0368.4170: Cryptography and Game Theory Ran Canetti and Alon Rosen

Lecture 10

23 December 2009 Fall 2009

Scribes: Margarita Vald and Ilia Gorelik

Part I

Mechanism Design

1 Introduction

So far we've seen various types of games, and investigated how they work. In this lecture we'll ask the opposite question: We want to obtain certain goals. How should we plan a game that will obtain these goals? This area is called **Mechanism Design**.

In modeling real-life situations by games, delicate modeling decisions have major effects on both the participants' behavior and on the decisions eventually taken in the game. Some situations of interest may be auctions and public projects, which have many variants:

- *Auction* - Are the bids given in sealed envelopes, or is it a public auction, with bids given orally? Are the items sold individually or together? The method of offering the bids effects what participants learn on each other, the bids they give, and eventually the result of the auction.
- *Public project (new highway, new library, park, defense system, etc.)* - Different ways of spreading the costs of the project across society affects the decision of whether the project is undertaken or not.

Natural questions to ask are:

- Is there an efficient mechanism where all parties agree to participate voluntarily and that leads to desirable outcomes overall?
- In elections, how do we cause voters to vote by their real preferences, without lying?

We will make the following assumptions:

- The players behave strategically. That is, they have a benefit function they want to maximize.
- The players have **private information** known only to them that affects their behavior.

For example:

- In auctions:
  - Each participant has his evaluation of the object (private information)
  - The participants may give a lower bid than their real evaluation (will want to minimize the payment)
  - The auctioneer will decide the winner and the price to pay.
    - We will focus on mechanisms in which there is a central entity that receives all input data and decides the result according to the rules of the game.
- In negotiation between a buyer and a seller:
  - Each participant has a value of the deal (private information)
  - The seller may claim that his value is higher than it's real value (increase the price)
  - The buyer may claim that his value is lower than it's real value (decrease the price)
    - We would like to plan a mechanism that enables the buyer and the seller to agree on a price profitable to both.

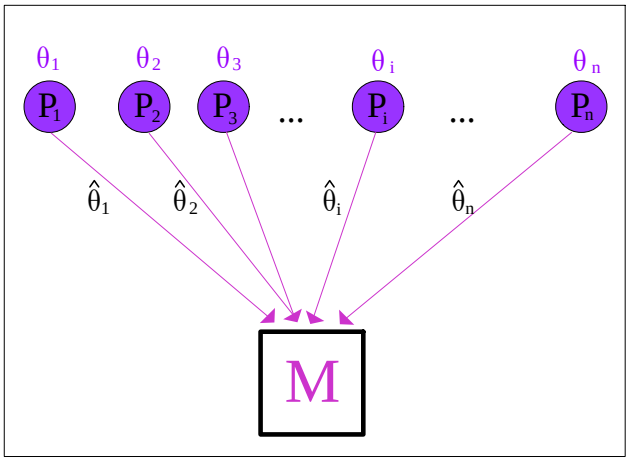

Figure 1: Mediated game model

2 Formal Model

2.1 Model Definition

In Game Theory, strategic interactions among n players are formally modeled as games. In this class, we focus on finite versions of mediated games of incomplete information. In their normal form, each player has his own private valuation θ_i, where $(\theta_1, ..., \theta_n)$ are drawn from some publicly known, joint distribution. Players do not communicate with each other. (Conceptually, they are isolated in separate rooms.) Instead, each player privately sends a report $\hat{\theta}_i$ to a trusted mediator M (see figure 1). The mediator then evaluates a specified, possibly probabilistic, finite function on the received reports so as to determine a public outcome d. The outcome d determines the result of the game: for instance, in public auctions, d determines which player receives the object.

Remark 1. Ideally, we would like the utilities to depend only on on private information. However, this would enable us to consider only limited mechanisms. In order to overcome this, we will introduce *payments*, which will allow for much more fine-tuned mechanisms.

That is, a game consists of:

- players- $P_1, ..., P_n$
- decisions - $d \in D$
- private valuation of players.
  - the private valuation of playerP_i is- $\theta_i \in \Theta_i$
- value function- $v_i : D \times \Theta_i \to \mathbb{R}$
  - The benefit of player P_i with private information θ_i from decision $d \in D$

Remark 2. P_i has valuation θ_i and announces $\hat{\theta}_i$ to the mechanism.

Example 3. Public project

A group of n people want to decide whether or not to build a certain public project.

- Cost of project - C (cost for participant $\frac{C}{n}$)

- Decision - $D = \{0, 1\}$ (1 - do / 0 - don't do the project)
- Benefit of player P_i from doing the project - θ_i
- value of player P_i: $v_i(d, \theta_i) = \begin{cases} 0 & d = 0 \\ \theta_i - \frac{C}{n} & d = 1 \end{cases}$

Remark 4. This formalism implicitly assumes that the cost of the project is spread evenly across society. Later we will see different ways to spread the cost.

Example 5. Allocating a product

An indivisible good is to be allocated to one member of the society. We have $P_1, ..., P_n$ potential buyers. We want to give the product to one of them.

- Decision $D = P_i$ where $i \in \{1, ..., n\}$
- Benefit of player P_i if obtain the product- θ_i
- value of player P_i: $v_i(d, \theta_i) = \begin{cases} \theta_i & d = P_i \\ 0 & d \neq P_i \end{cases}$
- Later, we will incorporate payments to establish the utility function.

2.2 Decision rules and efficiency

The society would like to make a decision that maximizes the overall benefit of it's members:

- For the public project, we would like a mechanism that will decide to build the project if the sum of benefits over all agents is more then its cost
- For the product allocation, we might want a mechanism that gives the product to the agent that has the maximal benefit from it

From now on, we denote by $\Theta = \Theta_1 \times \Theta_2 \times ... \times \Theta_n$.

Let us formalize this:

Definition 6 (Decision rule). a decision rule is a function $f : \Theta \to D$.

Definition 7 (Efficiency). A decision rule f is **efficient** if it maximize the overall benefit:

$$\forall \overrightarrow{\theta} \in \Theta, \ \forall d' \in D: \quad \sum_i v_i \left(f\left(\overrightarrow{\theta}\right), \theta_i \right) \geq \sum_i v_i(d', \theta_i)$$

In words, f finds the "best" decision for any set of valuation.

For example:

- The public project should be done only if the overall benefit is higher then the cost:

$$\mathrm{d} = 1 \Leftrightarrow \sum_i v_i(1, \theta_i) \geq \sum_i v_i(0, \theta_i) \ \Leftrightarrow \ \sum_i \theta_i \geq C$$

- Indivisible product - The efficient decision is $d = \underset{\mathrm{i}}{\mathrm{argmax}} \{\theta_i\}$

2.3 Payments

The definition of efficiency as the basis for decision making has a problem: We base it on a private information, which the decision maker doesn't actually have. We could ask the players to give their private value, but the answers will often be dishonest or deceptive.

For instance, in the public project, if the ith player's cost is $\frac{C}{n}$, then an agent with $\theta_i < \frac{C}{n}$ has an incentive to under-report his value, and claim he has no profit from the project, and hence try to manipulate the decision to his own benefit. In the same way, an agent with $\theta_i > \frac{C}{n}$ has an incentive to over-report his value (meaning $\theta_i = C$). This could result in wrong decision. Other decision mechanisms aimed to bypass this problem, such as voting and deciding whether to build the project by majority vote, could also result in a decision which is not efficient.

The question is, can we create incentives for the players to reveal their real private information? Many times the answer is yes. We can balance their interests by putting taxes (to reduce the profit). That is done by a payment function, which indicates how much the players receive.

Definition 8 (Payment function). A **payment function** of player P_i is

$$t_i : \Theta \to \mathbb{R}$$

For all the players,

$$t : \Theta \to \ \mathbb{R}^n \ .$$

Remark 9. The payment t_i could be negative, in which case the player needs to pay.

Definition 10 (Social Choice Function). A **social choice function** is a pair of decision and payment:

$$\hat{f}\left(\overrightarrow{\theta}\right) = \left(f\left(\overrightarrow{\theta}\right), t\left(\overrightarrow{\theta}\right)\right)$$

The utility of player P_i for a playout of the game is based on its benefit from the decision taken, plus the payment he receives from the mechanism (both are based on the declared values):

$$u_i(\hat{\theta}, d, \theta_i) = v_i\left(d, \theta_i\right) + t_i\left(\hat{\theta}\right)$$

Remark 11. A utility function of this form is called quasi-linear.

2.4 Dominant Strategies

Definition 12 (Strategy). A **strategy** is a function $m_i : \theta_i \mapsto \hat{\theta}_i$

Definition 13 (Dominant strategy). A strategy m_i is a **dominant strategy** for $\theta_i \in \Theta_i$ if :

$$\forall \theta_i' \ \forall \hat{\theta}_{-i} : \quad u_i\left(\hat{f}\left(m_i\left(\theta_i\right), \hat{\theta}_{-i}\right), \theta_i\right) \geq u_i\left(\hat{f}\left(\theta_i', \hat{\theta}_{-i}\right), \theta_i\right)$$

In words, A strategy m_i is a dominant strategy for player P_i on input θ_i if it is superior to all other strategies of this player, regardless of other players strategies.

Definition 14 (Incentive Compatible Mechanism). A mechanism is **incentive compatible** (or **truthful**) if for each player P_i with valuation $\theta_i \in \Theta_i$, $\hat{\theta}_i = \theta_i$ is a dominant strategy for $\theta_i \in \Theta_i$

2.4.1 The Grove Criterion

We would like to have mechanisms that are truthful: in such mechanisms participation is simple (since the best strategy is to tell the truth) and the results are obviously beneficial to all. Grove gives a criterion for the truthful strategy to be a dominant strategy.

Theorem 15 (Grove Criterion)**.** *if:*

- f *is an efficient decision rule*
- *for each player*P_i *there is a function* $h_i : \Theta_{-i} \to \mathbb{R}$ *such that:* $t_i(\hat{\theta}) = h_i(\hat{\theta}_{-i}) + \sum_{j \neq i} v_j \left(f(\hat{\theta}), \hat{\theta}_j \right)$

  In words, the payment function can be written as the sum of a function which does not depend on $\hat{\theta}_i$ *and the sum of the valuation of all other players evaluated on* $\hat{\theta}$*.*

then $\hat{f} = (f, t)$ *is dominant strategy truthful.*

Proof. By the way of contradiction, suppose f is an efficient decision rule, and that for each player P_i there is h_i that define together t:$\Theta \to \mathbb{R}^n$, But (f, t) is not dominant strategy incentive compatible.

Then, there is a player P_i, valuations $\theta \in \Theta$ and θ_i' such that θ_i' is more beneficial to P_i:

$$v_i(f(\theta_{-i}, \theta_i'), \theta_i) + t_i(\theta_{-i}, \theta_i') > v_i(f(\theta), \theta_i) + t_i(\theta)$$

.

We'll expand t_i explicitly:

$$v_i(f(\theta_{-i}, \theta_i'), \theta_i) + h_i(\theta_{-i}) + \sum_{j \neq i} v_j \left(f\left(\theta_{-i}, \theta_i'\right), \theta_j \right) > v_i(f(\theta), \theta_i) + h_i(\theta_{-i}) + \sum_{j \neq i} v_j \left(f\left(\theta\right), \theta_j \right)$$

Therefore: $\sum_{i=1}^{n} v_i \left(f\left(\theta_{-i}, \theta_i'\right), \theta_i \right) > \sum_{i=1}^{n} v_i \left(f\left(\theta\right), \theta_i \right)$

And $f\left(\theta_{-i}, \theta_i'\right)$ contradicts the efficiency of $f(\theta)$.

The conclusion is that $\hat{f} = (f, t)$ is dominant strategy incentive compatible. □

Example 16. Second price auction - a VCG mechanism.

The winner of the auction is the bidder who bid the biggest bid, but the winner pays only the second highest bid that was made.

- $f(\theta) = \underset{\text{i}}{\text{argmax}}\{\theta_i\}$.
- $t_i(\theta) = \begin{cases} 0 & f(\theta) \neq i \\ -\max\limits_{\text{j}\neq\text{i}} \{\theta_j\} & f(\theta) = i \end{cases}$

Claim 17. The VCG mechanism for second-price auction is dominant strategy truthful (DST).

Proof. We will show that the Grove criterion holds. Let $h_i\left(\theta_{-i}\right)$=$-\max\limits_{\text{d}\in\text{D}} \left\{ \sum\limits_{j \neq i}^{n} v_j \left(d, \theta_j\right) \right\}$= $-\max\limits_{\text{j}\neq\text{i}} \{\theta_j\}$.

- For the player with the highest value:
  - $\sum\limits_{j \neq i}^{n} v_j \left(f\left(\theta\right), \theta_j \right) = 0$
  - $h_i\left(\theta_{-i}\right)$=$-v_{2_{nd}}\left(d_{2_{nd}}, \theta_{2_{nd}}\right)$=$-\theta_{2_{nd}}$
  - Therefore, $t_i = -\theta_{2_{nd}}$
- For any other player:

- $\sum_{j\neq i}^{n} v_j\left(f\left(\theta\right),\theta_j\right)=\theta_{winner}$
- $h_i\left(\theta_{-i}\right)=-\max_{\mathrm{d}\in\mathrm{D}}\left\{\sum_{j\neq i}^{n} v_j\left(d,\theta_j\right)\right\}=-\theta_{winner}$
- Therefore, $t_i=0$

□

Example 18. Public project.

Recall the "public project" example on page 2 with project cost C. We change the mechanism, specifically change the payment structure, and the result is that the mechanism is truthful; we manage to drop the "manipulability" concern we had before.

- $f(\theta)=\begin{cases}1 & C<\sum_{i=1}^{n}\theta_i\\ 0 & C\geq\sum_{i=1}^{n}\theta_i\end{cases}$
- $t_i(\theta)=\begin{cases}\sum_{j\neq i}\theta_j-C & \sum_{j\neq i}\theta_j<C<\sum_{i=1}^{n}\theta_i \text{ and } \theta_i\geq 0\\ C-\sum_{j\neq i}\theta_j & \sum_{i=1}^{n}\theta_i<C<\sum_{j\neq i}\theta_j \text{ and } \theta_i<0\\ 0 & \text{otherwise}\end{cases}$

The payment function intuitively indicates the damage that each player causes society. This is equal to the difference between the total valuations of all other players and the cost of the project. The only players who pay are those who are pivotal: their valuation changes the decision.

Claim 19. The allocating mechanism presented is dominant strategy truthful (DST).

Proof. We will show that the Grove criterion holds by adding an extra player "government" whose valuation space is the singleton valuation, giving cost C to undertaking the project and 0 otherwise. i.e.,

$$v_{n+1}=\begin{cases}-C & C<\sum_{i=1}^{n}\theta_i\\ 0 & C\geq\sum_{i=1}^{n}\theta_i\end{cases}$$

Let $h_i\left(\theta_{-i}\right)=-\max_{\mathrm{d}\in\mathrm{D}}\left\{\sum_{j\neq i}^{n+1} v_j\left(d,\theta_j\right)\right\}=-\max\left\{\sum_{j\neq i}^{n}\theta_j-C,\,0\right\}$.

- If the social benefit $\sum\theta_i$ exceeds the cost C:
  - $\sum_{j\neq i}^{n+1} v_j\left(f\left(\theta\right),\theta_j\right)=\sum_{j\neq i}\theta_j-C$
  - In case $\sum_{j\neq i}^{n}\theta_j>C$
    - $h_i\left(\theta_{-i}\right)=C-\sum_{j\neq i}^{n}\theta_j$
    - Therefore, $t_i=0$
  - In case $\sum_{j\neq i}^{n}\theta_j\leq C$ (and $\theta_i\geq 0$)

* $h_i(\theta_{-i}) = 0$
* Therefore, $t_i = \sum_{j \neq i} \theta_j - C$

- If the cost of the project does not exceed the social benefit:
  - $\sum_{j \neq i}^{n+1} v_j(f(\theta), \theta_j) = 0$
  - In case $\sum_{j \neq i}^{n} \theta_j \leq C$
    * $h_i(\theta_{-i}) = 0$
    * Therefore, $t_i = 0$
  - In case $\sum_{j \neq i}^{n} \theta_j > C$ (and $\theta_i < 0$)
    * $h_i(\theta_{-i}) = C - \sum_{j \neq i}^{n} \theta_j$
    * Therefore, $t_i = C - \sum_{j \neq i}^{n} \theta_j$

□

Remark 20. Notice that $\sum_{i=1}^{n} t_i(\theta) < C$ (unless $\sum_{i=1}^{n} \theta_i = C$), thus the payments collected do not cover the project's cost. This is inevitable; otherwise, one could declare a strictly lower valuation $\hat{\theta}_i$ and still the project would commence.

Part II

Secure Computation

3 Preliminaries

In this section we give a brief recap of the notion of security from Lecture 4.

Given a joint computation among n players and a trusted party, Secure Computation aims to remove the trusted party without suffering any correctness or privacy losses. In this lecture we will focus on a special case of *secure function evaluation* in the *ideal/real* paradigm.

Recall: An *ideal evaluation* of a n-input, n-output function f using ideal functionality $\mathcal{F}$ consists of the following process.

- Each player P_i has a private input, x_i, and is assumed to be honest or malicious.
- A honest player P_i simply send his original x_i to the ideal functionality $\mathcal{F}$.
- Malicious players may instead perfectly coordinate their actions, so as to compute and report to the ideal functionality $\mathcal{F}$ alternative inputs x_i' for every malicious player P_i.
- The ideal functionality $\mathcal{F}$ then computes $(y_1, ..., y_n) = f(x_1', ..., x_n')$, and privately hands out y_i to each player P_i.
- Each party P_i gives its output y_i to the environment E, where honest parties give $\mathcal{F}$'s output and the malicious parties jointly decide what to output based on their original inputs and values received from $\mathcal{F}$.

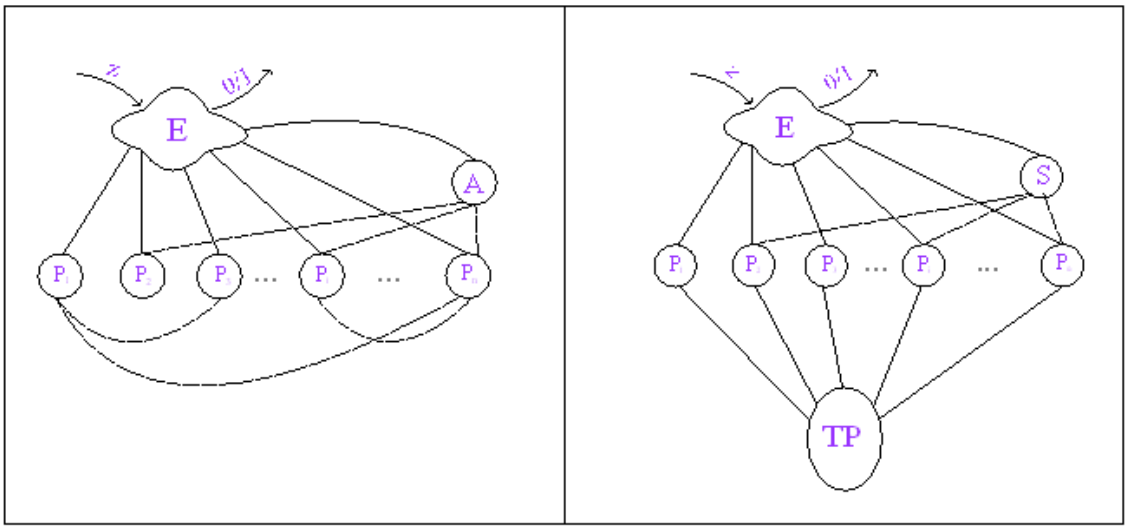

Figure 2: Real world vs. Ideal world

- Let $IDEAL_{\mathcal{F},S,E}\left(1^k, z\right)$ to be the random variable consisting of the environment E's output following an ideal-world execution with simulator S, where k is the security parameter.

A *real evaluation* of f consists of a protocol π executed by the n players alone:

- The players evaluate f on their private inputs by exchanging messages back and forth using a specified communication channel.
- Honest players always send their messages according to the protocol instructions.
- Malicious players may instead deviate from their instructions in an arbitrary manner, and even perfectly coordinate their communication strategies before and during the protocol.
- Let $REAL_{\pi,A,E}\left(1^k, z\right)$ be a random variable consisting of the output of the environment E following an execution of π with an adversary A, where k is the security parameter.

An *secure function evaluation* of f is a real evaluation of f that guarantees the same privacy and correctness as an ideal evaluation. The ideal setting captures the desired correctness and privacy achievable in the presence of malicious players. Secure computation guarantees that any damage that malicious players could be done in a real evaluation they could also do in the ideal one. See figure 2.

Definition 21. Let $\mathcal{F}$ be an ideal functionality, and let π an n-party protocol. A protocol π securely realizes $\mathcal{F}$ if for any PPT adversary A there exists a PPT adversary S s.t for all PPT environments E:

$$\left\{REAL_{\pi,A,E}\left(1^k, z\right)\right\}_{z\in\{0,1\}^*,k\in\mathbb{N}} \approx \left\{IDEAL_{\mathcal{F},S,E}\left(1^k, z\right)\right\}_{z\in\{0,1\}^*,k\in\mathbb{N}}$$

Remark 22. W.l.o.g we restrict attention to environments E that output a single bit.

Remark 23. For a more detailed version of the model see lecture 4.

4 Bridging Game Theory and Cryptography

Game Theory provides a wonderfully general notion of a mediated game. However, Game Theory does not address whether the players alone might be able to replace the trusted mediator, so as to play by themselves. Additionally, we would like to maintain

privacy of valuations with respect to other players; However, game theory does not take into account the privacy loss of the players.

Secure Computation, on the other hand, provides a general way of replacing trusted mechanisms with a specially-designed communication protocol in a way that preserves privacy of local information. Therefore, it may seem reasonable to replace the mediator by some multi-party secure computation. However, doing so can harm the fairness of the game and permits collusion between malicious players. For example, the GMW protocol uses randomness, which corrupted parties can use to collude.

A potential solution is to demand from the players not to co-operate and to punish players that do collude. Punishing is not always applicable since players can collude in ways that are not detectable by others. For example, one undetectable way to collude is by signaling: passing information by behaving in specific ways where the protocol allows a degree of freedom (for example, fixing one's randomness). Indeed, the ubiquity of randomness in secure protocols enables signaling that is undetectable by honest parties.

When payoffs are associated with the computation's outputs, it may beneficial for players to deviate from the protocol, for example by lying about their inputs.

Currently, secure computation does not model incentives for the players. Yet, if its correctness and privacy will be used for any real-world purpose, then the players will prefer some of its outputs to others, and will thus rationally respond to their incentives. Therefore, we should be careful in replacing the mediator with joint computation to ensure the incentive structure of the original is preserved: for instance, good strategies in the original game should remain good strategies in the new game.

Recall the Dodis-Halevi-Rabin result. Their result states that any equilibrium in the original game with the trusted correlating device remains an equilibrium in the modified game with general secure multi-party computation replacing the trusted device.

One may think that Dodis-Halevi-Rabin theorem should hold in our setting. However, the main difference is that the original mechanism assumes no communication among parties, while the standard secure function evaluation notion does not prevent communication among corrupted parties. Therefore, replacing the trusted mechanism by secure function evaluation protocol may lead to a game with different characteristics.

In addition, Dodis-Halevi-Rabin's result only guarantees that every equilibrium in the original game remains an equilibrium in the new game, in contrast to our case where we need in addition that the new game does not generate additional equilibria. (More generally, we need to guarantee that good strategies in the original game can be translated to good strategies in the new game. For instance, that any DST strategy in the original game translates to a DST strategy in the new game.)

We will introduce collusion-free secure function evaluation and argue that it suffices for replacing the mediator and maintain security even when all players pursue their selfish interests.

5 Collusion-free Secure Computation

Informally, collusion-freeness is the lack of ability of players to coordinate strategies adaptively and cooperate *during* the execution of the protocol.

To capture collusion freeness, we use a slightly different *ideal/real* paradigm, by explicitly adding incentives to both settings. The security notion means that any PPT external observer cannot distinguish the two games, even if he is allowed to interact freely with the players during the game.

The computation model is similar to the model presented in Lecture 4, except that there is a separate adversary for each corrupted party instead of one central adversary; these adversaries do not communicate with each other.

The definition of collusion-freeness is inspired by the definition of secure computation presented in Section 3, the main difference being 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 corresponding malicious parties in the real world. See figure 3

5.1 Execution in the Real World

In the real-world an n party protocol π is executed. The parties run the protocol π by exchanging messages in several rounds over a given communication channel. Let $I \subseteq [n]$ denote the set of corrupt parties. In the current setting each corrupted party

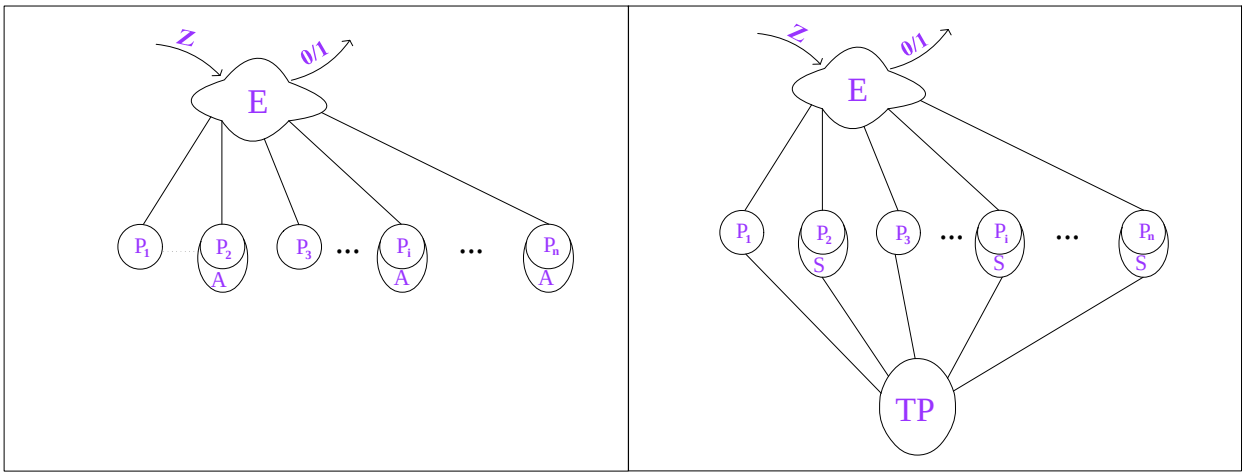

Figure 3: Real world vs. Ideal world

has its own adversary controlling him, as opposed to the previous definition where there is a central adversary controlling all the corrupted parties.

- The *input determination* and *computation* are the same as in Section 3.
- Given a set of adversarial strategies $A_I = \{A_i\}_{i \in I}$, let $REAL_{\pi, A_I, E}\left(1^k, z\right)$ denote the random variable consisting of the output of the environment E following an execution of π, where k is the security parameter

5.2 Execution in the Ideal World

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 $I \subseteq [n]$ denote the set of corrupt parties. The *ideal* execution proceeds as follows:

- *Input determination:* As in Section 3.
- *Result of the experiment:* Given a set of adversarial strategies $S_I = \{S_i\}_{i \in I}$ for the corrupted parties I (where S_i is the code running instead of the P_i's code), let $IDEAL_{\mathcal{F}, S_I, E}\left(1^k, z\right)$ be the random variable consisting of the environment E's output following an ideal world execution, where k is the security parameter.

In the current setting each corrupted party has its own isolated simulator controlling him, as opposed to the previous definition presented in section 3 where there is a central simulator controlling all the corrupted parties.

5.3 Collusion-Freeness

If we followed the standard definitional paradigm, we would require that for all I and any set of efficient real-world strategies $A_I = \{A_i\}_{i \in I}$ there should exist a set of efficient ideal-world strategies $S_I = \{S_i\}_{i \in I}$ such that the corresponding real- and ideal-world outcomes would be computationally indistinguishable. The problem with this approach is that it allows each S_i to depend on the subset I of corrupted parties as well as all the A_j (i.e., even for $j \neq i$), and thus this approach does not adequately model collusion-freeness. Since we want each S_i to depend only on A_i, we instead require for any adversary A the existence of a simulator S that makes the real and ideal worlds indistinguishable for all subsets I of corrupted parties.

Definition 24. Let $\mathcal{F}$ be an ideal functionality. A n-party protocol π collusion-resistant securely realizes $\mathcal{F}$ if for any PPT adversary A exists a PPT simulator S s.t for all PPT environment E:

$$\left\{REAL_{\pi,A,E}\left(1^k,z\right)\right\}_{z\in\{0,1\}^*,k\in\mathbb{N}} \approx \left\{IDEAL_{\mathcal{F},S,E}\left(1^k,z\right)\right\}_{z\in\{0,1\}^*,k\in\mathbb{N}}$$

Remark 25. Since the environment E corrupts the parties *one-by-one*, the simulator S does not depend on the subset I of corrupted parties.

6 Collusion-Free Multiparty Computation

Our goal is to construct a collusion-free protocol for secure computation of an arbitrary (poly-time) functionality $\mathcal{F}$. However, collusion-freeness is impossible in the plain model since the presence of randomness enables the parties to signal each other. In this section, we discuss two potential solutions that would capture collusion-freeness in a meaningful way. Each solution adds different assumptions to the plain model: the first solution relies on physical, public computation devices, and the second introduces a new entity called the *mediator*. The mediator-based solution uses randomness, whereas the physical solution doesn't.

6.1 A Solution via physical computation

The idea behind this solution is to implement GMW using physical implementations of cryptographic primitives. This approach enables us to eliminate the use of randomness. We first introduce the physical components of our protocol, and then describe the protocol in detail. This solution is inspired by the [3] solution.

6.1.1 Building Blocks

In this section we informally present the communication network in which the physical GMW games will be played.

Conceptually, we envisage a group of players, seating far apart around a large table, communicating via identical boxes (and super-boxes) and addition, multiplication, and shuffle machines. Informally, a player can choose a message or privately toss a coin (all participants can see that he tossed a coin, but nobody can see the result of the toss), write it on a piece of paper and put it into a new, empty box. So long as it is not opened, the box totally hides and guarantees the integrity of its content. Only the owner of a box can open it (in which case all the players are aware that he is opening it and he will be the only one to read its content). A player owns a box if it is physically close to him. By definition, the player originally locking a new box owns it. After that, ownership of a box can be transferred to another player by passing it to him. Furthermore, each box has a unique serial number, which only its owner can see. Other participants sitting around the table cannot see the serial number on the box.

A player can perform multiplication and addition of the values inside boxes by inserting two boxes into an appropriate machine, obtaining a single box containing the product or sum (respectively) of the values in the inserted boxes. The addition machine may accept either two boxes or m boxes as inputs (where m is the number of players). We can duplicate values using the addition machine.

Furthermore, a player can also publicly put four of his boxes into a new super-box SB, in which case none of them can be opened before SB. All super-boxes are again identical to each other, but are larger than (ordinary) boxes. The rules of ownership for super-box are the same as for boxes. There is only one possible way for a player i to open a super-box SB of his: all players observe that i has opened SB, and the player can choose only one sub-box inside which can be manipulated (e.g., opened or transferred). The remaining sub-boxes stay inside the locked superbox, which is placed in the middle of the table where none of the players can touch it.

Boxes and super-boxes always stay above the table and their transfers are always tracked by the players. The players can thus "mentally assign" to each box or super-box an identifier, j, insensitive to any possible change of ownership. The only exception is when a player i publicly puts his super-box into a shuffle machine: when it is taken out, the sub-boxes contents will remain unchanged and private, but their order is randomly permuted, in a way that is unpredictable to all players.

Lastly, at each moment a player may leave the room, thereby causing the game to abort.

Machinery To conclude, we have the following physical components: boxes, a machine to create boxes with content, multiplication and addition machines, and a machine to shuffle boxes into a super-box. We will represent '0' by an empty box and '1' by a box containing a piece of paper. The multiplication and addition machines will receive the boxes and create a new box with the appropriate content. (If the new box should represent '1', the machine will use a fresh piece of paper.)

In order to achieve the desired modeling we first define some ideal functionalities that have the properties of the physical devices:

6.1.2 Oblivious Transfer

Now we describe how to implement one-out-of-four oblivious transfer in the model described above.

Suppose Alice would like Bob to obtain one out of four values, in such way that Bob will not learn anything about the remaining three values and Alice won't know which value was chosen by Bob.

In order to achieve these properties, Alice will generate four boxes containing the values for the OT using the multiplication/addition machines and put those boxes into a new super-box. This super-box is shuffled using the shuffle machine. Afterwards Alice removes the super-box from the shuffle machine and passes it to Bob, who in turn opens the super-box and chooses exactly one sub-box.

This procedure is observable by all participants, who can verify that Alice and Bob abide by the rules of the model: for example, they verify that Bob opens exactly one box and places the super-box with the remaining sub-boxes locked in the middle of the table. However, the participants cannot see the serial number of the chosen sub-box.

By the construction, it is clear that Bob learns only one value: if Bob deviates and greedily opens more than one box, the other participants notice that and abort the protocol. Since the super-box was shuffled and passed to Bob, Alice and all other participants cannot see the serial number of Bob's chosen box, and therefore cannot know which of the four sub-boxes Bob chose.

The OT implementation described above will be invoked many times during the function evaluation.

Remark 26. The use of the shuffle machine in the implementation is vital to guaranteeing OT properties: without the shuffling, Alice will know which box Bob chose by the relative order of the chosen box in the super-box. For instance, if Bob chooses the third box in the super-box, Alice will know that it's the third box she inserted. The shuffling permutes the sub-boxes into an unpredictable order, therefore seeing which box was chosen (by its relative position in the super-box) won't reveal any information to Alice.

6.1.3 The Underlying Computational Model

We present a computational model similar to the model previously presented in class (see Lecture 4).

We would like to capture the situation of players sitting around a table, where each player can perform actions at any time (we assume time is discrete). This model is synchronous, and a computation in this model is a sequence of steps, where each step consists of two mini-steps, as follows:

1. Each machine is activated in some predefined order. Upon activation, the machine performs an internal computation, and possibly sends a message. In this mini-step, the "scheduler" collects all the messages, but does not deliver them yet.
   This captures the ability of players to perform actions at any time instant.
2. All messages collected in the first mini-step are delivered simultaneously to their recipients.
   This captures the ability of players to perform actions concurrently to other players' actions.

6.1.4 Ideal Functionalities for the Physical Assumptions

We present ideal functionalities that capture the physical properties of the physical actions in this model, as described above. Note that all actions in the physical world are public, and only the values (inputs and outputs) are private.

In order to capture the notions of *validity* (opened/unopened boxes) and *ownership* (who possesses a given box) in the physical world, we define the following ideal functionality, $\mathcal{F}_{\text{database}}$:

Functionality $\mathcal{F}_{\text{database}}$

Functionality $\mathcal{F}_{\text{database}}$ running with ideal functionalities $\mathcal{F}_{\text{ccon}}$, $\mathcal{F}_{\text{add}}$, $\mathcal{F}_{\text{mult}}$, $\mathcal{F}_{\text{shuffle}}$ and a memory M proceeds as follows:

Local variable $last\_cid = 0$

1. Upon receiving a tuple (owner, cid, P_i) from another ideal functionality, proceed as follows:
   (a) If the the tuple (cid, P_i, b) is recorded then send the message (valid,, cid, P_i, b) to that ideal functionality.
   (b) Otherwise, send (invalid_record).
2. Upon receiving a tuple $(\text{record}, cid, P_i, b)$ from another ideal functionality, proceed as follows:
   (a) If the the tuple (cid, P_j, b') is recorded for some party P_j then send the message (invalid_cid) to that ideal functionality.
   (b) Otherwise, add (cid, P_i, b) to M and send the message (recorded) to that ideal functionality.
3. Upon receiving a tuple $(\text{delete}, cid, P_i)$ from another ideal functionality, proceed as follows:
   (a) If the the tuple (cid, P_i, b) is recorded then delete (cid, P_i, b) from M and send the message (deleted) to that ideal functionality.
   (b) Otherwise, send the the message (invalid_record) to that ideal functionality
4. Upon receiving a tuple $(\text{delete\_read\_only}, cid, P_i)$ from another ideal functionality, proceed as follows:
   (a) If the the tuple $(\text{read\_only}, cid, P_i, b)$ is recorded then delete $(\text{read\_only}, cid, P_i, b)$ from M and send the message (deleted) to that ideal functionality.
   (b) Otherwise, send the the message (invalid_record) to that ideal functionality
5. Upon receiving a tuple $(\text{change\_owner}, cid, P_i, P_j)$ from another ideal functionality, proceed as follows:
   (a) If the the tuple (cid, P_i, b) is recorded then change the record in M to (cid, P_j, b) and send the message (changed) to that ideal functionality.
   (b) Otherwise, send the the message (invalid_record) to that ideal functionality.
6. Upon receiving a tuple $(\text{restrict\_access}, cid, P_i)$ from another ideal functionality, proceed as follows:
   (a) If the the tuple (cid, P_i, b) is recorded then change the record in M to $(\text{read\_only}, cid, P_i, b)$ and send the message (changed) to that ideal functionality.
   (b) Otherwise, send the the message (invalid_record) to that ideal functionality
7. Upon receiving a message (Generate_ID) from another ideal functionality, proceed as follows:
   (a) Compute $last\_cid = last\_cid + 1$.
   (b) Send ID $(last\_cid)$ to that ideal functionality.

Figure 4: The database functionality

Functionality $\mathcal{F}_{\text{database}}$, described in Figure 4, is a registrar of commitments ownership. It models boxes with serial numbers containing a value inside and possessed by some parties. Each record in the database represents an unopened box in the physical

world. The existence of access control enables us to model oblivious transfer.

Additionally, we model the actions available in the physical world. We need to capture the ability to create boxes with chosen or random content either privately or publicly. Furthermore, we need to capture the ability to pass a box to another player.

Functionality $\mathcal{F}_{\text{ccon}}$

Functionality $\mathcal{F}_{\text{ccon}}$ running with parties $P_1, ..., P_m$, $\mathcal{F}_{\text{database}}$ and an adversary $\overrightarrow{S}$ proceeds as follows:

1. Upon receiving a value (Content, sid, P_i, b) from P_i where $b \in \{0, 1\}$ proceed as follows:
   (a) Generate a unique cid (for instance $cid = last\_cid + 1$)
   (b) Record the tuple (cid, P_i, b)
   (c) Send the message (Receipt, sid, cid, P_i) to $P_1, ..., P_m$ and $\overrightarrow{S}$.
2. Upon receiving a value (Rand_content, sid, P_i) from P_i, proceed as follows:
   (a) Generate a unique cid
   (b) Choose a random bit $b \in \{0, 1\}$, and record the tuple (cid, P_i, b).
   (c) Send b to P_i and send the message (Receipt_rand, sid, cid, P_i) to $P_1, ..., P_m$ and $\overrightarrow{S}$.
3. Upon receiving a value (Public_content, sid, P_i, b) from P_i, proceed as follows:
   (a) Generate a unique cid
   (b) Record the tuple (cid, P_i, b).
   (c) Send the message (Receipt_public, sid, cid, P_i, b) to $P_1, ..., P_m$ and $\overrightarrow{S}$.
4. Upon receiving a value (Pass, sid, cid, P_i, P_j) from P_i, proceed as follows:
   (a) If the the tuple (cid, P_i, b) is recorded,
      i. change record to (cid, P_j, b)
      ii. send (Passed, sid, cid, P_i, P_j) to $P_1, ..., P_m$ and $\overrightarrow{S}$.
   (b) Otherwise, do nothing.

Figure 5: The functionality for committing chosen or random content

Functionality $\mathcal{F}_{\text{ccon}}$, described in Figure 5, proceeds as follows:

- The "commitment to a chosen content" mode is modeled by having $\mathcal{F}_{\text{ccon}}$ receive a value (Content, sid, P_i, b) from some party P_i (the committer). Here, sid is a session ID used to distinguish among various copies of $\mathcal{F}_{\text{ccon}}$; cid is the commitment ID, used to distinguish among the different commitments that take place within a single instance of $\mathcal{F}_{\text{ccon}}$**;** and $b \in \{0, 1\}$ is the value committed to. In response, $\mathcal{F}_{\text{ccon}}$ generates an unique commitment ID cid, lets all the parties and the adversary $\overrightarrow{S}$ know that P_i has committed to some value and that this value is associated with session ID sid and commitment ID cid. This is done by sending the message (Receipt, sid, cid, P_i) to all the parties and the adversary $\overrightarrow{S}$. This models the ability of players to choose a private value and publicly place it in a box.
- The "commitment to a random content" mode is modeled by having $\mathcal{F}_{\text{ccon}}$ receive a value (Rand_content, sid, P_i) from some party P_i. In response, $\mathcal{F}_{\text{ccon}}$ generates an unique commitment ID cid, randomly chooses a bit $b \in \{0, 1\}$, sends it to P_i, and informs all the parties and the adversary $\overrightarrow{S}$ that P_i has committed to some value. This is done in the same way as for commitments with chosen content.

This models the ability of players to choose a private random value using a designated machine and publicly place it in a box.

- The "commitment ownership passing" mode is modeled by having $\mathcal{F}_{\text{ccon}}$ receive a value (Pass, sid, cid, P_i, P_j) from some party P_i (the current owner). Here, in addition, the message includes the identity of the receiver. In response, $\mathcal{F}_{\text{ccon}}$ sends the message (Passed, sid, cid, P_i, P_j) to all the parties and the adversary $\overrightarrow{S}$. To avoid ambiguities, no two commitments with the same commitment ID are allowed.
  This models the ability of players to publicly pass a box to other players.

Next, we model the addition and multiplication machines. This is done in a straightforward manner.

Functionality $\mathcal{F}_{\text{add}}$

Functionality $\mathcal{F}_{\text{add}}$ running with parties $P_1, ..., P_m$,$\mathcal{F}_{\text{database}}$, and an adversary $\overrightarrow{S}$ proceeds as follows:

1. Upon receiving a value (Add, sid, P_i, cid, cid') from P_i, proceed as follows:
   (a) If the the tuples (cid, P_i, b) and (cid', P_i, b') are recorded,
      i. generate a unique $\hat{cid}$
      ii. record the tuple $\left(\hat{cid}, P_i, b + b'\right)$
      iii. send (Added, sid, cid, $cid'\hat{cid}$, P_i) to $P_1, ..., P_m$ and $\overrightarrow{S}$.
   (b) Otherwise, do nothing
2. Upon receiving a value (Add_m_values, sid, $cid_1, \ldots, cid_m$, P_i) from P_i, proceed as follows:
   (a) If the tuples $(cid_1, P_i, b_1), \ldots, (cid_m, P_i, b_m)$ are recorded,
      i. send $result = \sum_{j=1}^{m} b_j$ to P_i
      ii. delete the records $(cid_1, P_i, b_1), \ldots, (cid_m, P_i, b_m)$
      iii. send (Added_m_values, sid, $cid_1, \ldots, cid_m$, P_i) to$P_1, \ldots, P_m$ and $\overrightarrow{S}$.
   (b) Otherwise, do nothing

Figure 6: The functionality computing addition

Functionality $\mathcal{F}_{\text{add}}$, described in Figure 6, proceeds as follows:

- The "add two committed contexts" mode is modeled by having $\mathcal{F}_{\text{add}}$ receive a message (Add, sid, P_i, cid, cid') from some party P_i. Here cid,cid' are the commitment ID's of the addends, and $\hat{cid}$ is the commitment ID of the result. In response, $\mathcal{F}_{\text{add}}$ generates an unique $\hat{cid}$ (commitment ID of the result) and records the result. Then $\mathcal{F}_{\text{add}}$ lets all parties and the adversary $\overrightarrow{S}$ know that P_i has added two values associated with session ID sid and commitment IDs cid and cid', and that the resulting value is associated with session ID sid and commitment ID $\hat{cid}$.

Functionality $\mathcal{F}_{\text{mult}}$, described in Figure 7, proceeds in the same way as $\mathcal{F}_{\text{add}}$, *mutatis mutandis*.

Next, we model Oblivious Transfer (OT), which was described in Subsection 6.1.2.

Functionality $\mathcal{F}_{\text{mult}}$

Functionality $\mathcal{F}_{\text{mult}}$ running with parties $P_1, ..., P_m$, $\mathcal{F}_{\text{database}}$, and an adversary $\overrightarrow{S}$ proceeds as follows:

1. Upon receiving a value (Multiply, sid, P_i, cid, cid') from P_i, proceed as follows:
   (a) If the the tuples (cid, P_i, b) and (cid', P_i, b') are recorded,
      i. generate a unique $\hat{cid}$
      ii. record the tuple $\left(\hat{cid}, P_i, b \cdot b'\right)$
      iii. send (Multiplied, sid, cid, cid', $\hat{cid}$, P_i) to $P_1, ..., P_m$ and $\overrightarrow{S}$.
   (b) Otherwise, do nothing

Figure 7: The functionality computing multiplication

Functionality $\mathcal{F}_{\text{shuffle}}$

Functionality $\mathcal{F}_{\text{shuffle}}$ running with parties $P_1, ..., P_m$, $\mathcal{F}_{\text{database}}$, and an adversary $\overrightarrow{S}$ proceeds as follows:

1. Upon receiving a value (Shuffle, sid, cid_{00},cid_{01}, cid_{10}, cid_{11}, P_i, P_j) from P_i, proceed as follows:
   (a) If the the tuples (cid_{00}, P_i, b_{00}) , (cid_{01}, P_i, b_{01}),(cid_{10}, P_i, b_{10}), and (cid_{11}, P_i, b_{11}) are recorded
      i. change records to $(\text{read\_only}, cid_{00}, P_j, b_{00})$, $(\text{read\_only}, cid_{01}, P_j, b_{01})$,$(\text{read\_only}, cid_{10}, P_j, b_{10})$, and $(\text{read\_only}, cid_{11}, P_j, b_{11})$.
      ii. send (Passed_read_only, sid, cid_{00}, cid_{01}, cid_{10}, cid_{11}, P_i, P_j) to $P_1, ..., P_m$ and $\overrightarrow{S}$
   (b) Otherwise, do nothing
2. Upon receiving a value (Choose, sid, $cid_{\ell k}$, P_j) proceed as follows:
   (a) If the the tuple $(\text{read\_only}, cid_{\ell k}, P_j, b_{\ell k})$ is recorded
      i. generate a unique $\hat{cid}$
      ii. record the tuple $\left(\hat{cid}, P_j, b_{\ell k}\right)$
      iii. delete the records $(\text{read\_only}, cid_{1-\ell,1-k}, P_j,\ b_{1-\ell,1-k})$, $(\text{read\_only}, cid_{\ell,1-k}, P_j,\ b_{\ell,1-k})$, $(\text{read\_only}, cid_{\ell k}, P_j, b_{\ell k})$, and $(\text{read\_only}, cid_{1-\ell,k}, P_j,\ b_{1-\ell,k})$
      iv. send the message (Chosen, sid, $\hat{cid}$, P_j) to $P_1, ..., P_m$ and $\overrightarrow{S}$.
   (b) Otherwise, do nothing

Figure 8: The functionality for shuffling

Functionality $\mathcal{F}_{\text{shuffle}}$, which is used to models OT, proceeds as follows.

- The permuting operation is modeled by having $\mathcal{F}_{\text{shuffle}}$ receive a value (Shuffle, sid, cid_{00},cid_{01}, cid_{10}, cid_{11}, P_i, P_j) from some party P_i. Here, P_j is the receiver. In response, $\mathcal{F}_{\text{shuffle}}$ changes the owner of the commitment to P_j and the permission to "read-only". Then, it passes the commitments to P_j and lets all parties and the adversary $\overrightarrow{S}$ know that P_j is the new owner of the "read-only" commitments.
- The choosing operation is modeled by having $\mathcal{F}_{\text{shuffle}}$ receive a value (Choose, sid, $cid_{\ell k}$, P_j) from some party P_j. In response, $\mathcal{F}_{\text{shuffle}}$ generates an unique $\hat{cid}$ (new commitment ID for the chosen commitment), records the commitment

with it new ID, deletes the "read-only" commitments and informs the parties and the adversary that P_j has chosen a commitment that associated with session ID sid and commitment ID $\hat{cid}$.
This ensures that all players have no clue regarding the serial number of the chosen box.

- Destroying the boxes after choosing one of them is modeled by having $\mathcal{F}_{\text{shuffle}}$ delete all records associated with the OT after the receiving party chose his box. Deleting all records ensures that no additional information regarding Sender's input to the OT will be revealed.

Finally, we would like to capture the ability of players to publicly open a box they possess, while keeping its content private. This is done in a straightforward manner.

Functionality $\mathcal{F}_{\text{local\_open}}$

Functionality $\mathcal{F}_{\text{local\_open}}$ running with parties $P_1, ..., P_n$, $\mathcal{F}_{\text{database}}$, and an adversary $\overrightarrow{S}$ proceeds as follows:

1. Upon receiving a value (Local_open, sid, cid, P_i) from P_i, proceed as follows:
   (a) If the the tuple (cid, P_i, b) is recorded then send the message (Open, sid, cid, P_i, b) to P_i and (Opened, sid, cid, P_i) to $P_1, ..., P_n$ and $\overrightarrow{S}$.
   (b) Otherwise, do nothing.

Figure 9: The functionality that locally opens messages

Functionality $\mathcal{F}_{\text{local\_open}}$, described in Figure 9, proceeds as follows. $\mathcal{F}_{\text{local\_open}}$ is initiated by receiving a message (Local_open, sid, cid, P_i) from some party P_i. In response, $\mathcal{F}_{\text{local\_open}}$ hands the value (Open, sid, cid, P_i, b) to P_i and lets all parties and the adversary $\overrightarrow{S}$ know that P_i has opened a commitment he owns with the serial number cid.

Remark 27. These ideal functionalities communicate among themselves via secure channels. All record, delete, and ownership requests are done by sending an appropriate request to $\mathcal{F}_{\text{database}}$.

6.1.5 An Ideal Functionality for Secure Function Evaluation

We would like an ideal functionality that would capture multi-party computation in the model presented above. This ideal functionality proceeds in rounds, and at each round may leak information to the adversarial parties.

We would like to capture an ideal process for securely evaluating a function in a physical computation model. We would like to allow parties to act simultaneously at all times, therefore the functionality proceeds in rounds. At each round all parties may act if they wish to. Since each action can depend only on actions that precede it, we can linearize and discretize the actions through the use of rounds.

Moreover, we would like all actions to be transparent to all participants, that is: all parties should be aware of the actions taken so far. Our goal is to maintain the inputs' and outputs' privacy while making the action public. To that end, we have the functionality inform all parties about any defection.

Functionality $\mathcal{F}_{\text{sfe}}^{f}$

Functionality $\mathcal{F}_{\text{sfe}}^{f}$ for computing a PPT function f, running with parties $P_1, ..., P_n$ and an adversary $\overrightarrow{S}$, proceeds in rounds. In each round, it receives a message from each player. Each of these messages is one of the following:

1. An `okay` message.
2. An `abort` message.
3. A `cheat` message, followed by a description of the defection. (A detailed description is given in the sequel.)
4. An input to the function, followed by a message of one of the previous kinds. (This in particular allows the functionality to prevent output from some parties.)

In each round, the ideal functionality proceeds as follows:

1. If all the messages received from the corrupted parties in the current round are `okay` messages, then send a `continue` message to all the adversarial parties.
2. If at least one of the messages was an `abort` message, then:
   (a) Send an `abort` message to all players.
   (b) Send the identities of the aborting players to all parties.
3. If at least one of the messages was a `cheat` message, then:
   (a) Send an `abort` message to all players.
   (b) Send the identities of the cheating players and the description of the defections to all parties.
4. If the message received as an input to the function followed an `abort` or `cheat` message from the same player, then:
   (a) Evaluate the function upon receiving all inputs.
   (b) Send to all parties messages of type 2(b) or 3(b), as appropriate.
   (c) Send an `abort` message to the player who sent the `abort` or `cheat` message.
5. If the message received as an input to the function followed an `okay` message from the same player, then:
   (a) Evaluate the function upon receiving all inputs.
   (b) Send each party his respective private output.

Figure 10: The functionality for secure function evaluation

The `abort` message models the ability of players to stop participating in the protocol altogether, i.e., to leave the room.

The reason for $\mathcal{F}_{\text{sfe}}^{f}$ to provide the honest parties with the description of the deviations is that in a perfect physical world each player is aware of all other players' actions.

6.1.6 The Protocol

We now present a construction for m parties to compute any functionality that is expressed by an arithmetic circuit which consists of OR and AND gates. We assume that there are m parties $P_1, ..., P_m$, and each input consists of n bits. We further assume that each party holds the values of some of the input wires to the circuit, and in addition there are random input wires.

Our construction is similar to the GMW protocol in the honest-but-curious setting, where the goal is to propagate, via private computation, shares of the input wires of the circuit to shares of all wires of the circuit, so that finally we obtain shares of the

output wires of the circuit. In contrast, our construction will be resilient to Byzantine adversaries.

Step 0: Insuring the only faults are "abort faults" Since all the commitment ID's (cid's) are generated by $\mathcal{F}_{ccon}$, $\mathcal{F}_{add}$ and $\mathcal{F}_{mult}$ sequentially and each ideal functionality sends all the commitment IDs involved in the operation it performs ($\mathcal{F}_{ccon}$ in "pass", $\mathcal{F}_{add}$ in "Add" and $\mathcal{F}_{mult}$ in "Multiply") to all parties, each party P_i can verify that a party P_j is using the right boxs for the operations ("pass", "add", etc) by reconstructing the sequence of ID generation. If a party P_j doesn't follow the protocol then P_i will detect that and abort (this cheating will be included in players transcript). Moreover, all operations, the ID's of the parties involved in them, and the commitment ID's associated with them are fully public. Hence, each player can verify that the protocol is followed properly.

Step 1: Sharing the inputs Each party P_i generates public commitments to 0 and then to 1 using $\mathcal{F}_{\text{ccon}}$. To guarantee total order, P_i will generate his commitment only after P_{i-1} finishes committing to his public values. Afterwards, each party P_i (in increasing order of i) splits each of its input bits. That is, for every input bit x_j^i for $j = 1, ..., n$ and $k \neq i$ (in increasing order of k) the party P_i generates a commitment to a random bit r_j^k using $\mathcal{F}_{\text{ccon}}$. Then, party P_i sets his own share of his j-th input wire by generating a commitment to $x_j^i + \sum_{k \neq i} r_j^k$ using $\mathcal{F}_{\text{ccon}}$. Then, each party P_i (in increasing order of i) shares each of its input bits with all other parties (in order): P_i sends to party P_k its share of the j-th input wire of party P_i. Additionally, each party P_i (in increasing order of i) generates a commitment to a random bit using $\mathcal{F}_{\text{ccon}}$ for each of his random input wires.

During this process, each party verifies, as described in **Step 0**, that the proceedings are done correctly.

Step 2: Circuit emulation Proceeding by the order of wires, the parties use their commitments to the two input wires to a gate in order to privately compute shares for the output-wire of the gate. Suppose that the parties hold the commitments that correspond to the two input wires of some gate: that is, for $i = 1, ..., m$, party P_i holds the commitments to the shares a_i, b_i, where $a_1, ..., a_m$ are the shares of the first wire, and $b_1, .., b_m$ are the shares of the second wire. We consider the usual two cases:

Evaluation of OR gate:

Each party (in increasing order) sets its output wire to be $a_i + b_i$ by using $\mathcal{F}_{\text{add}}$. In other words, each party commits to the XOR of a_i and b_i using the functionality $\mathcal{F}_{\text{add}}$. That way, the shared output will be given by

$$\sum_{i=1}^{m}(a_i + b_i) = a + b$$

and that is the desired true output.

Evaluation of AND gate:

The true value is given by

$$\left(\sum_{i=1}^{m} a_i\right) \cdot \left(\sum_{i=1}^{m} b_i\right) = \sum_{i=1}^{m} a_i b_i + \sum_{i=1}^{n} \sum_{i<j\leq m} (a_i b_j + a_j b_i)$$

Thus, if every party P_i will have his share set to

$$w_i = a_i b_i + \sum_{i<j\leq m} (a_i b_j + a_j b_i)$$

then the true output will be given by $\sum w_i$.

Since each party P_i owns commitments to a_i, b_i it is left with the task of finding out the value of $a_i b_j + a_j b_i$ for every $i < j$. For every i, j such that $i < j$, it is P_j's responsibility to help the party P_i learn $a_i b_j + a_j b_i$. The solution is to use $\mathcal{F}_{\text{shuffle}}$: The party P_j will generate "read only" commitments to $a_i b_j + a_j b_i$ for every possible combination of (a_i, b_i) using $\mathcal{F}_{\text{ccon}}$, $\mathcal{F}_{\text{add}}$, $\mathcal{F}_{\text{mult}}$, and the public commitments to 0 and 1. Then, using $\mathcal{F}_{\text{shuffle}}$, it will transfer the commitment to the correct combination to party P_i by passing all four commitments to P_i and allowing P_i to choose the one that corresponds to his shares. That way,

the i-th party will obtain a commitment to the j-th share it needs in order to compute its own share using the suitable ideal functionalities.

Again, as in **Step 1** each party P_k verifies that P_i was using the right values (commitments) in the evaluation (in other word, in case of the OR gate evaluation, P_i indeed set w_i to be $a_i + b_i$ and not $a_{i+1} + b_{i-1}$for example), and that the calculations are done in the right order. And again, this is done as described in **Step 0**.

Step 3: Output reconstruction Once the shares of the circuit output wires are computed, each party passes its commitment to the share of each such wire to the party with which the wire is associated (in increasing order). This is done using $\mathcal{F}_{\text{ccon}}$. Once again the each party verifies that the right commitment was sent to the right party (as described in **Step 0**).

Then, all parties simultaneously adds all m shares using $\mathcal{F}_{add}$. Afterwards, all parties simultaneously open their boxes containing the sums and discover the function output using $\mathcal{F}_{\text{local\_open}}$.

Again, each party P_k verifies that P_i was using the right values (commitments) in the evaluation and that the openings are done in the right order (this is done as described in **Step 0**).

Remark 28. It is crucial that player P_i starts his computation only after player P_{i-1} had finished.

Remark 29. This protocol guarantees perfect fairness. No player can see his output while preventing other players from seeing theirs.

Remark 30. An important observation is that at any moment there exists only one legal action for only one party in the protocol. Thus, this protocol has exactly one valid transcript. This is unavoidable in order to prevent corrupted parties from signaling.

Theorem 31. *The P_{GMW} protocol collusion-resistant securely realizes $\mathcal{F}_{\text{sfe}}^{f}$ for any PPT function f.*

Proof. We prove this theorem only for the case of static corruptions, although it holds for adaptive corruptions as well. We have to show that for any subset I of corrupted parties and any adversarial strategies A_i for the corrupted parties, there exist simulators S_i ($i \in I$) such that the output of the environment is indistinguishable (as defined above). In our case, the output ensembles will be identically distributed.

High-level description of the simulator S_i. Recall that there is only one legal execution of the protocol, wherein at each moment only a single player should send messages (except for the last two rounds, in which all players simultaneously perform an action). Therefore, the simulator will internally run the adversary A_i and honestly simulate step-by-step all the communication in the protocol, aided by the leakage provided by the ideal functionality $\mathcal{F}_{\text{sfe}}^{f}$. If A_i aborts or cheats during that execution, S_i will detect that (as described in **Step 0** of the protocol) and report that to $\mathcal{F}_{\text{sfe}}^{f}$. Otherwise, S_i will send to $\mathcal{F}_{\text{sfe}}^{f}$ the value x_i' that the adversary used.

Detailed description of the simulator S_i. The simulation is done in a round-by-round manner. Since the ideal functionality $\mathcal{F}_{\text{sfe}}^{f}$ enforces round discipline, no simulator will proceed to round k unless all the simulators successfully passed round $k-1$ (where "successfully passed" is that all the internal adversaries neither cheated nor aborted in round $k-1$). Thus, if S_i is simulating round k, then all other simulators are simulating round k as well.

If the internal A_i cheats/aborts at round k: by the round-by-round simulation, we have had a fair game until now, and this is the first cheat/abort in the game. We inform the ideal functionality $\mathcal{F}_{\text{sfe}}^{f}$ about this by sending an appropriate `abort`/`cheat` message, sending A_i's response messages (if any), and outputting whatever A_i outputs. The `cheat` message sent to $\mathcal{F}_{\text{sfe}}^{f}$ is accompanied by the deviating message that S_i received from A_i at the kth round. If the deviating message is an 'open' request to $\mathcal{F}_{\text{local\_open}}$ (which is played by S_i), then S_i chooses a random bit and gives it to A_i.

If some other internal A_j's cheat/abort at round k: the simulator S_i will be informed at round k by $\mathcal{F}_{\text{sfe}}^{f}$ about the identities of the cheating players and how they cheated. According to the message received from $\mathcal{F}_{\text{sfe}}^{f}$, the simulator S_i will generate appropriate messages and send them to the internal A_i and will output whatever A_i outputs.

In **Step 1** of the protocol, S_i plays the role of $\mathcal{F}_{\text{ccon}}$ for A_i. In this step, A_i should request $n-1$ random shares and one commitment to chosen content. All of these will be supplied by S_i. In the process, S_i will discover the value x_i' that A_i uses as an input to the protocol. All of this holds only if A_i plays the protocol correctly (neither aborts nor cheats) until this step.

In all other phases of the protocol, up to opening the output wires (after all the parties have summed their output shares), the simulator S_i will honestly simulate all the communication in the protocol. (In case of someone aborting/cheating the simulator S_i will behave as described above.)

In the last step of the protocol, when the parties are supposed to open the commitments to their output wires, the simulator S_i collects the last protocol message from A_i, sends to $\mathcal{F}_{\text{sfe}}^f$ the x_i' recovered earlier and an `abort`/`cheat`/`ok` message as appropriate, receives from $\mathcal{F}_{\text{sfe}}^f$ the function's output and a description of aborts/cheatings (if any) that happened at this round, and (according to the messages received from the ideal functionality and A_i) generates messages and sends them to A_i and outputs whatever A_i outputs.

□

Remark 32. Recall that, according to our model of computation, in each round we first collect the messages for that round and only then do we deliver all of them. This allows the simulator to first collect those messages, then send an appropriate message to $\mathcal{F}_{\text{sfe}}^f$ and receive its response, and only then, relying on the information received from $\mathcal{F}_{\text{sfe}}^f$, generate suitable messages and deliver them to the internal adversary A_i.

Output distribution indistinguishability. All the players' actions are public since all actions are done via the ideal functionalities (capturing the corresponding physical actions), which inform all the players regarding the actions other players do. By the protocol's construction, there is only one legal run (transcript) of the protocol. Therefore, as mentioned previously, any player at any time can verify (even during the execution of the protocol) that the protocol executed correctly: that is, no cheating can go undetected.

On the one hand, if there occurred cheating/aborting at some round k in the real world, then in the ideal world, once the simulation reaches round k, we will discover the defection via the leakage provided by the ideal functionality $\mathcal{F}_{\text{sfe}}^f$ (as described in the simulation). If the defection occurs in the round of opening the output wires, then the other parties open their commitments to the summed output wires to obtain their output, and their transcripts include both the output value and the defection. Due to the leakage and the round-by-round simulation, each internal adversary's view is *identical* to its view in the real world. Moreover, the views of all internal adversaries are consistent with each other.

On the other hand, if there was no deviation, the simulators play honestly, making the view of the simulated adversaries in the simulation *identical* to the view of the real-world adversaries. Each simulator has only one legal transcript, which that simulator knows. The transcripts produced by the simulators would be the legal transcripts, which are consistent with each other, and the union of these transcripts represents the only legal execution of the protocol. In particular, the output of the function would be same as in the real world.

Remark 33. Each player knows the honest strategies of all players and can compute them.

6.2 Mediated Model Solution

In this section we present a mediated model for multiparty computation and show how to implement a general compiler for collusion-free multiparty computation in it. The implemented presented here is based on Alwen et al's compiler [1]'s compiler. Their idea is to add a central party, the *mediator*, which the parties partially trust: if the mediator is honest, the resulting protocol is secure against collusions; additionally, even if the mediator is corrupt, the resulting protocol will be secure according to the standard notion. The only requirement from the compiled protocol π is security against honest-but-curious adversaries.

6.2.1 Communication Model

In the mediated model, each party is able to communicate only with a mediator M. In this case the communication model is a star graph with M in the center. In real life this models a group of people sitting in different rooms without any communication devices while an external party (the mediator) walks from room to room telling each of the agents what was the result of the previous round. If the mediator is honest the parties are separated from each other. If the mediator is corrupt it can create coalitions between a set of parties (the corrupted ones) by passing all the information they want to each other using the dishonest mediator as a channel. See figure 11 for the model. Contrary to classical game theory, where the mediator is postulated to be honest, in cryptography no assumptions are made about the mediator, who may be either honest or corrupted.

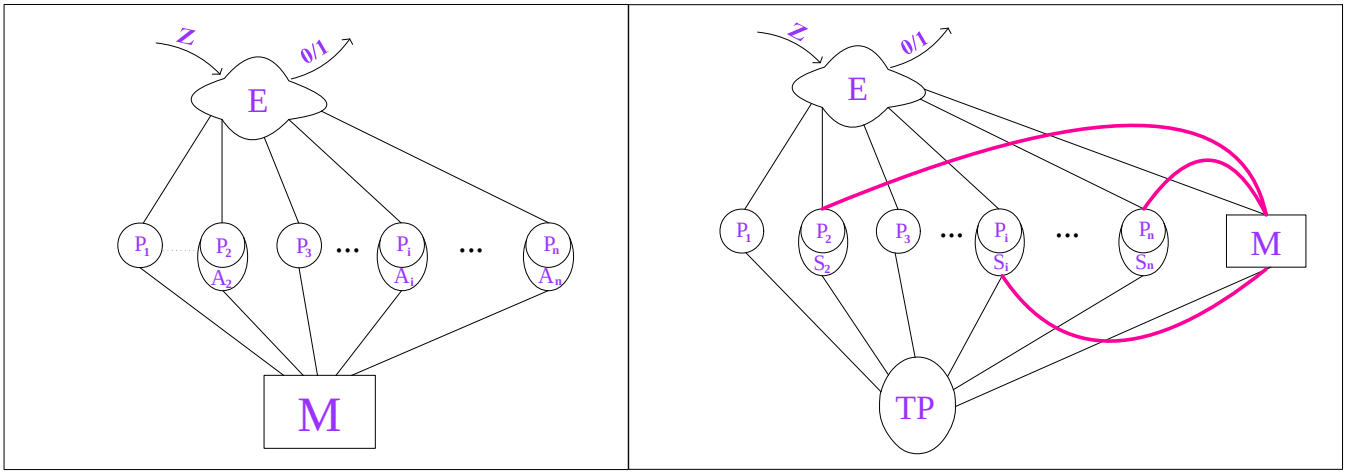

Figure 11: Real world vs. Ideal world in the mediated model (corrupted parties can use pink communication lines)

Definition 34. Let $\mathcal{F}$ be a functionality and Π be an $(n+1)$-party protocol computing $\mathcal{F}$ in the mediated model. say Π is a collusion-free protocol computing $\mathcal{F}$ if there is a set $\{\mathrm{Sim}_i\}_{i\in n}$ of efficiently-computable transformations such that, for every set $\mathcal{I} \in [n]$ of corrupt parties and any PPT strategies $\{A_i\}_{i\in n}$, setting $S_i = \mathrm{Sim}_i(1^k, A_i)$ for $i \in \mathcal{I}$ implies that the following two distributions are computationally indistinguishable:

$$\{REAL^{\text{mediated}}_{\Pi,A_I(z)}\left(1^k, x\right)\}_{z\in\{0,1\}^*,k\in\mathbb{N}} \approx \{IDEAL_{\mathcal{F},S_I(z)}\left(1^k, x\right)\}_{z\in\{0,1\}^*,k\in\mathbb{N}},$$

where $A_I = \{A_i\}_{i\in I}$.

Alwen et al do not consider full composability, as opposed to our previous definition.

Now we can formally state the main result of this section:

Theorem 35. *Given a (poly-time) functionality $\mathcal{F} = (f_1, ..., f_{n+1})$ and a protocol π which is secure in the standard sense, we construct a compiled protocol Π which uses a mediator M and satisfies the following properties:*

If the mediator M is honest then Π CF-realize $\mathcal{F}$, and Π realizes $\mathcal{F}$ (according to the standard security notion) even if the mediator M is corrupted.

6.2.2 Part 1: Broadcast "under cover" of message m by player P

We will first present a subroutine “Broadcast under cover” that implements a broadcast channel. Later on we present the compiler, which will use the subroutine. We assume that all the parties have public keys for encryption and signature $PK_1, ..., PK_n$, which are published by means of a presumed PKI.

Broadcast under cover is a way to force the mediator to broadcast honestly. In our model, all communication is done via the mediator, and if the mediator is corrupted, he can send different messages to different parties. We need the mediator to broadcast honestly in order to preserve the security of the underlying protocol π.

The protocol proceeds as follows:

1. P sends $m, s = \mathrm{sig}(m)$ to M. The signature is using the private key PK_p
2. M sends $c_i = \mathrm{com}(m, s)$ to each player P_i
3. Each P_i returns a signature $s_i = \mathrm{sig}(c_i)$ to M

(a) If any P_i fails to send a valid signature, then M set $\mathcal{C}_j = \mathrm{com}(\perp)$ for all j

(b) Else M sets $\mathcal{C}_j = \mathrm{com}(c_1, ..., c_n, s_1, ..., s_n)$ for each j

4. M sends $\mathcal{C}_j$ and proves in Zero Knowledge to each P_j that either:

   (a) $\mathcal{C}_j$ opens to $(c_1, ..., c_n, s_1, ..., s_n)$

   (b) each $c_k$opens to (m, s)

   (c) each s_k is a valid signature on c_k with respect to PK_k

   (d) s is a valid signature on m with respect to PK_p

   or

   (a) or $\mathcal{C}_i$ opens to $\perp$

Remark 36. If M is corrupt then some of the parties may get a commitment to $\perp$ and some a commitment to message m. If M is honest, the only way for a party to receive a commitment to $\perp$ is when some other party aborts.

6.2.3 Overview of the compiled protocol

In this part we illustrate the high-level structure of the protocol.

Let $P_1, ..., P_n$ be a set of n parties, each communicating with a mediator M, who wish to compute some (randomized) functionality $\mathcal{F} = (f_1, ..., f_{n+1})$ where each f_i maps $n + 1$ inputs to a single output. Let π be a protocol that securely computes $\mathcal{F}$ in the standard communication model with broadcast. (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:

- The parties run the underlying protocol π, where all the communication is done with the mediator's help:
  - Each message is generated jointly by the sending party and the mediator.
  - Only commitments to messages are sent, and the mediator re-commits to each party separately, so that the actual communication with each party appears to be independent of the communication with the other parties.
- The mediator commits to the messages that the good parties wish to send. (This is done using a PKI - each party signs its messages.)
- All parties have a consistent view of the messages sent by the bad parties. (This is done using the broadcast protocol from 6.2.3.)

Intuition When the mediator is honest, the only information he transfers among parties is the information the protocol instructs him to; that is, each party receives a commitment to the same message, generated using new randomness. Therefore, each party sees only independent commitments to messages rather than the messages themselves; furthermore, randomness is not transfered among the players. Hence, intuitively, the protocol is collusion-free.

When the mediator is dishonest, any damage that can be caused by the corrupted parties with the mediator's assistance can also be done in the ideal world: intuitively, this is since π securely realizes $\mathcal{F}$ in the standard communication model and broadcast "under covers" securely realizes broadcast channel. The protocol above, which consists of running π using broadcast "under covers", is therefore secure since it can be shown that the composition remains secure.

6.2.4 Part 2: The Compiler

Assume the protocol π proceeds in rounds where in each round a single party sends a message, and all messages are done over a broadcast channel.

The protocol consists of three stages. In the first stage, the parties commit to their inputs and random coins for protocol π. In the second stage, the parties simulate π, round-by-round, as follows. If it is P_i's turn to broadcast (for $i \in [n]$), then P_i runs GMW with the mediator on their internal states as input; the mediator obtains the next-message m along with a signature of P_i on this message (and the current round number). If it is the mediator's turn to broadcast, the mediator simply computes the next-message m on its own, and then runs broadcast "under covers" using m. As long as everyone behaves honestly, each party thus learns commitments to all messages of the protocol. In the third stage, the mediator runs GMW with each P_i to enable P_i to learn its output. We now describe the protocol formally.

Stage 1 - input commitment and randomness commitment:

1. Each P_i commits to his input x_i to M using randomness y_i
2. For each P_i
   - Each P_j broadcasts "under cover" (using the above protocol) $c_{ij} = \mathrm{com}(r_{ij})$, when r_{ij} is a random string.
     - r_{ij} will be used to determine the random string of P_j in that protocol: recall that $r_j = \oplus r_{ij}$

Stage 2 - round-by-round emulation of π: In each round k

1. (M learns the round-k message of π.)
   Let Party P_i be the sending party in the kth round of π .
   - P_i and M run two party GMW protocol to compute P_i's next message:
     - input for P_i:
       - the initial input x_i, and the commitment randomness y_i
       - the random string r_{ii} and the commitment to this random string c_{ii}
       - the output commitments of the broadcast till now $\mathrm{com^i}_1, \mathrm{com^i}_2, ..., \mathrm{com^i}_{k-1}$, as P_i received from M
       - the public keys $PK_1, ..., PK_n$
       - P_i's signing key sk_i
     - input for M:
       - the decommitment information for each $\mathrm{com^i}_1, \mathrm{com^i}_2, ..., \mathrm{com^i}_{k-1}$
     - output of P_i:
       - no output
     - output of M:
       - If the decommitment info given by M does not match $\mathrm{com^i}_1, \mathrm{com^i}_2, ..., \mathrm{com^i}_{k-1}$ and $PK_1, ..., PK_n$ given by P_i, then M gets an abort symbol.
       - Else, M gets m, which is P_i's next message according to $\pi$when run on input x_i, random input r_i, and given all the messages $m_1, ..., m_{k-1}$ that are committed to by $\mathrm{com^i}_1, \mathrm{com^i}_2, ..., \mathrm{com^i}_{k-1}$, and $s = \mathrm{sig}_{sk_i}(m)$.
2. M broadcast m as in the broadcast "under cover" protocol.
   - If P_i aborted during the computation of m then M broadcasts "under cover" the message $\perp$.

Stage 3 - output determination:

1. (P_i learn its output message of π.)
   Let Party P_i be the party to receive its output.

- P_i and M run two party GMW protocol to compute P_i's output (which is encrypted under PK_i):
  - input for P:
    - the commitment for the output message $c = \text{com}(m)$
    - the public keys$PK_1, ..., PK_n$
    - P_i's signing key sk_i
  - input for M
    - the decommitment information for c
  - output of P_i
    - If the decommitment info given by M does not match c and $PK_1, ..., PK_n$ given by P_i, then P_i gets an abort symbol.
    - Else, P_i gets $d = \underset{sk_i}{\text{Dec}}(m)$, where m is the the message committed to in c.
  - output for M
    - no output.

This protocol collusion-free realizes any functionality $\mathcal{F}$. We do not prove this here.

References

[1] J. Alwen, J. Katz, Y. Lindell, G. Persiano, A. Shelat, and I. Visconti. Collusion-Free Multiparty Computation in the Mediated Model. CRYPTO 2009:524-540.

[2] J. Katz, Y. Lindell. Collusion-Free Multiparty Computation in the Mediated Model.
available in http://eprint.iacr.org/2008/533.pdf

[3] S. Izmalkov, S. Micali, M. Lepinski. Rational Secure Computation and Ideal Mechanism Design. FOCS 2005:585-595

[4] Y. Mansour. Computational Game Theory course.
Available at http://www.cs.tau.ac.il/~mansour/course_games/course_games_03_04.html