Separating Broadcast from Cheater Identification

Yashvanth Kondi
Silence Laboratories (Deel)
yash@ykondi.net

Divya Ravi
University of Amsterdam
d.ravi@uva.nl

October 26, 2024

Abstract

Secure Multiparty Computation (MPC) protocols that achieve Identifiable Abort (IA) guarantee honest parties that in the event that they are denied output, they will be notified of the identity of at least one corrupt party responsible for the abort. Cheater identification provides recourse in the event of a protocol failure, and in some cases can even be desired over Guaranteed Output Delivery. However, protocols in the literature typically make use of broadcast as a necessary tool in identifying cheaters. In many deployments, the broadcast channel itself may be the most expensive component.

In this work, we investigate if it is inherent that MPC with IA should bear the full complexity of broadcast. As the implication of broadcast from IA has been established in previous work, we relax our target to circumvent this connection: we allow honest parties to differ in which cheaters they identify, nonetheless retaining the ability to prove claims of cheating to an auditor.

We show that in the honest majority setting, our notion of Provable Identifiable Selective Abort (PISA) can be achieved without a traditional broadcast channel. Indeed, broadcast in this setting—which we term Broadcast with Selective Identifiable Abort (BC-IA)—is achievable in only two point-to-point rounds with a simple echoing technique. On the negative side, we also prove that BC-IA is impossible to achieve in the dishonest majority setting.

As a general result, we show that any MPC protocol that achieves IA with r broadcasts, can be compiled to one that achieves PISA with $2(r+1)$ point to point rounds. As a practical application of this methodology, we design, implement, and benchmark a six-round honest majority threshold ECDSA protocol that achieves PISA, and can be deployed in any environment with point to point communication alone.

Contents

1 Introduction

Secure Multiparty Computation (MPC) protocols generally enable groups of parties to compute functions on their joint private inputs, under precisely defined constraints of what information may be leaked. Feasibility results have been established for decades, and the practicality of MPC has made major strides in recent years. While guarantees related to privacy of inputs are mostly standard across MPC protocols, their failure modes are not. The weakest such failure mode is "security with abort", in which an adversary is permitted to see the output of the computation itself, but deprive honest parties of it. This relatively weak guarantee allows for highly efficient protocols. Protocols that achieve "fairness" give the adversary a choice: obtaining the output for itself will imply that it is delivered to honest parties, but honest parties can be deprived of output if the adversary forgoes it as well. The strongest guarantee is that of "guaranteed output delivery", in which the adversary is unable to stop honest parties from receiving the output of the computation.

Foundational results [Cle86] have established that achieving fairness or guaranteed output is not just a matter of efficiency or complexity over security with abort, but in fact feasibility in different settings. In particular—assuming standard simulation based security—fairness is infeasible in the dishonest majority setting for general MPC, whereas security with abort is not just feasible in this setting, but quite practical. However, there does exist an alternative approach to address the failure mode of security with abort: identification of the party responsible for crashing the protocol, also known as *Identifiable Abort* (IA). Cheater identification may be used in combination with external punitive measures to serve as a strong disincentive to deviating from the protocol, for instance via financial penalties or reputational damage. In certain contexts, IA serves as a stepping stone towards guaranteed output, as cheating parties may be removed and the computation restarted.

Identifiable Abort is frequently studied as something of a compromise notion when guaranteed output is not possible, however we argue that in certain contexts cheater identification might even be preferable to guaranteed output. Note that guaranteed output protocols may detect and recover from cheats, but this does not imply the unambiguous identification of cheating parties. In many contexts, it is of far greater importance to identify the corrupt party rather than to deliver output to everyone. For instance,

- Threshold signatures are used to remove single points of failure in key management via MPC, and signing nodes may even all be operated by the same entity—who will likely be more interested in diagnosing which of its nodes has been compromised, than in receiving the signature.
- Institutions are unlikely to engage with each other at all (MPC or otherwise) if there is not a certain degree of mutual trust. In the event of a protocol failure, an acceptable explanation is that some party's node has been compromised by an external actor trying to attack the system. It is therefore imperative to identify which one, so that the situation may

be remedied immediately, and the system may resume operation after a reset.

- In many settings, MPC nodes operators are not strictly honest or malicious per se, but rational. Even if they do not collude, they act in their individual best interests. For example, if an operator knows that (1) output will be delivered regardless of its actions, and (2) it will not be held accountable for its actions, then it may be incentivized to conserve its resources and simply not participate in the protocol.

- Security with abort protocols already offer privacy in the event of cheats; the utility of guaranteed output is to prevent denial of service. The type of adversary to consider is therefore one that wishes to disrupt the protocol. In this regard, a disruptive party may still induce a relatively cumbersome worst case execution path in the guaranteed output case, with no consequences due to the lack of identification.

For these reasons, we argue that studying cheater identification is worthwhile, even in settings where guaranteed output delivery may be feasible. While IA for general MPC has been known to be feasible for as long MPC itself [GMW87b], the details for its practical realization are still the subject of ongoing research [BOSS20, CDKs24, BMRS24]. The classic GMW-style approach has each party prove that honest behaviour in zero-knowledge, whereas modern protocols use alternative techniques to avoid non-blackbox use of cryptography.

The Overhead of IA. While recent works have made progress towards efficient cheater identification, certain barriers remain to achieving efficiency and simplicity within the same realm as security with abort. There are essentially two ways that a misbehaving party can induce a protocol to crash:

- One way is by sending malformed protocol messages—this can be handled by employing zero-knowledge proofs [GMW87a], carefully opening randomness [CDKs24, BMRS24], or a hybrid of both.

- Another way is by simply staying silent when the protocol prescribes that it send a message. This is notoriously difficult to mitigate; if Bob complains that he did not receive a message from Alice that he was expecting on a private channel, whom do we blame? Given that private channels are inherently unverifiable, the standard solution is to route all communication through *verifiable broadcast*.

As IA is known to imply broadcast [CL17] most of these works express their efficiencies in terms of broadcast invocations. The usage of broadcast is not a distinguishing feature of any one work, and so the cost of realizing broadcast in the IA context is not typically explored in depth. Looking ahead, in this work we will identify certain relaxations to the broadcast primitive that are meaningful in the IA setting. However when left as an implementation detail

as in prior work, instantiating the broadcast channel generically has the potential to be the most expensive or cumbersome element of a real world deployment.

Instantiating Broadcast. We briefly review common options available to instantiate broadcast available today:

- **Blockchains.** One approach suggested in previous works is to use an external resource like a blockchain for broadcast [GMPS21, GKM+22, ZYP23]. This introduces assumptions external to the system along with accompanying liabilities, latencies, and financial costs. This may be acceptable in certain scenarios, but blockchains do not offer a general solution.
- **Broadcast protocols.** Assuming a Public Key Infrastructure (PKI), the feasibility of broadcast for any corruption threshold has been established for decades [DS83]. However, it is known that *deterministic* protocols for broadcast inherently suffer a round complexity that is linear in the corruption threshold [FL82, DS83]. While randomization can achieve $O(1)$ rounds in expectation [KK06], reasoning about composition is subtle [CCGZ19], and even the best known protocols are rather involved and have large constants [ADD+19].
- **Trusted coordinator.** A common system architecture is to have a 'coordinator' party that issues instructions, routes messages, and aggregates responses. In the security with abort model, such a coordinator is essentially untrusted, as it can at best compromise liveness of the protocol. However, using such a coordinator in the IA context to implement a broadcast channel (as has been suggested in previous work [Lin22]) substantially strengthens the trust assumption placed on the coordinator. For instance, a malicious coordinator may drop messages from an honest party, leading other parties to falsely identify it as corrupt.
- **Context-specific heuristics.** It may be possible in certain contexts to heuristically achieve the specific requirements that broadcast satisfies. For instance, certain deployments make use of third party messaging services to implement communcation channels, and these services may be queried to obtain logs of which parties in the MPC were non-responsive. We do not further discuss such heuristics in this work, as we focus on provable security with trust assumptions restricted to the parties involved in the MPC.

Relative to the security with abort setting where broadcast (with selective abort) can be achieved by a round of simply echoing messages [GL05], in the IA setting broadcast appears to induce substantial complexity. Towards achieving practical cheater identification with simple and complete specifications (i.e. instantiable over point-to-point channels only), we ask:

> *Can broadcast protocols in the Identifiable Abort setting be simpler and more efficient than using generic tools?*

Let us first clarify our target. We wish to enable a *provability* dimension to cheater identification, meaning that any aborting party must be able to convince an external auditor of the identity of a corrupt party—a form of "public identifiability" [BDO14, SV15, KZZ16]. However, we allow an important relaxation: honest parties need not be in agreement about which corrupt party they identify, or indeed whether there even was an abort. The lack of consensus amongst honest parties circumvents the direct connection between IA and broadcast, while provability to external auditors retains the strong disincentive for parties to cheat. We capture these requirements in a notion we call Provable Identifiable Selective Abort (PISA).

With our target notion for general MPC in place, we define a custom notion of broadcast as relevant to this setting—Broadcast with Selective Identifiable Abort (BC-IA). Roughly, BC-IA allows for honest receivers to terminate with either a publicly verifiable certificate of the dealer's misbehaviour (in the event of an abort), or a dealer-signed message that is consistent with any other non-aborting honest party's output.

1.1 Our Results

We give a multifaceted answer to our question about instantiating broadcast in the context of cheater identification, as outlined below:

1. **An Impossibility.** (Theorem 4.1) The $t < n$ setting does not permit a realization of BC-IA over point-to-point (p2p) channels, regardless of PKI.

2. **A Positive Result.** (Theorem 4.4) The $t < n/2$ setting permits a simple two-round BC-IA protocol over p2p channels, assuming a PKI and a synchronous network.

3. **A Generic Compiler.** (Theorem 5.3) Given a protocol that securely computes a functionality with Identifiable Abort in the presence of $t < n/2$ parties in r broadcast rounds, we obtain a protocol that computes the same functionality with PISA with tolerance to $t < n/2$ corruptions in $2(r+1)$ p2p rounds.

4. **A Concrete Application.** (Section 8) We present a targeted application of this methodology to derive a 6-round honest majority threshold ECDSA signing protocol that achieves PISA. While existing threshold ECDSA protocols make use of ideal broadcast to identify cheaters, ours runs over p2p channels alone, and as such can be deployed without additional dependencies. We implement and report benchmarks of our protocol to establish that it is efficient enough to use in practice. Along the way, we develop tools such as distributed key generation and verifiable secret sharing in this setting, which may be of independent interest when constructing other discrete logarithm based protocols that provide cheater identification.

We begin with a brief technical overview, following which we proceed to give our results in the same order as enumerated above.

1.2 Technical Overview

Broadcast. We first formalize BC-IA, which is an arguably minimal building block for Identifiable Abort when we wish for cheats to be certifiable. Note that certified cheats as well as the signed message are valid outputs for an honest party, meaning that consensus is not achieved (however, parties that output the signed message will be in agreement).

Roughly, we consider two grades of certified cheats: a certificate of non-responsiveness ω, or a certificate of cheating Ω. The former proves that the dealer did not send a message when it was expected to, whereas the latter proves conclusively that the dealer sent a malformed message in an attempt to cheat. We consciously choose to separate these two grades of disruptive behaviour, and leave it to higher level protocol logic to decide on appropriate penalties. For instance, non-responsiveness could be punished less harshly (up to a point) as network faults may be out of the parties' control.

Unfortunately, we prove that BC-IA is impossible to achieve over point-to-point channels alone with resilience to $n/2 \leq t < n$ corruptions. Intuitively, this is because a dishonest majority of corrupt parties could always collude to certify an honest sender as non-responsive. Recall that a party may output a message only if it has been *signed* by the dealer. This means that a non-responsive dealer must necessarily induce parties to output a certificate ω. The mechanism by which this certificate is produced must require the participation of a majority parties, so that it may not be abused by a corrupt minority. If a majority of parties is corrupt, then this mechanism can be invoked even against an honest dealer. Note that a PKI does not help circumvent this impossibility.

In the honest majority (i.e. $t < n/2$) setting, we construct a protocol in the PKI model that meets the above BC-IA notion—each party either terminates with a signed output by the dealer, or a certificate of the dealer's cheating. In particular, we show that a simple modification of the classic Goldwasser Lindell [GL05] echo broadcast yields our desired BC-IA primitive in just two point-to-point rounds per broadcast. Roughly, our modification is to agument echo broadcast so that parties sign and echo $\perp$ in the event that the dealer does not send them a signed message. A certificate ω can therefore be assembled to implicate the dealer, with $t + 1$ signed $\perp$ messages.

Provable Identifiability in MPC. In the spirit of BC-IA, we extend the notion of provable identifiability (without agreement) to the more general case of MPC. We introduce a new notion, namely Provable Identifiable Selective Abort (PISA), which guarantees that at the end of the MPC protocol, each honest party either obtains the output of the joint computation or a certifiable proof of cheating against a corrupt participant. Notably, unlike the case of BC-IA where certifiable cheats are limited to implicating only the dealer; PISA allows for any corrupt participant to be identified as the cheater. Further, as the name suggests, Provable Identifiable Selective Abort allows for the (provable) identification of different cheaters by different honest parties.

As BC-IA—a special case of PISA MPC—requires an honest majority, it

follows that PISA MPC does as well. We show that any honest majority IA protocol that uses a broadcast channel can be compiled into a PISA MPC protocol that runs over point-to-point channels only. This compiler demonstrates that consensus on the identified cheater can be traded off to avoid dependence on broadcast, while retaining provable cheater identification. At a high level, our compiler replaces each broadcast invocation of the underlying protocol with an invocation of BC-IA. While BC-IA does not guarantee liveness and safety, attempting to subvert these properties will provably identify a corrupt party—this suffices for the failure mode of PISA MPC. If no BC-IA instance aborts, the broadcast channel of the underlying IA protocol is live and safe, yielding a perfect emulation. In case the underlying IA protocol identifies a cheater, a certificate consisting of $t+1$ signatures on the identity of the corrupt party can be assembled by virtue of the honest majority. Overall, an MPC protocol achieving IA by making use of r broadcast invocations is compiled to a PISA protocol that makes use of $r+1$ BC-IA invocations, which in turn can be realized in $2(r+1)$ point-to-point rounds.

Lastly, we note that since the cheats are certifiable, broadcasting the certificates (via "true" broadcast) would achieve standard IA with consensus on the identified cheater. However, this requires the use of standard broadcast, which may be difficult to instantiate for the reasons outlined previously.

Application: Threshold ECDSA Signing. As a demonstration of the practical impact of our results, we apply our techniques to construct a threshold ECDSA signing protocol that achieves Identifiable Abort without broadcast. We choose this application as ECDSA signing is non-trivial due to its non-linear arithmetic, and yet poses a tractable problem as its circuit representation is relatively small. Moreover, threshold ECDSA signing is a widely deployed application of MPC, and the application is therefore of real-world relevance.

Existing protocols for threshold ECDSA signing with IA are designed for the dishonest majority setting [CGG+20, CCL+23]. In principle, we could apply our compiler to these protocols anyway—with $\lceil (n+1)/2 \rceil$ "signing parties" and $n/2$ "helper parties" to achieve $t < n/2$. However this would be overkill, as dishonest majority protocols are substantially more complex than honest majority ones. We therefore construct a new *honest majority* threshold ECDSA signing protocol that achieves IA in three broadcast rounds. We instantiate the broadcast channel with our BC-IA primitive, to obtain a protocol that runs in six point-to-point rounds.

Our new protocol leverages observations from the recent work Doerner et al. [DKLs24] to reduce ECDSA signing to Distributed Key Generation (DKG) and two secure multiplications. As building blocks, we design DKG and Verifiable Secret Sharing (VSS) protocols for the PISA setting. The VSS protocol is a simple adaptation of Pedersen's [Ped91], whereas the DKG protocol augments VSS to sample an *unbiased* key in two broadcast rounds. These protocols are designed to sample random values in secret shared form, while exposing each party's share in publicly committed form—either directly in the exponent, or masked as in Pedersen commitments. Consequently, parties are able

to prove honest behaviour (eg. when broadcasting their signature shares, which contain products of secrets) via lightweight Schnorr-like zero-knowledge proofs over these Pedersen commitments.

Finally, we report benchmarks of an implementation of our protocol with a standard 256-bit curve; its computation cost in the worst case (eg. when a cheater must be identified) ranges from 15ms for three parties, to 480ms for ten parties. Of course, real-world performance is likely to be determined by other factors like network constraints and adversarial slowdowns. We envision our protocol to be used in scenarios where Denial of Service attacks are a constant threat, and protocol aborts are rampant—were this not the case, one could use security with abort protocols. Given this, the performance of different protocols and approaches will depend to a large extent on deployment conditions.

2 Related Work

Broadcast and its variants. Without a setup (such as a PKI i.e. public-key infrastructure), broadcast is achievable if any only if $t < n/3$ [PSL80, LSP82] However, if we assume a PKI setup, (cryptographically-secure) broadcast is achievable even against a dishonest majority of corruptions, namely, $t < n$ [DS83]. With respect to round complexity of deterministic broadcast protocols, it is known that $t + 1$ rounds are necessary and sufficient to achieve security against t corruptions, even assuming access to a public-key infrastructure [FL82, DS83].

Typically, broadcast protocols in the literature by default consider the notion of guaranteed output delivery i.e. the protocol terminates with all parties outputting the agreed upon value. Broadcast with the weaker security guarantee of security with selective abort, was explored in the [GL05]. This weak version of broadcast is achievable in two rounds against $t < n$ corruptions, where honest parties may not agree on whether the protocol aborted or not (but it is guaranteed that honest parties do not output inconsistent values). [FGMv02, FGH$^+$02] showed that the notion of 'detectable broadcast', where corrupt parties can make the protocol abort but honest parties agree on whether the protocol aborted or not, is achievable against $t < n$ corruptions.

To the best of our knowledge, our work is the first to explicitly consider the notion of broadcast with identifiability. Our notion of Broadcast with Selective Identifiable Abort (BC-IA) allows every honest party who aborted to learn and verifiably prove to an external party that the sender cheated. It is incomparable to detectable broadcast in terms of guarantees – While detectable broadcast ensures that honest parties are in agreement about whether an abort occurred (which BC-IA does not provide), it does not enable identification or certification of the cheater. As mentioned earlier, in terms of resilience, detectable broadcast is achievable against $t < n$ corruptions; while BC-IA is feasible only against $t < n/2$ corruptions. However, in terms of round complexity BC-IA fares significantly better as it is achievable in two rounds which is in stark contrast to existing detectable broadcast protocols that comprise of $O(t)$ rounds.

On Identifiability in MPC. The most common security notion of identifiability that has been explored in MPC is identifiable abort (IA), which was introduced in [AL10] and is stronger than unanimous abort (UA). IA security guarantees that parties are in agreement about the identity of (at least one) cheater, in case the protocol results in an abort. It is known that broadcast is necessary for IA [CL17], therefore all IA protocols must necessarily rely on broadcast. Therefore, when broadcast is not available, the best one can hope for is some kind of selective identifiability, where honest parties may not agree on the identity of a cheater. This was captured in the notion of selective identifiable abort (SIA), introduced in [DRSY23] which guarantees that each honest party either obtains the output or learns the identity of a cheater, but it may so happen that different honest parties identify different cheaters. We enhance this notion to SIA to be certifiable i.e. the honest party who aborts not only learns the identity of the cheater, but also obtains a 'proof-of-cheating' certificate that is verifiable by any external party. This is reminiscent to the notion of IA with public verifiability / auditability in [AO12, BDO14, BOS16, CFY17] where either the correctness of the output is attested or a cheater can be found by the external party. The crucial difference is that such protocols achieving IA with public verifiability must necessarily rely on broadcast, which is not available in our setting. Therefore, we allow the party who aborted to obtain a certifiable proof of cheating *privately* which would suffice to convince any external party of the identity of the cheater, whenever needed.

Threshold ECDSA Signing. Given the practical relevance of the problem, there is a plethora of approaches to constructing threshold ECDSA signing. We refer the reader to Doerner et al. [DKLs24] for an overview. We will only touch upon the works that share some common features with ours. Damgård et al. [DJN+20] constructed an honest majority threshold ECDSA signing protocol in the security with abort model. While highly efficient, it is susceptible to denial of service attacks as a cheater can crash the protocol anonymously. Groth and Shoup [GS22] constructed an ECDSA signing protocol that can operate in an asynchronous network setting, however with a low corruption tolerance of $t < n/3$ that is inherent to the setting. Canetti et al. [CGG+20] and Castagnos et al. [CCL+23] constructed dishonest majority ECDSA signing protocols that can trace cheaters. However, they rely on verifiable broadcast channels, which we are determined to avoid. Nonetheless, the cryptographic machinery that they use (as necessary for the dishonest majority setting) will likely make their protocol much slower than ours in most scenarios, as discussed in Section 9.

3 Preliminaries

Corruption Model. We denote the set of parties as $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$. We consider a malicious adversary $\mathcal{A}$ who can statically corrupt up to a threshold $t < n/2$ among the n parties.

Security and Network Model. We follow the real /ideal world simulation paradigm and analyze the security of our protocols within the Universal Composability security framework of [Can01]. Our target ideal functionality for ECDSA signing is one that simply computes and outputs and ECDSA signature upon being requested by enough parties, along with appropriate provisions for identifying cheaters. This is in contrast to *idealized* threshold signatures in the UC framework [CGG+20] which allow the adversary to specify an arbitrary format for the strings. We assume that parties are connected via pairwise authenticated channels. We consider a synchronous network communication model where the computation proceeds in rounds and the messages sent in a round are assumed be delivered to the intended receiver(s) before the next round begins. Note that especially when we are concerned with certifying parties that do not send messages when they are supposed to, cheater identification is not meaningful in the asynchronous setting. This is because—as previously observed by Shoup [Sho23]—corrupt parties that do not send messages are indistinguishable from honest parties whose messages are adversarially delayed.

Building Blocks and Setup. Our PISA MPC protocol resulting from the compiler is in the $\mathcal{F}_{\mathsf{BC-IA}}$-hybrid model, where $\mathcal{F}_{\mathsf{BC-IA}}$ denotes the ideal functionality corresponding to Broadcast with Selective Identifiable Abort (BC-IA) and uses an IA MPC protocol realizing $\mathcal{F}_{\mathsf{IA}}$ (Appendix C.1) as a building block.

Our ECDSA protocol is in the $(\mathcal{F}_{\mathsf{BC-IA}}, \mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{DKG}})$-hybrid model, where $\mathcal{F}_{\mathsf{BC-IA}}, \mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{DKG}}$ denote ideal functionalities corresponding to BC-IA, Verifiable Secret Sharing (VSS) [1] and a Distributed Key Generation (DKG) protocol respectively. We provide formal descriptions of these ideal functionalities to capture our identifiability notion. BC-IA assumes the presence of a public-key infrastructure (PKI) setup. The VSS protocol is designed in the $\mathcal{F}_{\mathsf{BC}}$ model and uses an openable encryption scheme $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Open}, \mathsf{Vrfy})$ (Appendix A). Our DKG protocol is designed in the $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{VSS}})$-hybrid model. Both, our DKG protocol and ECDSA signing protocol make use of Simulation-Extractable Non-Interactive Zero-Knowledge Proofs (NIZKs) in the random oracle model, which are required to be straight-line simulatable, but witness extraction can be rewinding—this is because we only rely on extracting witnesses for the security argument, and not for simulating the protocol. This means that we can use Fiat-Shamir compiled NIZKs rather than more expensive Fischlin transform [Fis05] as typically required for UC security.

4 Broadcast With IA

4.1 Property-Based Definition

We begin with a property-based definition of Broadcast with Selective Identifiable Abort (BC-IA), although the definition we actually use in our protocols is

[1] We abuse standard notation here, our VSS protocol is a verifiable secret sharing of a uniformly chosen random secret as opposed to a secret chosen by a designated dealer.

formulated in the Real-Ideal paradigm and given in Section 4.3. The property-based definition is primarily meant to allow for easier comparisons with different flavours of broadcast in the literature, which are typically formulated with property-based definitions as well. We give our dishonest majority impossibility result relative to this property-based definition, illustrating that this barrier is not due to UC peculiarities.

(Property-Based Definition) Broadcast with Selective Identifiable Abort (BC-IA). Let $\{\mathcal{P}_i\}_{i\in[n]}$ denote a set of n parties, among which a designated party $\mathcal{P}_d$ (referred to as the dealer) holds a message msg as input. Consider a tuple of two algorithms $(\mathtt{Deal}, \mathtt{Audit})$.

$\mathtt{Deal}(\mathsf{msg}) \to \left(\mathbf{out}_i \in \{(m,\sigma_d), \omega, \Omega\}\right)_{i\in[n]}$ is an algorithm that takes the message msg as input and outputs a vector of n values, each of which can correspond to either a signed message (m, σ_d) by the dealer or a certificate of non-responsiveness ω, or a certificate of cheating Ω.

$\mathtt{Audit}(\mathbf{out}_i) \to \{\mathtt{accept}, \mathtt{reject}\}$ is an algorithm that takes $\mathbf{out}_i \in \{(m,\sigma_d), \omega, \Omega\}$ as input and outputs either `accept` or `reject`.

We require the following three properties of a BC-IA protocol with respect to an adversary $\mathcal{A}$ corrupting up to a threshold t of parties.

Validity: Informally, this property is the same as validity of standard broadcast, i.e. it guarantees that if $\mathcal{P}_d$ is honest, then all honest parties output $\mathcal{P}_d$'s input value. More formally, if at most t parties are corrupted and $\mathcal{P}_d$ is honest, then $\mathbf{out}_h = (\mathsf{msg}, \sigma_d)$ for each honest party $\mathcal{P}_h (h \in \mathcal{H})$. Here, σ_d denotes a valid signature on msg.

Consistency: Informally, this property guarantees that if there exists a pair of honest parties who output a signed message, they must in fact output the same signed message. More formally, if at most t parties are corrupted and a pair of honest parties $\mathcal{P}_i$ and $\mathcal{P}_j$ output $\mathbf{out}_i = (m^{(i)}, \sigma_d^{(i)})$ and $\mathbf{out}_i = (m^{(j)}, \sigma_d^{(j)})$ respectively, then $\mathbf{out}_i = \mathbf{out}_j$.

Defamation-Freeness: Informally, this property captures that the adversary $\mathcal{A}$ cannot produce a certificate that implicates an honest dealer. More formally, when the dealer $\mathcal{P}_d$ is honest and $\mathcal{A}$ controls at most t parties, then the probability that $\mathcal{A}$ outputs $\mathbf{out}_{\mathcal{A}} \in \{\omega, \Omega\}$ such that $\mathtt{Audit}(\mathbf{out}_{\mathcal{A}}) = \mathtt{accept}$ is negligible.

Unforgeability: Informally, this property guarantees that an adversary $\mathcal{A}$ will be unable to induce `Audit` to accept (msg, σ_d) if msg was not actually dealt by the honest dealer $\mathcal{P}_d$. More formally, even after $\mathcal{A}$ is allowed to instruct $\mathcal{P}_d$ to deal arbitrarily many messages $\{\mathbf{msg}\}$, the probability that $\mathcal{A}$ outputs $(\mathsf{msg}^*, \sigma_d^*)$ such that $\mathtt{Audit}(\mathsf{msg}^*, \sigma_d^*) = 1$ and $\mathsf{msg}^* \notin \{\mathbf{msg}\}$ is negligible.

Lastly, we point that the fact that the protocol terminates with the output of `Deal` implicitly captures *identifiability* i.e. each party either outputs a signed message by the dealer or a certificate (either of non-responsiveness or of cheating). We also note that the outputs are *transferrable* as they can be verified by any external party by means of verifying the dealer's signature or using the `Audit` algorithm. Importantly, auditing a transferred output can only verify a cheat, or that a dealer attempted to broadcast a value—auditing a single party's output can not verify consensus amongst honest parties.

The auditability properties we formulate above (unforgeability and defamation-freeness) mark a departure from standard broadcast definitions. Indeed, we show below that this notion is *impossible* to achieve when the adversary controls more than half the parties, even with a PKI. This is in contrast to standard broadcast definitions, which permit honest parties to terminate with unsigned canonical outputs when they detect malicious behaviour on the dealer's part—such notions can be satisfied by classic protocols [DS83]. This is not to say that our definition is strictly stronger, as ours allows honest parties to terminate without consensus.

4.2 Impossibility of BC-IA with $t \geq n/2$

We give the details of our argument below.

Theorem 4.1. *Consider a model where parties have access to a public-key infrastructure and only point-to-point channels. Then, Broadcast with Selective Identifiable Abort among n parties is impossible to achieve against $n/2 \leq t < n$ corruptions.*

Proof. Assume towards contradiction that protocol Π achieves BC-IA with tolerance to $n/2 \leq t < n$ corruptions. Let A and B denote two disjoint sets of participants, such that $|B| \geq n/2$ and the dealer $\mathcal{P}_d \in B$. Consider the following two scenarios:

Scenario 1: The adversary corrupts all parties in B including $\mathcal{P}_d$, and keeps them silent throughout the execution of Π. More specifically, $\mathcal{A}$ instructs each party in B to simply not send any messages. Note that this scenario is possible only because $\mathcal{A}$ is allowed to corrupt a majority of parties.

Scenario 2: The adversary corrupts all parties in A, and instructs them to follow the steps of Π honestly, *except* that they ignore any message sent by any party in B.

Upon inspection, it becomes clear that both scenarios are equivalent to a network adversary severing all connections across A and B. Moreover, the joint view of all parties in A is identical in both scenarios: it is simply derived by executing the protocol Π honestly without any messages from any party in B (messages to parties in B may as well be dropped as they induce effectively no response).

Let us now examine the output $\mathbf{out}_i$ of $\mathcal{P}_i \in A$ in either scenario. In order to deliver guarantees compliant with BC-IA, there are two possible options in the overwhelming majority of outcomes:

- Case 1: a certified cheat, $\mathbf{out}_i \in \{\omega, \Omega\}$.
- Case 2: a certified output, $\mathbf{out}_i = (\mathsf{msg}^*, \sigma_d^*)$.

We argue that both cases contradict the requirements of BC-IA itself. Recall that in Scenario 2, the dealer $\mathcal{P}_d$ is honest. Therefore, Case 1 directly contradicts defamation-freeness in that scenario; an honest dealer must not be implicated by a certified cheat. Simultaneously, Case 2 contradicts unforgeability: consider an $\mathcal{A}$ who does not request *any* dealt instances from the unforgeability challenger, and therefore any certified output $(\mathsf{msg}^*, \sigma_d^*)$ it is able to produce constitutes a breach of unforgeability. The adversary is able to obtain such a $(\mathsf{msg}^*, \sigma_d^*)$ simply by executing Π entirely amongst the corrupt set A—this follows Scenario 1 wherein all of B is effectively offline, and the premise of Case 2 ensures that $(\mathsf{msg}^*, \sigma_d^*)$ is produced amongst A regardless.

We have thus arrived at a contradiction, completing the proof of Theorem 4.1. □

We note that the above proof extends to the following weaker variants (thus, making the impossibility result stronger):

1. The proof holds even against a fail-stop adversary, as the only deviation from the protocol in the above proof is to drop messages.
2. Consider a weaker notion of BC-IA, where the cheat can be certified only jointly by honest parties (as opposed to individually by any honest party who aborts). The above proof extends easily to this notion as well, one simply considers the joint outputs of the entire set of parties in A (which the adversary can use).

Lastly, we highlight another interesting aspect of this impossibility result: We observe that the proof holds irrespective of the number of rounds that comprise Π. Notably, this does not contradict existing $O(t)$ round protocols that achieve standard broadcast with resilience to $t < n$ corruptions in the PKI model—our argument makes crucial use of the BC-IA requirement that all outputs must be certified (which is not required of standard broadcast).

4.3 Ideal Functionality

Especially given that we wish to invoke multiple instances in parallel, we formulate an ideal functionality that will allow us to achieve our desired flavour of BC-IA in the UC framework. We first give the functionality realized by our broadcast primitive, and then proceed to give the protocol itself. The `Audit` component of the property-based notion is replaced by a 'Transfer' interface here, intuitively to enable the transfer of protocol context to external parties.

Functionality 4.2. $\mathcal{F}_{\mathsf{BC-IA}}(n, t, d)$**: Broadcast-IA**

This functionality is parameterized by the party count n, the threshold t such that $n \geq 2t + 1$, and the index of the dealer d. This functionality is accessed by parties $\{\mathcal{P}_i\}_{i\in[n]}$ at most t of whom may be corrupt. Let the set of honest parties be indexed by $\mathcal{H}$.

Any party, indexed by a global identifier pid, may register at any point with this functionality. Each such party $\mathsf{P}_{\mathsf{pid}}$ is provided with an output $\mathsf{out}_{\mathsf{pid}}$ which can be transferred to any other party. The set of such parties is initially only $\{\mathcal{P}_i\}_{i\in[n]}$, and can be expanded as necessary subsequently.

Deal: On receiving $(\texttt{deal}, \mathsf{sid}, \mathsf{msg})$ from $\mathcal{P}_d$ such that $\mathsf{sid} =: \mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}'$ and sid is fresh, send $(\texttt{deal-req}, \mathsf{sid}, d)$ to $\mathcal{S}$. On receiving $(\texttt{ready}, \mathsf{sid})$ from all parties,

1. If $\mathcal{P}_d$ is not corrupt, then set $\mathbf{out} = \{\mathbf{out}_i = \mathsf{msg}\}_{i\in[n]}$.
2. Otherwise, receive $\mathbf{out} = \{\mathbf{out}_i \in \{\mathsf{msg}, \mathsf{cheat}, \perp\}\}_{i\in[n]}$ from $\mathcal{S}$.

Send $(\texttt{dealt}, \mathsf{sid}, \mathbf{out}_i)$ to each $\mathcal{P}_i$. Additionally, for each pid corresponding to every party $h \in \mathcal{H}$, set and store $\mathsf{out}_{\mathsf{pid}} = (\mathsf{sid}, \mathbf{out}_i)$.

Transfer: Upon receiving $(\texttt{transfer}, \mathsf{sid}, \mathsf{pid}, \mathsf{pid}^*)$ from both $\mathsf{P}_{\mathsf{pid}}$ and $\mathsf{P}_{\mathsf{pid}^*}$

1. If $\mathcal{P}_d$ is honest, send $\mathsf{out}_{\mathsf{pid}}$ to $\mathsf{P}_{\mathsf{pid}^*}$.
2. Otherwise, receive $\mathsf{out}_{\mathsf{pid}} \in \{\mathsf{msg}, \mathsf{cheat}, \perp\}$ from $\mathcal{S}$ and send to $\mathsf{P}_{\mathsf{pid}^*}$.

4.4 The Protocol

The protocol assumes a PKI and employs a simple echo broadcast technique along the lines of Goldwasser and Lindell [GL05], with a provision to output a certificate of non-responsiveness. Note that in the following description (and each subsequent one in the paper) the "send to all parties" instruction includes sending to oneself. The protocol is given informally below:

- **Round 1**: The dealer $\mathcal{P}_d$ signs its message m, and sends m along with its signature σ_d to all parties.
- **Round 2**: Each $\mathcal{P}_i$ that received (m, σ_d) from the dealer in Round 1 forwards it to all other parties, and any $\mathcal{P}_i$ that received no (valid) message in Round 1 sends $\perp$ to all parties instead. Either way, the forwarded message is accompanied by a signature σ_i.
- **Output**, per $\mathcal{P}_i$'s view: In case two conflicting $(m, \sigma_d), (m', \sigma'_d)$ values were received, this immediately implicates $\mathcal{P}_d$ as a cheater with a certificate Ω. In case $t + 1$ signed $\perp$ values were received, this collection of

signatures implicates $\mathcal{P}_d$ as non-responsive with a certificate ω. In the absence of either case of certifiable failure of $\mathcal{P}_d$, any (m, σ_d) received in the previous round can be safely output.

We can analyze all possible outcomes as follows:

1. If $\mathcal{P}_d$ is honest, $(n-t) \geq t+1$ parties will follow the protocol, and all honest parties output (m, σ_d). A live network ensures that no ω can be produced against $\mathcal{P}_d$, and an unforgeable signature scheme rules out any party deriving a valid Ω.

2. If the dealer sends conflicting $(m, \sigma_d), (m', \sigma'_d)$ to *any* pair of honest parties, *all* honest parties will output Ω.

3. If the dealer withholds (m, σ_d) from *all* honest parties in Round 1, then all honest parties will output ω. Note that it is always possible for the adversary to "upgrade" the cheat from ω to Ω in the view of some subset of honest parties by sending them conflicting $(m, \sigma_d), (m', \sigma'_d)$ values via the echo phase.

4. Conditioned on at least one honest party having received some (m, σ_d) from $\mathcal{P}_d$ and no honest party having received a conflicting (m', σ'_d) in Round 1, the adversary is free to induce each honest party to output any one of (m, σ_d), Ω, or ω in Round 2. Note that all honest parties that output (m, σ_d) are in agreement about m, i.e. the one received by the honest party/parties in Round 1.

Care must be taken in the interpretation of the output of any individual party, especially in the context of an external retrospective audit. While Ω or ω undeniably certify $\mathcal{P}_d$'s deviation from the protocol, a signed output (m, σ_d) does not certify m as the output of the protocol. This is because if $\mathcal{P}_d$ is corrupt, any colluding $\mathcal{P}_i$ could try and pass off an arbitrary (m', σ'_d) as a "certified" output to an auditor. However, a collection of $t+1$ parties willing to attest to (m, σ_d) having been broadcast can certify the statement—we will use this idea later on when holding parties accountable for broadcasting messages that are malformed *in context*. In particular, while certain malformed messages can immediately serve to implicate their sender (such as a non-verifying NIZK), others require establishing protocol context to explain why the message is malformed, such as a jointly sampled nonce. We give the full protocol below.

Protocol 4.3. $\pi_{\mathsf{BC}}(n, t, d)$**: Broadcast With IA**

This protocol is parameterized by the party count n, the threshold t such that $n \geq 2t+1$, and the index of the dealer d. This protocol is run by parties $\{\mathcal{P}_i\}_{i \in [n]}$. The protocol requires a PKI, where the key pair of $\mathcal{P}_i$ is given by $(\mathsf{sk}_i^{\mathsf{PKI}}, \mathsf{pk}_i^{\mathsf{PKI}})$.

In the event that any $\mathcal{P}_i$ aborts (with an accompanying certificate), it is taken by assumption that $\mathcal{P}_i$ sends the certificate to all before halting the protocol.

Deal:

1. On receiving $(\mathtt{deal}, \mathsf{sid}, \mathsf{msg})$ from the environment such that sid is fresh, $\mathcal{P}_d$ does the following:

   a. Set $\mathsf{dealmsg} = (\mathtt{deal}, \mathsf{sid}, \mathsf{msg})$ and sign it:

   $$\sigma^{\mathsf{PKI-d}} \leftarrow \mathsf{Sign}^{\mathsf{PKI}}(\mathsf{dealmsg})$$

   b. Send $(\mathsf{dealmsg}, \sigma^{\mathsf{PKI-d}})$ to each $\{\mathcal{P}_i\}_{i\in[n]}$

   This completes the first round.

2. Each party $\mathcal{P}_i$ does the following, ignoring any unsigned/malformed messages from the dealer:

   a. If some $(\mathsf{dealmsg} = (\mathtt{deal}, \mathsf{sid}, \mathsf{msg}), \sigma^{\mathsf{PKI-d}})$ was received from the dealer, set $\mathsf{echomsg}_i = (\mathtt{echo}, \mathsf{dealmsg}, \sigma^{\mathsf{PKI-d}})$. Else, set $\mathsf{echomsg}_i = (\mathtt{echo}, \mathsf{sid}, \perp)$.

   b. Set $\sigma_i^{\mathsf{PKI}} \leftarrow \mathsf{Sign}^{\mathsf{PKI}}_{\mathsf{sk}_i^{\mathsf{PKI}}}(\mathsf{echomsg}_i)$, and send $(\mathsf{echomsg}_i, \sigma_i^{\mathsf{PKI}})$ to all parties.

   This completes the second round.

3. After collecting all messages $(\mathsf{echomsg}_j, \sigma_j^{\mathsf{PKI}})$ sent in the above round (ignoring malformed messages), each $\mathcal{P}_i$ does the following:

   - If $\exists j, j' \in [n]$ such that $\mathsf{echomsg}_j \neq \mathsf{echomsg}_{j'}$ and both contain valid (but conflicting) $\mathsf{dealmsg}$ values, then set

   $$\mathbf{out}_i = \Big(\mathsf{sid}, \mathsf{cheat}, \Omega_i = (\mathsf{msg}_j, \mathsf{msg}_{j'}, \sigma_j^{\mathsf{PKI-d}}, \sigma_{j'}^{\mathsf{PKI-d}})\Big)$$

   where $(\mathsf{msg}_j, \sigma_j^{\mathsf{PKI-d}})$ and $(\mathsf{msg}_{j'}, \sigma_{j'}^{\mathsf{PKI-d}})$ are parsed from $\mathsf{echomsg}_j$ and $\mathsf{echomsg}_{j'}$ respectively.

   - Otherwise if $\exists \mathcal{J} \subseteq [n]$ such that $|\mathcal{J}| > t$ and $\mathsf{echomsg}_j = (\mathtt{echo}, \mathsf{sid}, \perp)$ for each $j \in \mathcal{J}$, set

   $$\mathbf{out}_i = \big(\mathsf{sid}, \perp, \omega_i = \{\mathsf{sid}, \mathsf{echomsg}_j, \sigma_j^{\mathsf{PKI}}\}_{j\in\mathcal{J}}\big)$$

   - Otherwise, set $\mathbf{out}_i = (\mathsf{sid}, \mathtt{dealt}, \mathsf{msg}_j, \sigma_j^{\mathsf{PKI}})$ for any j where σ_j^{PKI} verifies.

   Output $\mathbf{out}_i$ as computed above.

Transfer: On receiving $(\texttt{transfer}, \mathsf{sid}, i, j)$ from the environment such that sid is fresh, $\mathcal{P}_i$ sends $\mathbf{out}_i$ to $\mathcal{P}_j$. $\mathcal{P}_j$ accepts $\mathbf{out}_i$ if any of the following hold:

- $\mathbf{out}_i = (\mathsf{sid}, \mathsf{cheat}, \Omega_i)$ with Ω_i containing two different messages, both with valid signatures from $\mathcal{P}_d$.
- $\mathbf{out}_i = (\perp, \omega_i)$ with ω_i containing valid signatures by a set of at least $(t+1)$ distinct parties on the message $(\texttt{echo}, \mathsf{sid}, \perp)$.
- $\mathbf{out}_i = (\mathsf{sid}, \texttt{dealt}, \mathsf{msg}, \sigma^{\mathsf{PKI}})$, which contains a valid signed message of the dealer.

Theorem 4.4. *Assuming a synchronous network and a PKI, protocol $\pi_{\mathsf{BC}}(n,t,d)$ UC-realizes $\mathcal{F}_{\mathsf{BC-IA}}(n,t,d)$ in the presence of a malicious adversary that statically corrupts up to $t < n/2$ parties.*

5 PISA MPC

We are now ready to define and realize the notion of Provable Identifiable Selective Abort (PISA) for the case of general MPC. Informally, a protocol achieving PISA guarantees that upon termination, each honest party either obtains the output of the joint computation, or a certifiable proof of cheating against a corrupt party. Importantly, PISA allows honest parties to possibly identify different cheaters; however each honest party can provably implicate the cheater it identified to any external party. Recall that the necessity of an honest majority for general secure function evaluation with PISA over point to point channels follows from our impossibility in Section 4.2, as BC-IA is a special case.

5.1 Ideal Functionality for PISA

We begin with a formal description of the ideal functionality for this new notion. The adversary first learns the output of the computation, and then determines which honest parties receive this output. Each honest party that is denied the output is instead given the identity of a corrupt party (determined by the adversary) along with the type of cheating, and the ability to transfer this information to any third party. Along the lines of $\mathcal{F}_{\mathsf{BC-IA}}$, we provide this latter feature through a "transfer" interface.

Functionality 5.1. $\mathcal{F}_{\mathsf{PISA}}(n,t,f)$ **: Provable Identifiable Selective Abort**

This functionality is parameterized by the party count n and the threshold t such that $n \geq 2t+1$. It is accessed by parties $\{\mathcal{P}_i\}_{i\in[n]}$ at most t of whom may be corrupted by an ideal adversary $\mathcal{S}$. Let the set of honest parties be indexed by $\mathcal{H}$. It is also parameterized by a function $f : \mathbb{X}_1 \times \mathbb{X}_2 \cdots \times \mathbb{X}_n \to \mathbb{Y}$.

Computation: On receiving $(\texttt{compute}, \mathsf{sid}, x_i)$, where $x_i \in \mathbb{X}_i$ from every

party $\mathcal{P}_i$ for $i \in [n]$,

1. Compute $y := f(\{x_i\}_{i\in[n]})$.
2. Send $(\texttt{function-output}, \mathsf{sid}, y)$ to $\mathcal{S}$.
3. Receive $\textbf{output} = \{\textbf{output}_i \in \{y, (\mathsf{cheat}, c_i), (\perp, c_i)\}\}_{i\in[n]}$ from $\mathcal{S}$ in response, where each $c_i (i \in [n])$ corresponds to an index of a corrupt party.
4. Send $(\texttt{output}, \mathsf{sid}, \textbf{output}_i)$ to each $\mathcal{P}_i$. Additionally, for each pid corresponding to every honest party $h \in \mathcal{H}$, set and store $\mathsf{out}_{\mathsf{pid}} = (\mathsf{sid}, \textbf{output}_i)$.

Transfer: Upon receiving $(\texttt{transfer}, \mathsf{sid}, \mathsf{pid}, \mathsf{pid}^*)$ from both $\mathsf{P}_{\mathsf{pid}}$ and $\mathsf{P}_{\mathsf{pid}^*}$

1. If the party $\mathsf{P}_{\mathsf{pid}}$ is honest and has previously been given $\mathsf{out}_{\mathsf{pid}}$, set $\mathsf{out}_{\mathsf{pid}^*} = \mathsf{out}_{\mathsf{pid}}$ and send it to $\mathsf{P}_{\mathsf{pid}^*}$.
2. Otherwise, receive $\mathsf{out}_{\mathsf{pid}} \in \{y, (\mathsf{cheat}, c), (\perp, c)\}$ from $\mathcal{S}$, where c corresponds to an index of a corrupt party $\mathcal{P}_c \in \{\mathcal{P}_i\}_{i\in[n]}$, set $\mathsf{out}_{\mathsf{pid}^*} = \mathsf{out}_{\mathsf{pid}}$ and send it to $\mathsf{P}_{\mathsf{pid}^*}$.

We note that the functionality as written allows cheaters, as well as *computation outputs* to be certified to external parties. The latter does not harm feasibility of achieving this notion—the functionality itself could compute signed outputs for example—but may be dropped if desired for efficiency reasons.

5.2 Compiler

In this section, we present a compiler that transforms an MPC protocol Π_{IA} achieving Identifiable Abort (IA) in the $\mathcal{F}_{\mathsf{BC}}$-hybrid model to an MPC protocol Π_{PISA} that achieves PISA in the $\mathcal{F}_{\mathsf{BC-IA}}$ hybrid model. The compiler makes blackbox use of the protocol, and therefore the compiled protocol computes the same function except with PISA guarantees. Here, IA [AL10, IOZ14] (refer Appendix C.1 for ideal functionality) refers to the notion where all honest parties either obtain the output or agree on a common cheater, and $\mathcal{F}_{\mathsf{BC}}$ denotes the ideal functionality for standard broadcast. We assume an honest majority among the participants running Π_{IA} as well as Π_{PISA}. Note that this assumption is necessary for Π_{PISA}, as mentioned earlier.

The main idea of our compiler is to replace each invocation of broadcast by Π_{IA} with a corresponding one to BC-IA instead. We incrementally construct our compiler by explaining our ideas below. Consider a direct replacement of one such round of broadcast by a round of BC-IA. There are two possible outcomes:

1. None of the honest parties observed an abort in this BC-IA invocation. By consistency of BC-IA, broadcast has been emulated perfectly for Π_{IA}.

2. There is at least one aborting honest party. This party must have obtained a certifiable proof of cheating against the dealer of a BC-IA invocation. We instruct this party to transfer this certifiable cheat to other honest parties in the subsequent round, therefore inducing honest parties to halt the protocol and return this certificate of cheating as their Π_{PISA} output. However upon receiving such a certificate, before halting, each honest party echoes this certificate to all other honest parties. Note that this mechanism adds at most two rounds in total: the moment an honest party observes a cheat, all other honest parties are made aware in the next round, implying that all honest parties are guaranteed to terminate within two rounds of *any* honest party observing a cheat via BC-IA.

Consider the case that no aborts are observed in any BC-IA invocation. Then, the honest parties must have received all the messages relayed via broadcast in Π_{IA}, and are therefore able to compute their outputs as prescribed in Π_{IA}. By the guarantees of IA security, this results in one of the two following cases:

1. All honest parties obtain the output (say, y): In this case, the honest parties simply output y as output of Π_{PISA}.

2. All honest parties abort, but identify a common cheater (say, $\mathcal{P}_{\mathsf{c}}$): In this case, we instruct each party to announce the identity of the cheater via another invocation of BC-IA. Given the honest majority, each honest party is able to collect at least $t+1$ attestations of the fact that $\mathcal{P}_{\mathsf{c}}$ is corrupt, which suffices to convince any external auditor.

We note that a certificate as described above cannot be forged by an adversary to implicate an honest party because **(a)** No honest party will identify another honest party by the guarantees of Π_{IA}, and **(b)** the adversary only controls t parties, whereas $t+1$ attestations are required to assemble a certificate. This completes a high-level description of our compiler.

We give a formal description of the compiler below.

Protocol 5.2. $\Pi_{\mathsf{PISA}}(n,t,f)$**: MPC protocol achieving** PISA

This protocol is parameterized by the party count n and the threshold t such that $n \geq 2t+1$. It is also parameterized by a function $f : \mathbb{X}_1 \times \mathbb{X}_2 \cdots \times \mathbb{X}_n \to \mathbb{Y}$. The protocol is run by parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, of which any t may be corrupt. The output of this protocol is $f(x_1, \ldots, x_n)$, where $x_i \in \mathbb{X}_i$ denotes $\mathcal{P}_i$'s input for $i \in [n]$. Any failure in obtaining the output is accompanied by a certificate implicating a corrupt party. Note that for readability, we do not explicitly specify each time that a fresh session identifier sid is required.

Building Block: An R-round MPC protocol Π_{IA} realizing $\mathcal{F}_{\mathsf{IA}}$.

1. For each round $r \in [R]$ of Π_{IA}, do the following:

   a. (When $r \geq 2$) If any party transferred a cheat $(\mathsf{sid}, \mathtt{cert} = (\mathtt{type}, \mathsf{c}))$ to $\mathcal{P}_i$ in the previous BC-IA round, $\mathcal{P}_i$ transfers this to all parties (via the transfer interface of BC-IA as well), and terminates with

output $\textbf{output}_i = (\mathsf{type} \in \{\mathtt{cheat}, \perp\}, \mathsf{c})$ as appropriate.

b. Suppose Π_{IA} requires $\mathcal{P}_i$ to communicate via point-to-point channels, then $\mathcal{P}_i$ follows the same instructions of Π_{IA}.

c. Suppose Π_{IA} requires $\mathcal{P}_i$ to invoke $\mathcal{F}_{\mathsf{BC}}$ as sender with message msg, then $\mathcal{P}_i$ invokes $\mathcal{F}_{\mathsf{BC-IA}}(n, t, i)$ with $(\mathtt{deal}, \mathsf{sid}, \mathsf{msg})$.

d. Each $\mathcal{P}_j$ ($j \in [n]$) does the following:

- If the BC-IA round terminated with a cheat $(\mathsf{sid}, \mathtt{cert} = (\mathtt{type}, i))$ against dealer $\mathcal{P}_i$, then $\mathcal{P}_j$ transfers this output to all other parties using $(\mathtt{transfer}, \mathsf{sid}, j, k)$ for $k \in [n] \setminus j$ and terminates with output $\textbf{output}_j = (\mathsf{type} \in \{\mathtt{cheat}, \perp\}, i)$ as appropriate.
- Else $\mathcal{P}_j$ sets $\mathsf{msg}^{(i,r)} = \textbf{out}_j^i$ as the message broadcast by P_i in Round r, where $\textbf{out}_j^i$ denotes $\mathcal{P}_j$'s output of this BC-IA round corresponding to dealer $\mathcal{P}_i$. Continue to the next step in Π_{IA}.

2. Each $\mathcal{P}_j$ ($j \in [n]$) computes the output as per Π_{IA} using $\{\mathsf{msg}^{(i,r)}\}_{i\in[n],r\in[R]}$ and the point-to-point messages received so far. The output of Π_{IA} corresponds to either an output y or $(\mathtt{abort}, \mathsf{sid}, \mathsf{c})$, where c is the identity of the cheater revealed to all honest parties.

- If the output computed is y, then $\mathcal{P}_j$ outputs $\textbf{output}_j = y$ and terminates.
- Else, $\mathcal{P}_j$ invokes $\mathcal{F}_{\mathsf{BC-IA}}(n, t, j)$ with $(\mathtt{deal}, \mathsf{sid}, \mathsf{msg} = (\mathtt{abort}, \mathsf{c}))$.

3. Each $\mathcal{P}_j$ collects at least $t+1$ session identifiers $\textbf{sid}$ such that $(\mathsf{sid}, \mathtt{dealt}, (\mathtt{abort}, \mathsf{c}))$ was received from BC-IA for each $\mathsf{sid} \in \textbf{sid}$. This constitutes a certificate of cheating. In order to transfer this certificate to $\mathsf{P}_{\mathsf{pid}}$, parties $\mathcal{P}_j$ and $\mathsf{P}_{\mathsf{pid}}$ jointly invoke the BC-IA transfer interface for each $\mathsf{sid} \in \textbf{sid}$, and $\mathsf{P}_{\mathsf{pid}}$ verifies that they are all $(\mathsf{sid}, \mathtt{dealt}, (\mathtt{abort}, \mathsf{c}))$.

Round Complexity of Π_{PISA}. Observe that each invocation of $\mathcal{F}_{\mathsf{BC}}$ is replaced with an invocation of $\mathcal{F}_{\mathsf{BC-IA}}$, and there is an additional instance of BC-IA at the end of the protocol to announce the output of Π_{IA}. Therefore, if Π_{IA} has R BC rounds, then Π_{PISA} uses $(R+1)$ BC-IA rounds. Instantiating BC-IA with our 2 round protocol $\pi_{\mathsf{BC}}(n, t, d)$ (Protocol 4.3) yields an overall point-to-point round complexity of $2R + 2$ rounds for Π_{PISA}.

Note that the transfer of cheats detected (if any) in the BC-IA invocation corresponding to Round r of Π_{IA} replaces the BC-IA invocation of Round $(r+1)$, for $r \in [R-1]$. Therefore, this does not incur additional overhead in rounds. Furthermore, there is no need to transfer cheats detected in the BC-IA invocation corresponding to the last Round R of Π_{IA}, as the parties who did not detect a cheat can simply proceed to output computation of Π_{IA}.

Security Analysis. As there must exist a simulator $\mathcal{S}$ for any $\mathcal{A}_{\mathsf{IA}}$ (by virtue of security of Π_{IA}), we can use $\mathcal{S}$ to construct a simulator for Π_{PISA}. In order to

argue security, we show how any adversary $\mathcal{A}_{\mathsf{PISA}}$ for Π_{PISA} can be transformed into an adversary $\mathcal{A}_{\mathsf{IA}}$ for Π_{IA}. In particular, given a challenge execution of Π_{IA}, we construct an execution of Π_{PISA} which $\mathcal{A}_{\mathsf{PISA}}$ attempts to decide is real or simulated. Observe that Π_{PISA} is in essence a partial execution of Π_{IA}—constructing our reduction is therefore a matter of filtering the correct messages from Π_{IA} to Π_{PISA} for $\mathcal{A}_{\mathsf{IA}}$. We specify $\mathcal{A}_{\mathsf{IA}}$ below:

1. As long as $\mathcal{A}_{\mathsf{PISA}}$ does not abort any of the BC-IA invocations, $\mathcal{A}_{\mathsf{IA}}$ behaves the same as $\mathcal{A}_{\mathsf{PISA}}$ i.e. the messages sent in the BC-IA rounds in Π_{PISA} are simply sent as corresponding BC rounds in Π_{IA} and point-to-point messages are simply forwarded.

2. Consider the first time $\mathcal{A}_{\mathsf{PISA}}$ aborts a BC-IA invocation, say corresponding to round r of Π_{IA} by corrupt dealer $\mathcal{P}_i$. This can be done by allowing at least one honest party to detect a cheat either as output of BC-IA or via a transfer [2]. In this scenario, if none of the honest parties received the intended message, then $\mathcal{A}_{\mathsf{IA}}$ simply broadcasts $\perp$ on behalf of $\mathcal{P}_i$ in Round r of Π_{IA} . However, if at least some of the honest parties received a message m, then $\mathcal{A}_{\mathsf{IA}}$ does the following: Broadcast m on behalf of $\mathcal{P}_i$ in Round r of Π_{IA}. In the subsequent BC round, $\mathcal{A}_{\mathsf{IA}}$ filters messages sent by honest parties in round $(r+1)$ of Π_{IA} while forwarding them to $\mathcal{A}_{\mathsf{PISA}}$ in Π_{PISA}: Only the messages of honest parties who did not discover the cheat in the previous BC-IA round are forwarded. From the next round onwards, none of the messages from honest parties are forwarded (as this corresponds to discovering a cheat and terminating in Π_{PISA}).

3. $\mathcal{A}_{\mathsf{IA}}$ outputs whatever $\mathcal{A}_{\mathsf{PISA}}$ does.

Observe that the only difference between an execution of Π_{IA} with $\mathcal{A}_{\mathsf{IA}}$ and Π_{PISA} with $\mathcal{A}_{\mathsf{PISA}}$ is that $\mathcal{A}_{\mathsf{PISA}}$ could choose to make some honest parties abort earlier than others during the BC-IA invocations, unlike in Π_{IA} where all honest parties have a consistent view of the BC invocations at all times. Further, we note that the actions of $\mathcal{A}_{\mathsf{PISA}}$ during the last BC-IA invocation (to announce the output of Π_{IA}) does not affect the honest parties' computation (as their outputs are already determined once the R rounds of Π_{IA} have been emulated). With respect to honest parties that do not abort in Π_{PISA}, note that their messages can be simulated identically to the simulation in Π_{IA}. We can thus conclude that the indistinguishability of the simulation of Π_{PISA} is implied by security of Π_{IA}.

We are now ready to state the formal theorem.

Theorem 5.3. *Suppose a protocol Π_{IA} among parties $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ realizes $\mathcal{F}_{\mathsf{IA}}$ in the $\mathcal{F}_{\mathsf{BC}}$-hybrid model, where $n \geq 2t+1$. Then, the protocol Π_{PISA} realizes $\mathcal{F}_{\mathsf{PISA}}$ in the $\mathcal{F}_{\mathsf{BC-IA}}$ hybrid model, against a malicious adversary that corrupts $t < n/2$ parties.*

[2] Note that it is always possible for a corrupt party to transfer a cheat implicating a corrupt dealer.

We state the corollaries that follow when BC-IA is realized using our 2-round protocol 4.3 and suitable existing instantiations of Π_{IA} are plugged in: **(a)** the two-round IA protocol of [CGZ20] in the CRS model that uses broadcast in both rounds and **(b)** the four-round IA protocol of [CRSW22] in the plain model that uses broadcast in all rounds.

Corollary 5.4. *Assuming a common reference string and a PKI setup, there exists a 6-round* PISA *MPC protocol against a malicious adversary that corrupts $t < n/2$ parties, that uses only point-to-point channels.*

Corollary 5.5. *Assuming PKI, there exists a 10-round* PISA *MPC protocol against a malicious adversary that corrupts $t < n/2$ parties, that uses only point-to-point channels.*

Notably, the use of PKI in the above results in limited to realizing BC-IA.

As a final remark for this section, we note that in the above description, we assumed that the same set of parties participate in the underlying protocol Π_{IA}, as well as the compiled protocol Π_{PISA}. This necessitates an honest majority amongst the parties running Π_{IA}, as $\mathcal{F}_{\mathsf{PISA}}$ can be realized only with an honest majority. However, one could consider a setting where Π_{IA} is run among a dishonest majority of participants, say $\mathcal{M} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$, where $m \geq t + 1$; while an extended set of participants $\mathcal{N} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, where $n \geq 2t + 1$ are registered with $\mathcal{F}_{\mathsf{BC-IA}}$ as "helper" nodes (where $\mathcal{M} \subset \mathcal{N}$). We discuss a modified compiler for such a setting in Appendix C.2.

6 Verifiable Secret Sharing Over P2P Channels

In this section, we construct Verifiable Secret Sharing over point-to-point channels as an important building block towards our final ECDSA protocol. The intended outcome of this protocol is not just to share a secret, but to jointly sample a uniformly random secret in Shamir shared form. In addition, we wish for a commitment—a Pedersen commitment in particular—of each party's share to be made publicly available to all parties. Intuitively, all parties' Pedersen commitments will lie along a degree-t polynomial, and we model adversarial bias (of the commitments) by letting the adversary simply choose this polynomial. Note that the perfectly hiding nature of Pedersen commitments ensures that the secret itself stays perfectly hidden, and uniformly random.

6.1 Ideal Functionality

We begin with a description of the ideal functionality, and then proceed to describe the protocol.

Functionality 6.1. $\mathcal{F}_{\mathsf{VSS}}(\mathcal{G}, n, t)$**: Verifiable Secret Sharing**

This functionality is parameterized by the party count n, the threshold t such that $n \geq 2t+1$, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. This functionality

is accessed by parties $\{\mathcal{P}_i\}_{i\in[n]}$ at most t of whom may be corrupt. Let the set of honest parties be indexed by $\mathcal{H}$.

Share: On receiving $(\texttt{share}, \mathsf{sid})$ from all parties such that $\mathsf{sid} =: \mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}'$ and sid is fresh,

1. Sample $d \leftarrow \mathbb{Z}_q$, and set $\hat{G} = d \cdot G$.
2. Send $\hat{G}$ to the adversary $\mathcal{S}$.
3. Receive the following values from $\mathcal{S}$:
   - $\{\mathbf{vssout}_i \in \{\texttt{VSS-success}, \texttt{cert} = (\texttt{type}, \mathsf{c})\}\}_{i\in[n]}$, where $\texttt{type} \in \{\mathsf{cheat}, \perp\}$ specifies the type of cheating certificate and c corresponds to the index of the corrupt party implicated to $\mathcal{P}_i$ as the cheater.
   - $C, (x, \hat{x}), \{(\mathsf{sh}_i, \hat{\mathsf{sh}}_i)\}_{i\notin\mathcal{H}}$, where $C \in \mathbb{G}[X]$ is a degree t polynomial, and the rest are $\mathbb{Z}_q$ values subject to:

$$\mathsf{sh}_i \cdot G + \hat{\mathsf{sh}}_i \cdot \hat{G} = C(i) \quad \text{and} \quad x \cdot G + \hat{x} \cdot \hat{G} = C(0)$$

4. Sample the secret $\mathsf{s} \leftarrow \mathbb{Z}_q$ uniformly at random.
5. Compute 'opening information' $\hat{\mathsf{s}}$ of $C(0)$ for the above secret as $\hat{\mathsf{s}} = (x - \mathsf{s})/d + \hat{\mathsf{s}}$
6. Define degree t polynomials $f, \hat{f} \in \mathbb{Z}_q[X]$ such that

$$f(0) = \mathsf{s},\ \hat{f}(0) = \hat{\mathsf{s}} \quad \text{and} \quad \{f(i) = \mathsf{sh}_i,\ \hat{f}(i) = \hat{\mathsf{sh}}_i\}_{i\notin\mathcal{H}}$$

7. For each $i \in \mathcal{H}$ where $\mathbf{vssout}_i = \texttt{VSS-success}$, set $\mathbf{vssout}_i = (\texttt{VSS-success}, C, f(i), \hat{f}(i))$.

Send $(\texttt{shared}, \mathsf{sid}, \mathbf{vssout}_i)$ to each $\mathcal{P}_i$. Additionally, for each pid corresponding to every party $h \in \mathcal{H}$, set and store $\mathsf{vssout}_{\mathsf{pid}} = (\mathsf{sid}, \mathbf{vssout}_i)$.

Transfer: Upon receiving $(\texttt{cert-transfer}, \mathsf{sid}, \mathsf{pid}, \mathsf{pid}^*)$ from both $\mathsf{P}_{\mathsf{pid}}$ and $\mathsf{P}_{\mathsf{pid}^*}$

1. If $\mathsf{P}_{\mathsf{pid}}$ is honest and $\mathsf{vssout}_{\mathsf{pid}}$ is of the form $(\mathsf{sid}, \texttt{cert} = (\texttt{type}, \mathsf{c}))$, send $\mathsf{vssout}_{\mathsf{pid}}$ to $\mathsf{P}_{\mathsf{pid}^*}$.
2. Otherwise, receive $\mathsf{vssout}_{\mathsf{pid}}$ of the form $(\mathsf{sid}, \texttt{cert} = (\texttt{type}, \mathsf{c}))$ from $\mathcal{S}$ where c indexes a corrupt party, and send it to $\mathsf{P}_{\mathsf{pid}^*}$.

6.2 The Protocol

Before we give the protocol overview, we note that in this and subsequent protocols, we make use of the fact that the broadcast protocol π_{BC} accompanies dealt messages with the dealer's signature. In formal protocol descriptions we invoke broadcast through the $\mathcal{F}_{\mathsf{BC-IA}}$ functionality, and therefore must use its 'transfer' interface to access such signatures. In informal descriptions we refer to such signatures directly, as it is far more readable to follow the actual implementation.

Our honest execution path follows well-established techniques for this task, along the lines of Feldman [Fel87], Pedersen [Ped92], and Gennaro et al. [GJKR07], and requires a single broadcast round. Each $\mathcal{P}_i$ is instructed to sample secret polynomials $f_i, \hat{f}_i \in \mathbb{Z}_q^{\leq t}[X]$, and designate each $\mathcal{P}_j$ to receive $f_i(j), \hat{f}_i(j)$ privately while $C_i(j) = f_i(j) \cdot G + \hat{f}_i(j) \cdot \hat{G}$ is made public. The joint secret is therefore defined to be $\sum_i f_i(0)$. In case $\mathcal{P}_i$ is cheated by $\mathcal{P}_j$ in that its privately communicated $f_j(i), \hat{f}_j(i)$ do not match the public $C_j(i)$, party $\mathcal{P}_i$ saves the offending ciphertext as a certificate of $\mathcal{P}_j$'s misbehaviour, accompanied by a proof of correct decryption.

- **Broadcast Round 1** as initiated by each $\mathcal{P}_i$ for $i \in [t+1]$: Each $\mathcal{P}_i$ samples degree-t polynomials $f_i, \hat{f}_i \in \mathbb{Z}_q$, and defines polynomial $C_i(x) = f_i(x) \cdot G + \hat{f}_i(x) \cdot \hat{G}$.

  For each $j \in [n]$, $\mathcal{P}_i$ prepares a ciphertext $\mathbf{ct}_{ij} \leftarrow \mathsf{Enc}_{\mathsf{pk}_j}(f_i(j), \hat{f}_i(j))$. Finally, $\mathcal{P}_i$ broadcasts $(C_i, \mathbf{ct}_i = \{\mathbf{ct}_{ij}\}_{j \in [n]})$ via π_{BC}. Note that π_{BC} is run amongst all n parties.

- **Output** per $\mathcal{P}_i$: any certificate of cheating by π_{BC} is output, if it exists. In case $\exists j \in [t+1]$ such that decrypting $\mathbf{ct}_{ji}$ does not yield $f_j(i), \hat{f}_j(i)$ corresponding to C_j, output Ω that consists of its (proven) decryption. Otherwise, output $(C, f(i), \hat{f}(i)) = \sum_{j \in [t+1]} (C_j, f_j(i), \hat{f}_j(i))$.

Analysis. By security of π_{BC}, it holds that for each $j \in [n]$, every non-aborting honest $\mathcal{P}_i$ agrees on $(C_j, \mathbf{ct}_j)$. Therefore, any honest party that terminates successfully will output the same public commitment C. Moreover, every such $\mathcal{P}_i$ also outputs $f(i), \hat{f}(i)$ such that $f_j(i) \cdot G + \hat{f}_j(i) \cdot \hat{G} = C_j(i)$.

A rushing $\mathcal{P}_j$ can arbitrarily bias C by choosing C_j after seeing every other party's message, but this has *no* effect on the distribution of the joint secret, as each C_i perfectly hides $f_i(0)$.

Any aborting execution reduces to the following cases:

1. An abort in π_{BC}, which will be accompanied by a certificate Ω or ω as relevant.
2. The existence of $\mathbf{ct}_{ji}$ that does not decrypt to $f_j(i), \hat{f}_j(i)$ such that $f_j(i) \cdot G + \hat{f}_j(i) \cdot \hat{G} = C_j(i)$. This readily yields a certificate Ω as detailed in the protocol.

Therefore, any honest party that is not in agreement about C or in possession of a valid decommitment $f(i), \hat{f}(i)$ to $C(i)$ will be able to produce a certificate Ω or ω to implicate some corrupt $\mathcal{P}_j$ for the failure.

Protocol 6.2. $\pi_{\mathsf{VSS}}(\mathcal{G}, n, t)$**: Honest Majority VSS With IA**

This protocol is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. Additionally, it makes use of a *common random string* $\hat{G}$, whose discrete logarithm relative to G is unknown. The protocol runs once with parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, of which any t may be corrupt. The private output of this protocol for $\mathcal{P}_i$ is $(C, f(i), \hat{f}(i))$ where $C \in \mathbb{G}[X]$ is a common degree-t polynomial, and $f(i) \cdot G + \hat{f}(i) \cdot \hat{G} = C(i)$. Any failure in obtaining the output is accompanied by failure certificate which we assume is forwarded by the aborting party to others before terminating. We do not give explicit instructions for the transfer interface, but instead express inline how the proof of each cheat is transferred upon invocation.

This protocol functions in the $\mathcal{F}_{\mathsf{BC}}$-hybrid model, and makes use of an openable encryption scheme $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Open}, \mathsf{Vrfy})$ (Appendix A).

Share:

1. On receiving $(\mathtt{init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party $\mathcal{P}_i$ checks whether there exists a record of the form $(\mathsf{sid}, \cdot)$ in memory. If not, then each $\mathcal{P}_i$ for $i \in [t+1]$ does the following:
   a. Sample two degree t polynomials $f_i, \hat{f}_i \leftarrow \mathbb{Z}_q[X]$
   b. Define polynomial $C_i \in \mathbb{G}[X]$ such that $C_i(x) = f_i(x) \cdot G + \hat{f}_i(x) \cdot \hat{G}$
   c. Compute encrypted shares:
   $$\mathbf{ct}_i = \{\mathbf{ct}_{ij} \leftarrow \mathsf{Enc}_{\mathsf{pk}_j^{\mathsf{PKI}}}(f_i(j), \hat{f}_i(j))\}_{j \in [n]}$$
   d. Invoke an instance of $\mathcal{F}_{\mathsf{BC-IA}}(n, t, i)$ with $(\mathtt{deal}, \mathsf{sid}, \mathsf{msg} = (C_i, \mathbf{ct}_i))$ for a fresh sid.

   This completes the first phase.
2. Upon completion of the broadcast round, if $\exists j \in [t+1]$ such that the $\mathcal{F}_{\mathsf{BC}}$ instance in which $\mathcal{P}_j$ was the dealer resulted in an abort of $\mathsf{type} \in \{\mathtt{cheat}, \perp\}$, then $\mathcal{P}_i$ terminates with output $\mathbf{vssout}_i = (\mathsf{type}, j)$, and invokes the Transfer interface of $\mathcal{F}_{\mathsf{BC}}$ when invoked with this sid again.
3. Each party $\mathcal{P}_i$ does the following for $j \in [t+1]$:
   a. Obtain $f_j(i), \hat{f}_j(i) = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{ct}_{ji})$
   b. Verify that $f_j(i) \cdot G + \hat{f}_j(i) \cdot \hat{G} = C_j(i)$

- If this fails, open the ciphertext by computing

$$\zeta_{ji} = (f_j(i), \hat{f}_j(i), \pi_{\mathsf{ct}}) \leftarrow \mathsf{Open}(\mathsf{sk}_i, \mathbf{ct}_{ji})$$

and set

$$\Omega_i^j = (\texttt{bad-ct}, \zeta_{ji}, (\mathsf{sid}, \texttt{dealt}, (C_j, \mathbf{ct}_j), \sigma_j^{\mathsf{PKI}}))$$

If Ω_i^j is defined for some $j \in [t+1]$, then output $\mathbf{vssout}_i = (\texttt{cheat}, j)$, and transmit Ω_i^j in addition when invoked with the Transfer interface. Otherwise, compute the commitment polynomial and shares:

$$C = \sum_{i \in [t+1]} C_i$$
$$f(i) = \sum_{j \in [t+1]} f_j(i)$$
$$\hat{f}(i) = \sum_{j \in [t+1]} \hat{f}_j(i)$$

and output $(\texttt{VSS-success}, C, f(i), \hat{f}(i))$.

The formal theorem appears below, whose proof is deferred to the full version.

Theorem 6.3. *In the $\mathcal{F}_{\mathsf{BC}}$-hybrid model, $\pi_{\mathsf{VSS}}(\mathcal{G}, n, t)$ UC-realizes $\mathcal{F}_{\mathsf{VSS}}(\mathcal{G}, n, t)$ against a malicious adversary that corrupts up to $t < n/2$ parties.*

Zero sharing. As a special case of VSS, the protocol π_{Zero} generates a degree $n-1$ verifiable secret sharing of the constant value 0—this is accomplished by a straightforward tweak of π_{VSS} in which $t = n-1$ and the constant term of the polynomial is set to (and verified to be) zero. For completeness, we give this protocol in Appendix B. We denote the ideal functionality for this special case of VSS as $\mathcal{F}_{\mathsf{ZSS}}$ (which is defined similar to $\mathcal{F}_{\mathsf{VSS}}$ adopting the above tweaks).

7 Distributed Key Generation from VSS

Building on the previous section, we show how to construct *distributed key generation* (DKG) in the $\mathcal{F}_{\mathsf{VSS}}$-hybrid model, so that parties can establish a public $k \cdot G$ value such that k is secret shared. We first describe the ideal functionality for DKG, followed by the protocol description.

Functionality 7.1. $\mathcal{F}_{\mathsf{DKG}}(\mathcal{G}, n, t)$: Distributed Key Generation

This functionality is parameterized by the party count n, the threshold t such that $n \geq 2t+1$, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. This functionality

is accessed by parties $\{\mathcal{P}_i\}_{i\in[n]}$ at most t of whom may be corrupt. Let the set of honest parties be indexed by $\mathcal{H}$.

Dkeygen: On receiving $(\texttt{dkeygen}, \mathsf{sid})$ from all parties such that $\mathsf{sid} =: \mathcal{P}_1\|\ldots\|\mathcal{P}_n\|\mathsf{sid}'$ and sid is fresh,

1. Receive $\{(\mathsf{sk}_i, \mathsf{pk}_i)\}_{i\notin\mathcal{H}}$ from adversary $\mathcal{S}$ where $\mathsf{sk}_i \cdot G = \mathsf{pk}_i$.
2. Sample the secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q$ uniformly at random.
3. Define degree t polynomial $f \in \mathbb{Z}_q[X]$ such that $f(0) = \mathsf{sk}$ and $f(i) = \mathsf{sk}_i$ for $i \notin \mathcal{H}$.
4. Let $F \in \mathbb{G}[X]$ be the degree t polynomial such that $F(j) = f(j) \cdot G$ for $j \in [n]$.
5. Send F to $\mathcal{S}$ and receive in response $\{\mathbf{dkgout}_i \in \{\texttt{key-pair}, \texttt{cert} = (\texttt{type}, \mathsf{c})\}\}_{i\in[n]}$ from $\mathcal{S}$, where $\texttt{type} \in \{\mathsf{cheat}, \perp\}$ specifies the type of cheating certificate and c corresponds to the index of the corrupt party implicated to $\mathcal{P}_i$.

If $\mathbf{dkgout}_i = \texttt{key-pair}$, set $\mathbf{dkgout}_i = (\texttt{key-pair}, F, f(i))$. Send $(\texttt{DKG-done}, \mathsf{sid}, \mathbf{dkgout}_i)$ to each $\mathcal{P}_i$. Additionally, for each pid set and store $\mathsf{dkgout}_{\mathsf{pid}} = (\mathsf{sid}, \mathbf{dkgout}_i)$.

Transfer: Upon receiving $(\texttt{cert-transfer}, \mathsf{sid}, \mathsf{pid}, \mathsf{pid}^*)$ from both $\mathsf{P}_{\mathsf{pid}}$ and $\mathsf{P}_{\mathsf{pid}^*}$

1. If $\mathsf{P}_{\mathsf{pid}}$ is honest and $\mathsf{dkgout}_{\mathsf{pid}}$ is of the form $(\mathsf{sid}, \texttt{cert} = (\texttt{type}, \mathsf{c}))$, send $\mathsf{dkgout}_{\mathsf{pid}}$ to $\mathsf{P}_{\mathsf{pid}^*}$.
2. Otherwise, receive $\mathsf{dkgout}_{\mathsf{pid}}$ of the form $(\mathsf{sid}, \texttt{cert} = (\texttt{type}, \mathsf{c}))$ from $\mathcal{S}$ where c indexes a corrupt party, and send it to $\mathsf{P}_{\mathsf{pid}^*}$.

Recall that π_{VSS} establishes polynomials $f, \hat{f}$ such that each $\mathcal{P}_i$ holds $f(i), \hat{f}(i)$ that correspond to a public $C(i) = f(i) \cdot G + \hat{f}(i) \cdot \hat{G}$. Observe that $C(i)$ is a Pedersen commitment to which $\mathcal{P}_i$ knows an opening. Consider the polynomials $F, \hat{F} \in \mathbb{G}[X]$ such that $F(i) = f(i) \cdot G$ and $\hat{F}(i) = \hat{f}(i) \cdot \hat{G}$: observe that π_{VSS} essentially samples $F + \hat{F}$, and while this sum can be biased, F itself is *perfectly masked* by $\hat{F}$, and therefore uniformly random at the termination of π_{VSS}. Our π_{DKG} protocol therefore serves to simply unmask F by securely prising apart F and $\hat{F}$, so that F—and its discrete logarithm f—may be used as the DKG polynomial per the usual Feldman format [Fel87, GJKR07].

The above prising apart is accomplished in a single broadcast round after π_{VSS}: each $\mathcal{P}_i$ broadcasts $F_i = f(i) \cdot G$ (which implies a $\hat{F}_i = \hat{f}(i) \cdot \hat{G}$ given $C(i)$) along with a proof of knowledge of $\mathsf{DLog}_G F_i$ and $\mathsf{DLog}_{\hat{G}} \hat{F}_i$. Intuitively, due to the same reason that Pedersen commitments are binding, there is exactly one value of $F_i, \hat{F}_i$ for which $\mathcal{P}_i$ can produce such a proof. The value $f(0)$ therefore serves as the uniformly sampled secret key with the public $F(0)$, and each $\mathcal{P}_i$ holds a private $f(i)$ along with the public $F_i = f(i) \cdot G$.

In more detail, the protocol π_{DKG} proceeds as follows:

- **Broadcast Round 1**: All parties run π_{VSS} so that each $\mathcal{P}_i$ obtains $(C, f(i), \hat{f}(i))$ upon successful termination.
- **Broadcast Round 2:** Each $\mathcal{P}_i$ sets $F_i = f(i) \cdot G$ and $\hat{F}_i = \hat{f}(i) \cdot \hat{G}$, and prepares a NIZK to prove knowledge of their respective discrete logarithms: $\pi^i_{\mathsf{DL}\wedge} \leftarrow P_{\mathsf{DL}\wedge}((G, f(i), F_i), (\hat{G}, \hat{f}(i), \hat{F}_i))$. In addition, D is defined to be $\mathsf{CRHF}(C)$, towards certifying agreement on C as the output of the previous broadcast round.

  $\mathcal{P}_i$ then broadcasts $(D, \hat{F}_i, \pi^i_{\mathsf{DL}\wedge})$.
- **Output:** Every $\mathcal{P}_i$ checks that for $j \in [n]$ every $V_{\mathsf{DL}\wedge}(\pi^j_{\mathsf{DL}\wedge}, (G, F_j), (\hat{G}, \hat{F}_j)) = 1$, where $\hat{F}_j = C(j) - F_j$. In case $\exists j \in [n]$ for which this proof doesn't verify, $\mathcal{P}_i$ is able to produce a certificate Ω to implicate $\mathcal{P}_j$ as follows:

  1. Find $t+1$ (signed) messages broadcast in the previous round that all agree on D, and assemble them into a certificate ϖ_{BC}.
  2. Prepare a certificate Ω that consists of C to establish context, ϖ_{BC} that proves that C was indeed the outcome of π_{VSS}, and finally $\hat{F}_j, \pi^j_{\mathsf{DL}\wedge}$ accompanied by $\mathcal{P}_j$'s signature.

  As usual, any certificates of cheating produced in the course of π_{VSS} are output directly in case of an abort. If no cheating is detected, set $F \in \mathbb{G}[X]$ to be the degree t polynomial such that $F(j) = F_j$, and output $(F, f(i))$.

Analysis. By security of π_{VSS}, it holds that for each $j \in [n]$, every non-aborting honest $\mathcal{P}_i$ agrees on a degree t polynomial $C \in \mathbb{G}[X]$, and privately obtains $f(i), \hat{f}(i)$ such that $C(i) = f(i) \cdot G + \hat{f}(i) \cdot \hat{G}$. Therefore, $F_i = f(i) \cdot G$ and $\hat{F}_i = \hat{f}(i) \cdot \hat{G}$ as broadcast by honest parties in the second round lie on degree t polynomials F and $\hat{F}$ respectively.

A corrupt party $\mathcal{P}_j$ may choose to broadcast $F^*_j \neq F(j)$, but we argue that it will be unable to produce an accepting proof $\pi^{j*}_{\mathsf{DL}\wedge}$ for such a value, by constructing a reduction to the hardness of computing discrete logarithms in $\mathbb{G}$. Recall that $\mathsf{DLog}_G \hat{G}$ is unknown, and that the "honest" pair $f(j), \hat{f}(j)$ such that $C(j) = f(j) \cdot G + \hat{f}(j) \cdot \hat{G}$ is already fully specified by extrapolation of honest parties' $f(i), \hat{f}(i)$ values. The reduction therefore embeds the discrete log challenge in the adversary's view in the form of $\hat{G}$, and runs the protocol honestly (controlling $t+1$ uncorrupt parties) to obtain $f(j), \hat{f}(j)$ as an opening to the Pedersen commitment $C(j)$. Given that $\pi^{j*}_{\mathsf{DL}\wedge}$ is a simulation extractable proof of knowledge, if $\mathcal{P}_j$ is able to produce $\pi^{j*}_{\mathsf{DL}\wedge}$ that proves knowledge of $\mathsf{DLog}_G F^*_j$ and $\mathsf{DLog}_{\hat{G}} \hat{F}^*_j$ such that $(F^*_j, \hat{F}^*_j) \neq (F_j, \hat{F}_j)$, the corresponding witness $f^*(j), \hat{f}^*(j)$ can be efficiently extracted from the adversary with nearly the same probability. This yields another opening to Pedersen commitment $C(j)$,

which in combination with $f(j), \hat{f}(j)$ can be used to compute

$$\mathsf{DLog}_G \hat{G} = \frac{f(j) - f^*(j)}{\hat{f}^*(j) - \hat{f}(j)}$$

and therefore satisfy the discrete logarithm challenger with nearly the same probability a corrupt $\mathcal{P}_j$ is able to prove an inconsistent $F_j^* \neq F(j)$.

The adversary is then left with the following options:

1. Deviate from the protocol in π_{VSS}, which will result in an honest party obtaining certificate Ω or ω as appropriate.
2. Each corrupt $\mathcal{P}_j$ broadcasts the honest $(D, \hat{F}_j, \pi^j_{\mathsf{DL}\wedge})$, in which case each honest $\mathcal{P}_i$ outputs consistent $(F, f(i))$.
3. Some corrupt $\mathcal{P}_j$ broadcasts a tuple in which $\pi^j_{\mathsf{DL}\wedge}$ does not verify. This results in honest parties assembling a certificate Ω proving $\mathcal{P}_j$'s deviation from the protocol by: first establishing the polynomial F as output by π_{VSS} (certified by ϖ_{BC}, a collection of $t+1$ signatures on $D = \mathsf{CRHF}(C)$), and $\mathcal{P}_j$'s signature on the NIZK $\pi^j_{\mathsf{DL}\wedge}$ that does not verify relative to the statement implied by $C(j)$.

Therefore, in every scenario, each honest party $\mathcal{P}_i$ either outputs a consistent, uniformly random degree t polynomial $F \in \mathbb{G}[X]$ such that it holds $\mathsf{DLog}_G(F(i)) = f(i)$, or a certificate Ω (or ω) that conclusively proves malicious behaviour on the part of a corrupt $\mathcal{P}_j$.

Protocol 7.2. $\pi_{\mathsf{DKG}}(\mathcal{G}, n, t)$**: DKG With IA**

This protocol is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The protocol is designed in the $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{VSS}})$-hybrid model and makes use of a collision-resistant hash function CRHF and a simulation-extractable NIZK proof system $(P_{\mathsf{DL}\wedge}, V_{\mathsf{DL}\wedge})$ to prove knowledge of two discrete logarithms simultaneously. The private output for a successful execution of this protocol for $\mathcal{P}_i$ is $(F, f(i))$ where $F \in \mathbb{G}[X]$ is a common degree-t polynomial, and $f(i)$ is the discrete logarithm of $F(i)$.

We do not give explicit instructions for the transfer interface, but instead express inline how the proof of each cheat is transferred upon invocation.

Generate Key:

1. On receiving $(\texttt{init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party $\mathcal{P}_i$ checks whether there exists a record of the form $(\texttt{key-pair}, \mathsf{sid}, \mathsf{pk}, p(i))$ in memory. If not, then each $\mathcal{P}_i$ for $i \in [n]$ invokes $\mathcal{F}_{\mathsf{VSS}}(\mathbb{G}, n, t)$ in order to generate $(\texttt{VSS-success}, C, f(i), \hat{f}(i))$.
2. If the VSS round terminated with a cheat $(\mathsf{sid}, \texttt{cert} = (\texttt{type}, \texttt{c}))$ instead, then $\mathcal{P}_i$ terminates with output $\mathbf{dkgout}_i = (\texttt{type} \in \{\texttt{cheat}, \perp\}, j)$ as appropriate, and transfers this output to all other parties.

3. Otherwise, each party $\mathcal{P}_i$ then does the following:

   a. Compute a digest of the public polynomial generated by $\mathcal{F}_{\mathsf{VSS}}$ as $D = \mathsf{CRHF}(C)$

   b. Define $F_i = f(i) \cdot G$ and $\hat{F}_i = \hat{f}(i) \cdot \hat{G}$

   c. Prepare a PoK of discrete logarithm pair,

   $$\pi^i_{\mathsf{DL}\wedge} \leftarrow P_{\mathsf{DL}\wedge}((G, f(i), F_i), (\hat{G}, \hat{f}(i), \hat{F}_i))$$

   d. Invoke $\mathcal{F}_{\mathsf{BC-IA}}(n, t, i)$ with $\left(\mathtt{deal}, \mathsf{sid}, (D, F_i, \pi^i_{\mathsf{DL}\wedge})\right)$

4. If any party transferred a cheat $(\mathsf{sid}, \mathtt{cert} = (\mathtt{type}, \mathsf{c}))$ in place of a standard broadcast in the above round, terminate with output $\mathbf{dkgout}_i = (\mathsf{type} \in \{\mathtt{cheat}, \perp\}, \mathsf{c})$ as appropriate. Otherwise, upon completion of the broadcast round, if $\exists j \in [t+1]$ such that the $\mathcal{F}_{\mathsf{BC}}$ instance in which $\mathcal{P}_j$ was the dealer resulted in an abort of $\mathsf{type} \in \{\mathtt{cheat}, \perp\}$, then $\mathcal{P}_i$ terminates with output $\mathbf{dkgout}_i = (\mathsf{type}, j)$, and invokes the Transfer interface of $\mathcal{F}_{\mathsf{BC}}$ when invoked with this sid again.

5. Each $\mathcal{P}_i$ collects at least $t+1$ tuples of the form $\left(\mathsf{sid}, \mathtt{dealt}, (D, F_j, \pi^j_{\mathsf{DL}\wedge}), \sigma_j^{\mathsf{PKI}}\right)$ from the above broadcast round, the crucial detail being that D matches the one locally computed in Step 3a. These tuples are concatenated to form a certificate ϖ_{BC}, which serves to show that at least $t+1$ parties attest to D.

6. For every $j \in [t+1]$, each party $\mathcal{P}_i$ computes $\hat{F}_j = C(j) - F_j$ and checks that $V_{\mathsf{DL}\wedge}(\pi^j_{\mathsf{DL}\wedge}, (G, F_j), (\hat{G}, \hat{F}_j)) = 1$.

   Any proof $j \in [n]$ that fails readily yields a certificate

   $$\Omega^j_i = \left(\mathtt{bad\text{-}dkg\text{-}proof}, D, C, (\pi^j_{\mathsf{DL}\wedge}, (G, F_j), (\hat{G}, \hat{F}_j))\right)$$

   $\mathcal{P}_i$ terminates with output $\mathbf{dkgout}_i = (\mathtt{cheat}, j)$. In order to transfer this cheat certificate, $\mathcal{P}_i$ sends Ω^j_i along with invoking the transfer interface of $t+1$ instances of $\mathcal{F}_{\mathsf{BC-IA}}$ from the previous round in which the output $\mathbf{out}_i$ contains the same D.

7. In the event that all proofs pass, each $\mathcal{P}_i$ outputs $(\mathtt{DKG\text{-}done}, F, f(i))$, where F is the degree t polynomial such that $F(j) = F_j$ for each $j \in [n]$.

The formal theorem appears below, whose proof is deferred to the full version.

Theorem 7.3. *In the $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{VSS}})$-hybrid model, $\pi_{\mathsf{DKG}}(\mathcal{G}, n, t)$ UC-realizes $\mathcal{F}_{\mathsf{DKG}}(\mathcal{G}, n, t)$ against a malicious adversary that corrupts up to $t < n/2$ parties.*

8 Distributed ECDSA Signing With Identifiable Abort

We first give the functionality realized by our ECDSA protocol, which is adapted from Doerner et al. [DKLs24].

Functionality 8.1. $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$**: Threshold ECDSA**

This functionality is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The setup phase runs once with n parties, and the signing phase may be run many times between (varying) subgroups of parties indexed by $\mathbf{P} \subseteq [n]$ such that $|\mathbf{P}| = 2t + 1$. If any party is corrupt, then the adversary $\mathcal{S}$ may instruct the functionality to abort or fail, but in so doing it must reveal the index of one corrupt party.

Setup: On receiving $(\texttt{init}, \mathsf{sid})$ from some party $\mathcal{P}_i$ such that $\mathsf{sid} =: \mathcal{P}_1 \| \ldots \| \mathcal{P}_n \| \mathsf{sid}'$ and $i \in [n]$ and sid is fresh, send $(\texttt{init-req}, \mathsf{sid}, i)$ to $\mathcal{S}$. On receiving $(\texttt{init}, \mathsf{sid})$ from all parties,

1. Sample the joint secret and public keys, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ECDSAGen}(\mathcal{G})$.
2. Store $(\texttt{secret-key}, \mathsf{sid}, \mathsf{sk})$ in memory.
3. Send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ directly to $\mathcal{S}$.
4. On receiving $(\texttt{release}, \mathsf{sid})$ from $\mathcal{S}$, send $(\texttt{public-key}, \mathsf{sid}, \mathsf{pk})$ to all parties and store $(\texttt{pk-delivered}, \mathsf{sid})$ in memory. On receiving $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_c)$ where c is the index of a corrupt party, send $(\texttt{abort}, \mathsf{sid}, \mathcal{P}_c)$ to all parties and halt.

Signing: On receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$ from any party $\mathcal{P}_i$, parse $\mathsf{sigid} =: \mathbf{P} \| \mathsf{sigid}'$ such that $|\mathbf{P}| = 2t + 1$ and ignore the message if $i \notin \mathbf{P}$ or $\mathbf{P} \not\subseteq [n]$ or sigid is not fresh or if $(\texttt{pk-delivered}, \mathsf{sid})$ does not exist in memory. Otherwise, send $(\texttt{sig-req}, \mathsf{sid}, \mathsf{sigid}, i, m)$ directly to $\mathcal{S}$.
On receiving $(\texttt{sign}, \mathsf{sid}, \mathsf{sigid}, m)$ from each $\mathcal{P}_i$ for every $i \in \mathbf{P}$, sample $\sigma \leftarrow \mathsf{ECDSASign}(\mathcal{G}, \mathsf{sk}, m_{\mathbf{P}_1})$ and then take the appropriate action:

- If no corrupt parties are indexed by $\mathbf{P}$, send $(\texttt{signature}, \mathsf{sid}, \mathsf{sigid}, \sigma)$ to $\mathcal{P}_i$ for every $i \in \mathbf{P}$.
- Otherwise, send σ to $\mathcal{S}$ and receive in response $\{\mathbf{sigout}_i \in \{\texttt{signature}, \texttt{cert} = (\texttt{type}, \mathsf{c})\}\}_{i \in [n]}$ from $\mathcal{S}$, where $\texttt{type} \in \{\mathsf{cheat}, \perp\}$ specifies the type of cheating certificate and c corresponds to the index of the corrupt party implicated to $\mathcal{P}_i$.

If $\mathbf{sigout}_i = \texttt{signature}$, set $\mathbf{sigout}_i = (\texttt{signature}, \sigma)$. Send $(\texttt{sign-done}, \mathsf{sid}, \mathbf{sigout}_i)$ to each $\mathcal{P}_i$. Additionally, for each pid set and store $\mathsf{sigout}_{\mathsf{pid}} = (\mathsf{sid}, \mathbf{sigout}_i)$.

Transfer: Upon receiving $(\texttt{cert-transfer}, \mathsf{sid}, \mathsf{pid}, \mathsf{pid}^*)$ from both $\mathsf{P}_{\mathsf{pid}}$ and $\mathsf{P}_{\mathsf{pid}^*}$

1. If $\mathsf{P}_{\mathsf{pid}}$ is honest and $\mathsf{sigout}_{\mathsf{pid}}$ is of the form $(\mathsf{sid}, \texttt{cert} = (\texttt{type}, \mathsf{c}))$, send $\mathsf{sigout}_{\mathsf{pid}}$ to $\mathsf{P}_{\mathsf{pid}^*}$.
2. Otherwise, receive $\mathsf{sigout}_{\mathsf{pid}}$ of the form $(\mathsf{sid}, \texttt{cert} = (\texttt{type}, \mathsf{c}))$ from $\mathcal{S}$ where c indexes a corrupt party, and send it to $\mathsf{P}_{\mathsf{pid}^*}$.

8.1 The Protocol

ECDSA Tuples. Our protocol is built upon the "ECDSA tuple" technique of Abram et al. [ANO+22], which we describe here. Recall that an ECDSA signature σ on a message m consists of two components (r^{x}, s), where r^{x} is the x-coordinate of the elliptic curve point $R = k \cdot G$ (a public value), and the scalar $s = (\mathsf{SHA2}(m) + r^{\mathsf{x}} \cdot \mathsf{sk})/k \pmod q$. The idea behind the ECDSA tuple (first used by Lindell and Nof [LN18]) is that the numerator and denominator of s can each be individually revealed in masked form. In particular, if $\phi \leftarrow \mathbb{Z}_q$ is a uniformly random mask, then one can safely reveal the masked numerator $w = (\mathsf{SHA2}(m) + r^{\mathsf{x}} \cdot \mathsf{sk}) \cdot \phi$, and the masked denominator $u = k \cdot \phi$. Intuitively, u is simply a uniform value from $\mathbb{Z}_q$ (as ϕ acts as a one-time pad), and w is perfectly specified upon fixing u and the s component of the signature—this implies a straightforward simulation strategy wherein u is sampled uniformly, and w is computed as $s \cdot u$. Given the masked numerator and denominator, computing the signature itself is as simple as just taking their quotient.

The ECDSA tuple itself is correlated randomness that enables the computation of the masked numerator and denominator information theoretically, much like a Beaver triple enables secure multiplication. It consists of shares of the values $(k, \mathsf{sk}, \phi, u, v)$, where $v = \phi \cdot \mathsf{sk}$ and $u = \phi \cdot k$. Observe that the denominator u is already included in the tuple, and the numerator w can be computed as $\mathsf{SHA2}(m) \cdot \phi + r^{\mathsf{x}} \cdot v$, which is a linear combination of secrets as $\mathsf{SHA2}(m)$ and r^{x} are public.

Securely Computing ECDSA Tuples. Doerner et al. [DKLs24] loosely sketched how ECDSA tuples can be derived in the honest majority setting, which we make concrete here. Observe that ECDSA tuples are a depth-1 correlation, meaning that there is an arithmetic circuit that computes it with only a single layer of (fan-in 2) multiplication gates. If degree t Shamir shares of k, sk, ϕ are available, completing the tuple by deriving degree $2t$ shares of u, v is non-interactive—one multiplication comes "for free" in this setting. Therefore, honest majority ECDSA signing can follow the structure below:

1. Degree t shares of sk are available by virtue of Distributed Key Generation executed during the setup phase.

2. The following values are sampled in two rounds during signing:

   a. R, and degree t shares of its discrete logarithm k, via Distributed Key Generation.

   b. Degree t shares of ϕ, jointly sampled by each party dealing Shamir shares of random values.

   c. Two degree $2t$ sharings of zero, jointly sampled by each party dealing Shamir shares of zero.

3. Degree $2t$ shares of u, v are derived by each party locally multiplying its shares of ϕ and k, sk respectively, and adding the shares of zero to randomize the result. They also locally derive shares of $w = \mathsf{SHA2}(m) \cdot \phi + r^{\mathsf{x}} \cdot v$.

4. Parties broadcast their shares of u and w to reconstruct them, and output the signature as $(R, s = w/u)$.

Adding Identifiability. Assuming that Verifiable Secret Sharing and Distributed Key Generation are available with Identifiable Abort, we can augment the above structure to support identifiability. To begin with, Steps 2b and 2c employ VSS rather than naive joint sampling. This ensures that by the final broadcast round in Step 4, each party's secrets are available in publicly committed form. In particular, each party's share of k, sk is available directly in the exponent due to the respective DKGs, and their shares of ϕ and zero are available as Pedersen commitments via VSS. In the final broadcast round, each party attaches a NIZK to its shares of u and w, proving that they have been derived correctly relative to their committed secrets. These NIZKs are quite efficient, following from standard Schnorr-like sigma protocols.

Protocol Overview. Assuming that DKG, VSS, ZSS are available, ECDSA signing is a single additional broadcast round. We describe the protocol below.

- **Setup.** Parties execute DKG to derive a public degree t polynomial $F_{\mathsf{pk}} \in \mathbb{G}[X]$, such that each $\mathcal{P}_i$ holds sk_i where $\mathsf{sk}_i \cdot G = F_{\mathsf{pk}}(i)$. The public key is defined to be $\mathsf{pk} = F_{\mathsf{pk}}(0)$. The rest of the protocol below pertains to signing.

- **Broadcast Rounds 1 and 2.** Parties execute DKG, VSS, and two ZSS instances in parallel to derive the following values:

  – *Public:* Degree t polynomials $F_R, C^{\phi} \in \mathbb{G}[X]$, and degree $2t$ polynomials $Z_0, Z_1 \in \mathbb{G}[X]$ where $Z_0(0) = Z_1(0) = 0$. The signing nonce is determined as $R = F_R(0)$.

– *Private:* Each party $\mathcal{P}_i$ holds $k_i, f^\phi(i), \hat{f}^\phi(i), z_0(i), \hat{z_0}(i), z_1(i), \hat{z_1}(i)$ from $\mathbb{Z}_q$ such that,

$$\begin{aligned} k_i \cdot G &= F_R(i) & z_0(i) \cdot G + \hat{z_0}(i) \cdot \hat{G} &= Z_0(i) \\ f^\phi(i) \cdot G + \hat{f}^\phi(i) \cdot \hat{G} &= C^\phi(i) & z_1(i) \cdot G + \hat{z_1}(i) \cdot \hat{G} &= Z_1(i) \end{aligned}$$

Given these values, each $\mathcal{P}_i$ locally derives its share of the ECDSA tuple:

$$\big(k_i,\ \mathsf{sk}_i,\ f^\phi(i),\ u_i = f^\phi(i) \cdot k_i + z_1(i),\ v_i = f^\phi(i) \cdot \mathsf{sk}_i + z_0(i)\big)$$

following which $\mathcal{P}_i$'s share of the signature is set to u_i and $w_i = f^\phi(i) \cdot \mathsf{SHA2}(m) + r^\times \cdot v_i$.

Notice that each of $\mathcal{P}_i$'s private values are available in publicly committed form—either directly in the exponent as in $\mathsf{pk}_i, F_R(i)$, or as Pedersen commitments $C^\phi(i), Z_0(i), Z_1(i)$—and the signature shares w_i, u_i are deterministic functions of these private values. Therefore, $\mathcal{P}_i$ additionally computes NIZKs π_i^w, π_i^u which prove that w_i, u_i respectively are correctly derived relative to the public committed inputs. Intuitively, the relations proven by the NIZK are that the product of the openings to two Pedersen commitments is contained in a third, eg. the product of $f^\phi(i)$ and k_i (committed in $C^\phi(i)$ and $F_R(i)$ respectively) is committed in $u_i \cdot G - Z_1(i)$. We defer specifics to Protocol 8.2.

- **Broadcast Round 3.** Each $\mathcal{P}_i$ broadcasts two sets of values:
  – $D = \mathsf{CRHF}(F_{\mathsf{pk}}, F_R, C^\phi, Z_0, Z_1)$ to confirm the output of the DKG and VSS.
  – $w_i, u_i, \pi_i^w, \pi_i^u$ to complete the ECDSA signature.

  Upon receiving every $w_j, u_j, \pi_j^w, \pi_j^u$ value from all parties, each $\mathcal{P}_i$ first checks that all the NIZKs verify. If a single NIZK—sent by $\mathcal{P}_j$—fails verification, $\mathcal{P}_i$ assembles a certificate Ω that includes the following values:
  – (F_R, C^ϕ, Z_0, Z_1) to establish the context of the protocol.
  – $t+1$ signatures on $D = \mathsf{CRHF}(F_{\mathsf{pk}}, F_R, C^\phi, Z_0, Z_1)$ as received via broadcast this round, to establish agreement on the above protocol context.
  – $(w_j, u_j, \pi_j^w, \pi_j^u)$ along with $\mathcal{P}_j$'s signature as broadcast in this round.

  If all NIZKs pass verification, each $\mathcal{P}_i$ assembles the signature

$$\sigma = \left(R, s = \Big(\sum_{j \in [n]} \lambda_j \cdot w_j\Big) / \Big(\sum_{j \in [n]} \lambda_j \cdot u_j\Big)\right)$$

and outputs it, where $\{\lambda_j\}_{j \in [n]}$ is the set of Lagrange coefficients that interpolate the y-intercept of a degree $n-1$ polynomial.

Analysis. Assuming that VSS, DKG, and zero sharing are secure, we only need analyze the outcomes of the final broadcast round. We enumerate all possible cases below:

1. Some honest party $\mathcal{P}_i$ does not complete VSS/DKG. In this event, $\mathcal{P}_i$ must have obtained a certificate of cheating to implicate a corrupt $\mathcal{P}_j$, which it then forwards to all parties (who then echo and output it).

2. All honest parties agree upon $(F_{\mathsf{pk}}, F_R, C^\phi, Z_0, Z_1)$, and get at least $t+1$ signatures on their hashed digest D. At this point, all parties' inputs for the final round are fixed, and the values they are expected to broadcast are a simple deterministic function of these inputs (randomness for NIZKs notwithstanding). Assuming that each party $\mathcal{P}_i$ broadcasts *some* value $(w_i, u_i, \pi_i^w, \pi_i^u)$, there are two possible outcomes in each party's view:

   - *All NIZKs verify.* In this case, we argue that each (w_i, u_i) value is correctly derived. Recall that the NIZKs prove Pedersen commitment relations, in particular that the product of the first two is committed in the third. Each (corrupt) $\mathcal{P}_i$'s witness for the NIZK is completely specified by its DKG and VSS outputs—ignoring linear offsets, w_i is derived from the product of $f^\phi(i), \mathsf{sk}_i$, and u_i from the product of $f^\phi(i), k_i$. Producing accepting NIZKs for some $(w_i^*, u_i^*) \neq (w_i, u_i)$ entails using an alternative witness, i.e. an alternative opening to the same Pedersen commitments $C^\phi(i), Z_0(i), Z_1(i)$ generated by VSS. Invoking the extractor for the NIZK in this event therefore yields an alternative opening to the same Pedersen commitment (the original one being from VSS), which directly yields a reduction to computing the discrete logarithm $\mathsf{DLog}_{\hat{G}} G$.

     Therefore in this case, by correctness of the ECDSA tuple, it holds that a valid ECDSA signature $(R, s = (\sum_i \lambda_i w_i)/(\sum_i \lambda_i u_i))$ is obtained.

   - *Some NIZK π_i^w, π_i^u fails verification.* In this case, a certificate consisting of $F_{\mathsf{pk}}, F_R, C^\phi, Z_0, Z_1$ (along with $t+1$ signatures on their hashed digest D), and signed π_i^w, π_i^u themselves, is assembled to implicate $\mathcal{P}_i$ to an external auditor.

     This information is sufficient to run the verification algorithm and see that at least one of the NIZKs does not verify. Moreover, the $t+1$ signatures on the digest D (which must be verified by recomputing CRHF) validates that $F_{\mathsf{pk}}, F_R, C^\phi, Z_0, Z_1$ were indeed the output of their respective DKG/VSS instances, and that no honest party is in disagreement about the *statement* of the NIZK.

Therefore in each case, every honest party either obtains a valid ECDSA signature on the message, or a certificate implicating a corrupt party.

Protocol 8.2. $\pi_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$**: Honest Majority ECDSA With IA**

This protocol is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The setup phase runs once with parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, and the signing phase may be run many times between (varying) subsets of parties of size $2t + 1$. The protocol makes use of the ideal functionalities $\mathcal{F}_{\mathsf{BC}}$, $\mathcal{F}_{\mathsf{VSS}}$, $\mathcal{F}_{\mathsf{DKG}}$ as well as the simulation extractable NIZK proof system $(P_{\mathsf{prod}}, V_{\mathsf{prod}})$ to prove products of Pedersen-committed values. Let $\{\lambda_i\}_{i\in[n]}$ be the set of Lagrange coefficients required to interpolate the constant term of a degree $n - 1 = 2t$ polynomial, i.e. $\sum_{i\in[n]} \lambda_i \cdot f(i) = f(0)$ for any polynomial $f \in \mathbb{Z}_q^{\leq n}[X]$.

Setup:

1. On receiving $(\texttt{init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party $\mathcal{P}_i$ checks whether there exists a record of the form $(\texttt{DKG-done}, F_{\mathsf{pk}}, \mathsf{sk}_i)$ in memory. If not, then each $\mathcal{P}_i$ for $i \in [t + 1]$ invokes $\mathcal{F}_{\mathsf{DKG}}$ in order to generate $(\texttt{DKG-done}, F_{\mathsf{pk}}, \mathsf{sk}_i)$.

   In case $\mathcal{P}_i$ observes the DKG to fail, it transfers the appropriate certificate.

Signing:

1. Upon receiving the instruction to sign a message m with fresh signature ID sigid, each $\mathcal{P}_i$ does the following in parallel:
   a. Invoke $\mathcal{F}_{\mathsf{DKG}}(\mathcal{G}, n, t)$ to generate $(\texttt{DKG-done}, F_R, k_i)$
   b. Invoke $\mathcal{F}_{\mathsf{VSS}}(\mathcal{G}, n, t)$ to generate $(\texttt{VSS-success}, C^\phi, f^\phi(i), \hat{f}^\phi(i))$
   c. Invoke $\mathcal{F}_{\mathsf{ZSS}}(\mathcal{G}, n, 2t)$ twice to generate $(\texttt{ZSS-success}, Z_0, z_0(i), \hat{z_0}(i))$ and $(\texttt{ZSS-success}, Z_1, z_1(i), \hat{z_1}(i))$.

   This requires two broadcast rounds (when π_{DKG} is used to realize $\mathcal{F}_{\mathsf{DKG}}$).
2. If any of the VSS, ZSS, or DKG invoked above terminated with a cheat $(\mathsf{sid}, \texttt{cert} = (\texttt{type}, \mathsf{c}))$, then $\mathcal{P}_i$ terminates with output $\mathbf{sigout}_i = (\texttt{type} \in \{\texttt{cheat}, \perp\}, j)$ as appropriate, and transfers this output to all other parties.
3. Otherwise, each $\mathcal{P}_i$ does the following:
   a. Compute $R = F_R(0)$, and parse $(r^{\mathsf{x}}, r^{\mathsf{y}}) = R$
   b. Prepare signature shares

$$w_i = f^\phi(i) \cdot \mathsf{SHA2}(m) + f^\phi(i) \cdot \mathsf{sk}_i \cdot r^{\mathsf{x}} + z_0(i)$$

and

$$u_i = f^{\phi}(i) \cdot k_i + z_1(i)$$

c. Derive the Pedersen commitment randomness for the publicly committed versions of the above values,

$$\rho_{i,0} = -\frac{\left(\mathsf{SHA2}(m) \cdot \hat{f}^{\phi}(i) + \hat{z_0}(i)\right)}{r^{\mathsf{x}}} \quad \text{and} \quad \rho_{i,1} = -\hat{z_1}(i)$$

Prepare proofs that the above values were computed honestly:

$$\pi_i^w \leftarrow P_{\mathsf{prod}}^{G,\hat{G}}((f^{\phi}(i), \hat{f}^{\phi}(i)), (\mathsf{sk}_i, 0), (f^{\phi}(i) \cdot \mathsf{sk}_i, \rho_{i,0}))$$
$$\pi_i^u \leftarrow P_{\mathsf{prod}}^{G,\hat{G}}((f^{\phi}(i), \hat{f}^{\phi}(i)), (k_i, 0), (k_i \cdot f^{\phi}(i), \rho_{i,1}))$$

d. Compute digest $D = \mathsf{CRHF}(F_{\mathsf{pk}}, F_R, Z_0, Z_1, C^{\phi})$

e. Invoke $\mathcal{F}_{\mathsf{BC-IA}}(n, t, i)$ with $(\mathtt{deal}, \mathsf{sid}, (w_i, u_i, \pi_i^w, \pi_i^u))$.

f. Invoke $\mathcal{F}_{\mathsf{BC-IA}}(n, t, i)$ with $(\mathtt{deal}, \mathsf{sid}, D)$.

4. If any party transferred a cheat $(\mathsf{sid}, \mathtt{cert} = (\mathtt{type}, \mathsf{c}))$ in place of a standard broadcast in the above round, terminate with output $\textbf{sigout}_i = (\mathsf{type} \in \{\mathtt{cheat}, \perp\}, \mathsf{c})$ as appropriate. Otherwise, upon completion of the broadcast round, if $\exists j \in [t+1]$ such that the $\mathcal{F}_{\mathsf{BC}}$ instance in which $\mathcal{P}_j$ was the dealer resulted in an abort of $\mathsf{type} \in \{\mathtt{cheat}, \perp\}$, then $\mathcal{P}_i$ terminates with output $\textbf{sigout}_i = (\mathsf{type}, j)$, and invokes the Transfer interface of $\mathcal{F}_{\mathsf{BC}}$ when invoked with this sid again.

5. Otherwise, upon termination of the broadcast phase, each $\mathcal{P}_i$ does the following:

a. Parse $t+1$ signatures on R from the previous broadcast round, store this collection as ϖ_{BC}.

Any party $\mathcal{P}_j$ that broadcast a conflicting $D^* \neq D$ can be identified as a cheater with the following certificate:

$$\Omega_i^j = (\mathtt{bad\text{-}context}, \varpi_{\mathsf{BC}}, D^*)$$

where σ_j^{PKI} is the signature that accompanied the broadcast value D^* from $\mathcal{P}_j$.

b. For each $j \in [n]$, check $\mathcal{P}_j$'s proof by defining the statements

$$\mathsf{stmt}_0^j = \left(w_j \cdot G - \left(Z_0(j) + \mathsf{SHA2}(m) \cdot C^{\phi}(j)\right)\right) / r^{\mathsf{x}}$$

and

$$\mathsf{stmt}_1^j = u_j \cdot G - Z_1(j)$$

and validating the proofs π_j^w, π_j^u against them by checking

$$V_{\mathsf{prod}}^{G,\hat{G}}(C^\phi(j), \mathsf{pk}_j, \mathsf{stmt}_0^j, \pi_j^w) = V_{\mathsf{prod}}^{G,\hat{G}}(C^\phi(j), R_j, \mathsf{stmt}_1^j, \pi_j^u) = 1$$

A non-verifying proof yields the following certificate of cheating:

$$\Omega_i^j = \left(\texttt{bad-tuple}, j, (F_{\mathsf{pk}}, F_R, Z_0, Z_1, C^\phi, \varpi_{\mathsf{BC}}), (w_j, u_j, \pi_j^w, \pi_j^u, \sigma_j^{\mathsf{PKI}})\right)$$

$$\Omega_i^j = \left(\texttt{bad-tuple}, j, (F_{\mathsf{pk}}, F_R, Z_0, Z_1, C^\phi), (w_j, u_j, \pi_j^w, \pi_j^u, \sigma_j^{\mathsf{PKI}})\right)$$

$\mathcal{P}_i$ terminates with output $\textbf{sigout}_i = (\texttt{cheat}, j)$. In order to transfer this cheat certificate, $\mathcal{P}_i$ sends Ω_i^j along with invoking the transfer interface of $t+1$ instances of $\mathcal{F}_{\mathsf{BC-IA}}$ from the previous round in which the output $\textbf{out}_i$ contains the same D.

c. In case all checks pass, compute

$$s = \frac{\sum_{i\in[n]} \lambda_i \cdot w_i}{\sum_{i\in[n]} \lambda_i \cdot u_i}$$

and output the ECDSA signature (r^{x}, s)

The formal theorem appears below, whose proof is deferred to the full version.

Theorem 8.3. *In the* $(\mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{ZSS}}, \mathcal{F}_{\mathsf{DKG}})$*-hybrid model,* $\pi_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *UC-realizes* $\mathcal{F}_{\mathsf{ECDSA}}(\mathcal{G}, n, t)$ *against a malicious adversary that corrupts up to* $t < n/2$ *parties.*

The above protocol requires three broadcast rounds, which when instantiated with π_{BC} yields a protocol that requires six point-to-point rounds in total.

9 Performance

We give an accounting of the costs associated with a non-aborting instance of our protocol below. We give both asymptotic costs, as well as concrete costs for the commonly used 128-bit security level, i.e. for a 256-bit elliptic curve.

- **Broadcast** requires two p2p rounds, and broadcasting a message m induces $O(|m| + \lambda)$ bits of communication between each pair of parties (λ to account for the dealer's signature).
- **VSS** requires a single broadcast round (meaning two p2p rounds), in which $t+1$ parties broadcast n ciphertexts and a degree t polynomial each. This induces $O(tn\lambda)$ communication between each pair of parties. Concretely, each party sends $48n^2 + 32t + 64$ bytes to every party for a 256-bit curve

with ElGamal encryption. Computation for each party is dominated by interpolating the t different $\mathbb{G}[X]$ polynomials when checking consistency of their decrypted share with the corresponding public value—nt curve multiplications in total.

- **DKG** requires a VSS instance, and another broadcast (four p2p rounds total) of a hash digest, an elliptic curve point, and a NIZK proving knowledge of two discrete logarithms. The communication complexity in addition to the VSS is $O(n\lambda)$ between each pair of parties, which is entirely subsumed by VSS. Concretely, each party sends $48n^2 + 32t + 192$ bytes to every party for a 256-bit curve. Computation in addition to VSS is dominated by checking t NIZKs that each prove knowledge of two discrete logarithms in the curve—equivalent to verifying $2t$ signatures, subsumed by VSS overall.
- **ECDSA setup** is equivalent to a single DKG instance.
- **ECDSA signing** invokes three VSS instances and a single DKG, and additionally has each party broadcast signature shares (two $\mathbb{Z}_q$ values) with accompanying NIZKs. The dominating communication complexity term is that of the ($O(1)$ number of) VSS instances, as the signature shares and NIZKs only requires $O(n\lambda)$ bits between each party. Concretely, each party sends $192n^2 + 128n + 960$ bytes to every party for a 256-bit curve. Computation, in addition to the VSS and DKG, is dominated by verifying the $2n$ NIZKs that each prove product relations of Pedersen commitments—equivalent to verifying $18n$ signatures.

Note that an aborting instance costs *strictly less* than a successfully terminating instance. This is in contrast to alternative approaches like Canetti et al. [CGG+20] whose protocol has a dedicated cheater identification phase in case of a cheat.

Empirical Wallclock Time. We implemented our protocol in Rust, and benchmarked it on commodity hardware (Intel i7 14700, 32GB RAM). We averaged our numbers over 100 samples. We plot the results in Figure 1. To give a point of comparison, we benchmarked the state of the art dishonest majority (non-IA) protocol of Doerner et al. [DKLs24] in the same environment. We also include the running times of the only other threshold ECDSA protocol that offers identifiable abort, that of Canetti et al. [CGG+20] (numbers taken from Haitner et al. [HLNR23], which does not include the cost of the extra identification phase or broadcast channels). We varied the corruption threshold t from 1 to 10 for our comparison, meaning that the number n of signers in our protocol is about twice as many as that of Doerner et al. and Canetti et al. for the same threshold.

Note that this comparison is for the computation time alone, and does not include network costs. In the optimistic execution path where no party cheats, the fact that our protocol runs in six (p2p) rounds means that network latency

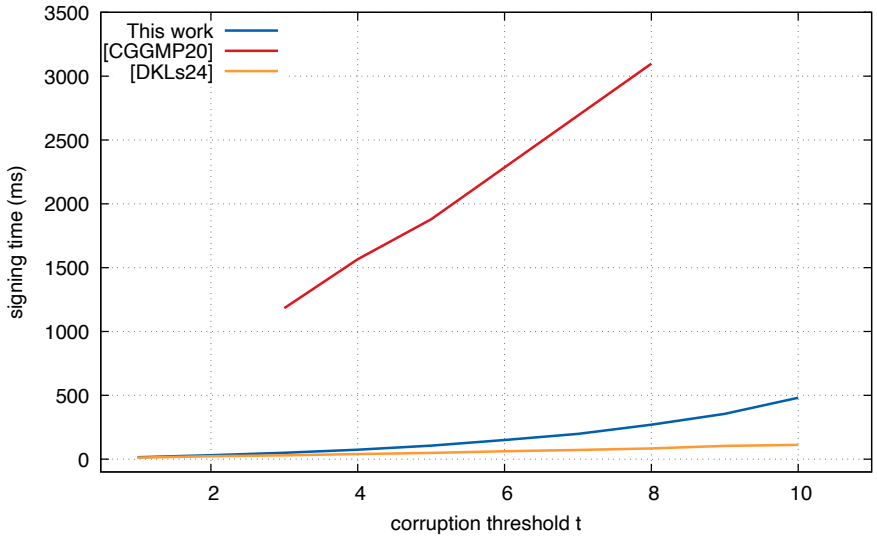

Figure 1: Comparison of signing times for state of the art ECDSA signing protocols, DKLs24 [DKLs24] and CGGMP20 [CGG+20]

will likely dictate wallclock time—compare this with only three p2p rounds for Doerner et al., or four *broadcast* rounds for Canetti et al. However, given that our protocol is meant to be used in scenarios where protocol deviations and failures are rampant, it is more interesting to compare the pessimistic paths. The worst slowdown an adversary can inflict on our protocol is to force each round to take the maximal possible time just under the timeout threshold. Therefore if the network timeout is set to NTO, then our protocol terminates in time roughly $6\mathsf{NTO}$ in the worst case. The protocol of Canetti et al. terminates in time $4\times$ the worst case timeout complexity of the underlying broadcast channel, which must also account for the fact that *honest* signing with their protocol can take several seconds for computation alone. The protocol of Doerner et al. does not achieve identifiability, and can be induced to terminate without output if a single party deviates from its instructions.

10 Acknowledgements

The authors would like to thank Vladyslav Khomenko for implementing and benchmarking the protocol, and for his feedback on drafts of the paper.

References

[ADD+19] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $O(1)$

rounds, expected $O(n^2)$ communication, and optimal resilience. In *FC 2019*, February 2019.

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, (2), April 2010.

[ANO+22] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. 2022.

[AO12] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In *ASIACRYPT 2012*, December 2012.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In *SCN 14*, September 2014.

[BMRS24] Carsten Baum, Nikolas Melissaris, Rahul Rachuri, and Peter Scholl. Cheater identification on a budget: MPC with identifiable abort from pairwise macs. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VIII*, volume 14927 of *Lecture Notes in Computer Science*, pages 454–488. Springer, 2024.

[BOS16] Carsten Baum, Emmanuela Orsini, and Peter Scholl. Efficient secure multiparty computation with identifiable abort. In *TCC 2016-B, Part I*, 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 *CRYPTO 2020, Part II*, August 2020.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, October 2001.

[CCGZ19] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. *Journal of Cryptology*, (3), July 2019.

[CCL+23] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts proactive and adaptive security. *Theor. Comput. Sci.*, 939:78–104, 2023.

[CDKs24] Ran Cohen, Jack Doerner, Yashvanth Kondi, and abhi shelat. Secure multiparty computation with identifiable abort via vindicating release. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VIII*, volume 14927 of *Lecture Notes in Computer Science*, pages 36–73. Springer, 2024.

[CFY17] Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. Catching MPC cheaters: Identification and openability. In *ICITS 17*, November / December 2017.

[CGG+20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In *ACM CCS 2020*, November 2020.

[CGZ20] Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In *EUROCRYPT 2020, Part II*, May 2020.

[CL17] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, (4), October 2017.

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

[CRSW22] Michele Ciampi, Divya Ravi, Luisa Siniscalchi, and Hendrik Waldner. Round-optimal multi-party computation with identifiable abort. In *EUROCRYPT 2022, Part I*, May / June 2022.

[DJN+20] Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. In *SCN 20*, September 2020.

[DKLs24] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ecdsa in three rounds. In *45th IEEE Symposium on Security and Privacy, SP 2024*. IEEE, 2024.

[DRSY23] Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Minimizing setup in broadcast-optimal two round MPC. In *EUROCRYPT*, volume 14005 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2023.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th FOCS*, October 1987.

[FGH+02] Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein, and Adam Smith. Detectable byzantine agreement secure against faulty majorities. In *21st ACM PODC*, July 2002.

[FGMv02] Matthias Fitzi, Nicolas Gisin, Ueli M. Maurer, and Oliver von Rotz. Unconditional byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In *EUROCRYPT 2002*, April / May 2002.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO 2005*, August 2005.

[FL82] Michael J Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, (1), January 2007.

[GKM+22] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 252–282. Springer, 2022.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, (3), July 2005.

[GMPS21] Vipul Goyal, Elisaweta Masserova, Bryan Parno, and Yifan Song. Blockchains enable non-interactive MPC. In *TCC 2021, Part II*, November 2021.

[GMW87a] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, May 1987.

[GMW87b] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO'86*, August 1987.

[GS22] Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Report 2022/506, 2022. https://eprint.iacr.org/2022/506.

[HLNR23] Iftach Haitner, Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. *IACR Cryptol. ePrint Arch.*, 2023. https://eprint.iacr.org/2018/987.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO 2014, Part II*, August 2014.

[KK06] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *CRYPTO 2006*, August 2006.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *EUROCRYPT 2016, Part II*, May 2016.

[Lin22] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Paper 2022/374, 2022. Paper has since been updated to remove section on Identifiable Abort. See https://eprint.iacr.org/archive/2022/374/20220322:132756.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In *ACM CCS 2018*, October 2018.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[Ped91] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In *EUROCRYPT'91*, April 1991.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO'91*, August 1992.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[Sho23] Victor Shoup. The many faces of schnorr. *IACR Cryptol. ePrint Arch.*, page 1019, 2023.

[SV15] Berry Schoenmakers and Meilof Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In *ACNS 15*, June 2015.

[ZYP23] Guy Zyskind, Avishay Yanai, and Alex 'Sandy' Pentland. Unstoppable wallets: Chain-assisted threshold ECDSA and its applications. *IACR Cryptol. ePrint Arch.*, page 832, 2023.

A Verifiable Decryption

We detail an El-Gamal based encryption scheme that allows for verifiably decryptable ciphertexts. Besides the standard key generation, encryption, and decryption algorithms, a verifiably decryptable scheme also consists of an "open" algorithm that outputs a proof of correct decryption, and a verification algorithm to check such proofs.

Protocol A.1. : Verifiably Decryptable Encryption Scheme

These protocols are parameterized by a security parameter $\lambda$and an elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$ such that $q \in \Omega(2^\lambda)$. The message space is $\mathbb{Z}_q^2$, i.e. a pair of $\mathbb{Z}_q$ elements. The protocols make use of a simulation extractable NIZK proof system $(P_{\mathsf{prod}}, V_{\mathsf{prod}})$ to prove that a Diffie-Hellman tuple is well-formed. Additionally, they make use of a hash function $H_q : \{0,1\}^* \mapsto \mathbb{Z}_q$ that is assumed to implement a random oracle.

$\mathsf{KeyGen}(1^\lambda)$**:** .

1. Sample $\mathsf{sk} \leftarrow \mathbb{Z}_q$, compute $\mathsf{pk} = \mathsf{sk} \cdot G$.
2. Output $(\mathsf{sk}, \mathsf{pk})$.

$\mathsf{Enc}_{\mathsf{pk}}(\mathbf{m}_0, \mathbf{m}_1)$**:** .

1. Sample $r \leftarrow \mathbb{Z}_q$, compute $R = r \cdot G$.
2. Compute $\mathsf{K} = r \cdot \mathsf{pk}$ and pads $\mathbf{k}_0 = H_q(0||\mathsf{K})$ and $\mathbf{k}_1 = H_q(1||\mathsf{K})$.
3. Set ciphertexts $\mathbf{ct}_0 = \mathbf{k}_0 + \mathbf{m}_0$ and $\mathbf{ct}_1 = \mathbf{k}_1 + \mathbf{m}_1$.
4. Output $(R, \mathbf{ct}_0, \mathbf{ct}_1)$

$\mathsf{Dec}_{\mathsf{sk}}(\mathbf{ct})$**:** .

1. Parse $(R, \mathbf{ct}_0, \mathbf{ct}_1) := \mathbf{ct}$
2. Compute $\mathsf{K} = \mathsf{sk} \cdot R$ and pads $\mathbf{k}_0 = H_q(0||\mathsf{K})$ and $\mathbf{k}_1 = H_q(1||\mathsf{K})$.
3. Set messages $\mathbf{m}_0 = \mathbf{ct}_0 - \mathbf{k}_0$ and $\mathbf{m}_1 = \mathbf{ct}_1 - \mathbf{k}_1$.
4. Output $(\mathbf{m}_0, \mathbf{m}_1)$

$\mathsf{Open}(\mathsf{sk}, \mathbf{ct})$**:** .

1. Parse $(R, \mathbf{ct}_0, \mathbf{ct}_1) := \mathbf{ct}$
2. Compute $\mathsf{K} = \mathsf{sk} \cdot R$ and pads $\mathbf{k}_0 = H_q(0||\mathsf{K})$ and $\mathbf{k}_1 = H_q(1||\mathsf{K})$.
3. Set messages $\mathbf{m}_0 = \mathbf{ct}_0 - \mathbf{k}_0$ and $\mathbf{m}_1 = \mathbf{ct}_1 - \mathbf{k}_1$.

4. Prove that $(R, \mathsf{pk}, \mathsf{K})$ is a DH tuple: $\pi_{\mathsf{DH}} \leftarrow P_{\mathsf{DH}}(\mathsf{sk}, (R, \mathsf{pk}, \mathsf{K}))$
5. Set opening information $\pi_{\mathsf{ct}} = (\mathsf{K}, \pi_{\mathsf{DH}})$
6. Output $(\mathbf{m}_0, \mathbf{m}_1, \pi_{\mathsf{ct}})$

$\mathsf{Vrfy}(\mathbf{ct}, \mathbf{m}_0, \mathbf{m}_1, \pi_{\mathsf{ct}})$**:** .

1. Parse $(R, \mathbf{ct}_0, \mathbf{ct}_1) := \mathbf{ct}$ and $(\mathsf{K}, \pi_{\mathsf{DH}}) := \pi_{\mathsf{ct}}$
2. Set pads $\mathbf{k}_0 = H_q(0||\mathsf{K})$ and $\mathbf{k}_1 = H_q(1||\mathsf{K})$.
3. Verify that $\mathbf{m}_0 \stackrel{?}{=} \mathbf{ct}_0 - \mathbf{k}_0$ and $\mathbf{m}_1 \stackrel{?}{=} \mathbf{ct}_1 - \mathbf{k}_1$.
4. Verify that $(R, \mathsf{pk}, \mathsf{K})$ is a DH tuple: $V_{\mathsf{DH}}(\pi_{\mathsf{DH}}, (R, \mathsf{pk}, \mathsf{K})) \stackrel{?}{=} 1$
5. Output 1 if all the above checks pass, 0 otherwise.

B Zero Sharing

We provide the zero-sharing protocol π_{Zero} for completeness.

Protocol B.1. $\pi_{\mathsf{Zero}}(\mathcal{G}, n, t)$**: Honest Majority Zero-sharing With IA**

This protocol is parameterized by the party count n, the threshold t, and the elliptic curve $\mathcal{G} = (\mathbb{G}, G, q)$. The protocol runs once with parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, of which any t may be corrupt. The private output of this protocol for $\mathcal{P}_i$ is $(C, f(i), \hat{f}(i))$ where $C \in \mathbb{G}[X]$ is a common degree-t polynomial, and $f(i) \cdot G + \hat{f}(i) \cdot \hat{G} = C(i)$. This protocol assumes a PKI and a broadcast protocol π_{BC}, and makes use of an openable encryption scheme $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Open}, \mathsf{Vrfy})$.

Differences from π_{VSS} are highlighted like this.

Share:

1. On receiving $(\mathtt{init}, \mathsf{sid})$ from the environment $\mathcal{Z}$, each party $\mathcal{P}_i$ checks whether there exists a record of the form $(\mathsf{sid}, \cdot)$ in memory. If not, then each $\mathcal{P}_i$ for $i \in [t+1]$ does the following:
   a. Sample two degree t polynomials $f_i, \hat{f}_i \leftarrow \mathbb{Z}_q[X]$, conditioned on $f_i(0) = \hat{f}_i(0) = 0$.
   b. Define polynomial $C_i \in \mathbb{G}[X]$ such that $C_i(x) = f_i(x) \cdot G + \hat{f}_i(x) \cdot \hat{G}$
   c. Compute encrypted shares:
   $$\mathbf{ct}_i = \{\mathbf{ct}_{ij} \leftarrow \mathsf{Enc}_{\mathsf{pk}_j^{\mathsf{PKI}}}(f_i(j), \hat{f}_i(j))\}_{j \in [n]}$$
   d. **Broadcast** $(C_i, \mathbf{ct}_i)$

This completes the first phase. Note that for every successfully terminated broadcast instance $j \in [t+1]$, party $\mathcal{P}_i$ has a signed output $(\mathsf{sid}, \texttt{dealt}, (C_j, \mathbf{ct}_j), \sigma_j^{\mathsf{PKI}})$.

2. Upon completion of the broadcast round, if $\exists j \in [t+1]$ such that π_{BC} failed to produce output when $\mathcal{P}_j$ was the dealer, then $\mathcal{P}_i$ terminates here and sends relevant failure certificate $(\mathbf{out}_i, \sigma_i^{\mathsf{PKI-o}})$ when activated with this sid again.

3. Each party $\mathcal{P}_i$ does the following for $j \in [t+1]$:

   a. Obtain $f_j(i), \hat{f}_j(i) = \mathsf{Dec}_{\mathsf{sk}_i}(\mathbf{ct}_{ji})$

   b. Verify that $f_j(i) \cdot G + \hat{f}_j(i) \cdot \hat{G} = C_j(i)$

   - If this fails, open the ciphertext by computing

   $$\zeta_{ji} = (f_j(i), \hat{f}_j(i), \pi_{\mathsf{ct}}) \leftarrow \mathsf{Open}(\mathsf{sk}_i, \mathbf{ct}_{ji})$$

   and set

   $$\Omega_i^j = (\texttt{bad-ct}, \zeta_{ji}, (\mathsf{sid}, \texttt{dealt}, (C_j, \mathbf{ct}_j), \sigma_j^{\mathsf{PKI}}))$$

   c. Verify that C_j is a degree-t polynomial, and that $C_j(0) = 0$. If this verification fails, set

   $$\Omega_i^j = (\texttt{bad-poly}, (\mathsf{sid}, \texttt{dealt}, (C_j, \mathbf{ct}_j), \sigma_j^{\mathsf{PKI}}))$$

If Ω_i^j is defined for some $j \in [t+1]$, then output $(\texttt{cheat-detected}, \Omega_i^j)$ and echo it to all parties. Otherwise, output

$$(\texttt{success}, C = \sum_{i \in [t+1]} C_i, f(i) = \sum_{j \in [t+1]} f_j(i), \hat{f}(i) = \sum_{j \in [t+1]} \hat{f}_j(i))$$

.

C Supplementary material for PISA MPC

C.1 Ideal Functionality for IA

For the sake of completeness, we recap the ideal functionality for IA below.

Functionality C.1.

SFE with Identifiable Abort $\mathcal{F}_{\mathsf{IA}}$. This functionality interacts with n actively participating parties denoted by $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and with an ideal adversary $\mathcal{S}$. It is also parameterized by a function $f : \mathbb{X}_1 \times \mathbb{X}_2 \cdots \times \mathbb{X}_n \to \mathbb{Y}$.

SFE: On receiving $(\texttt{compute}, \mathsf{sid}, x_i)$, where $x_i \in \mathbb{X}_i$ from every party $\mathcal{P}_i$

for $i \in [n]$,

1. Compute $y := f(\{x_i\}_{i\in[n]})$
2. Send $(\texttt{candidate-output}, \mathsf{sid}, y)$ to $\mathcal{S}$, and receive $(\texttt{stooge}, \mathsf{sid}, c)$ in response.
3. If c is the index of a corrupt party, then send $(\texttt{abort}, \mathsf{sid}, c)$ to all parties. Otherwise, send $(\texttt{output}, \mathsf{sid}, y)$ to all parties.

C.2 Compiler for dishonest majority

In the compiler in Section 5.2, we assumed that the same set of parties participate in the starting protocol Π_{IA} as well as the compiled protocol Π_{PISA}. This necessarily requires an honest majority, since $\mathcal{F}_{\mathsf{PISA}}$ can be realized only with an honest majority. However, one could consider a setting where Π_{IA} is run among a dishonest majority of participants, say $\mathcal{M} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$, where $m \geq t+1$; while an extended set of participants $\mathcal{N} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, where $n \geq 2t+1$ are registered with $\mathcal{F}_{\mathsf{BC-IA}}$ (where $\mathcal{M} \subset \mathcal{N}$).

Our compiler would not work in such a setting as it relies on there being an honest majority of participants in Π_{IA} crucially in the last step, where a party that is announced to be a cheater by $t+1$ parties executing Π_{IA} is implicated. In a setting where there is a dishonest majority of participants in Π_{IA}, we propose that the starting protocol Π_{IA} must have a stronger guarantee, namely its aborts must be *publicly-verifiable* [BDO14, SV15, BOSS20]. Publicly-verifiable IA refers to the notion where as in IA, the honest parties agree on the identity of at least one corrupted cheater; furthermore any external party (say, an auditor) can verify the correctness of computation or identify the cheating party, typically by looking at the protocol transcript on a public bulletin board.

Below, we sketch a compiler that transforms a publicly-verifiable IA protocol Π_{PVIA} among the parties in $\mathcal{M}$ in the $\mathcal{F}_{\mathsf{BC}}$-hybrid model to Π'_{PISA} among the parties in $\mathcal{N}$ in the $\mathcal{F}_{\mathsf{BC-IA}}$-hybrid model. Similar to our Protocol 5.2, the BC invocations in Π_{PVIA} are replaced with a BC-IA invocations in Π'_{PISA}. If a party detects a cheat during a BC-IA invocation, this cheat is transferred. Additionally the following steps are executed.

1. If a party in $\mathcal{N}$ obtains the dealer's signed message as the output during a BC-IA invocation, it initiates another BC-IA invocation as a dealer to echo this message. If the echo BC-IA invocations of at least $t+1$ parties in $\mathcal{N}$ attest to the same message m, then we certify m as being broadcast. Intuitively, this is done to emulate the effect of m being included in the "public transcript" of Π_{PVIA}.
2. An honest party in $\mathcal{M}$ who has not detected a cheat collects the outputs of these echo BC-IA invocations.

If none of the BC-IA invocations abort, then output computation can be carried out by parties in $\mathcal{M}$ identical to Π_{PVIA}. If this results in honest par-

ties obtaining the output y, they simply output y as output of Π'_{PISA} . Else, the honest parties must have identified a common cheater, say $\mathcal{P}_{\mathsf{c}}$. Note that although honest parties are in minority in $\mathcal{M}$, they can convince an auditor of the validity of the protocol transcript due to the following: For each message of Π_{PVIA}, an honest party in $\mathcal{M}$ can use its collection of the outputs obtained during the echo BC-IA invocations to show that this message has been attested by at least $(t+1)$ parties in $\mathcal{N}$. This must hold due to the validity of the echo BC-IA invocations by the $(t+1)$ honest parties in $\mathcal{N}$.

Therefore $\mathcal{P}_i$ can output $(\mathtt{cheat}, \mathsf{c})$ as output of Π'_{PISA} , with the corresponding certificate to be transferred as the above collection for each message of Π_{PVIA}. Note that this constitutes a valid proof of cheating as **(a)** the honest majority in $\mathcal{N}$ ensures that the 'public' transcript cannot be forged, as each broadcast message needs to be attested by at least $(t+1)$ participants of $\mathcal{N}$ and **(b)** the public verifiability of Π_{PVIA} guarantees that an auditor can detect the cheater by inspecting the public transcript. This completes the description of our modified compiler.