Collusion-Free Multiparty Computation in the Mediated Model

Joël Alwen[1], Jonathan Katz[2], Yehuda Lindell[3], Giuseppe Persiano[4], abhi shelat[5], and Ivan Visconti[4,⋆]

[1] New York University, USA
jalwen@cs.nyu.edu
[2] The University of Maryland, USA
jkatz@cs.umd.edu
[3] Bar-Ilan University, Israel
lindell@cs.biu.ac.il
[4] University of Salerno, Italy
{giuper,visconti}@dia.unisa.it
[5] University of Virginia, USA
shelat@virginia.edu

Abstract. *Collusion-free* protocols prevent subliminal communication (i.e., covert channels) between parties running the protocol. In the standard communication model, if one-way functions exist, then protocols satisfying any reasonable degree of privacy cannot be collusion-free. To circumvent this impossibility, Alwen, shelat and Visconti (CRYPTO 2008) 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 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., and resolve the main open questions in this area by showing a collusion-free protocol (in the mediated model) for computing any multi-party functionality.

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 (a.k.a., steganographic) fashion. As introduced by Lepinski, Micali, and shelat, a *collusion-free* protocol [13] rules out such covert

⋆ The research of the second and third authors was supported by US-Israel Binational Science Foundation grant #2004240. The research of the fifth and sixth authors has been supported in part by the European Commission through the ICT program under Contract ICT-2007-216646 ECRYPT II and through the FP6 program under contract FP6-1596 AEOLUS. Author shelat is supported by NSF grant 0845811.

communication. Unfortunately, in the standard communication model, if one way functions exist, then 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 that realize 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 [3,4,7,17,19]), 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 [10,11,13] (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 high-profile spectrum auctions [6].

One approach for constructing collusion-free protocols is to rely on the notion of *verifiable determinism* as introduced by Lepinski et al. [12,13]. Roughly speaking, verifiable determinism 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 [10,11,13] rely on strong physical assumptions such as secure envelopes and ballot boxes.

A completely different approach to the problem was recently suggested by Alwen, shelat and Visconti [1]. They proposed a 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 can be realized in many real 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.) Finally, we prove composition theorems in the mediated setting that may be useful in 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 few paragraphs provide 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 (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. 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.

Set-up assumptions. Alwen et al. assume *no* shared state between the parties, and this allows the mediator to potentially "fork" the parties (in the sense of [2]) into disjoint subsets running independent computations. To prevent this, Alwen et al. assume that even a dishonest mediator behaves honestly during a "set-up" phase. (See further discussion in Section 2.)

In addition to this model, we also analyze a model where there is assumed to be a trusted public-key infrastructure (PKI) such that all parties running the protocol know each others' public keys. These two set-up assumptions are incomparable: the first makes assumptions regarding the behavior of the mediator but can achieve a stronger notion of collusion-freeness; the second may be more realistic but requires the involvement of an external trusted party to set up the public key infrastructure.

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 with broadcast. (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.)

   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 due to the need to preserve security when the mediator is dishonest:

Authentication. The mediator should be prevented from modifying the messages of honest parties. To achieve this, we change $\mathcal{F}^{\pi}_{\mathsf{compute}}$ to output (msg, σ), where σ is a valid signature[1] by P_i on msg, and require the mediator to send commitments on *both* these values to the other parties. Furthermore, $\mathcal{F}^{\pi}_{\mathsf{compute}}$ will ensure that all previous commitments contain appropriately signed messages.

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 when the mediator is honest: if a party aborts, the mediator has no way of generating a (commitment to a) default message with an appropriate signature. We fix this by allowing the mediator in this case to commit to a "dummy message" with

[1] As discussed earlier, we consider two different set-up assumptions: public keys can be established either during a preamble phase, or via an external PKI. See further discussion in the following section.

no signature; we also change $\mathcal{F}^{\pi}_{\mathsf{compute}}$ so that if it detects a dummy message the mediator receives no output. The effect is that 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.

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 protocols 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

Standard cryptographic primitives. 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 the length of all commitments is a fixed function of the message length.

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 outputs of Gen, and we assume (without loss of generality) that one can efficiently decide whether a given (sk, pk) lies in $\mathsf{Range}(\mathsf{Gen})$. We assume the length of all valid signatures is some known, fixed function of the message length.

Security in the mediated model – preliminaries. We use the real/ideal paradigm for defining security, but our real and ideal worlds differ from the usual ones and collusion-freeness requires a new definition. Our real world is essentially standard except that all communication is between parties $P_1, \ldots, P_n$ and the mediator P_{n+1}. 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). In each ideal world we consider two possible set-up assumptions; see below. Collusion-freeness is defined 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 $P_1, \ldots, P_{n+1}$ have inputs $x_1, \ldots, x_{n+1}$ and run the protocol honestly, then each P_i receives output $f_i(x_1, \ldots, x_{n+1})$, distributed appropriately in case $\mathcal{F}$ is randomized.

Set-up assumptions. Alwen et al. [1] observe that if *no* setup is assumed, then—because all parties communicate only with the mediator—a dishonest mediator can "fork" the parties into disjoint groups, each running a separate computation (much as in [2]). To prevent this, parties must be able to "authenticate" to each other. (It is as an interesting open question to assume no setup and treat "forking" attacks directly, but this is not the focus of our work.)

Here, we study two set-up assumptions under which such authentication can be implemented. The first, termed trusted-PKI, assumes a PKI in the usual sense, with each party knowing the public keys of all the other parties running the protocol. We stress that we do not assume honestly generated keys, or require parties to prove knowledge of their keys; all we require is consistency.

While this approach is appealing, it loses something in the "spirit" of collusion-freeness since parties are now potentially able to follow strategies based on each others' identities. Nevertheless, we believe the trusted-PKI model is meaningful in the context of collusion-freeness. For one, public keys can be generated *before* inputs are given to the parties (and before the function being computed is agreed upon!), and so the use of any subliminal communication will be limited. Furthermore, parties who are not aware of each other before execution of the protocol will necessarily generate their public keys independently, whereas parties who *are* aware of each other before executing the protocol cannot be prevented anyway from communicating arbitrary information in advance.

We also consider the mediated-PKI model that provides stronger guarantees of collusion-freeness under a different assumption. Specifically, here we follow Alwen et al. [1] and assume that even a dishonest mediator follows the protocol during a "preamble" phase where a "pseudo-PKI" is established. Instead of viewing this as an assumption, one can also interpret this as a claim that *if* the mediator acts in a particular way *then* certain guarantees hold.

2.1 Execution in the Real World (with an Honest Mediator)

We first consider the real world (i.e., the mediated model) 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). For simplicity the channels to/from P_{n+1} are assumed to be private and authenticated.

In this section we assume the mediator is honest; we consider the case of a dishonest mediator in Section 2.4. Let $\mathcal{I} \subseteq [n]$ denote the set of corrupt parties[2] and denote by $\mathcal{H} = [n] \setminus \mathcal{I}$ the set of uncorrupted parties (not including the mediator). A real world execution begins with a "PKI establishment" stage for which we define two variants:

[2] In contrast to the usual case, here a meaningful definition is obtained even when $\mathcal{I} = [n]$.

- trusted-PKI: For $i \in \mathcal{H}$, party P_i honestly generates a signature key pair (pk_i, sk_i). Each corrupted party P_i may generate an arbitrary public key pk_i. Once all keys have been generated, each P_i is given the vector $\mathsf{KeyVec}_i := (pk_1, \ldots, pk_n)$ of public keys, and the mediator is given $\mathsf{KeyVec}^i_{n+1} := \mathsf{KeyVec}_i$ for all $i \in [n]$. (The notation is chosen to be consistent with the mediated-PKI setting below.) We say that KeyVec_i *matches* KeyVec^i_{n+1} if they are equal.
- mediated-PKI: For $i \in \mathcal{H}$, party P_i honestly generates a signature key pair (pk_i, sk_i). Each corrupted party P_i may generate an arbitrary public key pk_i. All parties send pk_i to the mediator. (If a party fails to send anything, the mediator uses a default public key.) The mediator chooses independent coins $\{r^j_i\}_{i,j \in [n]}$, computes $c^j_i = C(pk_i; r^j_i)$, and sends $\mathsf{KeyVec}_i := (c^i_1, \ldots, c^i_n)$ to party P_i. The mediator keeps the vectors of decommitments $\mathsf{KeyVec}^i_{n+1} := ((pk_1, r^i_1), \ldots, (pk_n, r^i_n))$. We say that KeyVec_i *matches* KeyVec^i_{n+1} if for all $j \in [n]$ the jth component of KeyVec^i_{n+1} is a valid decommitment to the jth component of KeyVec_i.

The remainder of the real-world execution is identical in both settings.

Input determination and protocol execution: Each party P_i (for $i \in [n+1]$) is given input x_i. Party P_i is also given auxiliary input aux_i (which honest players ignore) as well as independent random coins r_i. 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

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

to be the random variable consisting of the stated outputs following an execution of Π where the parties are given inputs $\boldsymbol{x} = \{x_1, \ldots, x_{n+1}\}$ and auxiliary inputs $\mathbf{aux} = \{\mathsf{aux}_1, \ldots, \mathsf{aux}_{n+1}\}$.

2.2 Execution in the Ideal World (with an Honest Mediator)

We continue to 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 $\mathcal{I} \subseteq [n]$ be the set of corrupted parties, and let $\mathcal{H} = [n] \setminus \mathcal{I}$ be the set of honest parties (other than the mediator) as before.

As in the previous section, we distinguish between two settings in the ideal world. The trusted-PKI setting includes a "PKI establishment" step where a PKI is established exactly as in the real world: for $i \in \mathcal{H}$, party P_i honestly generates signature keys (pk_i, sk_i). A corrupted P_i outputs any public key pk_i of its choice. For $i \in [n]$ party P_i is then given the vector $\mathsf{KeyVec}_i := (pk_1, \ldots, pk_n)$ of public keys. The $\mathsf{mediated\text{-}PKI}$ setting has no PKI establishment step.

The remainder of the ideal-world execution is identical in both settings.

Input determination: Each party P_i (for $i \in [n+1]$) is given their input x_i and auxiliary input aux_i (which an honest player ignores).
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 $i \in [n+1]$.

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

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

to be the random variable consisting of the stated outputs following an ideal-world execution where the parties are given inputs $\boldsymbol{x}$ and auxiliary inputs $\mathbf{aux}$ as specified.

2.3 Collusion-Freeness

Having defined the ideal and real models, we can now define collusion-freeness. 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.

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 that the following two distributions are computationally indistinguishable:*

$$\left\{\text{IDEAL}^{\mathsf{cf}}_{\mathcal{F}, \mathcal{S}_\mathcal{I}(\mathbf{aux})}(1^k, \boldsymbol{x})\right\}_{\boldsymbol{x}, \mathbf{aux} \in (\{0,1\}^*)^{n+1},\, k \in \mathbb{N}}$$

$$\left\{\text{REAL}^{\mathsf{mediated}}_{\Pi, \mathcal{A}_\mathcal{I}(\mathbf{aux})}(1^k, \boldsymbol{x})\right\}_{\boldsymbol{x}, \mathbf{aux} \in (\{0,1\}^*)^{n+1},\, k \in \mathbb{N}}$$

2.4 Security (with a Dishonest Mediator)

The definition in the case of a dishonest mediator is essentially the standard one for secure multi-party computation, with the exception being that honest parties cannot communicate directly in the real world. (We also incorporate a PKI establishment phase as in the prior sections.) Further details are given in the full version. A protocol satisfying both Definition 1 and the definition for the case of a dishonest mediator 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 Building Blocks

Our protocol uses some cryptographic primitives and tools which we review here.

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 $\mathcal{F}_{\mathsf{com}}((m,r),\lambda) = (\bot, C(m;r))$, 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

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

4. The signature functionality is defined as:

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

A protocol π securely computing $\mathcal{F}$ (in the standard sense): Let π be an $(n+1)$-party protocol that securely computes $\mathcal{F}$ in the usual 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 $t \leq n+1$ of 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 proved secure using a *black-box* simulator.
By using standard techniques, we may assume without loss of generality that:

- 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 round r.

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 messages of protocol 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 Π. 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 (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

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

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 $\boldsymbol{C}$; vector KeyVec; 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 $\mathsf{com}_j^{\mathsf{input}}$ and $\mathsf{com}_j^{\mathsf{rand}}$; a vector of commitments $\boldsymbol{C}_j$; vector KeyVec'; and a round number rid'. In addition, P_{n+1} sends a vector $\mathbf{dec}_j$.
- Upon receiving the above, $\mathcal{F}^{\pi}_{\mathsf{compute}}$ does:
  1. If KeyVec does not match KeyVec' then send $\perp$ to P_{n+1} and halt. Otherwise extract the parties' public keys $(pk_1, \ldots, pk_n)$ from KeyVec'.
  2. If $(\mathsf{com}^{\mathsf{input}}, \mathsf{com}^{\mathsf{rand}}, \boldsymbol{C}, \mathsf{rid}) \neq (\mathsf{com}_j^{\mathsf{input}}, \mathsf{com}_j^{\mathsf{rand}}, \boldsymbol{C}_j, \mathsf{rid}')$ or if $(sk_j, pk_j) \notin \mathsf{Range}(\mathsf{Gen})$, then send $\perp$ to P_{n+1} and halt.
  3. If $\mathsf{dec}^{\mathsf{input}}$ is not a valid decommitment to $\mathsf{com}_j^{\mathsf{input}}$, or $\mathsf{dec}^{\mathsf{rand}}$ is not a valid decommitment to $\mathsf{com}_j^{\mathsf{rand}}$, or $\mathbf{dec}_j$ does not contain valid decommitments to all the commitments in $\boldsymbol{C}_j$, then send $\perp$ to P_{n+1} and halt.
  4. Let $(\mathsf{msg}_1, \sigma_1), \ldots, (\mathsf{msg}_\ell, \sigma_\ell)$ be the committed values in $\boldsymbol{C}_j$. 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.
  5. Let x_j and r_j be the committed values in $\mathsf{com}_j^{\mathsf{input}}$ and $\mathsf{com}_j^{\mathsf{rand}}$ 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.

Fig. 1. The functionality computing the next message of π

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 call "mediator broadcast." The mediator P_{n+1} begins holding a message m, and at the end of the protocol each P_i obtains an (independent) commitment com_i to a message m_i. 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 $m_i = m'$ 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 aborts.* Our protocol $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ realizing $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ proceeds, roughly speaking in the following three stages:

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

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 $\mathsf{com}^{\mathsf{rand}}$, vector KeyVec, and a vector of r commitments $\boldsymbol{C}$. In addition, P_j sends two strings $\mathsf{dec}^{\mathsf{input}}$ and $\mathsf{dec}^{\mathsf{rand}}$.
- P_{n+1} inputs a pair of commitments $\mathsf{com}_j^{\mathsf{input}}$ and $\mathsf{com}_j^{\mathsf{rand}}$, vector KeyVec', and a vector of r commitments $\boldsymbol{C}_j$. In addition, P_{n+1} sends a vector $\mathbf{dec}_j$.
- Upon receiving the above, $\mathcal{F}^{\pi}_{\mathsf{output}}$ does:
  1. If KeyVec does not match KeyVec' then send $\perp$ to P_j and halt. Otherwise extract the parties' public keys $(pk_1, \ldots, pk_n)$ from KeyVec'.
  2. If $(\mathsf{com}^{\mathsf{input}}, \mathsf{com}^{\mathsf{rand}}, \boldsymbol{C}) \neq (\mathsf{com}_j^{\mathsf{input}}, \mathsf{com}_j^{\mathsf{rand}}, \boldsymbol{C}_j)$, then send $\perp$ to P_j and halt.
  3. If $\mathsf{dec}^{\mathsf{input}}$ (resp., $\mathsf{dec}^{\mathsf{rand}}$) is not a valid decommitment to $\mathsf{com}_j^{\mathsf{input}}$ (resp., $\mathsf{com}_j^{\mathsf{rand}}$), or $\mathbf{dec}_j$ does not contain valid decommitments to all the commitments in $\boldsymbol{C}_j$, then send $\perp$ to P_j and halt.
  4. Let $(\mathsf{msg}_1, \sigma_1), \ldots, (\mathsf{msg}_r, \sigma_r)$ be the committed values in $\boldsymbol{C}_j$. 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.
  5. Let x_j and r_j be the committed values in $\mathsf{com}_j^{\mathsf{input}}$ and $\mathsf{com}_j^{\mathsf{rand}}$. Compute the value $\textsc{out}_j$ 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 $\textsc{out}_j$ to P_j and halt.

Fig. 2. The functionality computing the output of π

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

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

- For $j \in [n]$, each P_j inputs a bit b_j and KeyVec_j.
- P_{n+1} inputs a message m, $\{\mathsf{KeyVec}^j_{n+1}\}_{j \in [n]}$, random coins $r_1, \ldots, r_n$, and a set $\mathcal{H} \subseteq [n]$.
- For all $i \in [n]$, if KeyVec_i and KeyVec^i_{n+1} do not match then set $\mathcal{H} := \mathcal{H} \setminus \{i\}$.
- Let $b = \bigwedge_i b_i$.
- 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 $b = 0$, 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}.

Fig. 3. Mediator broadcast

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 takes 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 the full version of the paper.

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.

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 broadcast (for $j \in [n]$), then P_j runs $\mathcal{F}^{\pi}_{\mathsf{compute}}$ with the mediator; thus, the mediator obtains the next π-message msg along with a signature of P_j on this message (and the current round number). If it is the mediator's turn to broadcast, the mediator simply computes the next π-message msg on its own, and then runs "mediator broadcast" using msg. As long as everyone behaves honestly, each party thus learns commitments to all messages of the protocol. In the third stage, the mediator runs $\mathcal{F}^{\pi}_{\mathsf{output}}$ with each P_j to enable P_j to learn its output. We now describe the protocol formally.

The protocol begins with each party P_i $(i \in [n])$ holding a vector KeyVec_i, and with the mediator holding $\{\mathsf{KeyVec}^i_{n+1}\}_{i \in [n]}$. Party P_i also holds input x_i and, if $i \in [n]$, 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 $\mathsf{dec}^{\mathsf{input}}_j = (x_j, s_j)$ to $\mathcal{F}_{\mathsf{com}}$. Let $\mathsf{com}^{\mathsf{input}}_j$ 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 $\mathsf{dec}^{\mathsf{rand}}_j = (r_j, s'_j)$ the output of P_j and by $\mathsf{com}^{\mathsf{rand}}_j$ 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 $\boldsymbol{C}_j = (\mathsf{com}_1^j, \ldots, \mathsf{com}_{i-1}^j)$ be the commitments that P_j output in the previous $i-1$ rounds.
- P_j and P_{n+1} run an instance of $\mathcal{F}^{\pi}_{\mathsf{compute}}$. Here, P_j sends $\mathcal{F}^{\pi}_{\mathsf{compute}}$ its commitments $\mathsf{com}_j^{\mathsf{input}}$ and $\mathsf{com}_j^{\mathsf{rand}}$, the vector of commitments $\boldsymbol{C}_j$, vector KeyVec_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 $\mathcal{F}^{\pi}_{\mathsf{compute}}$ the commitments $\mathsf{com}_j^{\mathsf{input}}$, $\mathsf{com}_j^{\mathsf{rand}}$, and $(\mathsf{com}_1^j$, $\ldots$, $\mathsf{com}_{i-1}^j)$; vector KeyVec_{n+1}^j; the round identifier $\mathsf{rid} = 0i$; and the decommitments $(\mathsf{dec}_1^j, \ldots, \mathsf{dec}_{i-1}^j)$.
- If $\mathcal{F}^{\pi}_{\mathsf{compute}}$ returns $\perp$ to P_{n+1}, then P_{n+1} sets $\mathsf{abort} = \mathsf{true}$ and $m_i = \mathsf{dummy}$. Otherwise, if $\mathcal{F}^{\pi}_{\mathsf{compute}}$ 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 $\mathsf{sid} = 1i$. P_{n+1} chooses random $r_1, \ldots, r_n$ and runs $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ with the other parties, where P_{n+1} provides input $(m_i, r_1, \ldots, r_n, \mathcal{H} = [n], \{\mathsf{KeyVec}_{n+1}^i\}_{i\in[n]})$ and every other party P_j provides input 1 and KeyVec_j.

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

Stage 3 – output determination: (Note that this emulates the rth and final round of π.)

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 $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ with all the other parties, where P_{n+1} provides input $(\mathsf{msg}_r, r_1, \ldots, r_n, \mathcal{H} = [n], \{\mathsf{KeyVec}_{n+1}^i\}_{i\in[n]})$ and every other party P_i provides input 1 and KeyVec_i. The mediator outputs OUT_{n+1}.

   Each party P_j defines com_r^j to be the commitment that it received from $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$. Note that P_{n+1} can compute the commitment, and knows the corresponding decommitment.
2. The mediator P_{n+1} runs $\mathcal{F}^{\pi}_{\mathsf{output}}$ with each P_j, where P_j provides input $\mathsf{com}_j^{\mathsf{input}}$, $\mathsf{com}_j^{\mathsf{rand}}$, KeyVec_j, $(\mathsf{com}_1^j, \ldots, \mathsf{com}_r^j)$, $\mathsf{dec}_j^{\mathsf{input}}$, and $\mathsf{dec}_j^{\mathsf{rand}}$, and P_{n+1} sends $\mathsf{com}_j^{\mathsf{input}}$, $\mathsf{com}_j^{\mathsf{rand}}$, KeyVec_{n+1}^j 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 $\mathcal{F}^{\pi}_{\mathsf{output}}$ in this step.

4 Proof of Security

All the results stated here apply to protocols run in either the trusted-PKI or mediated-PKI settings.

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

Theorem 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 ($\mathcal{F}_{\mathsf{com}}$, $\mathcal{F}_{\mathsf{ct}}$, $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$, $\mathcal{F}^{\pi}_{\mathsf{compute}}$, $\mathcal{F}^{\pi}_{\mathsf{output}}$)-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 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 2 follows along the lines of [5] and is given in the full version of the paper.

Theorem 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 3 requires ρ to be secure under (bounded) concurrent self-composition since a dishonest mediator may run concurrent executions with different honest parties. Theorem 3 follows immediately from the definition of m-bounded concurrent self composition; a proof is given in the full version of the paper.

We can now prove our main result:

Corollary 1. *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, in either the* trusted-PKI *or* mediated-PKI *settings.*

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[3] ρ that is secure under m-bounded concurrent self-composition [14,16] (m will be specified in the proof below);
- $\mathcal{F}^{\mathsf{sid}}_{\mathsf{bcast}}$ is instantiated using protocol $\Pi^{\mathsf{sid}}_{\mathsf{bcast}}$ (given in the full version of the paper) 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 1 and 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 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 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 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 mediated model.

References

1. Alwen, J., Shelat, A., Visconti, I.: Collusion-Free Protocols in the Mediated Model. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 497–514. Springer, Heidelberg (2008)
2. Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure Computation without Authentication. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 361–377. Springer, Heidelberg (2005)
3. Bohli, J.M., Steinwandt, R.: On Subliminal Channels in Deterministic Signature Schemes. In: Park, C.-s., Chee, S. (eds.) ICISC 2004. LNCS, vol. 3506, pp. 182–194. Springer, Heidelberg (2005)

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

4. Burmester, M., Desmedt, Y., Itoh, T., Sakurai, K., Shizuya, H., Yung, M.: A Progress Report on Subliminal-Free Channels. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 157–168. Springer, Heidelberg (1996)
5. Canetti, R.: Security and Composition of Multiparty Cryptographic Protocols. Journal of Cryptology 13(1), 143–202 (2000)
6. Cramption, P., Schwartz, J.: Collusive Bidding: Lessons from the FCC Spectrum Auctions. Journal of Regulatory Economics 17(3), 229–252 (2000)
7. Desmedt, Y.: Simmons' Protocol is not Free of Subliminal Channels. In: IEEE Computer Security Foundations Workshop, pp. 170–175 (1996)
8. Goldreich, O.: Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press, Cambridge (2004)
9. Hopper, N., Langford, J., von Ahn, L.: Provably Secure Steganography. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 77–92. Springer, Heidelberg (2002)
10. Izmalkov, S., Lepinski, M., Micali, S.: Verifiably Secure Devices. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 273–301. Springer, Heidelberg (2008)
11. Izmalkov, S., Micali, S., Lepinski, M.: Rational Secure Computation and Ideal Mechanism Design. In: Foundations of Computer Science (FOCS) 2005, pp. 585–595 (2005)
12. Lepinski, M., Micali, S., Shelat, A.: Fair Zero-Knowledge. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 245–263. Springer, Heidelberg (2005)
13. Lepinski, M., Micali, S., Shelat, A.: Collusion-Free Protocols. In: Symposium on Theory of Computing (STOC) 2005, pp. 543–552. ACM, New York (2005)
14. Lindell, Y.: Protocols for Bounded-Concurrent Secure Two-Party Computation in the Plain Model. Chicago Journal of Theoretical Computer Science (1), 1–50 (2006)
15. Lindell, Y.: Lower Bounds and Impossibility Results for Concurrent Self Composition. Journal of Cryptology 21(2), 200–249 (2008)
16. Pass, R.: Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. In: Symposium on Theory of Computing (STOC) 2004, pp. 232–241 (2004)
17. Simmons, G.: The Prisoners' Problem and the Subliminal Channel. In: Advances in Cryptology—Crypto 1983, pp. 51–67. Springer, Heidelberg (1983)
18. Simmons, G.: Cryptanalysis and Protocol Failures. Comm. ACM 37(11), 56–65 (1994)
19. Simmons, G.: The History of Subliminal Channels. In: Information Hiding Workshop, pp. 237–256 (1996)