Collusion-Free Multiparty Computation in the Mediated Model*

Jonathan Katz† Yehuda Lindell‡

September 26, 2008

Abstract

Collusion-free protocols prevent subliminal communication (i.e., covert channels) between parties running the protocol. In the standard communication model (and assuming the existence of one-way functions), protocols satisfying any reasonable degree of privacy cannot be collusion-free. To circumvent this impossibility result, Alwen et al. recently suggested the *mediated model* where all communication passes through a mediator; the goal is to design protocols where collusion-freeness is guaranteed as long as the mediator is honest, while standard security guarantees continue to hold if the mediator is dishonest. In this model, they gave constructions of collusion-free protocols for commitments and zero-knowledge proofs in the two-party setting.

We strengthen the definition of Alwen et al. in several ways, and resolve the key open questions in this area by showing a collusion-free protocol (in the mediated model) for computing any multi-party functionality.

*This research was supported by US-Israel Binational Science Foundation grant #2004240.

†Department of Computer Science, The University of Maryland, USA. Email: jkatz@cs.umd.edu

‡Department of Computer Science, Bar-Ilan University, Israel. Email: lindell@cs.biu.ac.il

1 Introduction

It is well known that two or more parties running some protocol can potentially embed "disallowed" communication in the protocol messages themselves; i.e., the parties can use the messages of the protocol as a *covert channel* to communicate in a subliminal (aka steganographic) fashion. A *collusion-free* protocol [13] rules out such covert communication. Unfortunately, in the standard communication model (and assuming one-way functions exist) it is impossible for any protocol whose messages have any entropy to be collusion free [9]. This seems to rule out collusion-free protocols (in the standard communication model) realizing any "interesting" level of privacy [13].

Although there has been some work addressing the issue of subliminal channels in certain limited contexts (mainly signature schemes [18, 7, 4, 20, 3]), the problem has, until recently, been largely ignored by the cryptographic community. Presumably this is because protocol designers generally assume a "worst-case" adversarial model, where if two parties are dishonest then they are assumed to be coordinating their actions and communicating out of band, anyway. Recent attention focused on applying cryptographic protocols in game-theoretic settings [13, 11, 10] (see also [12]), however, has re-invigorated interest in designing collusion-free protocols. Preventing subliminal communication is also important in other settings. For example, in a large-scale, distributed system where parties are chosen randomly (from a large pool of players) to run some protocol, the set of parties running a given instance of the protocol may not have had any chance to coordinate their actions in advance, and may have no way to communicate out of band; in this case, the protocol itself introduces a new vulnerability if it can be used as a means for players to initiate collusion, or to transfer information. The problem of subliminal communication is not just of theoretical interest: efforts to collude using covert channels have been observed in real-world auctions [6].

One approach for constructing collusion-free protocols is to rely on *verifiable determinism*. The basic idea goes back to Simmons [19], who applied it to DSA signatures; Lepinski et al. [12, 13] re-introduced the idea in the setting of general secure computation. (This latter setting is much more difficult since here privacy is also a concern.) Roughly speaking, a verifiably deterministic protocol ensures that at every point in the protocol there is only a *single* "valid" message that a player can send; if that player sends anything else, all other parties detect this and raise an alarm. This suffices to prevent covert communication. Unfortunately, all existing constructions of verifiably deterministic protocols for general secure computation [13, 11, 10] rely on strong physical assumptions such as secure envelopes and ballot boxes that cannot be realized unless the parties running the protocol are physically co-located. Furthermore, once parties are in close physical proximity it is unclear how to prevent them from signalling to each other using other means.

A completely different approach to the problem was recently suggested by Alwen et al. [1]. They proposed a real-world model in which each party is able to communicate only with a *mediator*. (I.e., the communication network is a star graph with the mediator at the center.) Rather than *remove* randomness from protocol messages, as when using verifiable determinism, this approach has the mediator *add* randomness to (i.e., re-randomize) the messages of the protocol in order to eliminate any subliminal communication. This, of course, assumes the mediator is honest; when the mediator is dishonest then corrupted parties can communicate freely using the mediator as a channel. In this case, the protocol is required to satisfy standard security guarantees.

The mediated model may, at first, appear unrealistic, but in fact the model can be realized in many real-life settings. As an example, recently in Israel the Maccabi Health Fund (a large HMO) ran an auction with several insurance companies as bidders. In this auction, the bidders came to the offices of the HMO and were seated in separate rooms, with no way to communicate with the outside world (participants were searched for cellphones and other wireless devices). The auction

proceeded in stages with an auctioneer going from room to room, informing the participants about the results of the previous round and taking their next bid. It would have been possible in this case to replace the auctioneer with a server mediating the communication between all parties.

1.1 Our Contributions

In addition to introducing a definition of collusion-freeness in the mediated model, Alwen et al. also gave the first constructions of collusion-free protocols in this setting. They showed protocols for commitment and zero-knowledge in the two-party case, but left open the questions of general secure computation as well as dealing with more than two parties. In this paper we solve these open questions, and show the first multi-party protocol for collusion-free computation of arbitrary functionalities in the mediated model. Feasibility is not trivial in this setting, in part because we aim to satisfy a *stronger* definition of security than that put forth by Alwen et al.; see below. (We view this strengthened definition as an additional contribution of our work.) As a third contribution, we also prove composition theorems in the mediated setting that should prove useful for future work.

The paragraphs that follow briefly describe the most important differences between our definition and that of Alwen et al. [1]; formal definitions are in Section 2. The next section provides a high-level overview of our protocol that emphasizes the technical difficulties that arise.

Aborts as a subliminal channel. The definition in [1] allows parties to communicate some large (but bounded) number of bits by *aborting* the protocol; specifically, in an r-round protocol each party can communicate roughly $\log r$ bits to all other parties. Moreover, Alwen et al. conjecture that this is unavoidable. We show that this conjecture is false. In our definition we allow only a *single* bit to be communicated, where this bit indicates whether some party aborted at some point in the protocol but does not reveal which parties aborted or in which rounds these aborts occurred. Achieving this stronger notion introduces many of the complications in designing our protocol.

Common randomness. The definition in [1] allows parties to use the protocol to agree on a shared random string, even if this is not implied by the functionality being computed.[1] (Indeed, the ideal model in [1, Def. 1] allows the distinct simulators for each party to use the same random tape R; see also the comments in [1] that follow the definition.) In game-theoretic contexts, in particular, this is problematic [11, 10]. Our definition of collusion-freeness rules out this possibility.

1.2 Overview of our Protocol

The discussion here omits certain details, and is meant only to illustrate the high-level structure of the protocol. A formal description of the protocol is given in Section 3.

Let $P_1, \ldots, P_n$ be a set of n parties, each communicating with a mediator P_{n+1}, who wish to compute some (randomized) functionality $\mathcal{F}$. Let π be a protocol that securely computes $\mathcal{F}$ in the standard communication model where all parties have access to a broadcast channel. (In fact, we assume without loss of generality that all messages in π are sent over the broadcast channel.) We compile π to obtain a collusion-free protocol Π in the following way. For each message msg sent by some party P_i in protocol π do:

1. P_i and the mediator run a protocol for secure two-party computation of a functionality $\mathcal{F}^{\pi}_{\mathsf{compute}}$ that outputs to the mediator the next message msg that P_i would send in the underlying execution of π. (A secure computation is needed since P_i will not actually know any of the messages sent by other parties in previous rounds of π; see step 2.)

[1] In [1] it is claimed that this drawback will be addressed in the full version, which we have not seen.

If the mediator does not obtain a valid msg (i.e., if P_i aborts or provides incorrect input to $\mathcal{F}^{\pi}_{\mathsf{compute}}$), then the mediator sets msg to some default value. (This step is essential if we wish to prevent parties from using aborts as a covert channel.)

2. The mediator sends independent *commitments* of msg to each of the other parties.

At the end of the protocol, the mediator runs a secure two-party computation with each party P_i that allows P_i to learn their output, as specified by protocol π.

It is not too difficult to argue that the above protocol is collusion-free when the mediator is honest. Intuitively, this is because each party sees only *independent commitments* to messages rather than the messages themselves. However, the following issues arise if the mediator is dishonest:

Authentication. The mediator should be unable to modify the messages of honest parties. To prevent this, we actually require $\mathcal{F}^{\pi}_{\mathsf{compute}}$ to output (msg, σ), where σ is a valid signature[2] by P_i on msg, and we then have the mediator send commitments on both these values to the other parties. Furthermore, $\mathcal{F}^{\pi}_{\mathsf{compute}}$ will require the mediator to prove that all previous commitments contain appropriately signed messages and, if not, will output an error.

Preventing subliminal channels based on aborts. Signing each message (as just described) prevents a dishonest mediator from modifying honest parties' messages, but introduces a potential problem with collusion-freeness: now, if a party P_j aborts then the mediator has no way of generating a (commitment to a) default message along with an appropriate signature. We fix this by allowing the mediator in this case to send a commitment to a "dummy message" with *no* signature; we also change $\mathcal{F}^{\pi}_{\mathsf{compute}}$ so that if it detects a dummy message it returns an error message to the mediator only. Thus, parties cannot detect whether anyone has aborted until the end of the protocol, and never learn which (or how many) parties aborted nor the round(s) in which an abort occurred. Furthermore, if a dishonest mediator ever uses a dummy message, then it receives error messages from then on.

Ensuring "broadcast". Protocol π is secure under the assumption that parties communicate over a broadcast channel. In our compiled protocol, where all communication is routed through the mediator, we need a way to ensure that a dishonest mediator sends (different commitments to) the *same* message to all parties. We implement this "mediator broadcast" by, roughly speaking, having the mediator (1) collect signatures from all parties on the committed messages; (2) send independent commitments on these signatures to all parties; and then (3) prove to each party independently that all parties have signed a commitment to the same underlying message. As above, in case of an abort we allow the mediator to send a "dummy commitment" to the parties.

Handling concurrency. When the mediator is honest, the two-party protocol computing $\mathcal{F}^{\pi}_{\mathsf{compute}}$, as well as the sub-protocols used to implement mediator broadcast, are run sequentially. But when the mediator is dishonest, it may run *concurrent* executions with the honest parties. We thus need all the two-party protocols being run to be secure under (bounded) concurrent self composition.

2 Definitions

Preliminaries. We denote the security parameter by k. A function $\mu(\cdot)$ is negligible if for every polynomial $p(\cdot)$ there exists a value N such that for all $k > N$ it holds that $\mu(k) < \frac{1}{p(k)}$. Let $X = \{X(1^k, a)\}_{a \in \{0,1\}^*, k \in \mathbb{N}}$ and $Y = \{Y(1^k, a)\}_{a \in \{0,1\}^*, k \in \mathbb{N}}$ be distribution ensembles. Then X

[2]We assume players begin the protocol with a consistent PKI. (We do not assume honestly-generated keys or require parties to prove knowledge of their keys.) See further discussion in the following section.

and Y are computationally indistinguishable, denoted $X \stackrel{c}{\equiv} Y$, if for every non-uniform polynomial-time distinguisher D there exists a negligible function $\mu(\cdot)$ such that for every $a \in \{0,1\}^*$,

$$|\Pr[D(X(1^k, a)) = 1] - \Pr[D(Y(1^k, a)) = 1]| < \mu(k).$$

Security in the mediated model – preliminaries. We use the real/ideal paradigm for defining security of protocols, but our real and ideal worlds differ somewhat from the standard ones and collusion-freeness is defined in a different manner than usual. Our real world is essentially standard except that all communication is between parties $P_1, \ldots, P_n$ and the mediator P_{n+1}, and there is no direct communication between P_i and P_j, for $i, j \leq n$, unless the mediator allows it. We define two different ideal worlds depending on whether the mediator is honest (and collusion-freeness is the goal) or dishonest (in which case we default to the standard notion of security).[3] We also explicitly incorporate a PKI into our real and ideal models; see the discussion below. As in [1], we capture collusion-freeness (when the mediator is honest) by requiring the existence of *independent* simulators, one for each malicious party, such that their joint output in the ideal world is indistinguishable from the joint output of the malicious parties in the real world.

Let $\mathcal{F} = (f_1, \ldots, f_{n+1})$ denote the functionality the parties wish to compute, where each f_i maps $n+1$ inputs to a single output. (We allow the mediator to provide input and receive output, something not done in [1].) We implicitly assume that any protocol under discussion for computing $\mathcal{F}$ is *correct*: i.e., if parties $P_1, \ldots, P_{n+1}$ hold a consistent PKI, have inputs $x_1, \ldots, x_{n+1}$, respectively, and run the protocol honestly, then each party P_i receives output $f_i(x_1, \ldots, x_{n+1})$, distributed appropriately in case these functions are randomized.

PKI. Alwen et al. [1] suggest that public keys for each party should be generated as part of the protocol itself. Unfortunately, if all communication is done via the mediator this means that a corrupt mediator can impersonate parties and "disconnect" the parties into disjoint groups running independent computations (as in [2]). For this reason, Alwen et al. assume a "broadcast-honest" mediator who reliably forwards all public keys at the beginning of the protocol. We simplify things by just assuming that all players begin the protocol with a consistent PKI. (We do not assume honestly generated keys or require parties to prove knowledge of their keys.)

Both our approach and that of Alwen et al. introduce the possibility that a party's public key can be used as a subliminal channel, but we do not view this as a serious concern. For one, public keys are generated *before* inputs are given to the parties, so any such communication will be independent of parties' inputs. Furthermore, parties that are not aware of each other before execution of the protocol will necessarily generate their keys independently, whereas parties that *are* aware of each other before executing the protocol cannot be prevented anyway from communicating an arbitrary amount of information in advance.

2.1 Execution in the Real World (the Mediated Model)

We first consider the real world in which an $(n+1)$-party protocol Π is executed. Channels are available only between the mediator P_{n+1} and P_i (for all i). The channels to/from P_{n+1} are assumed to be private and authenticated for simplicity.

We assume here that the mediator is honest, and consider a dishonest mediator in Section 2.3. Let $\mathcal{I} \subseteq [n]$ denote the set of corrupted parties[4] and denote by $\mathcal{H} = [n] \setminus \mathcal{I}$ the set of uncorrupted parties (not including the mediator). An execution in the real world proceeds as follows:

[3] In [1] a single ideal world encompassing both cases was given, but we find it simpler to treat them separately.

[4] We remark that, in contrast to the usual case, a meaningful definition is obtained even when $\mathcal{I} = [n]$.

PKI establishment: For $i \in \mathcal{H}$, party P_i honestly generates a pair of signature keys (pk_i, sk_i). Each corrupted party P_i may output any public key pk_i of its choice. At the end of this stage, all parties (including the mediator) are given the vector $(pk_1, \ldots, pk_n)$ of public keys.

Input determination and protocol execution: Each party P_i (for $1 \leq i \leq n+1$) is given input x_i. (We stress that this is done only *after* the PKI is established.) Party P_i is also given auxiliary input aux_i; honest players ignore this input. Independent random coins $r_1, \ldots, r_{n+1}$ for the parties are also chosen. The parties then run the protocol, with honest parties (including the mediator) acting as directed by Π, and corrupted parties behaving arbitrarily.

Result of the experiment: At the conclusion of the protocol, let OUT_i, for $i \in \mathcal{I}$, denote the entire view of the corrupted party P_i, and let OUT_i, for $i \in \mathcal{H} \cup \{n+1\}$, denote the final output of P_i (as dictated by Π). Given a set of adversarial strategies $\mathcal{A}_\mathcal{I} = \{\mathcal{A}_i\}_{i \in \mathcal{I}}$, define

$$\text{REAL}^{\mathsf{mediated}}_{\Pi, \mathcal{A}_\mathcal{I}(\vec{\mathsf{aux}})}(1^k, \vec{x}) \stackrel{\text{def}}{=} (\text{OUT}_1, \ldots, \text{OUT}_{n+1})$$

to be the random variable consisting of the stated outputs following an execution of Π where the parties are given inputs and auxiliary inputs as specified.

2.2 Collusion-Freeness (with an Honest Mediator)

Throughout this section, we assume the mediator is honest. In this ideal world, all parties communicate only with a trusted party computing $\mathcal{F}$. In particular, corrupted parties are unable to communicate with each other and therefore cannot communicate information about their inputs or coordinate their actions (beyond what they have agreed upon in advance). Let $\mathcal{I}$ be the set of corrupted parties, and let $\mathcal{H}$ be the set of honest parties (other than the mediator) as before. An execution in this ideal world proceeds as follows:

PKI establishment: A PKI is established exactly as in the real world. That is: for $i \in \mathcal{H}$, party P_i honestly generates signature keys (pk_i, sk_i). Each corrupted party P_i outputs any public key pk_i of its choice. All parties are then given the vector $(pk_1, \ldots, pk_n)$ of public keys.

Input determination: Each party P_i (for $1 \leq i \leq n+1$) is given their input x_i. Party P_i is also given auxiliary input aux_i; an honest player ignores this input. Independent random coins $\{r_i\}_{i \in \mathcal{I}}$ for the corrupted parties are also chosen.

An honest party sets $x'_i = x_i$ and sends x'_i to $\mathcal{F}$. A corrupted P_i may send any x'_i of its choice. Unless otherwise specified, if any $x'_i = \perp$ then all parties get output $\perp$ from $\mathcal{F}$. Otherwise, $\mathcal{F}$ hands $f_i(x'_1, \ldots, x'_{n+1})$ to party P_i, for $1 \leq i \leq n+1$.

Note that a malicious party who "aborts" by sending $\perp$ to $\mathcal{F}$ communicates (at most) one additional bit to all other parties beyond what is directly implied by $\mathcal{F}$. Furthermore, this decision to abort must be made independently of the output of $\mathcal{F}$ on the given inputs.

Result of the experiment: At the conclusion of the protocol, let OUT_i, for $i \in \mathcal{I}$, denote an arbitrary value output by P_i, and let OUT_i, for $i \in \mathcal{H} \cup \{n+1\}$, denote the value given to P_i by $\mathcal{F}$. Given a set of adversarial strategies $\mathcal{S}_\mathcal{I} = \{\mathcal{S}_i\}_{i \in \mathcal{I}}$, define

$$\text{IDEAL}^{\mathsf{cf}}_{\mathcal{F}, \mathcal{S}_\mathcal{I}(\vec{\mathsf{aux}})}(1^k, \vec{x}) \stackrel{\text{def}}{=} (\text{OUT}_1, \ldots, \text{OUT}_{n+1})$$

to be the random variable consisting of the stated outputs following an ideal-world execution where the parties are given inputs and auxiliary inputs as specified.

Having defined the ideal and real models, we can now define collusion-freeness of a protocol. If we followed the standard definitional paradigm, we would require that for all $\mathcal{I}$ and any set of efficient real-world strategies $\mathcal{A}_\mathcal{I} = \{\mathcal{A}_i\}_{i\in\mathcal{I}}$, there should exist a set of efficient ideal-world strategies $\mathcal{S}_\mathcal{I} = \{\mathcal{S}_i\}_{i\in\mathcal{I}}$ such that the corresponding real- and ideal-world outcomes are computationally indistinguishable. A deficiency of this approach is that it allows each $\mathcal{S}_i$ to depend on $\mathcal{I}$ as well as *all* the $\mathcal{A}_j$ (i.e., even for $j \neq i$), and thus this approach does not adequately model collusion-freeness. Since we want each $\mathcal{S}_i$ to depend only on $\mathcal{A}_i$, we instead require the existence of a set of efficient *transformations* $\{\mathsf{Sim}_i\}_{i\in[n]}$ such that setting $\mathcal{S}_i = \mathsf{Sim}_i(1^k, \mathcal{A}_i)$ for $i \in \mathcal{I}$ makes the real and ideal worlds indistinguishable. While this is not the most general definition possible, we achieve this definition in our work (and, moreover, all our transformations are black-box).

Definition 1 *Let $\mathcal{F}$ be a functionality, and Π an $(n+1)$-party protocol computing $\mathcal{F}$ in the mediated model.* Π *is a* collusion-free protocol computing $\mathcal{F}$ *if there is a set $\{\mathsf{Sim}_i\}_{i\in[n]}$ of efficiently-computable transformations such that, for all $\mathcal{I} \subseteq [n]$ and any* PPT *strategies $\{\mathcal{A}_i\}_{i\in\mathcal{I}}$, setting $\mathcal{S}_i = \mathsf{Sim}_i(1^k, \mathcal{A}_i)$ for $i \in \mathcal{I}$ implies*

$$\left\{\mathrm{IDEAL}^{\mathsf{cf}}_{\mathcal{F},\mathcal{S}_\mathcal{I}(\mathsf{a}\vec{\mathsf{u}}\mathsf{x})}(1^k, \vec{x})\right\}_{\vec{x},\mathsf{a}\vec{\mathsf{u}}\mathsf{x}\in\in(\{0,1\}^*)^{n+1},\, k\in\mathbb{N}} \stackrel{\mathrm{c}}{\equiv} \left\{\mathrm{REAL}^{\mathsf{mediated}}_{\Pi,\mathcal{A}_\mathcal{I}(\mathsf{a}\vec{\mathsf{u}}\mathsf{x})}(1^k, \vec{x})\right\}_{\vec{x},\mathsf{a}\vec{\mathsf{u}}\mathsf{x}\in(\{0,1\}^*)^{n+1},\, k\in\mathbb{N}}.$$

2.3 Security (with a Dishonest Mediator)

The definition in the case of a dishonest mediator is essentially the standard one, with the main exception being that honest parties cannot communicate directly in the real world. For completeness, we include the definition in Appendix A.

A protocol satisfying the definitions of both this and the previous section will be called a collusion-free protocol securely computing $\mathcal{F}$.

3 Collusion-Free Multiparty Computation in the Mediated Model

We construct a collusion-free protocol Π for secure computation of an arbitrary (poly-time) functionality $\mathcal{F} = (f_1, \ldots, f_{n+1})$. We first introduce the components of our protocol, and describe the protocol in full detail in Section 3.4. High-level intuition for the protocol was given in Section 1.2.

3.1 Tools and Building Blocks

Our protocol uses a number of standard cryptographic primitives and tools, which we review here.

Some standard primitives and functionalities:

- **Commitments.** Let C be a perfectly binding commitment scheme, where $C(m; r)$ denotes a commitment to m using random coins r. The decommitment of $\mathsf{com} = C(m; r)$ is $\mathsf{dec} = (m, r)$. We assume all commitments are some known, fixed function of the message length.
- **Signature schemes.** Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme that is existentially unforgeable under adaptive chosen-message attacks. $\mathsf{Range}(\mathsf{Gen})$ denotes the set of possible outputs of Gen, and we assume that one can decide in polynomial time whether a given (sk, pk) lies in $\mathsf{Range}(\mathsf{Gen})$. (This is without loss of generality, since we may always take sk to be the random coins used by Gen.) We assume all valid signatures are some known, fixed function of the message length.

– **Two-party functionalities.** We use ideal functionalities to model various sub-protocols used in Π. Standard functionalities we use are the commitment functionality $\mathcal{F}_{\mathsf{com}}$, the coin-tossing functionality $\mathcal{F}_{\mathsf{ct}}$, the zero-knowledge functionality $\mathcal{F}_{\mathsf{zk}}$, and the signature functionality $\mathcal{F}_{\mathsf{Sign}}$:

1. $\mathcal{F}_{\mathsf{com}}$ is defined by $\mathcal{F}_{\mathsf{com}}((m,r),\lambda) = (\perp, C(m;r))$, where λ denotes the empty string.
2. The coin-tossing functionality is defined by $\mathcal{F}_{\mathsf{ct}}(1^\ell,\lambda) = ((r,s), C(r;s))$, where $|r| = \ell$ and both r and s are uniformly distributed.
3. Let R be an $\mathcal{NP}$-relation. Functionality $\mathcal{F}_{\mathsf{zk}}$ for the relation R is defined by

$$\mathcal{F}_{\mathsf{zk}}((x,w),x') = \begin{cases} (\perp, R(x,w)) & \text{if } x = x' \\ (\perp, 0) & \text{otherwise} \end{cases}$$

4. The signature functionality is defined as:

$$\mathcal{F}_{\mathsf{Sign}}((sk,pk,m),(pk',m')) = \begin{cases} (\perp, \mathsf{Sign}_{sk}(m)) & \text{if } (pk,m) = (pk',m') \text{ and} \\ & \quad (sk,pk) \in \mathsf{Range}(\mathsf{Gen}) \\ (\perp,\perp) & \text{otherwise} \end{cases}$$

A protocol π that securely computes $\mathcal{F}$ (in the standard sense): Let π be an $(n+1)$-party protocol that securely computes $\mathcal{F}$ in the standard sense [8], in the standard communication model where all parties have access to a public (but authenticated) broadcast channel. Precisely, π is secure-with-designated-abort for any number $t \leq n+1$ corrupted parties, where the mediator P_{n+1} is designated as the party who can prematurely abort the protocol. Roughly speaking, this means that the protocol guarantees privacy and correctness regardless of how many parties are corrupted, and guarantees output delivery and complete fairness as long as the mediator is not corrupted. For technical reasons, we also assume that π is proven secure using a *black-box* simulator.

We assume the following about π, all of which can be ensured using standard techniques:

– All messages in π have the same, fixed length. In any given round only a single party broadcasts, and the identity of the party who broadcasts depends on the current round number only.

– Say π has r rounds. Then P_{n+1} learns its output in round $r-1$; party P_{n+1} broadcasts in round r; and every other party learns its output in this final round.

Dummy commitments: As described in Section 1.2, everything the mediator sends to the parties will be "wrapped" inside a commitment. When all parties behave honestly, these will all be commitments to legitimate protocol messages of π along with a digital signature. If some party P_i aborts (or otherwise deviates from the protocol), however, an honest mediator will not be able to generate a valid commitment of this sort (in particular, the mediator will be unable to forge an appropriate signature). Nevertheless, we do not want some other party to learn that P_i aborted the protocol. We achieve this by allowing the mediator to send special "dummy commitments" to a distinguished value dummy. (I.e., a dummy commitment takes the form $C(\mathsf{dummy};r)$.) For the sake of concreteness, dummy can be taken to be a string of 0s of the appropriate length if we require that all legitimate messages be prefixed by a '1'.

3.2 Oblivious Computation of π

The general structure of protocol Π, as described in Section 1.2, has the mediator send to each P_j *commitments* to all the protocol messages of π. Thus, P_j cannot compute its π-messages directly

Functionality $\mathcal{F}^{\pi}_{\mathsf{compute}}$

The parties' public keys are $(pk_1, \ldots, pk_n)$. Functionality $\mathcal{F}^{\pi}_{\mathsf{compute}}$ runs with two parties P_j and P_{n+1} and works as follows:

- P_j inputs a pair of commitments $\mathsf{com}^{\mathsf{input}}$ and $\mathsf{com}^{\mathsf{rand}}$; a vector of commitments $\vec{C}$; and a round number rid. In addition, P_j sends two strings $\mathsf{dec}^{\mathsf{input}}$ and $\mathsf{dec}^{\mathsf{rand}}$, and its signing key sk_j.
- P_{n+1} inputs a pair of commitments $\mathsf{com}^{\mathsf{input}}_j$ and $\mathsf{com}^{\mathsf{rand}}_j$; a vector of commitments $\vec{C}_j$; and a round number rid'. In addition, P_{n+1} sends a vector $\vec{\mathsf{dec}}_j$.
- Upon receiving the above, $\mathcal{F}^{\pi}_{\mathsf{compute}}$ does:
  1. If $(\mathsf{com}^{\mathsf{input}}, \mathsf{com}^{\mathsf{rand}}, \vec{C}, \mathsf{rid}) \neq (\mathsf{com}^{\mathsf{input}}_j, \mathsf{com}^{\mathsf{rand}}_j, \vec{C}_j, \mathsf{rid}')$ or if $(sk_j, pk_j) \notin \mathsf{Range}(\mathsf{Gen})$, then send $\perp$ to P_{n+1} and halt.
  2. If $\mathsf{dec}^{\mathsf{input}}$ is not a valid decommitment to $\mathsf{com}^{\mathsf{input}}_j$, or $\mathsf{dec}^{\mathsf{rand}}$ is not a valid decommitment to $\mathsf{com}^{\mathsf{rand}}_j$, or $\vec{\mathsf{dec}}_j$ does not contain valid decommitments to all the commitments in $\vec{C}_j$, then send $\perp$ to P_{n+1} and halt.
  3. Let $(\mathsf{msg}_1, \sigma_1), \ldots, (\mathsf{msg}_\ell, \sigma_\ell)$ be the committed values in $\vec{C}_j$. If any of these are dummy values, send $\perp$ to P_{n+1} and halt. For $1 \leq i \leq \ell$, let ℓ_i denote the index of the party who is supposed to broadcast in round i of π. If there exists an i such that (1) $\ell_i \neq n+1$ and (2) $\mathsf{Vrfy}_{pk_{\ell_i}}((\mathsf{msg}_i, 0i), \sigma_i) \neq 1$, then send $\perp$ to P_{n+1} and halt.
  4. Let x_j and r_j be the committed values in $\mathsf{com}^{\mathsf{input}}_j$ and $\mathsf{com}^{\mathsf{rand}}_j$ respectively. Compute the next message msg that party P_j would send in protocol π when running with input x_j, random tape r_j, and after receiving messages $\mathsf{msg}_1, \ldots, \mathsf{msg}_\ell$. In addition, compute $\sigma = \mathsf{Sign}_{sk_j}(\mathsf{msg}, \mathsf{rid})$. Send (msg, σ) to P_{n+1} and halt.

Figure 1: The functionality computing the next message of π

(since it cannot directly observe the π-messages of other parties), but must instead compute these messages by executing a two-party protocol with the mediator. Specifically, we define a functionality $\mathcal{F}^{\pi}_{\mathsf{compute}}$ that computes the next π-message of P_j along with a signature of P_j on that message, and a functionality $\mathcal{F}^{\pi}_{\mathsf{output}}$ that enables P_j to obtain its π-output. (The actual functionalities we require are more complex because we must also check for incorrect behavior on the part of the mediator.) These are defined formally in Figures 1 and 2. Observe that only the mediator P_{n+1} receives output from $\mathcal{F}^{\pi}_{\mathsf{compute}}$, and only P_j receives output from $\mathcal{F}^{\pi}_{\mathsf{output}}$.

3.3 Mediator Broadcast

Protocol π assumes that all parties communicate over a broadcast channel. When the mediator is corrupt, we therefore must ensure that the mediator sends (commitments to) the *same* message to all honest parties. Note that checking for signatures on protocol messages, as done by $\mathcal{F}^{\pi}_{\mathsf{compute}}$ and $\mathcal{F}^{\pi}_{\mathsf{output}}$, only ensures that this holds for the messages of *honest* parties; it does not prevent a dishonest mediator from sending different messages on behalf of *corrupted* parties (who may collude with the mediator and sign multiple messages).

We achieve the above using what we refer to as "mediator broadcast". The mediator P_{n+1} begins holding a message m, and at the end of the protocol each party P_i obtains an (independent) commitment com_i to a message m_i that is unknown to P_i. (Note that m_i is well-defined, since C is perfectly binding.) The desired functionality is, informally, as follows: If all parties are honest, then $m_i = m$ for all P_i. If P_{n+1} is honest, then there is an $m' \in \{m, \mathsf{dummy}\}$ such that $m_i = m'$

Functionality $\mathcal{F}^{\pi}_{\mathsf{output}}$

The parties' public keys are $(pk_1, \ldots, pk_n)$. Functionality $\mathcal{F}^{\pi}_{\mathsf{output}}$ runs with two parties P_j and P_{n+1} and works as follows:

- P_j inputs a pair of commitments $\mathsf{com}^{\mathsf{input}}$ and $\mathsf{com}^{\mathsf{rand}}$, and a vector of r commitments $\vec{C}$. In addition, P_j sends two strings $\mathsf{dec}^{\mathsf{input}}$ and $\mathsf{dec}^{\mathsf{rand}}$.
- P_{n+1} inputs a pair of commitments $\mathsf{com}^{\mathsf{input}}_j$ and $\mathsf{com}^{\mathsf{rand}}_j$, and a vector of r commitments $\vec{C}_j$. In addition, P_{n+1} sends a vector $\vec{\mathsf{dec}}_j$.
- Upon receiving the above, $\mathcal{F}^{\pi}_{\mathsf{compute}}$ does:
  1. If $(\mathsf{com}^{\mathsf{input}}, \mathsf{com}^{\mathsf{rand}}, \vec{C}) \neq (\mathsf{com}^{\mathsf{input}}_j, \mathsf{com}^{\mathsf{rand}}_j, \vec{C}_j)$, then send $\perp$ to P_j and halt.
  2. If $\mathsf{dec}^{\mathsf{input}}$ (resp., $\mathsf{dec}^{\mathsf{rand}}$) is not a valid decommitment to $\mathsf{com}^{\mathsf{input}}_j$ (resp., $\mathsf{com}^{\mathsf{rand}}_j$), or $\vec{\mathsf{dec}}_j$ does not contain valid decommitments to all the commitments in $\vec{C}_j$, then send $\perp$ to P_j and halt.
  3. Let $(\mathsf{msg}_1, \sigma_1), \ldots, (\mathsf{msg}_r, \sigma_r)$ be the committed values in $\vec{C}_j$. If any of these are dummy values, send $\perp$ to P_j and halt. For $1 \leq i \leq r$, let ℓ_i denote the index of the party who is supposed to broadcast in round i of π. If there exists an i such that (1) $\ell_i \neq n+1$ and (2) $\mathsf{Vrfy}_{pk_{\ell_i}}((\mathsf{msg}_i, 0i), \sigma_i) \neq 1$, then send $\perp$ to P_j and halt.
  4. Let x_j and r_j be the committed values in $\mathsf{com}^{\mathsf{input}}_j$ and $\mathsf{com}^{\mathsf{rand}}_j$. Compute the value OUT_j that party P_j would output in protocol π when running with input x_j, random tape r_j, and after receiving the messages $\mathsf{msg}_1, \ldots, \mathsf{msg}_r$. Send OUT_j to P_j and halt.

Figure 2: The functionality computing the output of π

for all honest parties P_i. If P_{n+1} is dishonest, then there is an m' such that $m_i \in \{m', \mathsf{dummy}\}$ for all honest parties P_i. This is a weak form of broadcast, but suffices for our application.

In Figure 3, we formally define a functionality $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$, parameterized by a session id sid, implementing the above. (An honest mediator chooses $r_1, \ldots, r_n$ uniformly at random, and sets $\mathcal{H} = [n]$; an honest P_i sends $b_i = 1$.) We stress that the functionality *always* outputs a commitment for each party, *even if some (dishonest) party uses input* $\perp$. Our protocol $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ realizing $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ proceeds, roughly speaking in the following three stages:

1. P_{n+1} sends $\mathsf{com}_i = C(m; r_i)$ to each party P_i.
2. P_i generates a signature σ_i on $(\mathsf{com}_i, \mathsf{sid})$, and sends σ_i to P_{n+1}.
3. If any P_i fails to send a valid signature, then P_{n+1} sends (independent) dummy commitments to all parties. Otherwise, P_{n+1} sends an independent commitment to $(\mathsf{com}_1, \sigma_1, \ldots, \mathsf{com}_n, \sigma_n)$ to all parties. In either case, P_{n+1} then proves to each party in zero knowledge that the commitments it sent take one of these forms.

The actual protocol $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ is slightly more complex. Furthermore, for technical reasons we do not use commitments, signatures, or zero-knowledge proofs directly but instead work in the ($\mathcal{F}_{\mathsf{com}}$, $\mathcal{F}_{\mathsf{Sign}}$, $\mathcal{F}_{\mathsf{zk}}$)-hybrid model. The complete protocol and a proof of security are given in Appendix B.

3.4 A Protocol Π for Collusion-Free Secure Computation

We now describe a collusion-free protocol Π that securely computes $\mathcal{F}$ in the ($\mathcal{F}_{\mathsf{com}}$, $\mathcal{F}_{\mathsf{ct}}$, $\mathcal{F}^{\pi}_{\mathsf{compute}}$, $\mathcal{F}^{\pi}_{\mathsf{output}}$, $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$)-hybrid model. When these functionalities are realized using protocols designed for the mediated model, we obtain a protocol for the real mediated model.

$\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$

Functionality $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$ runs with $P_1, \ldots, P_n, P_{n+1}$ as follows:

- For $j \in [n]$, each P_j inputs a bit b_j. (If P_j sends $\perp$, set $b_j = 0$.) Let $b = \bigwedge_i b_i$.
- The mediator P_{n+1} inputs a message m, random coins $r_1, \ldots, r_n$, and a set $\mathcal{H} \subseteq [n]$.
- If $b = 1$, then:
  - For $i \in \mathcal{H}$, send $\mathsf{com}_i = C(m; r_i)$ to P_i.
  - For $i \in [n] \setminus \mathcal{H}$, send $\mathsf{com}_i = C(\mathsf{dummy}; r_i)$ to P_i.

  If $b = 1$, then for $i \in [n]$ send $\mathsf{com}_i = C(\mathsf{dummy}; r_i)$ to P_i. In either case, send b to P_{n+1}.

Figure 3: "Mediator broadcast"

Our protocol consists of three stages. In the first stage, the parties commit to their inputs and random coins for a protocol π that securely computes $\mathcal{F}$ (in the standard sense). In the second stage, the parties simulate π, round-by-round, as follows. If it is P_j's turn to speak (for $j \in [n]$), then P_j runs $\mathcal{F}_{\mathsf{compute}}^{\pi}$ with the mediator; in this way, the mediator obtains the next message msg of the protocol along with a signature of P_j on this message (and the current round number). If it is the mediator's turn to speak, then the mediator simply computes the next message msg of the protocol on its own. The mediator then runs "mediator broadcast" using msg and the signature. As long as everyone behaves honestly, each party thus learns commitments to all messages of the protocol. In the third stage, the mediator runs $\mathcal{F}_{\mathsf{output}}^{\pi}$ with each P_j to enable P_j to learn its output. We now describe the protocol formally.

The protocol begins with all parties holding the same vector of public keys $\vec{pk} = (pk_1, \ldots, pk_n)$; party P_i also holds input x_i and, if $1 \leq i \leq n$, its own secret key sk_i.

Stage 1 – input commitment and coin tossing:

1. Each P_j executes $\mathcal{F}_{\mathsf{com}}$ with P_{n+1}, where P_j chooses random s_j and provides input $\mathsf{dec}_j^{\mathsf{input}} = (x_j, s_j)$ to $\mathcal{F}_{\mathsf{com}}$. Let $\mathsf{com}_j^{\mathsf{input}}$ be the commitment received by P_{n+1} from $\mathcal{F}_{\mathsf{com}}$.
2. Each P_j executes $\mathcal{F}_{\mathsf{ct}}$ with P_{n+1}, where the input length ℓ is the number of coins needed to run π. We denote by $\mathsf{dec}_j^{\mathsf{rand}} = (r_j, s'_j)$ the output of P_j and by $\mathsf{com}_j^{\mathsf{rand}}$ the output of P_{n+1}.

Stage 2 – round-by-round emulation of π: The mediator P_{n+1} initializes $\mathsf{abort} = \mathsf{false}$. Then, for $i = 1$ to $r - 1$, the parties run the following:

1. **(P_{n+1} learns the round-i message of π.)**

   Case 1: Party P_j, for $1 \leq j \leq n$, is supposed to broadcast in the ith round of π.

   - Let $\vec{C}_j = (\mathsf{com}_1^j, \ldots, \mathsf{com}_{i-1}^j)$ be the commitments that P_j output in the previous $i-1$ rounds.
   - P_j and P_{n+1} run an instance of $\mathcal{F}_{\mathsf{compute}}^{\pi}$. Here, P_j sends $\mathcal{F}_{\mathsf{compute}}^{\pi}$ its commitments $\mathsf{com}_j^{\mathsf{input}}$ and $\mathsf{com}_j^{\mathsf{rand}}$, the vector of commitments $\vec{C}_j$, the round identifier $\mathsf{rid} = 0i$, the decommitments $\mathsf{dec}_j^{\mathsf{input}}$ and $\mathsf{dec}_j^{\mathsf{rand}}$, and its signing key sk_j.

     P_{n+1} sends $\mathcal{F}_{\mathsf{compute}}^{\pi}$ the commitments $\mathsf{com}_j^{\mathsf{input}}$ and $\mathsf{com}_j^{\mathsf{rand}}$, the commitments $(\mathsf{com}_1^j, \ldots, \mathsf{com}_{i-1}^j)$, the round identifier $\mathsf{rid} = 0i$, and the decommitments $(\mathsf{dec}_1^j, \ldots, \mathsf{dec}_{i-1}^j)$.
   - If $\mathcal{F}_{\mathsf{compute}}^{\pi}$ returns $\perp$ to P_{n+1}, then P_{n+1} sets $\mathsf{abort} = \mathsf{true}$ and $m_i = \mathsf{dummy}$. Otherwise, if $\mathcal{F}_{\mathsf{compute}}^{\pi}$ returns $(\mathsf{msg}_i, \sigma_i)$ to P_{n+1}, then P_{n+1} sets $m_i = (\mathsf{msg}_i, \sigma_i)$.

Case 2: P_{n+1} is supposed to broadcast in the ith round of π:

– If $\mathsf{abort} = \mathsf{true}$ then P_{n+1} sets $m_i = \mathsf{dummy}$. If $\mathsf{abort} = \mathsf{false}$ then P_{n+1} locally computes the message msg_i as instructed by π (this is possible since P_{n+1} sees all π-messages "in the clear"), and sets $m_i = \mathsf{msg}_i$.

2. **(P_{n+1} "broadcasts" the round-i message of π.)** Let $\mathsf{sid} = 1i$. The mediator P_{n+1} chooses random $r_1, \ldots, r_n$ and runs $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$ with all the other parties, where P_{n+1} provides input $(m_i, r_1, \ldots, r_n, \mathcal{H} = [n])$ and every other party P_j provides input 1.

   Each party P_j defines com_i^j to be the commitment that it received from $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$. Note that P_{n+1}, given its output from $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$, can compute the commitments $\{\mathsf{com}_i^j\}_{j\in[n]}$, and knows the corresponding decommitments.

Stage 3 – output determination:

1. If $\mathsf{abort} = \mathsf{true}$ then P_{n+1} sets $\mathsf{msg}_r = \mathsf{dummy}$ and sets $\text{OUT}_{n+1} = \perp$. If $\mathsf{abort} = \mathsf{false}$ then P_{n+1} computes its π-output OUT_{n+1} and final message msg_r locally (it can do this since P_{n+1} sees all π-messages "in the clear"). In either case, the mediator then sets $\mathsf{sid} = 1r$, chooses random $r_1, \ldots, r_n$, and runs $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$ with all the other parties, where P_{n+1} provides input $(\mathsf{msg}_r, r_1, \ldots, r_n, \mathcal{H} = [n])$ and every other party P_i provides input 1. The mediator outputs OUT_{n+1}.

   Each party P_j defines com_r^j to be the commitment that it received from $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$. Note that P_{n+1} can compute the commitment, and knows the corresponding decommitment.

2. The mediator P_{n+1} runs $\mathcal{F}_{\mathsf{output}}^{\pi}$ with each P_j, where P_j provides input $\mathsf{com}_j^{\mathsf{input}}$, $\mathsf{com}_j^{\mathsf{rand}}$, $(\mathsf{com}_1^j, \ldots, \mathsf{com}_r^j)$, $\mathsf{dec}_j^{\mathsf{input}}$, and $\mathsf{dec}_j^{\mathsf{rand}}$, and P_{n+1} sends $\mathsf{com}_j^{\mathsf{input}}$, $\mathsf{com}_j^{\mathsf{rand}}$, the commitments $(\mathsf{com}_1^j, \ldots, \mathsf{com}_r^j)$, and the decommitments $(\mathsf{dec}_1^j, \ldots, \mathsf{dec}_{i-1}^j)$.

   Each party P_j outputs the value it receives from $\mathcal{F}_{\mathsf{output}}^{\pi}$ in this step.

4 Proof of Security

We first prove that Π is a collusion-free protocol that securely computes $\mathcal{F}$ in the $(\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{ct}}, \mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}, \mathcal{F}_{\mathsf{compute}}^{\pi}, \mathcal{F}_{\mathsf{output}}^{\pi})$-hybrid model. A proof of the following appears in Appendix C.

Theorem 4.1 *Let π be a protocol that securely computes $\mathcal{F}$ (as required in Section 3.1); let C be a perfectly binding commitment scheme; and let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a secure signature scheme. Then protocol Π from the previous section is a collusion-free protocol for securely computing $\mathcal{F}$ in the $(\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{ct}}, \mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}, \mathcal{F}_{\mathsf{compute}}^{\pi}, \mathcal{F}_{\mathsf{output}}^{\pi})$-hybrid model.*

We now show that when the ideal-world functionalities are instantiated using protocols satisfying appropriate definitions of security, we obtain a collusion-free protocol that securely computes $\mathcal{F}$ in the real mediated model. We obtain this as a corollary of the following composition theorems.

Theorem 4.2 *Let Π be a collusion-free protocol computing $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, where Π contains polynomially many sequential calls to $\mathcal{G}$, and let ρ be a collusion-free protocol computing $\mathcal{G}$. Then the composed protocol Π^{ρ} is a collusion-free protocol computing $\mathcal{F}$ in the real mediated model.*

A proof of Theorem 4.2 follows along the lines of [5] and is given in Appendix D.1.

Theorem 4.3 *Let $\mathcal{G}$ be a two-party functionality, and let Π be a multi-party protocol that securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model for concurrent self composition. Assume further that Π only makes calls to $\mathcal{G}$ (i.e., there are no other messages in Π), and that P_{n+1} plays the role of the second party in all calls to $\mathcal{G}$. Let m denote the overall number of calls to $\mathcal{G}$ in Π.*

If ρ is a two-party protocol that securely computes $\mathcal{G}$ under m-bounded concurrent self-composition, then the composed protocol Π^ρ securely computes $\mathcal{F}$ in the real mediated model.

Note that even if Π instructs the parties to make *sequential* calls to $\mathcal{G}$, Theorem 4.3 requires ρ to be secure under (bounded) concurrent self-composition since a dishonest mediator may run concurrent executions with different honest parties. Theorem 4.3 follows immediately from the definition of m-bounded concurrent self composition; a proof is given in Appendix D.2.

We can now prove our main result:

Corollary 4.4 *Let $\mathcal{F}$ be a polynomial-time, multi-party functionality. Then assuming the existence of enhanced trapdoor permutations, there exists a collusion-free protocol for securely computing $\mathcal{F}$ in the real mediated model.*

Proof: Let Π^{cf} denote the protocol of Section 3.4, where:

- C is a perfectly binding commitment scheme and $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is a secure signature scheme;
- π securely computes $\mathcal{F}$ (in the standard communication model) as specified in Section 3.1;
- $\mathcal{F}_{\mathsf{com}}$, $\mathcal{F}_{\mathsf{ct}}$, $\mathcal{F}^{\pi}_{\mathsf{compute}}$, and $\mathcal{F}^{\pi}_{\mathsf{output}}$ are instantiated by a single protocol[5] ρ that is secure under m-bounded concurrent self-composition [14] (m will be specified in the proof below);
- $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ is instantiated using protocol $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ of Appendix B, where $\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{Sign}}$, and $\mathcal{F}_{\mathsf{zk}}$ are realized by the same protocol ρ as above.

Note that Π^{cf} is defined in the real mediated model, and all the components above can be constructed under the assumption that enhanced trapdoor permutations exist. We now prove that Π^{cf} is a collusion-free protocol securely computing $\mathcal{F}$.

In the case of an honest mediator, this follows directly from Theorems 4.1 and 4.2 using the fact that the "mediator broadcast" protocol of Section 3.3 is collusion-free and the observation that any two-party protocol secure in the standard sense is trivially collusion-free. (If ρ is secure under m-bounded concurrent self-composition, it is also secure in the stand-alone sense.)

In the case of a dishonest mediator, the proof is slightly more involved since the hybrid-world protocol Π, as specified, does not fulfill the requirements of Theorem 4.3 (because $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ is not a two-party protocol). Nevertheless, observe that $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ consists only of calls to the two-party functionalities $\mathcal{F}_{\mathsf{com}}$, $\mathcal{F}_{\mathsf{Sign}}$, and $\mathcal{F}_{\mathsf{zk}}$. Thus, if we define Π' to be the same as protocol Π but using $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ instead of $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$, it follows that Π' *does* fulfill the requirements of Theorem 4.3. Observing that this changes the output distribution by at most a negligible amount (by security of $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$), we have that Π' securely computes $\mathcal{F}$ in the $(\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{ct}}, \mathcal{F}_{\mathsf{Sign}}, \mathcal{F}_{\mathsf{zk}}, \mathcal{F}^{\pi}_{\mathsf{compute}}, \mathcal{F}^{\pi}_{\mathsf{output}})$-hybrid model. Using an appropriate protocol ρ as required by Theorem 4.3, where m is the total number of ideal calls in Π', we conclude that $\Pi^{\mathsf{cf}} = \Pi'^{\rho}$ securely computes $\mathcal{F}$ in the real mediated model. ■

[5] This means we simply "wrap" these functionalities in one larger functionality, and have parties provide an additional input selecting which sub-functionality to run.

References

[1] J. Alwen, A. Shelat, and I. Visconti. Collusion-Free Protocols in the Mediated Model. *Advances in Cryptology — Crypto 2008*, LNCS vol. 5157, pages 497–514, Springer, 2008.

[2] B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure Computation without Authentication. *Advances in Cryptology — Crypto 2005*, LNCS vol. 3621, pages 361–377, Springer, 2005.

[3] J.-M. Bohli and R. Steinwandt. On Subliminal Channels in Deterministic Signature Schemes. *Information Security and Cryptology — ICISC 2004*, LNCS vol. 3506, pages 182–194, Springer, 2005.

[4] M. Burmester, Y. Desmedt, T. Itoh, K. Sakurai, H. Shizuya, and M. Yung. A Progress Report on Subliminal-Free Channels. *Information Hiding Workshop*, LNCS vol. 1174, pages 157–168, Springer, 1996.

[5] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[6] P. Cramption and J. Schwartz. Collusive Bidding: Lessons from the FCC Spectrum Auctions. *Journal of Regulatory Economics* 17(3): 229–252, 2000.

[7] Y. Desmedt. Simmons' Protocol is not Free of Subliminal Channels. *IEEE Computer Security Foundations Workshop*, pages 170–175, 1996.

[8] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, 2004.

[9] N. Hopper, J. Langford, and L. von Ahn. Provably Secure Steganography. *Advances in Cryptology — Crypto 2002*, LNCS vol. 2442, pages 77–92, Springer, 2002.

[10] S. Izmalkov, M. Lepinski, and S. Micali. Verifiably Secure Devices. *Theory of Cryptography Conference (TCC) 2008*, LNCS vol. 4948, pages 273–301, Springer, 2008.

[11] S. Izmalkov, S. Micali, and M. Lepinski. Rational Secure Computation and Ideal Mechanism Design. *Proc. 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 585–595, IEEE, 2005.

[12] M. Lepinski, S. Micali, and A. Shelat. Fair Zero-Knowledge. *Theory of Cryptography Conference (TCC) 2005*, LNCS vol. 3378, pages 245–263, Springer, 2005.

[13] M. Lepinski, S. Micali, and A. Shelat. Collusion-Free Protocols. *Proc. 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 543–552, ACM, 2005.

[14] Y. Lindell. Protocols for Bounded-Concurrent Secure Two-Party Computation in the Plain Model. *Chicago Journal of Theoretical Computer Science* 2006(1): 1-50, 2006.

[15] Y. Lindell. Lower Bounds and Impossibility Results for Concurrent Self Composition. *Journal of Cryptology* 21(2): 200-249, 2008.

[16] R. Pass. Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. *Proc. 36th Anuual ACM Symposium on Theory of Computing (STOC)*, pages 232–241, ACM, 2004.

[17] R. Pass and A. Rosen Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. *Proc. 44th Anuual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 404–413, IEEE, 2003.

[18] G. Simmons. The Prisoners' Problem and the Subliminal Channel. *Advances in Cryptology — Crypto '83*, pages 51–67.

[19] G. Simmons. Cryptanalysis and Protocol Failures. *Comm. ACM* 37(11): 56–65, 1994.

[20] G. Simmons. The History of Subliminal Channels. *Information Hiding Workshop*, LNCS vol. 1174, pages 237–256, Springer, 1996.

A Definitions of Security for a Dishonest Mediator

In this section, we assume a dishonest mediator. The real-world model is, of course, unchanged. Since a corrupted mediator can allow arbitrary communication between corrupted parties, however, we treat all corrupted players as a single coordinated adversary $\mathcal{A}$ and syntactically re-write the definition that way. (Note that honest parties still have no direct communication with each other.) The definition that we present is otherwise the *standard* definition of security with abort, meaning that fairness and output delivery are not guaranteed. We allow any number of corruptions.

Let $\mathcal{I} \subset [n] \cup \{n+1\}$ denote the set of corrupted parties including the mediator, and denote by $\mathcal{H} = [n] \setminus \mathcal{I}$ the set of uncorrupted parties. An execution in the real world proceeds as follows:

PKI establishment: A PKI is established as before. We stress that, even though the mediator is dishonest, we continue to assume that a consistent PKI is established. (We view the PKI as something that is established long before protocol execution begins.)

Input determination and protocol execution: $\mathcal{A}$ is given the inputs $\{x_i\}_{i\in\mathcal{I}}$ of the corrupted parties, along with auxiliary information aux. Each honest P_i is given input x_i. Independent random coins $r_{\mathcal{A}}$ and $\{r_i\}_{i\in\mathcal{H}}$ for the parties are also chosen. The parties then execute the protocol with honest players acting as directed by Π, and the adversary $\mathcal{A}$ behaving arbitrarily.

Result of the experiment: At the conclusion of the protocol, let $\text{OUT}_{\mathcal{A}}$ denote the entire view of $\mathcal{A}$, and let OUT_i, for $i \in \mathcal{H}$, denote the final output of the honest P_i (as dictated by Π). Given an adversary $\mathcal{A}$, define

$$\text{REAL}_{\Pi,\mathcal{A}(\mathsf{aux})}(1^k, \vec{x}) \stackrel{\text{def}}{=} (\text{OUT}_{\mathcal{A}}, \{\text{OUT}_i\}_{i\in\mathcal{H}})$$

to be the random variable consisting of the stated outputs following an execution of Π where the parties are given inputs (and $\mathcal{A}$ given auxiliary input) as specified.

The second ideal world. Let $\mathcal{I}$ and $\mathcal{H}$ be as above. As in the previous ideal model, honest parties only communicate with the trusted party computing $\mathcal{F}$. Because the mediator is dishonest, corrupted parties may now freely communicate with each other and we therefore view these parties as a single coordinated adversary $\mathcal{S}$. An execution in this ideal world proceeds as follows:

PKI establishment: There is no PKI in this ideal world. Note that this is the usual convention when defining security of protocols, and the only reason we include a PKI in the ideal world defined in Section 2.2 is because there we needed to explicitly model information "leaked" via corrupted parties' public keys.

Input determination: The adversary $\mathcal{S}$ is given the set of inputs $\{x_i\}_{i\in\mathcal{I}}$ of the corrupted parties, along with some auxiliary information aux. Each honest party P_i is given its own input x_i. Independent random coins $r_{\mathcal{S}}$ are also chosen.

An honest party sets $x'_i = x_i$ and sends x'_i to $\mathcal{F}$. The adversary $\mathcal{S}$ sends $\{x'_i\}_{i\in\mathcal{I}}$ to $\mathcal{F}$, where these may be determined arbitrarily. $\mathcal{F}$ then hands $\{f_i(x'_1,\ldots,x'_{n+1})\}_{i\in\mathcal{I}}$ to $\mathcal{S}$, after which $\mathcal{S}$ specifies a set $\emptyset \subseteq \mathcal{H}' \subseteq \mathcal{H}$. Each honest party $i \in \mathcal{H}'$ is given $f_i(x'_1,\ldots,x'_{n+1})$, whereas each $i \in \mathcal{H}\setminus\mathcal{H}'$ is given $\perp$.

Note that output delivery, agreement, and fairness cannot be guaranteed regardless of the number of corruptions, since the corrupted mediator controls all communication.

Result of the experiment: At the conclusion of the protocol, let $\text{OUT}_{\mathcal{S}}$ be an arbitrary value output by $\mathcal{S}$, and let OUT_i, for $i \in \mathcal{H}$, denote the value given to P_i by $\mathcal{F}$. Define

$$\text{IDEAL}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(1^k,\vec{x}) \stackrel{\text{def}}{=} (\text{OUT}_{\mathcal{S}},\{\text{OUT}_i\}_{i\in\mathcal{H}})$$

to be the random variable consisting of the stated outputs following an ideal-world execution where the parties are given inputs and auxiliary inputs as specified.

Here, the definition takes the standard form:

Definition 2 *Let $\mathcal{F}$ be a functionality, and Π an $(n+1)$-party protocol computing $\mathcal{F}$ in the mediated model. Π is a* protocol securely computing *$\mathcal{F}$ if for all $\mathcal{I}$ and every probabilistic polynomial-time adversary $\mathcal{A}$ corrupting the parties in $\mathcal{I}$, there is a probabilistic polynomial-time adversary $\mathcal{S}$ corrupting the same parties with:*

$$\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{aux}\in(\{0,1\}^*)^{n+1},\,k\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\Pi,\mathcal{A}(\mathsf{aux})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{aux}\in(\{0,1\}^*)^{n+1},k\in\mathbb{N}}.$$

B A Protocol for Mediator Broadcast

We describe a protocol $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ implementing mediator broadcast, and prove its security. Parties begin holding the same values for sid and $\vec{pk}$, and each honest P_j holds a signing key sk_j. The mediator holds as input a message m of some fixed length, coins $r_1,\ldots,r_n$, and a set $\mathcal{H}$. (Note that for an honest mediator running $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ in our protocol, it will always be the case that $r_1,\ldots,r_n$ are chosen at random and $\mathcal{H}=[n]$.) Each party P_j holds a bit b_j. (For an honest party $b_j = 1$.) The protocol proceeds as follows:

1. P_{n+1} chooses random values $\rho'_1,\ldots,\rho'_n$. Then P_{n+1} runs $\mathcal{F}_{\mathsf{com}}$ with each P_j, where the mediator uses (m,ρ'_j) as its input. The output of P_j from $\mathcal{F}_{\mathsf{com}}$ is denoted com'_j.

2. P_{n+1} runs $\mathcal{F}_{\mathsf{Sign}}$ with each P_j, where P_{n+1} sends $(pk_j,(\mathsf{com}'_j,\mathsf{sid}))$ as its input. P_j sends $(sk_j,pk_j,(\mathsf{com}'_j,\mathsf{sid}))$ as its input if $b_j = 1$, and sends $\perp$ otherwise.

   Let σ_j be the output given by $\mathcal{F}_{\mathsf{Sign}}$ to P_{n+1}.

3. If $\sigma_i \neq \perp$ for all i, P_{n+1} sets $b = 1$ and then runs two instances of $\mathcal{F}_{\mathsf{com}}$ with each P_j:

   - P_{n+1} chooses random ρ_j and sends $((\mathsf{com}'_1,\sigma_1,\ldots,\mathsf{com}'_n,\sigma_n),\rho_j)$ to the first instance of $\mathcal{F}_{\mathsf{com}}$.

- If $j \in \mathcal{H}$, the mediator P_{n+1} sends (m, r_j) to the second instance of $\mathcal{F}_{\mathsf{com}}$. If $j \notin \mathcal{H}$, the mediator P_{n+1} sends (dummy, r_j) to the second instance of $\mathcal{F}_{\mathsf{com}}$.

If there exists an i with $\sigma_i = \perp$, P_{n+1} sets $b = 0$ and then runs two instances of $\mathcal{F}_{\mathsf{com}}$ with each P_j:

- P_{n+1} chooses random ρ_j and sends (dummy, ρ_j) to the first instance of $\mathcal{F}_{\mathsf{com}}$.
- P_{n+1} sends (dummy, r_j) to the second instance of $\mathcal{F}_{\mathsf{com}}$.

Let $\overline{\mathsf{com}}_j, \mathsf{com}_j$ denote the output of P_j from the first (resp., second) instance of $\mathcal{F}_{\mathsf{com}}$.

4. Define relation R, parameterized by sid and $\vec{pk}$, as follows: $R\left((\overline{\mathsf{com}}, \mathsf{com}),\ (\overline{\mathsf{dec}}, \vec{\mathsf{dec}}, \mathsf{dec})\right) = 1$ iff all the following hold:
   (a) $\overline{\mathsf{dec}}$ is a valid decommitment to $\overline{\mathsf{com}}$. Let $(\mathsf{com}'_1, \sigma_1, \ldots, \mathsf{com}'_n, \sigma_n)$ be the underlying message in this case.
   (b) dec is a valid decommitment to com. Let m be the underlying message in this case.
   (c) Either $m = \mathsf{dummy}$, or the following hold:
      i. $\vec{\mathsf{dec}}'$ contains valid decommitments to each of $\mathsf{com}'_1, \ldots, \mathsf{com}'_n$. Let m_i denote the message committed to by com'_i in this case.
      ii. $m = m_1 = \cdots = m_n$ and, for all i, we have $\mathsf{Vrfy}_{pk_i}((\mathsf{com}'_i, \mathsf{sid}), \sigma_i) = 1$.

   P_{n+1} runs $\mathcal{F}_{\mathsf{zk}}$ (for the above relation R) with each P_j. The input of P_j is $\overline{\mathsf{com}}_j, \mathsf{com}_j$, and the input of P_{n+1} contains these commitments and the appropriate decommitments.

5. Each party P_j, for $j \in [n]$, decides on its output as follows: if it receives '1' from $\mathcal{F}_{\mathsf{zk}}$, then it outputs com_j; otherwise, it outputs $C(\mathsf{dummy}; 0^k)$. The mediator P_{n+1} outputs b.

We highlight that the above protocol consists only of calls to the ideal functionalities $\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{Sign}}$, and $\mathcal{F}_{\mathsf{zk}}$; there are no other protocol messages. We now prove security of $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$:

Theorem B.1 *Let C be a perfectly binding commitment scheme, and let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a secure signature scheme. Then $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ is a collusion-free protocol for securely computing $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ in the $(\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{Sign}}, \mathcal{F}_{\mathsf{zk}})$-hybrid model.*

Proof: We consider separately the case of an honest mediator and a dishonest mediator.

Claim B.2 *Let C be a perfectly binding commitment scheme. Then $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ is a collusion-free protocol computing $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ in the $(\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{Sign}}, \mathcal{F}_{\mathsf{zk}})$-hybrid model.*

Proof: Here we have an honest mediator. Let $\mathcal{I}$ denote the set of corrupted parties. We need to show independent transformations $\{\mathsf{Sim}_i\}_{i \in \mathcal{I}}$ that satisfy Definition 1. In fact, our transformations will be black-box. Sim_i, given oracle access to P_i, does as follows:

0. Sim_i runs P_i to generate pk_i. Then Sim_i is given the vector of public keys $\vec{pk} = (pk_1, \ldots, pk_n)$, a bit b_i, and an auxiliary input aux_i, and it runs P_i on these inputs.

1. Sim_i simulates an ideal call to $\mathcal{F}_{\mathsf{com}}$ by giving P_i a commitment com'_i to a 0-string of the appropriate length.

2. Sim_i simulates an ideal call to $\mathcal{F}_{\mathsf{Sign}}$ by extracting inputs $(sk_{i,1}, pk_{i,1}, (\mathsf{com}'_{i,1}, \mathsf{sid}_{i,1}))$ from P_i, and returning output $\perp$ to P_i. Then:

   - If $(pk_{i,1}, \mathsf{com}'_{i,1}, \mathsf{sid}_{i,1}) \neq (pk_i, \mathsf{com}'_i, \mathsf{sid})$ or $(sk_{i,1}, pk_i) \notin \mathsf{Range}(\mathsf{Gen})$ (these include the case when P_i's input is $\perp$), then Sim_i sends 0 to the trusted party computing $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$.
   - Otherwise, Sim_i sends 1 to the trusted party computing $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$.

   Sim_i receives in return a commitment com_i.

3. Sim_i simulates the first ideal call to $\mathcal{F}_{\mathsf{com}}$ by giving P_i a commitment $\overline{\mathsf{com}}_i$ to a 0-string of the appropriate length. It simulates the second call to $\mathcal{F}_{\mathsf{com}}$ by giving P_i the commitment com_i.

4. Sim_i then simulates an ideal call to $\mathcal{F}_{\mathsf{zk}}$ by extracting $(\overline{\mathsf{com}}_{i,2}, \mathsf{com}_{i,2})$ from P_i. Then:

   - If $(\overline{\mathsf{com}}_{i,2}, \mathsf{com}_{i,2}) \neq (\overline{\mathsf{com}}_i, \mathsf{com}_i)$, then Sim_i returns 0 to P_i.
   - Otherwise, Sim_i returns 1 to P_i

   Finally, Sim_i outputs whatever P_i outputs, and halts.

It is straightforward to see that Definition 1 holds for the $\{\mathsf{Sim}_i\}_{i\in\mathcal{I}}$ defined as above. Indeed, the only difference between the hybrid-world execution of protocol $\Pi_{\mathsf{bcast}}^{\mathsf{sid}}$ and the ideal-world execution of $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$ is with regard to the commitments $\{\mathsf{com}'_i, \overline{\mathsf{com}}_i\}_{i\in\mathcal{I}}$. Since, when the mediator is honest, these commitments are generated *independently* with fresh random coins in each experiment, computational hiding of C implies that the two experiments are indistinguishable. ■

Claim B.3 *Let C be a perfectly binding commitment scheme, and let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a secure signature scheme. Then $\Pi_{\mathsf{bcast}}^{\mathsf{sid}}$ securely computes $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$ in the $(\mathcal{F}_{\mathsf{com}}, \mathcal{F}_{\mathsf{Sign}}, \mathcal{F}_{\mathsf{zk}})$-hybrid model.*

Proof: We now have a dishonest mediator. Let $\mathcal{I}$ denote the set of corrupted parties, including the mediator, and let $\mathcal{H}$ denote the honest parties. In the proof that follows, we assume that every honest party P_i holds input $b_i = 1$; this is justified since, in our protocol that makes calls to $\Pi_{\mathsf{bcast}}^{\mathsf{sid}}$, this will always be the case.

Given an adversary $\mathcal{A}$ controlling the parties in $\mathcal{I}$, we construct a simulator $\mathcal{S}$ with black-box access to $\mathcal{A}$ that works as follows:

0. For each $i \in \mathcal{H}$, the simulator $\mathcal{S}$ runs $(sk_i, pk_i) \leftarrow \mathsf{Gen}(1^k)$. The simulator runs $\mathcal{A}$ to obtain the public keys $\{pk_i\}_{i\in\mathcal{I}}$. Let $\vec{pk} = (pk_1, \ldots, pk_n)$ denote the vector of public keys.

   $\mathcal{S}$ is given inputs $\{x_i\}_{i\in\mathcal{I}}$ and auxiliary input aux. It runs $\mathcal{A}$ on these inputs and $\vec{pk}$.

1. For each $j \in \mathcal{H}$, the simulator $\mathcal{S}$ simulates an ideal call to $\mathcal{F}_{\mathsf{com}}$ by extracting (m_j, ρ'_j) from $\mathcal{A}$. Let $\mathsf{com}'_j = C(m_j; \rho'_j)$.

   If all the $\{m_j\}_{j\in\mathcal{H}}$ are equal, then $\mathcal{S}$ sets $m^* = m_1$. Otherwise $\mathcal{S}$ sets $m^* = \mathsf{dummy}$.

2. For each $j \in \mathcal{H}$, the simulator simulates an ideal call to $\mathcal{F}_{\mathsf{Sign}}$ as follows: it first extracts $(pk_{j,n+1}, (\mathsf{com}'_{j,n+1}, \mathsf{sid}_{n+1}))$ from $\mathcal{A}$ and then:

   - If $(pk_{j,n+1}, (\mathsf{com}'_{j,n+1}, \mathsf{sid}_{n+1})) = (pk_j, (\mathsf{com}'_j, \mathsf{sid}))$, then $\mathcal{S}$ computes the signature $\sigma_j = \mathsf{Sign}_{sk_j}(\mathsf{com}'_j, \mathsf{sid})$ and gives σ_j to $\mathcal{A}$ as the output of $\mathcal{F}_{\mathsf{Sign}}$.
   - Otherwise, $\mathcal{S}$ gives $\perp$ to $\mathcal{A}$ as the output of $\mathcal{F}_{\mathsf{Sign}}$.

3. For each $j \in \mathcal{H}$, the simulator $\mathcal{S}$ simulates two ideal calls to $\mathcal{F}_{\mathsf{com}}$. Let $(\bar{m}_j, \rho_j)$ denote the inputs submitted by $\mathcal{A}$ to the first instance of $\mathcal{F}_{\mathsf{com}}$, and let $\overline{\mathsf{com}}_j = C(\bar{m}_j; \rho_j)$. Let (m'_j, r_j) denote the inputs submitted by $\mathcal{A}$ to the second instance of $\mathcal{F}_{\mathsf{com}}$, and let $\mathsf{com}_j = C(m'_j; r_j)$.

4. $\mathcal{S}$ initializes $\mathcal{H}' = \mathcal{H}$. Then for each $j \in \mathcal{H}$ it does:

   (a) Extract the inputs $(\overline{\mathsf{dec}}_j, \vec{\mathsf{dec}}'_j, \mathsf{dec}_j)$ submitted by $\mathcal{A}$ to the relevant instance of $\mathcal{F}_{\mathsf{zk}}$.

   (b) If $\mathcal{F}_{\mathsf{zk}}$ would output 1 to P_j (note that $\mathcal{S}$ can easily compute this, since an honest P_j would input $(\overline{\mathsf{com}}_j, \mathsf{com}_j)$ to $\mathcal{F}_{\mathsf{zk}}$), then
      - If $m'_j = m^*$, do nothing.
      - If $m'_j = \mathsf{dummy}$, remove j from $\mathcal{H}'$.
      - If $m'_j \notin \{m^*, \mathsf{dummy}\}$, set $\mathsf{fail} = \mathsf{true}$.

   (c) If $\mathcal{F}_{\mathsf{zk}}$ would output 0 to P_j, then set $r_j = 0^k$ and remove j from $\mathcal{H}'$.

5. $\mathcal{S}$ sends $m, r_1, \ldots, r_n, \mathcal{H}'$ to $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$ on behalf of P_{n+1}, and sends 1 to $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$ on behalf of the other parties in $\mathcal{I}$. It then outputs whatever $\mathcal{A}$ outputs, and halts.

It is straightforward to see that $\mathcal{S}$, as above, satisfies Definition 2 (under the assumption that every honest party P_i holds input $b_i = 1$). Indeed, the view of $\mathcal{A}$ in a hybrid-world execution of protocol $\Pi_{\mathsf{bcast}}^{\mathsf{sid}}$ is distributed identically to its view when run as a subroutine by $\mathcal{S}$ in an ideal-world execution of $\mathcal{F}_{\mathsf{bcast}}^{\mathsf{sid}}$. The only difference is with regard to the outputs of the honest parties, where a difference occurs only in case fail is set to true at the end of $\mathcal{S}$'s execution. (If fail is false then in the hybrid world every honest party P_j outputs a commitment com_j to a message $m_j \in \{m^*, \mathsf{dummy}\}$, just as in the ideal world.) But it is easy to see what whenever fail is set to true then $\mathcal{A}$ has forged a valid signature with respect to one of the honest party's public keys. ■

The two claims above prove the theorem. ■

C Proof of Theorem 4.1

We prove Theorem 4.1 by separately proving that Π satisfies Definitions 1 and 2.

C.1 Collusion-Freeness of Π (Honest Mediator)

Let $\mathcal{I}$ denote the set of corrupted parties. We show independent transformations $\{\mathsf{Sim}_j\}_{j \in \mathcal{I}}$ that satisfy Definition 1. The proof is quite straightforward, given the construction of Π, since parties essentially learn nothing until the final round of the protocol. We remark that, in this proof, we rely only on the hiding property of the commitment scheme C.

Sim_j, given oracle access to P_j, does as follows:

1. Sim_j runs P_j to generate pk_j. Then Sim_j is given the vector of public keys $(pk_1, \ldots, pk_n)$, an input x_j, and auxiliary input aux_j. It runs P_j on these inputs.

2. Sim_j simulates $\mathcal{F}_{\mathsf{com}}$ by extracting $\mathsf{dec}_j^{\mathsf{input}} = (x'_j, s_j)$ from P_j. Let $\mathsf{com}_j^{\mathsf{input}} = C(x'_j; s_j)$.

3. Sim_j simulates an ideal call to $\mathcal{F}_{\mathsf{ct}}$ by choosing random $\mathsf{dec}_j^{\mathsf{rand}} = (r_j, s'_j)$ and giving these values to P_j. Let $\mathsf{com}_j^{\mathsf{rand}} = C(r_j; s'_j)$.

4. Initialize $\mathsf{abort} = \mathsf{false}$. Then for $i = 1$ to r:

(a) If it is P_j's turn to speak in π, then Sim_j simulates a call to $\mathcal{F}^{\pi}_{\mathsf{compute}}$ as follows:
- Let $\vec{C}_j = (\mathsf{com}^j_1, \ldots, \mathsf{com}^j_{i-1})$ be the commitments given to P_j in the previous rounds.
- Extract inputs $(\overline{\mathsf{com}}^{\mathsf{input}}, \overline{\mathsf{com}}^{\mathsf{rand}}, \overline{C}, \mathsf{rid}, \overline{\mathsf{dec}}^{\mathsf{input}}, \overline{\mathsf{dec}}^{\mathsf{rand}}, sk_j)$ from P_j.
- If $(\overline{\mathsf{com}}^{\mathsf{input}}, \overline{\mathsf{com}}^{\mathsf{rand}}, \overline{C}, \overline{\mathsf{dec}}^{\mathsf{input}}, \overline{\mathsf{dec}}^{\mathsf{rand}}) \neq (\mathsf{com}^{\mathsf{input}}_j, \mathsf{com}^{\mathsf{rand}}_j, \vec{C}_j, \mathsf{dec}^{\mathsf{input}}_j, \mathsf{dec}^{\mathsf{rand}}_j)$, or $\mathsf{rid} \neq 0i$, or $(sk_j, pk_j) \notin \mathsf{Range}(\mathsf{Gen})$, then set $\mathsf{abort} = \mathsf{true}$.
- In any case, Sim_j returns $\perp$ to P_j.

(b) Sim_j simulates a call to $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ by extracting a bit b_j from P_j. If $b_j = 0$ then Sim_j sets $\mathsf{abort} = \mathsf{true}$. In any case, Sim_j returns to P_j a commitment com^j_i to a 0-string of the appropriate length.

5. If $\mathsf{abort} = \mathsf{false}$ then Sim_j sends x'_i to the trusted party computing $\mathcal{F}$. Otherwise, Sim_j sends $\perp$ to the trusted party computing $\mathcal{F}$. In return, Sim_j is given output OUT_j.

6. To complete the simulation, Sim_j simulates a call to $\mathcal{F}^{\pi}_{\mathsf{output}}$:
- Let $\vec{C}_j = (\mathsf{com}^j_1, \ldots, \mathsf{com}^j_r)$.
- Extract $(\overline{\mathsf{com}}^{\mathsf{input}}, \overline{\mathsf{com}}^{\mathsf{rand}}, \overline{C}, \overline{\mathsf{dec}}^{\mathsf{input}}, \overline{\mathsf{dec}}^{\mathsf{rand}})$ from P_j. If this is not equal to $(\mathsf{com}^{\mathsf{input}}_j, \mathsf{com}^{\mathsf{rand}}_j, \vec{C}_j, \mathsf{dec}^{\mathsf{input}}_j, \mathsf{dec}^{\mathsf{rand}}_j)$ then Sim_j gives $\perp$ to P_j. Otherwise, Sim_j gives OUT_j to P_j. It then halts and outputs whatever P_j outputs.

It is straightforward to see that Definition 1 holds for the $\{\mathsf{Sim}_j\}_{j\in\mathcal{I}}$ defined above. Indeed, the only difference between the hybrid-world experiment in which Π is run, and the ideal-world execution with ideal functionality computing $\mathcal{F}$, is with regard to the commitments $\{\mathsf{com}^j_i\}_{i\in[r],j\in\mathcal{I}}$. But since these commitments are computed *independently* with fresh random coins in each experiment, the hiding property of C immediately implies that these two experiments are indistinguishable.

C.2 Security of Π (Corrupt Mediator)

In this section we prove that Π satisfies Definition 2; i.e., that it securely computes $\mathcal{F}$ in the case of a dishonest mediator, in a hybrid model where a trusted party computes the functionalities $\mathcal{F}_{\mathsf{com}}$, $\mathcal{F}_{\mathsf{ct}}$, $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$, $\mathcal{F}^{\pi}_{\mathsf{compute}}$ and $\mathcal{F}^{\pi}_{\mathsf{output}}$. We are thus given a single, coordinated adversary $\mathcal{A}$ a set of corrupted parties $\mathcal{I}$ that includes the mediator, and we construct a single simulator $\mathcal{S}$ controlling the same parties. The simulator $\mathcal{S}$ uses the *black-box* simulator $\mathcal{S}_\pi$ that is guaranteed to exist for the protocol π that securely computes $\mathcal{F}$ in the stand-alone model.

We let $\mathcal{H} = [n] \setminus \mathcal{I}$ denote the set of honest parties. Let z denote the auxiliary input received by the adversary and let x_i be the input of party P_i. The simulator $\mathcal{S}$ works as follows:

Simulation of the PKI setup:

1. For each $i \in \mathcal{H}$, the simulator $\mathcal{S}$ runs $(sk_i, pk_i) \leftarrow \mathsf{Gen}(1^k)$. It also runs $\mathcal{A}$ to obtain the public keys $\{pk_i\}_{i\in\mathcal{I}}$. Let $\vec{pk} = (pk_1, \ldots, pk_n)$ denote the vector of public keys.

Simulation of input commitment/coin-tossing stage:

1. $\mathcal{S}$ invokes $\mathcal{A}$ with inputs $\{x_i\}_{i\in\mathcal{I}}$, auxiliary input z, and the vector of public keys $\vec{pk}$.

2. For every $j \in \mathcal{H}$, the simulator $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{com}}$ by sending to P_{n+1} the commitment $\mathsf{com}^{\mathsf{input}}_j = C(0^k; s_j)$ for a random s_j. (Since $\mathcal{A}$ controls P_{n+1}, from now on we will refer to $\mathcal{S}$ handing messages directly to $\mathcal{A}$ and not to P_{n+1}.)

3. For every $j \in \mathcal{H}$, the simulator $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{ct}}$ by handing $\mathcal{A}$ the commitment $\mathsf{com}_j^{\mathsf{rand}} = C(0^\ell; s'_j)$, where ℓ is the length of the random-tape that is generated by $\mathcal{F}_{\mathsf{ct}}$.

Simulation of the round-by-round emulation stage:
$\mathcal{S}$ invokes $\mathcal{S}_\pi$ to simulate the messages of π to $\mathcal{A}$; recall that $\mathcal{S}_\pi$ is a black-box simulator and so it does not receive any auxiliary input, and works by querying the adversary with vectors of messages sent in the protocol. We let $(\alpha_1, \ldots, \alpha_\ell)$ denote the messages corresponding to the first ℓ rounds of the protocol. If j denotes a round of π in which a malicious party speaks, we may assume that $\mathcal{S}_\pi$ does not query the adversary with $(\alpha_1, \ldots, \alpha_j, \ldots, \alpha_\ell)$ unless it has previously queried the adversary with vector $(\alpha_1, \ldots, \alpha_{j-1})$ and received message α_j in return. We further assume that whenever $\mathcal{S}_\pi$ sends a vector of length ℓ to the adversary then it is always a corrupted party who speaks in round $\ell + 1$. Finally, we assume that $\mathcal{S}_\pi$'s output is a vector containing all r messages sent in the protocol.

$\mathcal{S}$ runs $\mathcal{S}_\pi$ as follows:

- When $\mathcal{S}_\pi$ sends a vector of messages $(\alpha_1, \ldots, \alpha_\ell)$ to its adversary in the simulation of π, let $\ell' < \ell$ denote the largest index such that a malicious party speaks in round ℓ'. Thus, the values $\alpha_{\ell'+1}, \ldots, \alpha_\ell$ all denote messages that are sent by honest parties in π.

  1. $\mathcal{S}$ rewinds $\mathcal{A}$ to the point where it received response $\alpha_{\ell'}$. Then $\mathcal{S}$ simulates rounds $\ell' + 1$ through ℓ of the round-by-round emulation phase as follows. For $i = \ell' + 1, \ldots, \ell$:
     (a) Say honest party P_j speaks in round i. $\mathcal{S}$ first extracts $\mathcal{A}$'s input to the relevant instance of $\mathcal{F}^\pi_{\mathsf{compute}}$. Then:
        - If the input is not valid (i.e., would cause $\mathcal{F}^\pi_{\mathsf{compute}}$ to send $\perp$ as output), then $\mathcal{S}$ returns $\perp$ to $\mathcal{A}$.
        - Let $(\mathsf{msg}_1, \sigma_1), \ldots, (\mathsf{msg}_{i-1}, \sigma_{i-1})$ denote the committed values in the vector $\vec{C}_j$ submitted by $\mathcal{A}$ (cf. step 3 of $\mathcal{F}^\pi_{\mathsf{compute}}$ in Figure 1). If $\mathsf{msg}_k \neq \alpha_k$ for some k, then $\mathcal{S}$ also returns $\perp$ to $\mathcal{A}$. (This is different than what would happen in an execution of Π, but implies that $\mathcal{A}$ has forged a signature of an honest party.)
        - Otherwise, $\mathcal{S}$ returns (α_i, σ_i) to $\mathcal{A}$, where $\sigma_i = \mathsf{Sign}_{sk_j}(\alpha_i, 0i)$.

     (b) $\mathcal{S}$ then simulates a call to $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ by extracting the relevant inputs from $\{P_j\}_{j\in\mathcal{I}}$, returning the relevant outputs to $\{P_j\}_{j\in\mathcal{I}}$, and computing outputs for the honest parties in the obvious way.
  2. In round $\ell + 1$ it is the turn of some corrupted party to speak. Say that the next round in which an honest party speaks is round $\ell'' + 1$, and that honest party P_k speaks in that round. Then for $i = \ell + 1$ to ℓ'', the simulator $\mathcal{S}$ simulates calls to $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ by extracting the relevant inputs from $\{P_j\}_{j\in\mathcal{I}}$, handing the relevant outputs to $\{P_j\}_{j\in\mathcal{I}}$, and computing the outputs of $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ for P_k; these will be a sequence of commitments $\mathsf{com}^k_{\ell+1}, \ldots, \mathsf{com}^k_{\ell''}$. From the inputs to all the calls of $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$, the simulator can determine the messages $(\alpha_{\ell+1}, \sigma_{\ell+1}), \ldots, (\alpha_{\ell''}, \sigma_{\ell''})$ underlying these commitments. If any of these are dummy commitments, these are treated as an abort. $\mathcal{S}$ returns $(\alpha_{\ell+1}, \ldots, \alpha_{\ell''})$ to $\mathcal{S}_\pi$.

- When $\mathcal{S}_\pi$ wishes to send inputs $\{x'_i\}_{i\in\mathcal{I}}$ to the trusted party computing $\mathcal{F}$, then $\mathcal{S}$ sends $\{x'_i\}_{i\in\mathcal{I}}$ to its trusted party and hands to $\mathcal{S}_\pi$ the outputs $\{\text{OUT}_i\}_{i\in\mathcal{I}}$ it receives in return.

- When $\mathcal{S}_\pi$ outputs the final vector of messages $(\alpha_1, \ldots, \alpha_r)$, then $\mathcal{S}$ rewinds $\mathcal{A}$ to the point where it received response α_r and proceeds to the next step of the simulation.

Simulation of output determination stage:

1. $\mathcal{S}$ initializes $\mathcal{H}' = \mathcal{H}$. For each honest party P_j, the simulator $\mathcal{S}$ simulates $\mathcal{F}^{\pi}_{\mathsf{output}}$ with $\mathcal{A}$.
   - If the inputs of $\mathcal{A}$ would cause $\mathcal{F}^{\pi}_{\mathsf{output}}$ to send $\perp$, then $\mathcal{S}$ removes j from $\mathcal{H}'$.
   - Let $(\mathsf{msg}_1, \sigma_1), \ldots, (\mathsf{msg}_r, \sigma_r)$ denote the committed values in the vector $\vec{C}_j$ submitted by $\mathcal{A}$ (cf. step 3 of $\mathcal{F}^{\pi}_{\mathsf{output}}$ in Figure 2). If $\mathsf{msg}_k \neq \alpha_k$ for some k, then $\mathcal{S}$ also removes j from $\mathcal{H}'$. (Again, this event implies that $\mathcal{A}$ has forged a signature of an honest party.)
2. $\mathcal{S}$ sends $\mathcal{H}'$ to the trusted party and halts, outputting whatever $\mathcal{A}$ outputs.

We let $\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(1^k, \vec{x})$ denote the output of the honest parties and $\mathcal{S}$ in the ideal-world execution above, and let $\text{HYB}_{\Pi,\mathcal{A}(z)}(1^k, \vec{x})$ denote the output of the honest parties and $\mathcal{A}$ in the hybrid-model execution of Π. We sketch the proof that these two experiments are computationally indistinguishable. The proof proceeds using two hybrid experiments:

Hybrid H_1: Here we execute $\text{HYB}_{\Pi,\mathcal{A}(z)}(1^k, \vec{x})$, except that we change the way $\mathcal{F}^{\pi}_{\mathsf{compute}}$ and $\mathcal{F}^{\pi}_{\mathsf{output}}$ are defined. Specifically, for a round i in which an honest party speaks in π, let msg^*_i denote the message output by the invocation of $\mathcal{F}^{\pi}_{\mathsf{compute}}$ in that round. (I.e., msg^*_i is the round-i message generated by an honest party in π via the call to $\mathcal{F}^{\pi}_{\mathsf{compute}}$.) Then we add a step between steps 3 and 4 of both $\mathcal{F}^{\pi}_{\mathsf{compute}}$ and $\mathcal{F}^{\pi}_{\mathsf{output}}$ as follows:

- If there is an i where an honest party speaks in round i but $\mathsf{msg}_i \neq \mathsf{msg}^*_i$, then return $\perp$ to P_{n+1}.

It is not hard to see that the outcome of this experiment is statistically close to $\text{HYB}_{\Pi,\mathcal{A}(z)}(1^k, \vec{x})$, since the above rule is only applied when $\mathcal{A}$ has forged a signature of an honest party.

Hybrid H_2: This experiment is the same as H_1, except that we substitute commitments to garbage for the outputs of $\mathcal{F}_{\mathsf{com}}$ and $\mathcal{F}_{\mathsf{ct}}$ in the input commitment/coin-tossing phase. It is straightforward to show that the output distributions of H_2 and H_1 are computationally indistinguishable by reduction to the hiding property of the commitment scheme.

The indistinguishability of the output distributions in H_2 and $\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(1^k, \vec{x})$ follows from the assumption that $\mathcal{S}_\pi$ is a "good" simulator for π. In particular, given a vector of messages $\vec{\alpha} = (\alpha_1, \ldots, \alpha_r)$, a distinguisher D can run the same steps as in experiment H_2, but using $\vec{\alpha}$ instead of generating the π-messages itself. Now, if $\vec{\alpha}$ is generated by a real execution of π, then the experiment is exactly H_2. In contrast, if $\vec{\alpha}$ is the output of $\mathcal{S}_\pi$, then the experiment is exactly $\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(1^k, \vec{x})$. This completes the proof.

D Composition Theorems in the Mediated Model

Once again, we deal separately with the case of an honest mediator and a dishonest mediator.

D.1 Proof of Theorem 4.2

Here we have an honest mediator, and prove a composition theorem for the property of collusion-freeness. A proof of the theorem follows exactly along the lines of [5], and so we only sketch the details. We assume for simplicity that Π makes only a single call to ρ though, as in [5], the proof easily extends to the general case.

Let $\mathcal{I} \subset [n]$ denote a set of corrupted parties, and let $\{\mathcal{A}_i\}_{i\in\mathcal{I}}$ denote a collection of probabilistic polynomial time adversaries executing the real-world protocol Π^ρ. We first construct a collection $\{\mathcal{A}_i^\rho\}_{i\in\mathcal{I}}$ of adversaries attacking the real-world protocol ρ. Adversary $\mathcal{A}_i^\rho$ is defined as follows:

> $\mathcal{A}_i^\rho$ ignores its input, and views its auxiliary input as an internal state of $\mathcal{A}_i$. It then runs $\mathcal{A}_i$ from this state, with messages to/from the real-world mediator relayed to $\mathcal{A}_i$. When $\mathcal{A}_i$ halts, $\mathcal{A}_i^\rho$ outputs the current internal state of $\mathcal{A}_i$ and halts.

Collusion-freeness of ρ implies the existence of a set of efficient transformations $\{\mathsf{Sim}_i^\rho\}_{i\in\mathcal{I}}$ such that, setting $\mathcal{S}_i^{\mathcal{G}} = \mathsf{Sim}_i^\rho(\mathcal{A}_i^\rho)$, we obtain adversaries $\{\mathcal{S}_i^{\mathcal{G}}\}_{i\in\mathcal{I}}$ running in an ideal-world computation of $\mathcal{G}$ for which

$$\left\{\text{IDEAL}^{\mathsf{cf}}_{\mathcal{G},\mathcal{S}_{\mathcal{I}}^{\mathcal{G}}(\mathsf{a\vec{u}x})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{a\vec{u}x}\in(\{0,1\}^*)^{n+1},\,k\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\text{REAL}^{\mathsf{mediated}}_{\rho,\mathcal{A}_{\mathcal{I}}^\rho(\mathsf{a\vec{u}x})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{a\vec{u}x}\in(\{0,1\}^*)^{n+1},\,k\in\mathbb{N}}. \quad (1)$$

Next, we construct adversaries $\{\mathcal{A}_i^\Pi\}_{i\in\mathcal{I}}$ running Π in the $\mathcal{G}$-hybrid model. Adversary $\mathcal{A}_i^\Pi$ is defined as follows:

- Given input x_i and auxiliary input aux_i, run $\mathcal{A}_i$ on these inputs up to the round in which $\mathcal{G}$ is called.
- Run $\mathcal{S}_i^{\mathcal{G}}$ using arbitrary input but with the auxiliary input set equal to the current state of $\mathcal{A}_i$. When $\mathcal{S}_i^{\mathcal{G}}$ sends input x_i' to its trusted party computing $\mathcal{G}$, use this input in the current call to $\mathcal{G}$ and return the output from $\mathcal{G}$ to $\mathcal{S}_i^{\mathcal{G}}$.
- The output of $\mathcal{S}_i^{\mathcal{G}}$ is an internal state of $\mathcal{A}_i$. Run $\mathcal{A}_i$ from this state until the end of the protocol, then output whatever $\mathcal{A}_i$ outputs and halt.

As in [5], it follows from Equation (1) that

$$\left\{\text{HYB}^{\mathcal{G}}_{\Pi,\mathcal{A}_{\mathcal{I}}^\Pi(\mathsf{a\vec{u}x})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{a\vec{u}x}\in(\{0,1\}^*)^{n+1},\,k\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\text{REAL}^{\mathsf{mediated}}_{\Pi^\rho,\mathcal{A}_{\mathcal{I}}(\mathsf{a\vec{u}x})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{a\vec{u}x}\in(\{0,1\}^*)^{n+1},\,k\in\mathbb{N}}.$$

Collusion-freeness of Π in the $\mathcal{G}$-hybrid model implies the existence of a set of efficient transformations $\{\mathsf{Sim}_i^\Pi\}_{i\in\mathcal{I}}$ such that, setting $\mathcal{S}_i^\Pi = \mathsf{Sim}_i^\Pi(\mathcal{A}_i^\Pi)$, we obtain adversaries $\{\mathcal{S}_i^\Pi\}_{i\in\mathcal{I}}$ running in an ideal-world computation of $\mathcal{F}$ for which

$$\left\{\text{HYB}^{\mathcal{G}}_{\Pi,\mathcal{A}_{\mathcal{I}}^\Pi(\mathsf{a\vec{u}x})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{a\vec{u}x}\in(\{0,1\}^*)^{n+1},\,k\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\text{IDEAL}^{\mathsf{cf}}_{\mathcal{F},\mathcal{S}_{\mathcal{I}}^\Pi(\mathsf{a\vec{u}x})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{a\vec{u}x}\in\in(\{0,1\}^*)^{n+1},\,k\in\mathbb{N}},$$

and thus

$$\left\{\text{IDEAL}^{\mathsf{cf}}_{\mathcal{F},\mathcal{S}_{\mathcal{I}}^\Pi(\mathsf{a\vec{u}x})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{a\vec{u}x}\in\in(\{0,1\}^*)^{n+1},\,k\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\text{REAL}^{\mathsf{mediated}}_{\Pi^\rho,\mathcal{A}_{\mathcal{I}}(\mathsf{a\vec{u}x})}(1^k,\vec{x})\right\}_{\vec{x},\mathsf{a\vec{u}x}\in(\{0,1\}^*)^{n+1},\,k\in\mathbb{N}},$$

as desired. To complete the proof, we need only argue that $\mathcal{S}_i^\Pi$ depends only on $\mathcal{A}_i$ (and not the entire collection $\{\mathcal{A}_i\}_{i\in\mathcal{I}}$). Following the chain of constructions above, we see that this is the case:

- By construction, $\mathcal{A}_i^\rho$ depends only on $\mathcal{A}_i$.
- By collusion-freeness of ρ, we have that $\mathcal{S}_i^{\mathcal{G}}$ depends only on $\mathcal{A}_i^\rho$.
- By construction, $\mathcal{A}_i^\Pi$ depends only on $\mathcal{S}_i^{\mathcal{G}}$ and $\mathcal{A}_i$.
- By collusion-freeness of Π in the $\mathcal{G}$-hybrid model, $\mathcal{S}_i^\Pi$ depends only on $\mathcal{A}_i^\Pi$.

D.2 Proof of Theorem 4.3

Let Π be a protocol that securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model for concurrent self composition, where Π makes m calls to $\mathcal{G}$ and P_{n+1} plays the role of the second party in all calls to $\mathcal{G}$. Let ρ be a two-party protocol that securely computes $\mathcal{G}$ under m-bounded concurrent self-composition. (See Appendix E for appropriate definitions.) We need to prove that the composed protocol Π^ρ satisfies Definition 2; that is, that it is secure when the mediator is dishonest. A proof of this statement boils down to the observation that, when considering a malicious mediator, there are only syntactic differences between the experiment in which Π^ρ is run in the real mediated model and the experiment in which multiple concurrent executions of ρ are run. Specifically, the instructions of Π for each honest party define a set of input-selecting machines as in the definition of concurrent self composition (see Appendix E). Furthermore, since Π makes at most m calls to $\mathcal{G}$, the security of ρ under m-bounded concurrent self composition implies that for every $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that the output of an execution of Π^ρ in the real model with adversary $\mathcal{A}$ is computationally indistinguishable from an execution of Π with $\mathcal{S}$ in the $\mathcal{G}$-hybrid model. By the assumed security of Π in the $\mathcal{G}$-hybrid model, we then have that for every $\mathcal{S}$ there exists an $\mathcal{S}'$ such that the output of an execution of Π with $\mathcal{S}$ in the $\mathcal{G}$-hybrid model is indistinguishable from the output of an ideal execution with $\mathcal{S}'$ and a trusted party computing $\mathcal{F}$. Combining these results, we conclude that Π^ρ securely computes $\mathcal{F}$ as required.

E Security Under Bounded-Concurrent Self Composition

In this section we present the definitions for security under m-bounded concurrent self composition [15]. Loosely speaking, this relates to a setting where a single protocol computing a functionality $\mathcal{F} = (f_1, f_2)$ is run m times concurrently, and is the only protocol being run. We focus only on two-party protocols, as this is all we use in this work.

Overview. Fix a two-party functionality $\mathcal{F} = (f_1, f_2)$. Security of a protocol ρ computing $\mathcal{F}$ is, as usual, analyzed by comparing what an adversary can do in the real world to what it can do in an ideal world that is clearly secure. In both worlds, we will consider parties who perform multiple (concurrent) computations of $\mathcal{F}$. In the real world, these computations are performed by having the parties run multiple executions of ρ; in the ideal world, these computations are done by having the parties interact in multiple sessions with a trusted party who computes $\mathcal{F}$ on their behalf.

We assume an adversary who takes part in all computations of $\mathcal{F}$, always playing the role of P_2. We also have several parties $P_1^1, P_1^2, \ldots$ taking part in various computations of $\mathcal{F}$, always playing the role of P_1. In both the real and ideal worlds, the adversary may coordinate its actions in the various executions/sessions, but each honest P_1^j may choose its input depending on prior outputs it has received, but otherwise acts independently in each execution/session.

The process by which an honest P_1^j determines its inputs for the various computations of $\mathcal{F}$ is modeled by a probabilistic polynomial-time Turing machine M_1^j called an *input-selecting machine*. The input for the next computation is determined as a function of the initial input x^j of P_1^j, the number of computations of $\mathcal{F}$ initiated by P_1^j thus far, and any outputs that were obtained by P_1^j from previous computations of $\mathcal{F}$ that have already concluded. We assume M_1^j always produces input values of the same fixed, known length (that is polynomial in k).

Concurrent computation in the ideal model. We have an adversary $\mathcal{S}$ who always plays the role of P_2. The ideal world execution proceeds as follows:

Inputs: Each P_1^j is given initial input x^j; $\mathcal{S}$ is given initial input y and auxiliary input z.

Session initiation: $\mathcal{S}$ initiates a new session by sending a $(\mathsf{start}-\mathsf{session}, j)$ message to the trusted party. The trusted party then sends $(\mathsf{start\text{-}session}, i)$ to P_1^j, where i is the index of the session (i.e., this is the ith session to be initiated with P_1^j).

Honest party sends input to trusted party: Upon receiving $(\mathsf{start\text{-}session}, i)$ from the trusted party, the honest party P_1^j applies its input-selecting machine M_1^j to its initial input x^j, the session number i, and any outputs it has received in previous sessions. This results in a value x_i^j, and P_1^j sends (i, x_i^j) to the trusted party.

$\mathcal{S}$ sends input to the trusted party and receives output: Whenever the adversary wishes, it may send a message (j, i, y_i^j) to the trusted party, for any y_i^j of its choice. Upon sending this pair, it receives back $(j, i, f_2(x_i^j, y_i^j))$ where x_i^j is the value that P_1^j previously sent the trusted party. (If i start-session messages have not yet been sent to the trusted party, then the (j, i, y_i^j) message from the adversary is ignored. In addition, once an input indexed by j, i has already been sent by the adversary, the trusted party ignores any subsequent such messages.)

Adversary instructs trusted party to answer honest party: When the adversary sends a message of the type $(\mathsf{send\text{-}output}, j, i)$ to the trusted party, the trusted party sends $(i, f_1(x_i^j, y_i^j))$ to the honest party P_1^j, where x_i^j and y_i^j are the respective inputs sent by P_1^j and the adversary for this session. (If (j, i, x_i^j) and (j, i, y_i^j) have not yet been received by the trusted party, then this message is ignored.)

Outputs: An honest party P_1^j always outputs the vector of outputs that it received from the trusted party. Formally, whenever it receives an output, it writes it to its output-tape. Thus, the outputs do not appear in ascending order according to the session numbers, but rather in the order they are received by P_1^j. The adversary may output an arbitrary function of its auxiliary input z, initial input y, and the outputs obtained from the trusted party.

The ideal execution of $\mathcal{F}$ (with security parameter k, input-selecting machines $\overline{M} = (M_1^1, \ldots)$, initial inputs $(\vec{x}, y)$, and auxiliary input z to $\mathcal{S}$), denoted $\text{IDEAL}_{\mathcal{F},\mathcal{S}(z),\overline{M}}(1^k, \vec{x}, y)$, is defined as the pair consisting of the outputs of the honest parties and the output of $\mathcal{S}$ from the above ideal execution.

Execution in the real model. We next consider the real model in which a real two-party protocol is executed (and there is no trusted party). Formally, a two-party protocol ρ is defined by two sets of instructions for parties P_1 and P_2, respectively. A protocol is said to be polynomial-time if the running-time of each party in a *single execution* is bounded by a fixed polynomial in the security parameter k, irrespective of the length of the input.

Let $\mathcal{F}$ be as above and let ρ be a probabilistic polynomial-time, two-party protocol for computing $\mathcal{F}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adversary controlling P_2, and let $P_1^1, \ldots$ be honest. The parties run concurrent executions of the protocol, where an honest party P_1^j follows the instructions of ρ_1 in all of the executions. Here, the ith session for some party P_1^j is initiated by the adversary's sending a start-session message to the honest party. The honest party then applies its input-selecting machine to its initial input, the session number i, and its previously received outputs, and obtains the input for this new session. Upon the conclusion of an execution of ρ, the honest party writes its output from that execution on its output-tape. The scheduling of all messages is controlled by the adversary. That is, the execution proceeds as follows. The adversary sends a message of the form (i, α) to the honest party P_i^j. The honest party then adds the message α to the view of its ith execution of ρ and replies according to the instructions of ρ and this view. The adversary continues by sending another message, and so on.

In the setting of m-bounded concurrency, the scheduling by the adversary must fulfill the following condition: for every execution, from the time that execution begins until the time that it ends, messages from at most m other executions can be sent. This definition of concurrency covers the case when m executions are run simultaneously; it also includes the more general case where more than m executions take place overall, but each execution overlaps with at most m other executions. In this setting, the value m is fixed ahead of time, and the protocol design may depend on the choice of m. The real m-bounded concurrent execution of ρ (with security parameter k, input-selecting machines $\overline{M} = (M_1^j, \ldots)$, initial inputs $(\vec{x}, y)$, and auxiliary input z to $\mathcal{A}$), denoted $\text{REAL}^m_{\rho,\mathcal{A}(z),\overline{M}}(1^k, \vec{x}, y)$, consists of the outputs of the honest parties and the output of the adversary in the above execution.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, a protocol is secure if for every real-model adversary $\mathcal{A}$ and input-selecting machines $\overline{M}$, there exists an ideal model adversary $\mathcal{S}$ such that for all initial inputs $\vec{x}, y$, the outcome of an ideal execution with $\mathcal{S}$ is computationally indistinguishable from the outcome of a real protocol execution with $\mathcal{A}$. Notice that the order of quantifiers is such that $\mathcal{S}$ may depend on $\overline{M}$. Thus, $\mathcal{S}$ knows the *strategies* used by the honest parties to choose their inputs. However, $\mathcal{S}$ does not know the initial inputs of the honest parties, nor the random tapes used by its input-selecting machines. We now present the definition:

Definition 3 *Let $\mathcal{F}$ and ρ be as above, and let $m = m(k)$ be a fixed polynomial. Protocol ρ is said to* securely compute $\mathcal{F}$ under m-bounded concurrent self composition *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ controlling P_2, and every set of probabilistic polynomial-time input-selecting machines $\overline{M}$, there exists an non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ controlling P_2, such that*

$$\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(z),\overline{M}}(1^k, \vec{x}, y)\right\}_{k\in\mathbb{N};\vec{x},y,z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\text{REAL}^m_{\rho,\mathcal{A}(z),\overline{M}}(1^k, \vec{x}, y)\right\}_{n\in\mathbb{N};\vec{x},y,z\in\{0,1\}^*} .$$

The hybrid model for concurrent self composition. In, e.g., Theorem 4.3 we refer to "the hybrid model for concurrent self composition". What we mean by this is a model whereby the parties have access to a trusted party who behaves exactly as described in the ideal model above. The only reason we call it a "hybrid" model is that there is another protocol that instructs the parties how to behave.