S}$ receives a value V_A from the adversary, and compares this with V'_A, which is computed based on previously extracted values known to $\mathcal{S}$. If they differ, it then defines a key guess Δ' that is sent to $\mathcal{F}_{\mathsf{spVOLE}}$. If the guess is incorrect, then $\mathcal{F}_{\mathsf{spVOLE}}$ aborts and $\mathcal{S}$ also aborts. Otherwise, $\mathcal{S}$ sends a response to the corrupt P_A and sends P_A's extracted input to $\mathcal{F}_{\mathsf{spVOLE}}$.

In [WYKW21], it is argued that when using $\mathcal{S}$, the ideal execution is computationally indistinguishable from the real execution.

We now define the extractor $\mathcal{E}$. Recall that $\mathcal{E}$ may observe all the inputs to the hybrid functionalities $\mathcal{F}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{OT}}, \mathcal{F}_{\mathsf{EQ}}$, as well as all communication. Since there are no CRS-like functionalities in use, the only task of $\mathcal{E}$ is to write to its extractor tape the relevant inputs to $\mathcal{F}_{\mathsf{spVOLE}}$. By observing the inputs to the hybrid functionalities, $\mathcal{E}$ has all the same information that $\mathcal{S}$ uses to define

P_A's inputs. In particular, when $\mathcal{F}_{\mathsf{EQ}}$ is called by P_A, $\mathcal{E}$ can use the previously extracted values to check V_A, and if needed, define the key guess Δ' and write this to the tape. Then, if the protocol did not abort, $\mathcal{E}$ writes P_A's extracted input, computed the same way as $\mathcal{S}$, to its tape. Since the values $\mathcal{E}$ writes to its extractor tape can be computed exactly the same way as the values sent by $\mathcal{S}$ to $\mathcal{F}_{\mathsf{spVOLE}}$, it follows that Π_{spVOLE} is online-extractable.

Online Extractability of the VOLE Setup. Since Π_{spVOLE} uses a smaller VOLE functionality as setup, we need to show that this can also be implemented with online extractability. We consider the main VOLE protocol from [Roy22], which is in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model. In our case, the party P_A translates to the VOLE sender in [Roy22]. Most of the challenges in that security proof come from extracting the sender's input when it is corrupt; simulating the view of the adversary is actually trivial, and done exactly as in the protocol. Online extractability is therefore straightforward, as the extractor can simply run the simulator to obtain the extracted inputs.

D Homomorphic Commitment

Lemma 5 (Lemma 3, restated). *Suppose $P_S \in \mathcal{A}$ introduces errors of the form δ^i_j with party P_i, for $j \in [m+1]$, in the* Random *command in Π_{HCom} (Fig. 3). If the consistency check passes, then every pair of parties (P_S, P_i), for $i \in [1, n]$ hold a secret sharing of $l_j \cdot \Delta^i$, for $j \in [m]$. In other words, $\delta^i_j = 0$, for every i and $j \in [m]$, except with probability* $1/|\mathbb{F}|$.

Proof. Let the seed used with party P_1 be the "correct" seed, denoted by s. If a corrupted sender P_S used a different seed s^i in step 2 with a party P_i, this will result in additive errors δ^i_j in the $\langle \ell_j \rangle$ values that are committed to P_i.

In step 4d of the consistency check, $\mathcal{A}$ may send an incorrect MAC value it sends to each P_i; let us denote this by $\widetilde{M}^S_i$, and by M^S_i the actual MAC derived from the authenticated values. Then, the following relation needs to hold in order for the adversary to pass the check with party P_i,

$$\begin{aligned}\widetilde{M}^S_i &= (C + \sum_{j=1}^{m} \chi_j \cdot \delta^i_j + \delta^i_{m+1}) \cdot \Delta^i + K^i_S \\ &= M^S_i + (\sum_{j=1}^{m} \chi_j \cdot \delta^i_j + \delta^i_{m+1}) \cdot \Delta^i\end{aligned}$$

This gives $\widetilde{M}^S_i - M^S_i = (\sum_{j=1}^{m} \chi_j \cdot \delta^i_j + \delta^i_{m+1}) \cdot \Delta^i$. Since the χ_j values are sampled after $\mathcal{A}$ picks the errors δ^i_j, and at least one value of δ^i_j, for $j \in [m]$, is non-zero, and $\mathcal{A}$ does not know Δ^i for the honest parties, then the value on the right of the equation is uniformly random to $\mathcal{A}$. Therefore, the probability of $\mathcal{A}$ passing the check is at most $1/|\mathbb{F}|$.

Theorem 5 (Theorem 1, restated). *Protocol Π_{HCom} UC-securely realises the functionality $\mathcal{F}_{\mathsf{HCom}}$ assuming a broadcast channel in the presence of a malicious adversary that can statically corrupt up to $n-1$ parties, in the $(\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Rand}})$-hybrid model.*

Proof. We have two cases of corruption. One is when the adversary corrupts the sender and some of the receivers and the other is when the adversary corrupts a set of only receivers.

For both cases, we construct a PPT Simulator ($\mathcal{S}$) that runs the adversary ($\mathcal{A}$) as a subroutine, and is given access to $\mathcal{F}_{\mathsf{HCom}}$. It internally emulates the functionalities $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}, \mathcal{F}_{\mathsf{Rand}}$ and we implicitly assume that it passes all communication between $\mathcal{A}$ and the environment ($\mathcal{Z}$).

The parties controlled by the $\mathcal{A}$ are indicated by $\mathcal{P}_{\mathcal{A}}$ and the honest parties by $\mathcal{P}_{\mathcal{H}}$. The sender is denoted by P_S and receiver parties are denoted by $\mathcal{P}_R$. The $\mathcal{S}$ also keeps track of a flag that is set to 0 initially, and set to 1 if the $\mathcal{A}$ cheats in any of the steps.

Adversary corrupts a subset of receivers and P_S (Case 1). The simulation proceeds as follows:

Initialize: $\mathcal{S}$ receives Init and emulates $\mathcal{F}^{\mathsf{prog}}_{\mathsf{VOLE}}$. It receives Δ^i from P_i, where $P_i \in \mathcal{P}_{\mathcal{A}}$ and $P_i \in \mathcal{P}_R$ and stores it.

Input: $\mathcal{S}$ receives a message d from the adversary. $\mathcal{S}$ uses the next unused l_j that was generated in Random and sends $(\mathsf{Input}, \mathsf{id}_x, x)$, where $x = d + l_j$, to $\mathcal{F}_{\mathsf{HCom}}$, and stores (id_x, x).

Linear Operation: These are local operations. $\mathcal{S}$ computes $z = \alpha \cdot x + \beta \cdot y + \gamma$ (also the corresponding MAC), picks a new id id_z, stores $(z, \{M^S_i[\![z]\!]\}_{i \in \mathcal{P}_H})$, and sends $(\mathsf{LinComb}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y, \alpha, \beta, \gamma)$ to $\mathcal{F}_{\mathsf{HCom}}$.

Random:
1. $\mathcal{S}$ receives (Extend, s^i) from P_S for $i \in \mathcal{P}_H$. $\mathcal{S}$ also receives $\boldsymbol{M}^S_i$ from a corrupt P_S, for all $i \in \mathcal{P}_H$, and stores them. If P_S sends inconsistent seeds, $\mathcal{S}$ sets $\mathsf{flag} = 1$. We do not simulate the case when both the sender and receiver are corrupt.
2. If $\mathsf{flag} = 0$, $\mathcal{S}$ sets $\boldsymbol{u}^S = \mathsf{Expand}(s)$ and stores $\langle l_j \rangle = \{u^S_j, w^S_j\}$ as P_S's shares. If $\mathsf{flag} = 1$, $\mathcal{S}$ arbitrarily chooses one of the seeds received and computes $\langle l_j \rangle$ with it.
3. $\mathcal{S}$ samples $\chi_1, \dots, \chi_n \in \mathbb{F}_{p^r}$ and sends them to $\mathcal{A}$ to emulate $\mathcal{F}_{\mathsf{Rand}}$.
4. $\mathcal{S}$ stores $\widetilde{C}^S$, $\left(\widetilde{M}^S_i\right)_{i \in [1, \mathcal{P}_H]}$ it receives from P_S.
5. If $\widetilde{C}^S = C^S$ and $\mathsf{flag} = 0$, send $(\mathsf{Random}, \mathsf{id}_l, l, P_S)$ to $\mathcal{F}_{\mathsf{HCom}}$, along with s, where s is the seed received in step 1.
6. If $\widetilde{C}^S = C^S$ and $\mathsf{flag} = 1$, or $\widetilde{C}^S \neq C^S$, send abort to $\mathcal{F}_{\mathsf{HCom}}$ and abort.

Private Opening: $\mathcal{S}$ receives $(z, \widetilde{M}_i^S[\![z]\!])$, where z is a previously stored value and i is the index of the party to open to. It checks if $\widetilde{M}_i^S[\![z]\!] = M_i^S[\![z]\!]$, since the simulator knows what the MAC on z is supposed to be. If the MACs are not consistent, it aborts.

Batch Opening: $\mathcal{S}$ receives $(\boldsymbol{z}, \widetilde{M}_i^S[\![\boldsymbol{z}]\!])$, where $\boldsymbol{z}$ are a set of stored values, for all $i \in \mathcal{P}_H$. It checks if $\widetilde{M}_i^S[\![\boldsymbol{z}]\!] = M_i^S[\![\boldsymbol{z}]\!]$, since the simulator knows what the MAC on $\boldsymbol{z}$ is supposed to be. If the MACs are not consistent, it aborts.

Output: $\mathcal{S}$ receives $(z, \widetilde{M}_i^S[\![z]\!])$, where z is a previously stored value, for party P_i. It checks if $\widetilde{M}_i^S[\![z]\!] = M_i^S[\![z]\!]$, since the simulator knows what the MAC on z is supposed to be. If the MACs are not consistent, it aborts.

We need to argue that an adversary $\mathcal{A}$ cannot distinguish whether it interacts with Π_{HCom} or the simulator $\mathcal{S}$ equipped with $\mathcal{F}_{\mathsf{HCom}}$. First we'll prove indistinguishability of the simulator when $P_S \in \mathcal{P}_\mathcal{A}$ along with a subset of the receivers.

During **Initialize**, in both worlds the adversary picks its own random values Δ^i for the corrupt receivers. In **Input**, in the real world, $\mathcal{A}$ sends a message d, which is supposed to be $x - l_j$, where l_j is unused random value generated in **Random**. In the ideal world, the adversary sends a message d and the simulator extracts the adversary's input by computing $d - l_j$ since it knows l_j, and sends it to $\mathcal{F}_{\mathsf{HCom}}$. For **Random**, in the ideal world, the $\mathcal{S}$ receives the seeds and adversary's MACs. Then the $\mathcal{S}$ decides to abort if it received inconsistent seeds. If $\mathcal{A}$ cheats by sending inconsistent seeds, $\mathcal{S}$ always aborts where as in the real world the $\mathcal{A}$ can pass the check with probability $1/p^r$, as proven in Lemma 3. Therefore, **Random** is indistinguishable except with negligible probability. In the **Output phase**, the simulator always aborts if the MAC sent by the adversary is incorrect, as it knows what the correct MAC is supposed to be. In the real world, $\mathcal{A}$ can send an inconsistent MAC and still pass the check, if it manages to guess the honest parties' Δ values correctly, which happens with negligible probability. The argument is similar for the **Private Opening** and **Batch Opening** commands.

Adversary corrupts only a subset of receivers (Case 2). The simulation proceeds as follows:

Initialize: $\mathcal{S}$ receives Init from $\mathcal{A}$ and emulates $\mathcal{F}_{\mathsf{VOLE}}^{\mathsf{prog}}$. It receives Δ^i from P_i, where $P_i \in \mathcal{P}_\mathcal{A}$ and stores it. $\mathcal{S}$ sends Init to $\mathcal{F}_{\mathsf{HCom}}$.

Input: $\mathcal{S}$ samples d uniformly, sends it to the adversary, and sends $(\mathsf{Input}, \mathsf{id}_x)$ to $\mathcal{F}_{\mathsf{HCom}}$.

Linear Operation: These are local operations. $\mathcal{S}$ computes $K_i^S[\![z]\!] = \alpha \cdot K_i^S[\![x]\!] + \beta \cdot K_i^S[\![]\!]y$ for all $i \in \mathcal{P}_\mathcal{A}$. It sends $(\mathsf{LinComb}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y, \alpha, \beta)$ to $\mathcal{F}_{\mathsf{HCom}}$.

Random:

1. $\mathcal{S}$ receives Extend from P_i for $i \in \mathcal{P}_\mathcal{A}$ and receives $\boldsymbol{v}^i$ from $\mathcal{A}$.

2. $\mathcal{S}$ samples $\chi_1, \ldots, \chi_n \in \mathbb{F}_{p^r}$ and sends them to $\mathcal{A}$ to emulate $\mathcal{F}_{\mathsf{Rand}}$.
3. $\mathcal{S}$ picks C^S, $\left(M_i^S\right)_{i\in[1,n]}$ such that the check in step 4e passes, and sends them to $\mathcal{P}_{\mathcal{A}}$.

Private Opening: $\mathcal{S}$ sends $(\mathsf{PrivOpen}, \mathsf{id}_z, P_i)$ on behalf of $\mathcal{P}_{\mathcal{A}}$ to $\mathcal{F}_{\mathsf{HCom}}$ to privately open the value to P_i. It receives z from $\mathcal{F}_{\mathsf{HCom}}$, and picks $M_i^S[\![z]\!]$ such that check passes and sends it to P_i.

Output: $\mathcal{S}$ sends $(\mathsf{Output}, \mathsf{id}_z)$ to $\mathcal{F}_{\mathsf{HCom}}$ to receive the output z. It computes $M_i^S[\![z]\!] = K_S^i[\![z]\!] + \Delta^i \cdot z$ and then outputs $(z, M_i^S[\![z]\!])$, for all $i \in \mathcal{P}_H$.

Now we'll provide the indistinguishability argument for the case when $P_S \notin \mathcal{P}_{\mathcal{A}}$. During **Initialize**, in both worlds the adversary picks its own random values Δ^i for the corrupt receivers. In **Input**, in the real world $\mathcal{A}$ receives a random share of the senders input. In the ideal world, $\mathcal{A}$ receives a random message d. For **Random**, in the real world $\mathcal{S}$ sends C^S, $\left(M_i^S\right)$. The adversary will check whether $M_i^S = C^S \cdot \Delta^i + K_S^i[\![C]\!]$ but $\mathcal{S}$ knows Δ^i and $K_S^i[\![C]\!]$ so it can pick C^S, $\left(M_i^S\right)$ accordingly so that the check always passes. In the **Output** (and similarly in **Private Opening** and **Batch Opening**, $\mathcal{S}$ receives the output z from $\mathcal{F}_{\mathsf{HCom}}$ and computes the appropriate MAC which sends to $\mathcal{A}$.

Functionality $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$

Parameters: Finite field $\mathbb{F}_p$. The functionality runs between a sender P_S and a set of receiver parties $\mathcal{P}_R = \{P_1, \ldots, P_n\}$. We assume all parties have agreed upon public identifiers id_x, for each variable x used in the computation. For a vector $\boldsymbol{x} = (x_1, \ldots, x_m)$, we write $\mathsf{id}_{\boldsymbol{x}} = (\mathsf{id}_{x_1}, \ldots, \mathsf{id}_{x_m})$.

Inherits **Input**, **Linear Operation**, **Random**, **Output**, and **Private Opening** from $\mathcal{F}_{\mathsf{HCom}}$.

Abort Behaviour: $\mathcal{A}$ may corrupt any subset $\mathcal{I} \subset P_S \cup \{\mathcal{P}_R\}$. At any point in the protocol, it may send $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \neq \emptyset$ and $\mathcal{J} \subseteq \mathcal{I}$, upon which the functionality will send $(\mathsf{Abort}, \mathcal{J})$ to all parties and aborts.

Fig. 17: Functionality for a Homomorphic Commitment with Identifiable Abort

E Proofs of Theorems 2, 3

Theorem 6 (Theorem 2, restated). *Let Π be a sender-receiver protocol that UC-securely realises a functionality $\mathcal{F}$ with active security and dishonest majority, and supports online extractability when the sender and a subset of receivers are corrupt. Let $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a EUF-CMA secure signature scheme. Then the compiled protocol $\Pi_{\mathsf{Cmp}}^{\mathsf{IA}}$ securely realises $\mathcal{F}$ with active security in the CRS model, and achieves identifiable abort.*

Proof. We have two cases of corruption. In the first case, the sender and a subset of the receivers is corrupt and in the second case the sender is honest and only a subset of receivers is corrupt. For both cases, we construct a PPT Simulator ($\mathcal{S}$) that runs the adversary ($\mathcal{A}$) as a subroutine, and is given access to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$.

Whenever $\mathcal{A}$ communicates with $\mathcal{F}_{\mathsf{CRS}}$, $\mathcal{S}$ calls the extractor $\mathcal{E}$ which picks whichever CRS it wants and $\mathcal{S}$ forwards it to $\mathcal{A}$. The$^{\sim}$(tilde) symbol is used to indicate the potentially inconsistent views received from $\mathcal{A}$.

The adversary $\mathcal{A}$ corrupts a subset of receivers and P_S (Case 1). The simulation proceeds as follows.

1. $\mathcal{S}$ receives the random tapes that $\mathcal{A}$ picked.
2. $\mathcal{S}$ samples $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{H}})$ for the honest receivers, broadcasts $\mathsf{pk}_{\mathcal{H}}$, and receives the corresponding $\mathsf{pk}_{\mathcal{A}}$ from $\mathcal{A}$.
3. $\mathcal{S}$ picks random tapes for the honest parties and runs the Π protocol honestly. During the execution of Π, $\mathcal{S}$ sees all messages between the adversary and the honest parties and forwards them to $\mathcal{E}$ who outputs $\mathcal{A}$'s inputs on its extractor tape. As $\mathcal{E}$ writes on its extractor tape, $\mathcal{S}$ forwards $\mathcal{A}$'s inputs to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ for all the relevant commands. Additionally:
   - $\mathcal{S}$ adds a signature to every message it sends, as in the compiled protocol.
   - When $\mathcal{S}$ receives $(m_{S,j}^{r}, \sigma_{S,j}^{r})$ it verifies the signature and if the check fails, it broadcasts $(\mathsf{Complain}, P_S)$. If P_S fails to broadcast a valid signature during $\mathsf{Complain}$, $\mathcal{S}$ sends (Abort, P_S) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ and aborts.

Abort. Depending on which party aborted in the execution of Π, $\mathcal{S}$ takes care of each case as follows.

1. Abort by an honest receiver P_i:
   (a) $\mathcal{S}$ broadcasts $(\mathsf{abort}, \mathsf{view}_i, \rho_i)$ for the honest receiver P_i who aborted.
   (b) $\mathcal{S}$ sends (Abort, P_S) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ and aborts with output abort_S.
2. Abort by corrupt receiver P_j:
   (a) $\mathcal{S}$ receives $(\mathsf{abort}, \mathsf{view}_i, \rho_i)$ from $\mathcal{A}$ for some P_i.
   (b) $\mathcal{S}$ will run $\mathsf{VerifyAbort}$ $(\mathsf{pk}_i, \mathsf{view}_i, \rho_i)$ to establish if P_i aborted.
   (c) If P_i indeed aborted, $\mathcal{S}$ will send (Abort, P_S) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. Else it will send (Abort, P_i) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$.
3. Abort by corrupt sender P_S:
   (a) $\mathcal{S}$ receives abort from $\mathcal{A}$.
   (b) $\mathcal{S}$ sends $(\mathsf{view}_i, \rho_i)$ to $\mathcal{A}$ for all honest P_i
   (c) $\mathcal{A}$ broadcasts $(\widetilde{\mathsf{view}}_{S,i}, \mathsf{view}_i, \rho_i)$ for some P_i.
   (d) $\mathcal{S}$ runs $\mathsf{IA.Identify}\,(\mathsf{pk}_i, \mathsf{view}_i, \rho_i, \mathsf{pk}_S, \mathsf{view}_{S,i})$. If P_i aborted then send (Abort, P_S) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. In the unusual case where P_i is also corrupt but did not abort send (Abort, P_i) to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$.

We will now prove why $\mathcal{A}$ cannot distinguish if it is interacting with Π^{IA} or the simulator $\mathcal{S}$ equipped with $\mathcal{F}^{\mathsf{IA}}$.

In step 1, in both worlds $\mathcal{A}$ chooses and broadcasts its own random tapes. In step 2 in both worlds $\mathcal{A}$ receives public keys for the honest parties and broadcasts the keys that it picked. For step 3, because the honest parties have no input and their messages only depend on what the corrupt sender sends so $\mathcal{S}$ can perfectly match those messages. In step 5 if $\mathcal{A}$ has received a complaint message, it will broadcast $(m^r_{S,j}, \sigma^r_{S,j})$ in both worlds. In step 6, $\mathcal{A}$ receives an abort message in both worlds. The abort in the real world will happen if the signature check fails. Because of the security of the signature scheme the probability of $\mathcal{A}$ fooling an honest party is negligible.

In the abort phase we have three cases:

Abort by honest receiver. In step 1(a) $\mathcal{S}$ broadcasts $(\mathsf{abort}, \mathsf{view}_i, \rho_i)$. This is identically distributed to the real world, again, because $\mathcal{S}$ ran the real protocol Π using honestly generated random tapes, and because the receivers have no input.

Abort by corrupt receiver. The simulator doesn't send anything in this case so it's straightforward to argue indistinguishability.

Abort by corrupt sender. In step 3(b) $\mathcal{A}$ receives view_i, ρ_i and the argument is the same as in step 1(a). The only way a corrupt sender can frame an honest receiver is by producing some incriminating view but that only happens with negligible probability due to the security of the signature scheme.

Finally, we also consider the outputs of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ seen by the environment. In case of abort, we already argued above that a corrupt party will always be identified. For the non-abort outputs of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$, we rely on the online extractability property, which guarantees that the inputs to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ sent by $\mathcal{S}$ (from the extractor tape) are indistinguishable from those in the original simulation of Π, for $\mathcal{F}_{\mathsf{HCom}}$. Therefore, the non-aborting outputs of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ are distributed the same as those in the original simulation, and indistinguishable from the real world.

The adversary $\mathcal{A}$ corrupts only a subset of receivers (Case 2). The simulation proceeds as follows.

1. $\mathcal{S}$ receives the random tapes that $\mathcal{A}$ picked.
2. $\mathcal{S}$ samples and broadcasts $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{H}})$ for the honest parties and receives the corresponding $\mathsf{pk}_{\mathcal{A}}$ from $\mathcal{A}$.
3. $\mathcal{S}$ runs the simulator $\mathcal{S}_{\Pi}$:
   (a) Whenever $\mathcal{S}_{\Pi}$ sends messages to $\mathcal{F}_{\mathsf{HCom}}$, $\mathcal{S}$ forwards these messages to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ for all the relevant commands.
   (b) When $\mathcal{S}$ receives $(m^r_{i,S}, \sigma^r_{i,S})$ it verifies the signature and if the check fails, it broadcasts $(\mathsf{Complain}, P_i)$. If the check passes $\mathcal{S}$ sends $(m^r_{i,S})$ to $\mathcal{S}_{\Pi}$.
4. $\mathcal{A}$ either broadcasts $(m^r_{i,S}, \sigma^r_{i,S})$ or sends nothing.
5. $\mathcal{S}$ sends (Abort, P_i) to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ and aborts.

Abort. Depending on which party aborted during the execution of $\mathcal{S}_{\Pi}$, $\mathcal{S}$ takes care of each case as follows.

1. Abort by an honest receiver P_i:
   - This will never happen since the sender is honest so we can ignore it.
2. Abort by a corrupt receiver P_j:
   (a) $\mathcal{S}$ receives $(\mathsf{abort}, \mathsf{view}_i, \rho_i)$ from $\mathcal{A}$ for some P_i.

   (b) $\mathcal{S}$ will run $\mathsf{VerifyAbort}\ (\mathsf{pk}_i, \mathsf{view}_i, \rho_i)$ to establish if P_i cheated. This is not really necessary.
   (c) $\mathcal{S}$ will send (Abort, P_i) to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ and abort.
3. Abort by an honest sender P_S:
   (a) $\mathcal{S}$ broadcasts abort.
   (b) $\mathcal{A}$ sends $\widetilde{\mathsf{view}}_j$ and the opening to the random tape ρ_j for all corrupt parties P_j or sends nothing (in which case $\mathcal{S}$ launches a complaint and the views need to be broadcast). If $\mathcal{A}$ doesn't broadcast these for some party P_i, then $\mathcal{S}$ sends abort_i to the functionality.
   (c) $\mathcal{S}$ runs $\mathsf{IA.Identify}\ \left(\mathsf{pk}_j, \mathsf{view}_j, \rho_j, \mathsf{pk}_S, \mathsf{view}_{S,j}\right)$ for all corrupt parties P_j to establish who cheated.
   (d) $\mathcal{S}$ broadcasts $(\mathsf{view}_{S,j}, \mathsf{view}_j, \rho_j)$ for the particular P_j who cheated.
   (e) $\mathcal{S}$ sends (Abort, P_j) to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ and aborts.

We will now prove why $\mathcal{A}$ cannot distinguish if it is interacting with Π^{IA} or the simulator $\mathcal{S}$ equipped with $\mathcal{F}^{\mathsf{IA}}$.

In step 1, in both worlds $\mathcal{A}$ chooses and broadcasts its own random tapes. In step 2 in both worlds $\mathcal{A}$ receives public keys for the honest parties and broadcasts the keys that it picked. In step 3(b), we know that $\mathcal{S}_\Pi$ is a good simulator for Π so the view it simulates is indistinguishable from the real world.

In the abort phase we have three cases. In 2(c) $\mathcal{S}$ aborts with the same probability as in the real world. In step 3(d) $\mathcal{S}$ broadcasts $(\mathsf{view}_{S,j}, \mathsf{view}_j, \rho_j)$. $\mathcal{S}$ can reproduce $\mathsf{view}_{S,j}$ because it consists only of messages received from $\mathcal{A}$. $\square$

Theorem 7 (Theorem 3, restated). *Let Π be a protocol that UC-securely realises a functionality $\mathcal{F}$ with active security and dishonest majority. Let $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a EUF-CMA secure signature scheme. Then the compiled protocol Π_{Cmp} securely realises $\mathcal{F}$ with active security in the CRS model and using broadcast, and achieves the identifiable cheating property.*

Proof. UC-Security: If Π has any calls to hybrid functionalities, they are replaced with the corresponding protocols in the CRS model. This is allowed according to the UC theorem, so it does not break security. The transformed protocol securely realises $\mathcal{F}$ in the CRS model. The simulator $\mathcal{S}$ for a static adversary $\mathcal{A}$ corrupting up to $n-1$ parties works as follows:

1. $\mathcal{S}$ emulates the CRS by picking it according to the simulator for Π and sends it to the adversary.
2. $\mathcal{S}$ records pk from $\mathcal{A}$, picks keys on behalf of the honest parties, and sends the public keys to $\mathcal{A}$.
3. Assume that P_i was supposed to send a message to P_j in a given round of the protocol. There can be three different kinds of communication between parties P_i, P_j:
   (a) P_i is corrupt, but not P_j: $\mathcal{S}$ receives a message and the corresponding signature from $\mathcal{A}$. If the signature does not verify, $\mathcal{S}$ asks $\mathcal{A}$ to broadcast the signature. If $\mathcal{A}$ does not broadcast the signature, or if it does and the signature it broadcasted does not verify, parties abort with abort_i.

(b) P_i is honest, and P_j is corrupt: $\mathcal{S}$ runs the simulator for Π to get the message m that P_i was supposed to send in the current round. $\mathcal{S}$ signs m under the keys it picked for P_i. It sends $m, \mathsf{Sig}(m)$ to the receiver P_j and waits for a response from $\mathcal{A}$. It forwards $\mathcal{A}$'s response to the functionality $\mathcal{F}$.

(c) Both parties are corrupt: This case is trivial to simulate.

4. Whenever $\mathcal{S}$ is supposed to send a message to the functionality, it does whatever the simulator for Π does to compute the message to be sent and forwards it to the functionality.

Indistinguishability is straightforward to argue as the protocol messages of Π which the adversary sees (and the messages to $\mathcal{F}$) are identically distributed as in the regular simulation. In order to distinguish between the ideal world and the real world, one must therefore break UC security of the underlying protocol.

Identifiable cheating: Consider an adversary $\mathcal{A}$ who wins the experiment $\mathsf{Exp}^{\mathsf{ic}}_{\mathcal{A},\Pi}(\lambda)$ with some non-negligible probability. The adversary can win in one of two ways. The first is when the adversary corrupts some party and misbehaves, but $\mathsf{Identify}$ does not output any party as corrupt. The second is when the adversary manages to make $\mathsf{Identify}$ output an honest party, say P_j, as the corrupt party.

Assume it wins due to the first scenario. In this case, no parties are in conflict, meaning that none of the honest parties broadcasted a $\mathsf{complain}$ message and the $\mathsf{Identify}$ algorithm did not identify any party as misbehaving. Let the message that the adversary incorrectly generated be $\tilde{m}^l_{i,j}$ (according to ρ_i), and $\tilde{m}^l_{i,j} \neq m^l_{i,j}$. However, $\mathsf{Identify}$ always identifies a party P_i as cheater if it produces an inconsistent message. Therefore, it is a contradiction that the adversary can misbehave with an inconsistent message and not get caught by $\mathsf{Identify}$.

The other case can only happen if $\mathcal{A}$ has forged a signature of some honest party. Assume that P_i was identified as the cheater in round l. Since round l was when the party P_i was identified, all the messages until round $l-1$ should have been consistent. This means the messages P_i sent in round l will be correct and have signatures on $(P_i||P_j||m^l_{i,j}||l)$. If $\mathcal{A}$ can instead produce a valid signature on $(P_i||P_j||\tilde{m}^l_{i,j}||l)$ with $\tilde{m}^l_{i,j} \neq m^l_{i,j}$ then a successful $\mathcal{A}$ can directly be used to construct an attacker on the EUF-CMA property of the signature scheme with a loss factor n in success probability as the reduction has to guess which simulated honest party to use to embed the challenge pk in.

F Proof of Theorem 4

Proof. We construct a PPT simulator $\mathcal{S}$ that runs the adversary $\mathcal{A}$ as a subroutine, and is given access to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$. It internally emulates the functionalities $\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}^{\mathsf{IA},1}_{\mathsf{HCom}}, \ldots, \mathcal{F}^{\mathsf{IA},n}_{\mathsf{HCom}}, \mathcal{F}_{\mathsf{Commit}}$ and we implicitly assume that it passes all communication between $\mathcal{A}$ and the environment $\mathcal{Z}$.

Parties controlled by the adversary are indicated by $\mathcal{P}_{\mathcal{A}}$ and the honest parties are denoted by $\mathcal{P}_H$.

For simplicity, we specify behavior as if the adversary uses $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}, \mathcal{F}_{\mathsf{Commit}}$ or $\mathcal{F}_{\mathsf{Rand}}$ truthfully as specified in the protocol. If any of the functionalities abort, or a dishonest party does not send a command in a round to a functionality as it was supposed in the protocol, then the simulator will simply collect the sets of corrupted parties that are identified by the hybrid functionalities as $\mathcal{J}$ and immediately send $(\mathsf{Abort}, \mathcal{J})$ to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ as parties would do in **Abort 3** of the protocol.

Initialize: $\mathcal{A}$ chooses its random tape ρ_i for each dishonest party $P_i \in \mathcal{P}_{\mathcal{A}}$ and sends them to $\mathcal{F}_{\mathsf{Commit}}$ which is emulated by $\mathcal{S}$. For each honest party $P_i \in \mathcal{P}_H$, the simulator emulates making a commitment via $\mathcal{F}_{\mathsf{Commit}}$ to $\mathcal{A}$.

Random Input: Let P_i be the party to receive l inputs. If $P_i \in \mathcal{P}_H$ then the simulator just emulates running the protocol and sends $(\mathsf{RandInput}, P_i, l)$ in the name of each dishonest party P_j to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ whenever $\mathcal{A}$ sends $\mathsf{PrivOpen}$ to the respective $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j}$. If P_i is dishonest:

1. For every $P_j \in \mathcal{P}_{\mathcal{A}}$ that sends $(\mathsf{Random}, \mathsf{id}_{\boldsymbol{r}}, l)$ to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j}$ send $(\mathsf{RandInput}, P_i, l)$ in its name to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ and note the committed value as $\boldsymbol{r}_j$. For every honest party $P_j \in \mathcal{P}_H$, simulate committing to these values via each simulated session of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j}$ for a randomly chosen $\boldsymbol{r}_j$.
2. Let P_{j*} be a designated honest party. Once $\mathcal{S}$ obtains $(\mathsf{id}_{\boldsymbol{r}}, \boldsymbol{r})$ from $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ the simulator emulates opening the random value $\boldsymbol{r}_j$ to P_i by $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j}$ of any honest $P_j \neq P_{j*}$ with $(\mathsf{PrivOpen}, \mathsf{id}_{\boldsymbol{r}})$. For P_{j*}, it instead lets $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j*}$ send the value $\delta = \boldsymbol{r} - \sum_{j\in[n], j\neq j*} \boldsymbol{r}_j$ to P_i.

Linear Operation: These are a local operations so they need not be simulated.

Triple Generation:

1. $\mathcal{S}$ runs $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$, which is secure with identifiable cheating, by picking dummy random tapes for each $P_i \in \mathcal{P}_H$. If an abort occurs in $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$, the simulator opens its commitment to the honest parties' random tapes and receives the opening from $\mathcal{A}$ for its tapes. Then, $\mathcal{S}$ sends $\{\mathsf{view}_i\}_{i\in\mathcal{P}_H}$ for each honest party to $\mathcal{A}$ and receives $\{\mathsf{view}_i\}_{i\in\mathcal{P}_{\mathcal{A}}}$ for all the parties $\mathcal{A}$ controls. It runs the $\mathsf{Identify}$ algorithm with input $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, \rho_2, \ldots, \rho_n, \mathsf{view}_1, \ldots, \mathsf{view}_n)$. If $\mathsf{Identify}$ only identifies dishonest parties $\mathcal{J}$, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{J})$ to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ and terminates. If $\mathsf{Identify}$ outputs $\perp$ or also an honest party, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{P}_{\mathcal{A}})$ to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ and terminates.
2. $\mathcal{S}$ emulates each session of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ by receiving $\mathcal{A}$'s shares of the triples it obtains from $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ and storing them, and also consistently inputting its own shares into $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ sessions consistent with the outputs it obtained from $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$.
3. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Rand}}$ by picking a random value t and random combiners $\chi_1, \ldots, \chi_l$, and sending them to $\mathcal{A}$.
4. $\mathcal{S}$ receives commands to each $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ from $\mathcal{A}$ needed to compute $\langle t \cdot a - a' \rangle_C$ and $\langle b - b' \rangle_C$ for each triple. It honestly computes the corresponding shares

for the honest parties. $\mathcal{S}$ then opens its shares via Output to $\mathcal{A}$ to open $t \cdot a - a'$ and $b - b'$ and waits for $\mathcal{A}$ to open its shares.

5. $\mathcal{S}$ honestly computes $\langle \sigma \rangle_C$ for the honest parties and sends the shares to $\mathcal{A}$ via the opening of $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. It receives $\mathcal{A}$'s shares of $\langle \sigma \rangle_C$ and checks if $\sigma = 0$.
   (a) If $d = 0$ and all multiplication triples are consistent, then $\mathcal{S}$ sends $(\mathsf{TripGen}, l)$ in the name of each dishonest party to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$.
   (b) If $d = 0$ but there are inconsistent multiplication triples committed to, then $\mathcal{S}$ simply sends $(\mathsf{Abort}, \mathcal{P}_{\mathcal{A}})$ to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ and aborts.
   (c) If $d \neq 0$, $\mathcal{S}$ broadcasts an Abort, opens the honest parties' random tape commitments as well as all triple shares it has committed to via $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. Then, it waits to receive $\mathcal{A}$'s openings of its random tape commitments and triple share commitments via $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$. $\mathcal{S}$ then broadcasts $\{\mathsf{view}_i\}_{i \in \mathcal{P}_H}$ and waits for the $\mathcal{A}$'s views $\{\mathsf{view}_i\}_{i \in \mathcal{P}_{\mathcal{A}}}$. Using all the views and the random tapes, $\mathcal{S}$ can now run the $\mathsf{Identify}$ algorithm as in the protocol. As above, if $\mathsf{Identify}$ identifies dishonest parties $\mathcal{J}$, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{J})$ to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ and terminates. If $\mathsf{Identify}$ outputs $\perp$ or also an honest party, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{P}_{\mathcal{A}})$ to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ and terminates. If no cheaters are detected in $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$, $\mathcal{S}$ checks if the $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ output shares that $\mathcal{A}$ opened via $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA}}$ are consistent with the shares $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ generated. If $\mathcal{S}$ then identifies parties where these are different, then it sends $(\mathsf{Abort}, \mathcal{J})$ to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$, where $\mathcal{J}$ is the set of parties with inconsistent triple commitments. If no such party could be identified, then $\mathcal{S}$ sends $(\mathsf{Abort}, \mathcal{P}_{\mathcal{A}})$ to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$.

Output: On receiving $(\mathsf{Output}, \mathsf{id}_x)$ from $P_i \in \mathcal{P}_{\mathcal{A}}$ to $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},i}$, the simulator forwards the message to $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$, from which it gets the value x. $\mathcal{S}$ knows $\mathcal{A}$'s shares of the output as they are committed in $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},i}$, so it picks shares for one honest party P_{j*} (as in the input phase) such that they add up to x and makes $\mathcal{F}_{\mathsf{HCom}}^{\mathsf{IA},j*}$ output the correct share to $\mathcal{P}_{\mathcal{A}}$.

Indistinguishability: We argue that no computationally bounded environment can distinguish between the real world and ideal world executions, except with probability $1/p + \mathsf{negl}(\lambda)$.

For all operations except **Triple Generation**, it is trivial to see that simulation and real protocol are perfectly indistinguishable in its abort behavior and in terms of consistency as $\mathcal{S}$ does the same as **Abort 3** and each hybrid functionality only identifies dishonest parties as cheaters. We can therefore focus on **Triple Generation**.

First we look at the part running $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ or where it may abort. In the ideal world, if we have an abort during $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ then the simulator will always abort with dishonest parties only. In the real world, the algorithm $\mathsf{Identify}$ may identify no cheater at all or even an honest party. But since $\Pi_{\mathsf{Trip}}^{\mathsf{IC}}$ has the identifiable cheating property, by Definition 7 this can only occur with probability $\mathsf{negl}(\lambda)$.

In the ideal world, the simulation of the triple check always aborts if a committed triple is incorrect (i.e. even if $d = 0$). In the real protocol, this may not be the case. By a standard argument (see e.g. [LN17, Lemma 3.5]), the probability of this event happening to allow distinguishability is $1/(p-1)$.

Next, consider the case where $d \neq 0$ and the triple check turns to cheater identification. There, if $\mathsf{Identify}$ identifies an honest party then this is distinguishable between real and ideal world as $\mathcal{S}$ always aborts in the ideal world, but this can happen with only with probability $\mathsf{negl}(\lambda)$. The other abort that can happen is if no cheater is identified from running $\mathsf{Identify}$ on $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ and from the opening of all $\mathcal{F}^{\mathsf{IA}}_{\mathsf{HCom}}$ sessions, i.e. each party consistently committed to the outputs of $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$, but these did not form consistent multiplication triples. In this case, $\mathcal{S}$ always aborts with $\mathcal{P}_{\mathcal{A}}$ in the ideal world. In the real world, since $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ UC-securely implements $\mathcal{F}_{\mathsf{Triple}}$, and each party acted honestly during $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ (except with probability $\mathsf{negl}(\lambda)$ as otherwise $\mathsf{Identify}$ would have identified the party deviating from the protocol as having cheated by the Identifiable Cheating of $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$), the outputs of $\Pi^{\mathsf{IC}}_{\mathsf{Trip}}$ in case of no abort must be valid multiplication triples except with probability $\mathsf{negl}(\lambda)$ by assumption. □

G Online Phase Protocol

In Fig. 19, we present the protocol $\Pi^{\mathsf{IA}}_{\mathsf{MPC}}$ for the online phase with identifiable abort. It allows to share inputs, perform linear operations on sharings, multiply shared values and to reconstruct (output) a shared value. The protocol uses $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ to realize the sharing of secrets and the individual MPC protocol steps as follows:

1. To share an input x, party P_i uses a random input r created by $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ via $\mathsf{RandInput}$. P_i broadcasts the value $\delta = x - r$, whereupon all parties add δ to the commitment to r held in $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$, thus generating a commitment to x.
2. Similarly, to output a shared value x, parties simply send Output to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$.
3. To perform a linear operation on shared values, the parties simply call $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ to perform the linear operation on the committed values.
4. Finally, to perform a multiplication between sharings of x, y, the parties use a multiplicative random triple a, b, c generated also by $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$. To multiply, they follow the following steps:
   (a) The parties compute sharings of $\alpha = x - a, \beta = y - b$ using linear operations with $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$.
   (b) The parties use $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ to open α, β to all parties.
   (c) The parties compute $\gamma = \alpha \cdot \beta$ and set $z = a \cdot \beta + b \cdot \alpha + c - \gamma$ using $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$, which is a linear operation.

The only non-trivial step in the online phase is the multiplication, which follows the so-called Beaver circuit randomization paradigm [Bea92]. Since both $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ and the broadcast channel have identifiable abort, the overall protocol has identifiable abort as well.

We now sketch a proof of security for the protocol outlined above:

Theorem 8. *The protocol $\Pi^{\mathsf{IA}}_{\mathsf{MPC}}$ UC-securely implements the functionality $\mathcal{F}^{\mathsf{IA}}_{\mathsf{MPC}}$ in the $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$-hybrid model and using a broadcast channel with perfect security against any active attacker corrupting at most $n-1$ of the n parties.*

Functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{IA}}$

Parameters: Finite field $\mathbb{F}_p$, parties $P_1, \ldots, P_n$. The adversary is allowed to corrupt a subset of parties, denoted by $\mathcal{I}$. The computation happens over $\mathbb{F}_p$.

Input: On receiving $(\mathsf{Input}, P_i, \mathsf{id}_x, x)$ from P_i and $(\mathsf{Input}, P_i, \mathsf{id}_x)$ from all other parties:

1. Store (id_x, x).
2. Send $(\mathsf{Stored}, \mathsf{id}_x)$ to every party.

Linear Operation: On receiving $(\mathsf{LinComb}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y, \alpha, \beta, \gamma)$ from every P_i where $\mathsf{id}_x, \mathsf{id}_y$ are assigned, id_z is unassigned and where $\alpha, \beta, \gamma \in \mathbb{F}_p$, compute $z = \alpha \cdot x + \beta \cdot y + \gamma$ and store (id_z, z).

Multiplication: On receiving $(\mathsf{Mult}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y)$ from all parties where $\mathsf{id}_x, \mathsf{id}_y$ have been assigned and id_z is unassigned:

1. Store $(\mathsf{id}_z, x \cdot y)$.
2. Send $(\mathsf{Multiplied}, \mathsf{id}_z)$ to all parties.

Output: Upon receiving $(\mathsf{Output}, \mathsf{id}_x)$ from all parties and where id_x is assigned:

1. Send x to $\mathcal{A}$.
2. If $\mathcal{A}$ sends $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \subseteq \mathcal{I}, \mathcal{J} \neq \emptyset$, then send $(\mathsf{Abort}, \mathcal{J})$ to all the parties and terminate.
3. If $\mathcal{A}$ sends $\mathsf{Deliver}$ then output x to all parties.

Corrupt Party Behaviour: Whenever the adversary is supposed to send a value, it can choose to not send a value at all, triggering an abort. The functionality receives $(\mathsf{Abort}, \mathcal{J})$, where $\mathcal{J} \subseteq \mathcal{I}$ from $\mathcal{A}$ and $\mathcal{J} \neq \emptyset$, sends it to all the parties and terminates.

Fig. 18: Functionality for MPC

Proof. We only sketch the simulator here, as it directly follows from the observations above. For it, observe that the simulator simulates the hybrid functionality $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$. If $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ aborts at any point or the adversary does not send any messages, then $\mathcal{S}$ identifies the cheaters the same way as in the real protocol.

During **Input** for a dishonest P_i we extract the adversarial input x by observing the difference between the privately opened value r from $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ and the broadcast δ, which is then input in the name of P_i to $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{IA}}$. For an honest P_i, we simply broadcast a uniformly random δ.

No messages are sent during **Linear Operation**. For **Multiplication**, the simulator sfollows the protocol but lets $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ open uniformly random values for α, β.

During **Output** the simulator first obtains the output x from $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{IA}}$. It then makes $\mathcal{F}_{\mathsf{Prep}}^{\mathsf{IA}}$ output x as well, and any abort is forwarded to $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{IA}}$.

Protocol $\Pi^{\mathsf{IA}}_{\mathsf{MPC}}$

Parameters: Finite field $\mathbb{F}_p$, parties $P_1, \ldots, P_n$. The adversary is allowed to corrupt a subset of parties, denoted by $\mathcal{I}$. The computation happens over $\mathbb{F}_p$. We let l be a fixed constant for amortization.

The parties have access to a broadcast channel and an instance of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$. If at any point $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ outputs $(\mathsf{Abort}, \mathcal{J})$ or a set of parties $\mathcal{J}$ did not send their expected messages via the broadcast channel, then all parties abort with set $\mathcal{J}$.

Input: Party P_i wants to input $x \in \mathbb{F}_p$ for an unused id_x:

1. All parties check if there is an unused output of $\mathsf{RandInput}$ for party P_i. If so, then let id_r be the identifier of that output. If not, then all parties send $(\mathsf{RandInput}, P_i, l)$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ such that they get l identifiers. Then, let id_r be the first such fresh identifier.
2. P_i finds the value $r \in \mathbb{F}_p$ associated to id_r.
3. P_i broadcasts $\delta = x - r$.
4. Upon receiving δ, all parties send $(\mathsf{LinComb}, \mathsf{id}_x, \mathsf{id}_r, \perp, 1, 0, \delta)$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$.
5. All parties consider the random value id_r as used.

Linear Operation: The parties directly forward the respective message to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$.

Multiplication: To multiply two values represented by $\mathsf{id}_x, \mathsf{id}_y$ obtaining a new value represented by id_z that so far is unassigned, the parties do the following:

1. The parties check if there is an unused triple generated by $\mathsf{TripleGen}$ of $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$. If yes, then let $\mathsf{id}_a, \mathsf{id}_b, \mathsf{id}_c$ be the identifiers of that triple. If not, then all parties send $(\mathsf{TripleGen}, l)$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$, wait for the outputs and use the first fresh triple $\mathsf{id}_a, \mathsf{id}_b, \mathsf{id}_c$.
2. Each party sends $(\mathsf{LinComp}, \mathsf{id}_\alpha, \mathsf{id}_x, \mathsf{id}_a, 1, -1, 0)$ and $(\mathsf{LinComp}, \mathsf{id}_\beta, \mathsf{id}_y, \mathsf{id}_b, 1, -1, 0)$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$.
3. Each party sends $(\mathsf{Output}, \mathsf{id}_\alpha)$ and $(\mathsf{Output}, \mathsf{id}_\beta)$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$, publicly reconstructing α, β.
4. Each party locally computes $\gamma = \alpha \cdot \beta$ and sends $(\mathsf{LinComb}, \mathsf{id}_f, \mathsf{id}_a, \mathsf{id}_b, \beta, \alpha, -\gamma)$ as well as $(\mathsf{LinComb}, \mathsf{id}_z, \mathsf{id}_f, \mathsf{id}_c, 1, 1, 0)$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$.
5. All parties consider the triple $\mathsf{id}_a, \mathsf{id}_b, \mathsf{id}_c$ as used.

Output: To reveal a value id_x that is defined, the parties send $(\mathsf{Output}, \mathsf{id}_x)$ to $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ and obtain the value x.

Fig. 19: Online phase for MPC with Identifiable Abort

To see that the simulation is perfect, observe that the value δ in the real protocol is uniformly random as the uniformly random r has been used to compute it, while it is also uniformly random in the ideal world. The same argument can be made for α and β, which are also perfectly indistinguishable. For **Output** we also have perfect indistinguishability as $\mathcal{F}^{\mathsf{IA}}_{\mathsf{Prep}}$ outputs the same correct value in the simulation and the real protocol. As no honest party is ever detected as cheater in $\Pi^{\mathsf{IA}}_{\mathsf{MPC}}$ by definition and $\mathcal{F}^{\mathsf{IA}}_{\mathsf{MPC}}$ outputs the exact same aborting parties as $\Pi^{\mathsf{IA}}_{\mathsf{MPC}}$, the statement follows. $\square$