Fair and Robust Multi-Party Computation using a Global Transaction Ledger

Aggelos Kiayias
aggelos@di.uoa.gr

Hong-Sheng Zhou
hszhou@vcu.edu

Vassilis Zikas
vzikas@inf.ethz.edu

June 10, 2015

Abstract

Classical results on secure multi-party computation (MPC) imply that fully secure computation, including fairness (either all parties get output or none) and robustness (output delivery is guaranteed), is impossible unless a majority of the parties is honest. Recently, cryptocurrencies like Bitcoin where utilized to leverage the fairness loss in MPC against a dishonest majority. The idea is that when the protocol aborts in an unfair manner (i.e., after the adversary receives output) then honest parties get compensated by the adversarially controlled parties.

Our contribution is three-fold. First, we put forth a new formal model of secure MPC with compensation and we show how the introduction of suitable ledger and synchronization functionalities makes it possible to express completely such protocols using standard interactive Turing machines (ITM) circumventing the need for the use of extra features that are outside the standard model as in previous works. Second, our model, is expressed in the universal composition setting with global setup and is equipped with a composition theorem that enables the design of protocols that compose safely with each other and within larger environments where other protocols with compensation take place; a composition theorem for MPC protocols with compensation was not known before. Third, we introduce the first robust MPC protocol with compensation, i.e., an MPC protocol where not only fairness is guaranteed (via compensation) but additionally the protocol is guaranteed to deliver output to the parties that get engaged and therefore the adversary, after an initial round of deposits, is not even able to mount a denial of service attack without having to suffer a monetary penalty. Importantly, our robust MPC protocol requires only a *constant* number of (coin-transfer and communication) rounds.

Keywords: Bitcoin, MPC, Fairness, Robustness

1 Introduction

Secure multiparty computation (MPC) enables a set of parties to evaluate the output of a known function $f(\cdot)$ on inputs they privately contribute to the protocol execution. The design of secure MPC protocols, initiated with the seminal works of Yao [Yao82] and Goldreich et al. [GMW87] has evolved to a major effort in computer security engineering. Beyond privacy, a secure MPC protocol is highly desirable to be *fair* (either all parties learn the output or none) and *robust* (the delivery of the output is guaranteed and the adversary cannot mount a "denial of service" against the protocol). Achieving fairness and robustness in a setting where there is an arbitrary number of corruptions, as desirable as it may appear, is prohibited by strong impossibility results stemming from the work of Cleve [Cle86] who showed that coin-flipping is infeasible in any setting where there is no honest majority among parties that execute the protocol. These impossibility results, combined with the importance of the properties that they prevent, strongly motivate the exploration of alternate – yet still realistic – models that would enable fair and robust MPC protocols.

With the advent of Bitcoin [Nak08] and other decentralized cryptocurrencies, the works of [ADMM14a, ADMM14b, BK14, KB14] showed a new direction for circumvention of the impossibility results regarding the fairness property: enforcing fairness could be achieved through imposing monetary penalties. In this setting a breach of fairness by the adversary is still possible but it results in the honest parties collecting a compensation in a way that is determined by the protocol execution. At the same time, in case fairness is not breached, it is guaranteed that no party loses any money (despite the fact that currency transfers may have taken place between the parties). The rationale here is that a suitable monetary penalty suffices in most practical scenarios to force the adversary to operate in the protocol fairly.

While the main idea of fairness with penalties sounds simple enough, its implementation proves to be quite challenging. The main reason is that the way a crypto-currency operates does not readily provide a trusted party that will collect money from all participants and then either return it or redistribute it according to the pre-agreed penalty structure. This is because crypto-currencies are decentralized and hence no single party is ever in control of a money transfer beyond the owner of a set of coins. The mechanism used in [ADMM14a,ADMM14b,BK14,KB14] to circumvent the above problem is the capability[1] of the Bitcoin network to issue transactions that are "time-locked", i.e., become valid only after a specific time and prior to that time may be superseded by other transactions that are posted in the public ledger. Superseded time-locked transactions become invalid and remain in the ledger without ever being redeemed.

While the above works are an important step for the design of MPC protocols with properties that circumvent the classical impossibility results, several critical open questions remain to be tackled; those we address herein are as follows.

Our Results. Our contribution is three-fold. First, we put forth a new formal model of secure MPC with compensation and we show how the introduction of suitable ledger and synchronization functionalities makes it possible to express completely such protocols using standard interactive Turing machines (ITM) circumventing the need for the use of extra features that are outside the standard model (in comparison, the only previous model [BK14] resorted to specialized ITM's that utilize resources outside the computational model[2]). Second, our model

[1] Note that this feature is currently not fully supported.

[2] An ITM with the special features of "wallet" and "safe" was introduced in [BK14] to express the ability of ITM's to store and transfer "coins." Such coins were treated as physical quantities that were moved between players but also locked in safes in a way that parties were then prevented to use them in certain ways (in other words such safes were not local but were affected from external events).

is equipped with a composition theorem that enables the design of protocols that compose safely with each other and within larger environments where other protocols with compensation take place; a composition theorem for this class of protocols was not known before. Third, we introduce the first robust MPC protocol with compensation, i.e., an MPC protocol where not only fairness is guaranteed (via compensation) but additionally the protocol is guaranteed to deliver output to the parties that get engaged and therefore the adversary is not even able to mount a denial of service attack without having to suffer a monetary penalty. In more details we have the following.

- We put forth a new model that utilizes two ideal functionalities and express the ledger of transactions and a clock in the sense of [KMTZ13] that is connected to the ledger and enables parties to synchronize their protocol interactions. Our ledger functionality enable us to abstract all the necessary features of the underlying cryptocurrency. Contrary to the only previous formalization approach [BK14, KB14], our modeling allows the entities that participate in an MPC execution to be regular interactive Turing machines (ITM) and there is no need to equip them with additional physical features such as "safes" and "locks." Furthermore the explicit inclusion of the clock functionality (which is only alluded to in [BK14, KB14]) reveal the exact dependencies between the ledger and the clock functionality that are necessary in order for MPC with compensation protocols to be properly described. We express our model within a general framework that we call Q-fairness and may be of independent interest as it can express meaningful relaxations of fairness in the presence of a global ideal functionality.

- We prove a composition theorem that establishes that protocols in our framework are secure in a universally composable fashion. Our composition proof treats the clock and ledger functionalities as global setups in the sense of [CDPW07, CJS14]. We emphasize that this is a critical design choice: the fact that the ledger is a global functionality ensures that any penalties that are incurred to the adversary that result to credits towards the honest parties will be globally recognized. This should be contrasted to an approach that utilizes regular ideal functionalities which may be only accessible within the scope of a single protocol instance and hence any penalty bookkeeping they account may vanish with the completion of the protocol. Providing a composition theorem for MPC protocols with compensation was left as an open question in [BK14].

- We finally present a new protocol for fair and robust secure MPC with compensation. The robustness property we prove guarantees that once the protocol passes an initial round of deposits, parties are guaranteed to obtain output or be compensated. This is in contrast to fair MPC with compensation [ADMM14a, ADMM14b, BK14, KB14] where the guarantee is that compensation takes place only in case the adversary obtains output while an honest party does not. To put it differently, it is feasible for the adversary to lead the protocol to a deadlock where no party receives output however the honest parties have wasted resources by introducing transactions in the ledger. We remark that it is in principle possible to upgrade the protocols of [ADMM14a, ADMM14b, BK14, KB14] to the robust MPC setting by having them perform an MPC with identifiable abort, cf. [GMW87, IOZ14], (in such protocol the party that causes the abort can be identified and excluded from future executions). However even using such protocol the resulting robust MPC with compensation will need in the worst case a linear number of deposit/communication rounds in the number of malicious parties. Contrary to that, our robust protocol can be instantiated so that it requires a constant number of deposit/communication rounds

independently of the number of parties that are running the protocol. Our construction uses time-locked transactions in a novel way to ensure that parties do progress in the MPC protocol or otherwise transactions are suitably revertible to a compensation for the remaining parties. The structure of our transactions is quite more complex than what can be presently supported by bitcoin; we provide a high level overview of how our protocol can be implemented via Ethereum[3] contracts.

Related work. Beyond the previous works [ADMM14a, ADMM14b, BK14, KB14] in fair MPC with compensation there is a number of other works that attempted to circumvent the impossibility results for fairness in the setting of dishonest majority by considering alternate models. Contrary to the approach based on cryptocurrencies these works give an advantage to the protocol designer with respect to the adversarial strategy for corruption. For instance, in [GKM+13] a rational adversary is proposed and the protocol designer is privy to the utility function of the adversary. In [ALZ13] a reputation system is used and the protocol designer has the availability of the reputation information of the parties that will be engaged in the protocol. Finally in [GGJ+15] a two tiered model is proposed where the protocol designer is capable of distinguishing two distinct sets of servers at the onset of the computation that differ in terms of their corruptibility.

Global setups were first put forth in [CDPW07] motivated by notion of deniability in cryptographic protocols. In our work we utilize global functionalities for universal composition (without the deniability aspect) as in [CJS14] where a similar approach was taken for the case of the use of the random oracle as a global setup functionality for MPC.

Fairness can also be considered from the resource perspective, cf. [BN00, Pin03, GMPY06], where it is guaranteed due to the investment of proportional resources between the parties running the protocol, and the optimistic perspective, cf. [ASW97, ASW98, CC00], where a trusted mediator can be invoked in the case of an abort. We finally note that without any additional assumptions, due to the impossibility results mentioned above, one can provide fairness only with certain high probability that will be affecting the complexity of the resulting protocol, see, e.g., [GK09] and references therein.

2 Model

In this section, we lay down a formal framework for designing composable fair protocols in the presence of globally available trusted resources. In the first two subsections, we introduce shared (in the sense of the GUC model [CDPW07]) functionalities $\bar{\mathcal{G}}_{\text{CLOCK}}$ and $\bar{\mathcal{G}}_{\text{LEDGER}}$ respectively to formulate the trust resources that provided by Bitcoin-like systems. Then in subsection 2.3, we put forth a new formal model of secure MPC with compensation: we introduce the notion of Q-fairness, and formulate it via a wrapper functionality; we then consider the realization of such wrapper functionality, and further provide a composition theorem. In the end of this section, a sampling functionality for producing correlated randomness is also described. This will be used as setup in our protocol design.

2.1 Global Clock Functionality and Synchronous Protocol Executions

We here first define a shared clock functionality $\bar{\mathcal{G}}_{\text{CLOCK}}$. This functionality can be viewed as an extension of the clock functionality that was defined by Katz et al. [KMTZ13]. The main intuition behind the clock functionality is that when all honest parties agree to move to the

[3] `http://www.ethereum.org`.

next "clock tick", then the clock functionality will increase its state, τ, by 1. We remark that there are differences between our formulation and that by Katz et al. [KMTZ13]: there, the clock is for a single protocol; however, here we intend to have the clock to be accessed *globally*. In addition, in [KMTZ13], the functionality state is binary while here, in our formulation, the state τ is a positive integer. The detailed description of the clock functionality can be found in Figure 1.

Functionality $\bar{\mathcal{G}}_{\text{CLOCK}}$

Shared functionality $\bar{\mathcal{G}}_{\text{CLOCK}}$ is globally available to all participants. The shared functionality is parameterized with variables τ and $\mathcal{P}$.

Initially, $\tau := 0$ and $\mathcal{P} := \emptyset$.

- Upon receiving (REGISTER, sid) from some party P, then set $\mathcal{P} := \mathcal{P} \cup \{P\}$ and initialize $d_P := 0$.
- Upon receiving (CLOCK-UPDATE, sid) from some party $P \in \mathcal{P}$, then set $d_P := 1$.

  At any moment, if $d_P = 1$ for all $P \in \mathcal{H}$, where $\mathcal{H}$ denotes the set of honest parties, then set $\tau := \tau + 1$, and reset $d_P := 0$ for all $P \in \mathcal{P}$.
- Upon receiving (CLOCK-READ, sid) from some participant, return (CLOCK-READ, sid, τ) to the requestor.

Figure 1: The clock functionality.

Next, we elaborate and explain how to use the global clock to design synchronous protocols. To capture synchronous protocol execution in the global $\bar{\mathcal{G}}_{\text{CLOCK}}$ model we use the compiler idea from [KMTZ13]. First, as is the case in real-life synchronous protocols we assume that the protocol participants have agreed on the starting time τ_0 of their protocol and also on the duration of each round. We abstract this knowledge by assuming the parties know a function $\texttt{Round2Time} : \mathbb{Z} \to \mathbb{Z}$ which maps protocol rounds to time (according to the global clock). In particular, $\texttt{Round2Time}(0)$ is the time τ_0 in which a common reference string may be included in the ledger. For $\rho \in \mathbb{Z}^+$, $\texttt{Round2Time}(\rho)$ is the time in which the ρth round of the protocol should be completed. That is, if at time $\texttt{Round2Time}(\rho) + 1$ a party P_i has not received P_j's ρ-round message, then it takes this message to be a default message $\perp$. To make sure that no party proceeds to round $\rho + 1$ of the protocol before all honest parties have completed round ρ, we require that any two protocol rounds are at least two clock-ticks apart (see [KMTZ13] for a discussion); formally, for all $\rho \geq 0$, it holds that $\texttt{Round2Time}(\rho + 1) \geq \texttt{Round2Time}(\rho) + 2$.

A synchronous protocol in the above setting proceeds as follows:

- Upon receiving any activation , party P_i first queries the clock (i.e., send $\bar{\mathcal{G}}_{\text{CLOCK}}$ the message (READ, sid)) to find out the current time. Denote by τ the time reported by $\bar{\mathcal{G}}_{\text{CLOCK}}$.
- P_i checks that $\tau \geq \texttt{Round2Time}(0)$, and halts if this is not the case. Otherwise, P_i checks whether it has completed its protocol instructions for rounds $0, \ldots, \rho_c$, where $\rho_c = \max\{\rho \text{ s.t. } \texttt{Round2Time}(\rho) \leq \tau\}$. If this is not the case, P_i executes its next pending instruction (for round ρ_c);[4] otherwise, P_i sends (CLOCK-UPDATE, sid) to the clock.[5]

[4] Note that in (G)UC the parties lose activation whenever the send or output a message; thus it might be the case that they need to be activated (receive messages) multiple times in any given round in order to complete their protocol.

[5] Recall that CLOCK-UPDATE signals to the clock that, according to the current view of the party the clock should proceed.

2.2 Global Ledger Functionality

Functionality $\bar{\mathcal{G}}_{\text{LEDGER}}$ provides the abstraction of a public ledger in Bitcoin-like systems (e.g., Bitcoin, Litecoin, Namecoin, Ethereum, etc). Intuitively, the public ledger could be accessed globally by protocol parties or other entities including the environment $\mathcal{Z}$. Protocol parties or the environment can generate transactions; and these valid transactions will be gathered by a set of ledger maintainers (e.g., miners in Bitcoin-like systems) in certain order as the state of the ledger. More concretely, whenever the ledger maintainers receive a vector of transactions $\vec{\mathtt{tx}}$, they first add the transactions in a buffer, assuming they are valid with respect to the existing transactions and the state of the ledger; thus, in this way a vector of transactions is formed in the buffer. After certain amount of time, denoted by $\mathtt{T}$, all transactions in the buffer will be "glued" into the ledger state in the form of a block. In Bitcoin, $\mathtt{T}$ is 10 minutes (approximately); thus in about every 10 minutes, a new block of transactions will be included into the ledger, and the ledger state will be updated correspondingly.

Functionality $\bar{\mathcal{G}}_{\text{LEDGER}}$

Shared functionality $\bar{\mathcal{G}}_{\text{LEDGER}}$ is globally available to all participants. The shared functionality is parameterized with a predicate $\mathsf{Validate}$, a constant $\mathtt{T}$, and variables state, buffer and $\mathsf{counter}$.

Initially, $\mathsf{state} := \emptyset$, $\mathsf{buffer} := \emptyset$, and $\mathsf{counter} := 0$.

- Upon receiving $(\text{SUBMIT}, \mathsf{sid}, \vec{\mathtt{tx}})$ from some participant, send $(\text{CLOCK-READ}, \mathsf{sid})$ to $\bar{\mathcal{G}}_{\text{CLOCK}}$ and receive $(\text{CLOCK-READ}, \mathsf{sid}, \tau)$ from $\bar{\mathcal{G}}_{\text{CLOCK}}$. If $\mathsf{Validate}(\mathsf{state}, (\mathsf{buffer}, \vec{\mathtt{tx}})) = 1$, then set $\mathsf{buffer} := \mathsf{buffer}||\vec{\mathtt{tx}}$. If $|\tau - \mathtt{T} \cdot \mathsf{counter}| > \mathtt{T}$, then set $\mathsf{state} := \mathsf{state}||\mathsf{Blockify}(\tau, \mathsf{buffer})$ and $\mathsf{buffer} := \emptyset$ and $\mathsf{counter} := \mathsf{counter} + 1$.
- Upon receiving $(\text{READ}, \mathsf{sid})$ from some participant, return $(\text{READ}, \mathsf{sid}, \mathsf{state})$ to the requestor.

Figure 2: The public ledger functionality.

To enable the ledger to be aware of time, the ledger maintainers are allowed to "read" the state of another publicly available functionality $\bar{\mathcal{G}}_{\text{CLOCK}}$ defined above.

We remark that all gathered transactions should be "valid" which is defined by a predicate $\mathsf{Validate}$. In different systems, predicate $\mathsf{Validate}$ will take different forms. For example, in the Bitcoin system, the predicate $\mathsf{Validate}$ should make sure that for each newly received transaction that transfers v coins from the original wallet address $\mathtt{address}_o$ to the destination wallet address $\mathtt{address}_d$, the original wallet address $\mathtt{address}_o$ should have v or more than v coins, and the transaction should be generated by the original wallet holder (as shown by the issuance of a digital signature). Furthermore, prior to each vector of transactions becoming block, the vector is passed through a function $\mathsf{Blockify}(\cdot)$ that homogenizes the sequence of transactions in the form of a block. Moreover, in some systems like Bitcoin, it may add a special transaction called a "coinbase" transaction that implements a reward mechanism for the ledger maintainers.

In Figure 2 we provide the details of the ledger functionality.

2.3 Q-fairness and Secure Computation with Fair Compensation

In this subsection, we provide a formal framework for secure computation with fair compensation. In the spirit of [GMPY06], our main tool is a wrapper functionality. Our wrapper

functionality is equipped with a predicate[6] $\mathsf{Q}_{\bar{\mathcal{G}}}$ which can be used to "monitor" the state of the global setup $\bar{\mathcal{G}}$. If certain "bad" event occurs (such as an abort), then the predicate $\mathsf{Q}_{\bar{\mathcal{G}}}$ will be triggered, and the functionality will halt. With foresight Q will ensure that parties that abort are compensated. Any implementation of the wrapped functionality should ensure that such a halt event is not triggered with non-negligible probability. We will call such an implementation a Q-fair implementation.

Our framework can be defined with respect to any global setup that upon receiving a READ symbol it returns its public state trans. Let $\bar{\mathcal{G}}$ be such a global ideal functionality and let $\mathsf{Q}_{\bar{\mathcal{G}}}$ a predicate with respect to such $\bar{\mathcal{G}}$. Let $\mathcal{F}$ be a secure function evaluation (SFE) functionality which is fair in the sense of [GMPY06]: it returns output in two different ways: (i) delayed delivery: $(\text{DELIVER}, \mathsf{sid}, m, P)$ signifying delayed output delivery[7] of m to party P, (ii) fair delivery: $(\text{FAIR-DELIVER}, \mathsf{sid}, (m, P_{i_1}), \ldots, (m, P_{i_k}), (m_{\mathcal{S}}, \mathcal{S}))$ that results in simultaneous[8] delivery of outputs $m_{i_1}, \ldots m_{i_k}$ to parties $P_{i_1}, \ldots, P_{i_k}$ and output $m_{\mathcal{S}}$ to $\mathcal{S}$.

The wrapper functionality $\mathcal{W}$ that will be used in the definition of Q-fair secure computation is given in Figure 3. The intuition is that the Q predicate will be applied on the public state of $\bar{\mathcal{G}}$ whenever the wrapper "breaks" the fair delivery of the inner functionality $\mathcal{F}$. We are now ready to define Q-fairness with respect to a global functionality.

Definition 2.1. *We say protocol π realizes functionality $\mathcal{F}$ with $\mathsf{Q}_{\bar{\mathcal{G}}}$-fairness with respect to global functionality $\bar{\mathcal{G}}$, provided the following statement is true. For all adversaries $\mathcal{A}$, there is a simulator $\mathcal{S}$ so that for all environments $\mathcal{Z}$ it holds:*

$$\text{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})}_{\mathcal{S},\mathcal{Z}}$$

The following lemma captures the intuition that computing the inner functionality is at least as strong as computing it with Q-fairness.

Lemma 2.2. *Let $\bar{\mathcal{G}}$ be a global ideal functionality. Let $\mathsf{Q}_{\bar{\mathcal{G}}}$ be any predicate with respect to $\bar{\mathcal{G}}$. Then for all adversaries $\mathcal{A}$, there is a simulator $\mathcal{S}$ so that for all environments $\mathcal{Z}$, it holds:*

$$\text{EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\sigma,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})}_{\sigma,\mathcal{S},\mathcal{Z}}$$

The above lemma can be easily proved, because the simulator $\mathcal{S}$ by returning $(\text{FAIR-DELIVER}, \mathsf{sid}, \mathsf{mid})$ to the wrapper functionality, can avoid the predicate $\mathsf{Q}_{\bar{\mathcal{G}}}$ be triggered.

More generally, the protocol σ realizes $\mathcal{H}$ with $\mathsf{Q}'_{\bar{\mathcal{G}}}$ fairness using a functionality $\mathcal{F}$ with fairness $\mathsf{Q}_{\bar{\mathcal{G}}}$ provided that for all adversaries $\mathcal{A}$, there is a simulator $\mathcal{S}$ so that for all environments $\mathcal{Z}$, it holds:

$$\text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{H},\mathsf{Q}'_{\bar{\mathcal{G}}})}_{\mathcal{S},\mathcal{Z}}$$

We note that, both protocol π and the functionality $(\mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}}), \bar{\mathcal{G}})$ are with respect to the global functionality $\bar{\mathcal{G}}$[9]. By following the very similar proof idea of proving UC/GUC composition [Can01, CDPW07], we can prove the following:

[6] Whenever it is clear from the context we may drop the subscript $\bar{\mathcal{G}}$.

[7] Delayed output delivery is a standard (G)UC mechanism where the adversary is allowed to schedule the output at a time of its choosing.

[8] Given that in the (G)UC framework no simultaneous message delivery is supported, the term "simultaneous" will refer to "fetch mode delivery" as defined in [KMTZ13].

[9] In GUC framework [CDPW07], this is also called, $\bar{\mathcal{G}}$-subroutine respecting.

Wrapper Functionality $\mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})$

Functionality $\mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})$ interacts with a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, the adversary $\mathcal{S}$ and the environment $\mathcal{Z}$, as well as shared functionality $\bar{\mathcal{G}}$. The functionality is parameterized with an n-party functionality $\mathcal{F}$ and a predicate $\mathsf{Q}_{\bar{\mathcal{G}}}$.

- *Submitting an input.* Upon receiving $(\text{LABEL}, \mathsf{sid}, x)$ from a party P, it forwards $(\text{LABEL}, \mathsf{sid}, x)$ to $\mathcal{F}$. In case $\mathcal{F}$ produces a message μ for $\mathcal{S}$ it forwards $((\text{LABEL}, \mathsf{sid}, P), \mu)$ to $\mathcal{S}$.
- *Associating a label to a party.* On input $(\text{LABEL}, \mathsf{sid}, P, L)$ from $\mathcal{S}$ it records $L_{\mathsf{sid},P} = L$ and returns $(\text{LABEL}, \mathsf{sid}, P, L)$ to P.
- *Generating delayed output.* On input a message from $\mathcal{F}$ marked $(\text{DELIVER}, \mathsf{sid}, m, P)$ it forwards m to party P via delayed output.
- *Registering fair output.* On input a message from $\mathcal{F}$ that is marked for fair delivery $(\text{FAIR-DELIVER}, \mathsf{sid}, \mathsf{mid}, (m_1, P_{i_1}), \ldots, (m_k, P_{i_k}), (m_{\mathcal{S}}, \mathcal{S}))$, it forwards $(\mathsf{mid}, P_{i_1}, \ldots, P_{i_k}, m_{\mathcal{S}})$ to $\mathcal{S}$.
- *Delayed fair output delivery.* Upon receiving $(\text{FAIR-DELIVER}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{S}$ then provided that a message $(\mathsf{mid}, \ldots)$ has been delivered to the adversary, all pairs (m_j, P_j)'s associated with mid are sent to P_j's simultaneously.
- $\mathsf{Q}_{\bar{\mathcal{G}}}$*-fair delivery.* Upon receiving $(\text{Q-DELIVER}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{S}$ then provided that a message $(\mathsf{mid}, \ldots)$ has been delivered to the adversary operate as follows. For each pair of the form (m, P) associated with mid where P is corrupted, the adversary $\mathcal{S}$ receives (m, P). For each party P that is corrupted the pair (m, P) in mid is marked as `delivered`. Every other pair in mid is marked `undelivered`. Subsequently perform the following.
  - On input a message $(\text{DELIVER}, \mathsf{sid}, \mathsf{mid}, P)$ from $\mathcal{S}$, provided that the record mid contains the pair (m, P) that is `undelivered` then send READ to $\bar{\mathcal{G}}$, denote the responce by trans and if $\neg\mathsf{Q}_{\bar{\mathcal{G}}}(\mathsf{sid}, L_{\mathsf{sid},P}, 1, \mathsf{trans})$ then halt. Else, the party P will receive m and the pair (m, P) in mid is marked `delivered`.
  - On input a message $(\text{ABORT}, \mathsf{sid}, \mathsf{mid}, P)$ from $\mathcal{S}$, provided that the record mid contains the pair (m, P) that is `undelivered` then send READ to $\bar{\mathcal{G}}$, denote the responce by trans and if $\neg\mathsf{Q}_{\bar{\mathcal{G}}}(\mathsf{sid}, L_{\mathsf{sid},P}, 0, \mathsf{trans})$ then halt. Else, the party P will receive $\perp$ and the pair (m, P) in mid is marked `aborted`.

Figure 3: The wrapper functionality.

Lemma 2.3. *Let $\mathsf{Q}_{\bar{\mathcal{G}}}$ be a predicate with respect to global functionality $\bar{\mathcal{G}}$. Let π be a protocol that realizes the functionality $\mathcal{F}$ with $\mathsf{Q}_{\bar{\mathcal{G}}}$-fairness. Let σ be a protocol in $(\mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}}), \bar{\mathcal{G}})$-hybrid world. Then for all adversaries $\mathcal{A}$, there is a simulator $\mathcal{S}$ so that for all environments $\mathcal{Z}$, it holds*

$$\text{EXEC}^{\bar{\mathcal{G}}}_{\sigma^\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})}_{\sigma,\mathcal{S},\mathcal{Z}}$$

Theorem 2.4. *Let $\mathsf{Q}_{\bar{\mathcal{G}}}$ and $\mathsf{Q}'_{\bar{\mathcal{G}}}$ be predicates with respect to global functionality $\bar{\mathcal{G}}$. Let π be a protocol that realizes the functionality $\mathcal{F}$ with $\mathsf{Q}_{\bar{\mathcal{G}}}$-fairness. Let σ be a protocol in $(\mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}}), \bar{\mathcal{G}})$-hybrid world that realizes the functionality $\mathcal{H}$ with $\mathsf{Q}'_{\bar{\mathcal{G}}}$-fairness. Then for all adversaries $\mathcal{A}$, there is a simulator $\mathcal{S}$ so that for all environments $\mathcal{Z}$ it holds:*

$$\text{EXEC}^{\bar{\mathcal{G}}}_{\sigma^\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{H},\mathsf{Q}'_{\bar{\mathcal{G}}})}_{\mathcal{S},\mathcal{Z}}$$

Please see appendix for the proof.

We are now ready to instantiate the notion of Q-fairness with a compensation mechanism.

Computation with Fair Compensation. For the case when $\bar{\mathcal{G}}$ implements the Bitcoin-like ledger and $\mathsf{Q}_{\bar{\mathcal{G}}}$ provides compensation of c bitcoin, where $c > 0$, in the case of an abort the predicate $\mathsf{Q}_{\bar{\mathcal{G}}}(\mathsf{sid}, L, bit, \mathsf{trans})$ operates as follows: it parses L as a pair of $(\mathtt{address}, \mathsf{sk})$ where $\mathtt{address}$ is a bitcoin address and sk is the corresponding secret-key. Then it parses trans as a bitcoin ledger that contains transactions. Transactions in trans can also be marked with metadata. The following hold :

- If $bit = 1$, then $\mathsf{Q}_{\bar{\mathcal{G}}}$ outputs true if and only if the balance of all transactions (both incoming and outgoing) that concern $\mathtt{address}$ in trans and carry the meta-data sid is greater equal to 0.
- If $bit = 0$, then $\mathsf{Q}_{\bar{\mathcal{G}}}$ outputs true if and only if the balance of all transactions (both incoming and outgoing) that concern $\mathtt{address}$ in trans and carry the meta-data sid is greater equal to c.

2.4 Correlated Randomness as a Sampling Functionality

Our protocols are in the *correlated randomness* model, i.e., they assume that the parties initially, before receiving their inputs, receive appropriately correlated random strings. In particular, the parties jointly hold a vector $\vec{R} = (R_1, \ldots, R_n) \in (\{0,1\}^*)^n$, where P_i holds R_i, drawn from a given efficiently samplable distribution $\mathcal{D}$. This is, as usual, captured by giving the parties initial access to an ideal functionality $\mathcal{F}_{\text{CORR}}^{\mathcal{D}}$, known as a *sampling functionality*, which, upon receiving a default input from any party, samples $\vec{R}$ from $\mathcal{D}$ and distributes it to the parties. Hence, a protocol in the correlated randomness model is formally an $\mathcal{F}_{\text{CORR}}^{\mathcal{D}}$-hybrid protocol. Formally, a sampling functionality $\mathcal{F}_{\text{CORR}}^{\mathcal{D}}$ is parameterized by an efficiently computable sampling distribution $\mathcal{D}$ and the (ID's of the parties in) the player set $\mathcal{P}$.

Functionality $\mathcal{F}_{\text{CORR}}^{\mathcal{D}}$

Functionality $\mathcal{F}_{\text{CORR}}^{\mathcal{D}}$ interacts with a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, the adversary $\mathcal{S}$ and the environment $\mathcal{Z}$. The functionality is parameterized with a distribution sampler $\mathcal{D}$.

- Upon receiving $(\text{REQUEST}, \mathsf{sid})$ from any party or the adversary, set $\vec{R} = (R_1, \ldots, R_n) \leftarrow \mathcal{D}$ and for each $P_i \in \mathcal{P}$ send $(\text{REQUEST}, \mathsf{sid}, R_i)$ to P_i (or to the adversary if P_i is corrupted).

Figure 4: The correlated randomness functionality.

3 Our fair protocol compiler

In this section we present our fair protocol compiler. Our compiler compiles a synchronous protocol $\pi_{\mathtt{sh}}$ which is secure (i.e., private) against a corrupted majority in the semi-honest correlated randomness model (e.g, an OT-hybrid protocol where the OT's have been pre-computed) into a protocol π which is secure with fair-compensation in the malicious correlated randomness model. The high-level idea is the following: We first compile $\pi_{\mathtt{sh}}$ into a protocol in the malicious correlated randomness model, which is executed over a broadcast channel and is secure with

publicly identifiable abort. (Roughly, this means that someone observing the protocol execution can decide, upon abort, which party is not executing its code.) This protocol is then transformed into a protocol with fair compensation as follows: Every party (after receiving his correlated randomness setup) posts to the ledger transactions that the other parties can claim only if they, later, post transactions that prove that they follow their protocol. Transactions that are not claimed this way are returned to the source address; thus, if some party does not post such a proof it will not be able to claim the corresponding transaction, and will therefore leave the honest parties with a positive balance as their transactions will be refunded. Observe that these are not standard Bitcoin transactions, but they have a special format which is described in the following.

Importantly, the protocol we describe is guaranteed to either produce output in as many (Bitcoin) rounds are the rounds of the original malicious protocol, or to compensate all honest parties. This *robustness* property is achieved by a novel technique which ensures that as soon as the honest parties make their initial transaction, the adversary has no way of preventing them from either computing their output or being compensated. Informally, our technique consists of splitting the parties into "islands" depending on the transactions they post (so that all honest parties are on the same island) and then allowing them either compute the function within their island, or if they abort to get compensated. (The adversary has the option of being included or not in the honest parties' island.)

3.1 MPC with Publicly Identifiable Abort

As a first step in our compiler we invoke the semi-honest to malicious with identifiable abort compiler of Ishai, Ostrovsky, and Zikas [IOZ14] (hereafter referred to as the *IOZ compiler*). This compiler takes a semi-honest protocol $\pi_{\mathtt{sh}}$ in the correlated randomness model and transforms it to a protocol in the malicious correlated randomness model (for an appropriate setup) which is secure with identifiable abort, i.e., when it aborts, every party learns the identity of a corrupted party. The compiler in [IOZ14] follows the so called GMW paradigm [GMW87], which in a nutshell has every party commit to its input and randomness for executing the semi-honest protocol $\pi_{\mathtt{sh}}$ and then has every party run $\pi_{\mathtt{sh}}$ over a broadcast channel, where in each round ρ every party broadcast his round ρ messages and proves in zero-knowledge that the broadcasted message is correct, i.e., that he knows input and randomness that are consistent with the initial commitments and the (public) view of the protocol so far. The main difference of the IOZ compiler and the GMW compiler is that the parties are not only committed to their randomness, but they are also committed to their entire setup string, i.e., their private component of the correlated randomness. More concretely, the resulting malicious protocol $\pi_{\mathtt{m}}$ (which is based on the compiler in [IOZ14]) has the following properties:

- Every party is committed to his setup, i.e., the part of the correlated randomness it holds. That is, every party P_i receives from the setup his randomness (which we refer to as P_i's *private component* of the setup) along with one-to-many commitments[10] on the private components of all parties. Wlog, we also assume that a common-reference string (CRS) and a public-key infrastructure (PKI) are included in every party's local randomess.
- The malicious protocol uses *only* the broadcast channel for communication.
- Given the correlated randomness setup, the malicious protocol is completely deterministic. This is achieved in [IOZ14] by ensuring that all the randomness used in the protocol, even the

[10]These are commitments that can be opened so that every party agrees on whether or not the opening succeeded.

one needed for the zero-knowledge proofs, is part of the correlated randomness distributed by the sampling functionality.[11]

- $\pi_{\mathtt{m}}$ starts off by having every party broadcast a one-time pad encryption of its input with its (committed) randomness and a NIZK that it knows the input and randomness corresponding to the broadcasted message.
- By convention, the next-message function of the malicious protocol is such that if in any round the transcript seen by a party is an aborting transcript, i.e., is not consistent with an accepting run of the semi-honest protocol, then the party outputs $\perp$ (see below for a detailed formulation).
- There is a (known) upper bound on the number $\rho_{\mathtt{m}}$ of rounds of $\pi_{\mathtt{m}}$.

We also stress that, given appropriate setup, the IOZ-compiler achieve information-theoretic security, and needs therefore to build information-theoretic commitments and zero-knowledge proofs. As in this work we are only after computational security, we modify the IOZ compiler so that we use (computationally) UC secure one-to-many commitments [CLOS02] and computationally UC secure non-interactive zero-knowledge proofs (NIZKs) instead if their i.t. instantiation suggested in [IOZ14]. Both the UC commitment and the NIZKs can be built in the CRS model. Moreover, the use of UC secure instantiations of zero-knowledge and commitments ensures that the resulting protocol will be (computationally) secure.

Using the setup within a subset of parties. A standard property of many protocols in the correlated-randomness model is that once the parties in $\mathcal{P}$ have received the setup, any subset $\mathcal{P}' \subset \mathcal{P}$ is able to use it to perform a computation of a $|\mathcal{P}'|$-party function amongst them while ignoring parties in $\mathcal{P} \setminus \mathcal{P}'$. More concretely, assume the parties in $\mathcal{P}$ have been handed a setup distributed allowing them to execute some protocol π for computing any $|\mathcal{P}|$-party function f; then for any $\mathcal{P}' \subseteq \mathcal{P}$, the parties in $\mathcal{P}$ can use their setup within a protocol $\pi|_{\mathcal{P}'}$ to compute any $|\mathcal{P}'|$-party function $f|_{|\mathcal{P}'|}$. This property which will prove very useful for obtaining computation with robustness or compensation, is also satisfied by the IOZ protocol, as the parties in $\mathcal{P}'$ can simply ignore the commitments (public setup component) corresponding to parties in $\mathcal{P} \setminus \mathcal{P}'$.

Making Identifiability Public. The general idea of our protocol is to have every party issue transactions by which he commits to transferring a certain amount of bitcoin per party for each protocol round. All these transactions are issued at the beginning of the protocol execution, but every party can claim the "committed" coins transferred to him associated to some protocol round ρ only under the following conditions: (1) the claim is posted in the time-interval corresponding to round ρ; (2) the party has claimed all his transferred bitcoins associated to the previous rounds; and (3) the party has posted a transaction which includes his valid message for round ρ.

In order to ensure that a party cannot claim his bitcoins unless he follows the protocol, the ledger (more concretely the validation predicate) should be able to check that the party is indeed posting a message corresponding to its next protocol message. In other words, in each round ρ, P_i's round-ρ message acts a witness for P_i claiming all the bitcoins transferred to him associated with this round ρ. To this direction we make the following modification to the protocol: Let $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ denote the n-party function we wish to compute, and let F^{+1} be the $(n+1)$-party function which takes input x_i from each P_i, $i \in [n]$, and no input from P_{n+1} and outputs y_i to each P_i and a special symbol (e.g., 0) to P_{n+1}. Clearly, if $\pi_{\mathtt{sh}}$ is a semi-honest n-party protocol for computing f over broadcast, then the $n+1$ protocol $\pi'_{\mathtt{sh}}$ in which every P_i

[11] As an example, the challenge for the zero-knowledge proofs is generated by the parties opening appropriate parts of their committed random strings.

with $i \in [n]$ executes $\pi_{\mathtt{sh}}$ and P_{n+1} simply listens to the broadcast channel is a secure protocol for F^{+1}.

Now if $\pi'_{\mathtt{m}}$ denotes the $(n+1)$-party malicious protocol which results by applying the above modified IOZ compiler on the $(n+1)$-party semi-honest protocol for computing the function F^{+1}, then, by construction this protocol computes function F^{+1} with identifiable abort and has the following properties:

- Party P_{n+1} does not make any use of his private randomness whatsoever; this is true because he broadcasts no messages and simply verifies the broadcasted NIZKs.
- If some party P_i, $i \in [n]$ deviates from running $\pi_{\mathtt{sh}}$ with the correlated (committed) randomness as distributed from the sampling functionality, then this is detected by all parties, including P_{n+1} (and protocol $\pi'_{\mathtt{m}}$ aborts with this party). This follows by the soundness of the NIZK which P_i needs to provide proving that he is executing $\pi_{\mathtt{sh}}$ in every round.

Due to P_{n+1}'s role as an observer who gets to decide if the protocol is successful (P_{n+1} outputs 0) or some party deviated (P_{n+1} observes that the corresponding NIZK verification failed) in the following we will refer to P_{n+1} in the above protocol as the *judge*. The code of the judge can be used by anyone who has the public setup and wants to follow the protocol execution and decide whether it should abort or not given the parties' messages. Looking ahead, the judge's code in the protocol will be used by the ledger to decide wether or not a transaction that claims some committed coins is valid.

3.2 Special Transactions supported by our Ledger

In this section we specify the Validate and the Blockify predicates that are used for achieving our protocol's properties. More specifically, our protocol uses the following type of transactions which transfer v coins from wallet $\mathtt{address}_i$ to wallet $\mathtt{address}_j$ conditioned on a statement Σ:

$$\mathbb{B}_{v,\mathtt{address}_i,\mathtt{address}_j,\Sigma,\mathtt{aux},\sigma_i,\tau} \tag{1}$$

where σ_i is a signature of the transaction, which can be verified under wallet $\mathtt{address}_i$; τ is the time-stamp, i.e., the current value of the clock when this transaction is posted by the ledger—note that this timestamp is added by the ledger and not by the users,—$\mathtt{aux} \in \{0,1\}^*$ is an arbitrary string[12]; and the statement Σ consists of three arguments, i.e., $\Sigma = (\mathtt{arg}_1, \mathtt{arg}_2, \mathtt{arg}_3)$, which are processed by the validate predicate in order to decide if the transaction is valid (i.e., if it will be included in the ledger's next block). The validation happens by processing the arguments of Σ in a sequential order, where if while processing of some argument the validation rejects, algorithm Validate stops processing at that point and this transaction is dropped. The arguments are defined/processed as follows:

Time-Restrictions: The first argument is a pair $\mathtt{arg}_1 = (\tau_-, \tau_+) \in \mathbb{Z} \times (\mathbb{Z}^+ \cup \{\infty\})$ of points in time. If $\tau_- > \tau_+$ then the transaction is invalid (i.e., it will be dropped by the ledger). Otherwise, before time τ_- the coins in the transaction "remain" blocked, i.e., no party can spend them; from time τ_- until time τ_+, the money can be spent by the owner of wallet $\mathtt{address}_j$ provided that the spending statement satisfies also the rest of the requirements/arguments in statement Σ (listed below). After time τ_+ the money can be spent by the owner of wallet $\mathtt{address}_i$ without any additional restrictions (i.e., the rest of the arguments in Σ are not parsed and the transaction is dealt with as a standard Bitcoin transaction with receiver $\mathtt{address}_i$). As a special case, if $\tau_+ = \infty$ then the transferred coins

[12]This string will be included to the Ledger's state as soon as the transaction is posted and can be, therefore, referred to by other spending statements.

can be spent from $\texttt{address}_j$ at any point (provided the spending statement is satisfied); we say then that the transaction is *time-unrestricted*,[13] otherwise we say that the transaction is *time restricted*.

Spending Link: Provided that the processing of the first argument, as above, was not rejecting, the validate predicate proceeds to the second argument, which is a unique anchor, $\texttt{arg}_2 = \alpha \in \{0,1\}^*$. Informally, this serves as a unique identifier for linked transactions; that is, when $\alpha \neq \perp$, then the Validate algorithm of the ledger looks in the ledger's state and buffer to confirm that the balance of transactions to/from the wallet address $\texttt{address}_i$ with this anchor $\texttt{arg}_2$ is at least $v' > v$ coins. That is, the sum of bitcoins in existing valid transaction (in the state of in the buffer) with receiver address $\texttt{address}_i$ and anchor $\texttt{arg}_2$ minus the sum of bitcoins in existing valid transaction (in the state or in the buffer) with sender address $\texttt{address}_i$ and anchor $\texttt{arg}_2$ is greater than v. If this is not the case then the transaction is rendered invalid; otherwise the validation of this argument succeeds and the algorithm proceeds to the next argument. To spice up the terminology, we call a transaction which has a non-$\perp$ anchor argument a *linked transaction*.

State-Dependent Condition: The last argument to be validated is $\texttt{arg}_3$, which is a relation $\mathcal{R} : \mathcal{S} \times \mathcal{B} \times \mathcal{T} \to \{0,1\}$, where $\mathcal{S}$, $\mathcal{B}$, and $\mathcal{T}$ are the domains of possible ledger-states, ledger-buffers, and transactions, respectively (in a given encoding). This argument defines which type of transactions can spend the coins transferred in the current transaction. That is, in order to spend the coins, the receiver needs to submit a transaction $\texttt{tx} \in \mathcal{T}$ such that $\mathcal{R}(\mathsf{state}, \mathsf{buffer}, \texttt{tx}) = 1$.

We point out that as with standard Bitcoin transactions, the validation predicate will always also check validity of the signature σ_i with respect to the wallet $\texttt{address}_i$. Moreover, the standard Bitcoin transactions can trivially be casted as transactions of the above type by setting $\alpha = \perp$ and $\Sigma = ((0,\infty), \perp, \mathcal{R}_\emptyset)$, where $\mathcal{R}_\emptyset$ denotes the relation which is always true.

The Blockify algorithm groups transactions in the current buffer and adds a timestamp.[14]

3.3 The protocol

Let $\pi'_{\mathtt{m}}$ denote the above described malicious protocol. Consistently to our terminology, let $\texttt{Round2Time}(1)$ denote the time in which the parties have agreed to start the protocol execution. Wlog, assume that $\texttt{Round2Time}(1) > 2\mathtt{T}+2$. [15] Furthermore, for simplicity, we assume that each party P_i receives its input x_i with its first activation from the environment at time $\texttt{Round2Time}(1)$ (if some honest party does not have an input by that time it will execute the protocol with a default input, e.g., 0).

Informally, the protocol proceeds as follows: In a pre-processing step, before the parties receive input, the parties invoke the sampling functionality for $\pi'_{\mathtt{m}}$ to receive their correlated randomness.[16] The public component of this randomness includes their protocol-associated wallet $\texttt{address}_i$ which they output (to the environment). The environment is then expected to submit $\rho_{\mathtt{m}}$ special (as above) coin-transfers for each pair of parties $P_i \in \mathcal{P}$ and $p_j \in \mathcal{P}$; the source wallet-address for each such transaction is P_i's, i.e., $\texttt{address}_i$ and the target wallet-address

[13] This is the case with standard Bitcoin transactions.

[14] This is the absratction needed for our protocol; in actual crypto-currencies Blockify would have a more complicated functionality (cf. Section 2.2).

[15] That is we assume that at least four Bitcoin rounds (recall that $\mathtt{T}$ denotes the duration of a Bitcoin round) plus four extra clock-ticks have passed from the begging of the experiment.

[16] In an actual application, the parties will use an unfair protocol for computing the correlated randomness. As this protocol has no inputs, an abort will not be unfair (i.e., the simulator can always simulate the view of the adversary in an aborting execution.)

for is P_j's, i.e., $\texttt{address}_j$, and the corresponding anchors are as follows: $\alpha_{i,j,\rho} = (\mathsf{pid}, i, j, \rho)$, for $(i, j, \rho) \in [n]^2 \times [\rho_{\mathtt{m}}]$, where[17] pid is the GUC protocol ID for $\pi'_{\mathtt{m}}$. Since by assumption, $\texttt{Round2Time}(1) > 2\mathtt{T}+2$, the environment has sufficient time to submit these transaction so that by the time the protocol strarts they have been posted on the ledger.

At time $\texttt{Round2Time}(1) - \mathtt{T}$ the parties receive their inputs and initiate the protocol execution by first checking that sufficient funds are allocated to their wallets linked to the protocol executions by appropriate anchors, as above. If some party does not have sufficient funds then it broadcast an aborting message and all parties abort. (Note that this is a fair abort and no party has spent any time into making transactions.) Otherwise, parties make the special transactions that commit them (see below) into executing the protocol, and then proceed into claiming them one-by-one by executing their protocol in a round-by-round fashion.

Note that the protocol is executed in Bitcoin rounds so that the parties have enough time to claim their transaction. In fact, every protocol round is stretched to a Bitcoin rounds, i.e., $\texttt{Round2Time}(i+1) - \texttt{Round2Time}(i) \geq \mathtt{T}$, which will guarantee that any transaction submitted for round ρ, $\rho = 1, \ldots, \rho_{\mathtt{m}} - 1$, of the protocol, has been posted on the ledger by the beginning of round $\rho + 1$. By using a constant round malicious protocol $\pi'_{\mathtt{m}}$ (e.g., the modified compiled protocol from [IOZ14] instantiated with a constant round semi-honest protocol) we can ensure that our protocol will terminate in a constant number of Bitcoin rounds and every honest party will either receive its input, or will have a positive balance in its wallet. As already mentioned, achieving such a robustness property with known protocols would require a linear (in the number of corrupted parties) number of rounds which for large player sets is unrealistic.

Remark 3.1 (On availability of funds)**.** *Unlike existing works, we choose to explicitly treat the issue of how funds become available to the protocol by making the off-line transfers external to the protocol itself (i.e., the environment takes care of them). However, the fact that the environment is in charge of "pouring" money into the wallets that are used for the protocol does not exclude that the parties might be actually the ones having done so. Indeed, the environment's goal is to capture everything that is done on the side of, before, or after the protocol, including other protocols that the parties might have participated in. By giving the environment enough time to ensure these transactions are posted we ensure that some honest party not having enough funds corresponds to an environment that makes the computation abort (in a fair way and only in the pre-processing phase, before the parties have invested time into posting protocol transactions).*

Another issue to be discussed, is how we arrange that the balance of honest parties is positive in case of an abort. This is achieved as follows by exploiting the power of our special transactions: we require that the auxiliary string of a transaction of a party P_j which claims a committed transaction for some round ρ includes his ρ-round protocol message. We then have the relation of this transaction be such that it evaluates to 1 if only if this is indeed P_j's next message. Thus, effectively the validate predicate implements the judge and can, therefore, decide if some party aborted: if some party broadcasts a message that would make the judge abort, then the validate predicate drops the corresponding transaction and all claims for committed transactions corresponding to future rounds, thus, all other parties are allowed to reclaim their committed bitcoins starting from the next round.

The last question is: how is the ledger able to know which parties should participate in the protocol? Here is the problem: The adversary might post in the first round (as part of the committing transaction for the first round) a fake, maliciously generated setup. Since the ledger is not part of the correlated randomness sampling, it would be impossible to decide which is

[17] Recall that we assume $|\mathcal{P}| = n$.

the good setup. We solve this issue by a the following technique that is inspired by [BCL$^+$05]: The ledger[18] groups together parties that post the same setup; these parties form "islands", i.e, subsets of $\mathcal{P}$. For each such subset $\mathcal{P}' \subseteq \mathcal{P} \cup \{P_{n+1}\}$ which includes the judge P_{n+1}, the ledger acts as if the parties in $\mathcal{P}'$ are executing the protocol $\pi'_{\mathtt{m}}|_{\mathcal{P}'}$ (which, recall, is the restriction of $\pi'_{\mathtt{m}}$ to the parties in $\mathcal{P}'$) for computing the $|\mathcal{P}'|$-party function $F^{+1}|_{\mathcal{P}'}(\vec{x})$ defined as follows: let the function to be computed be $f(\vec{x})$, where $\vec{x} = (x_1, \ldots, x_n)$, and F^{+1} be as above, then $F^{+1}|_{\mathcal{P}'}(\vec{x}) = F^{+1}(\vec{x}_{\mathcal{P}'})$ where $\vec{x}_{\mathcal{P}'} = (x'_1, \ldots, x'_n)$ with $x'_i = x_i$ for $P_i \in \mathcal{P}'$ and x'_i being a default value for every $P_i \notin \mathcal{P}'$. This solves the problem as all honest parties will be in the same island $\mathcal{P}' \subset \mathcal{P}$ (as they will all post the same value for public randomness); thus if the adversary chooses not to post this value on behalf of some corrupted party, he is effectively setting this party's input to a default value, a strategy which is easily simulatable. (Of course, the above solution will allow the adversary to also have "islands" of only corrupted parties that might execute the protocol, but this is also a fully simulatable strategy and has not effect on fair-compensation whatsoever—corrupted parties are not required to have a positive balance upon abort).

The final protocol $\pi^{\mathbb{B}}_{\mathtt{m}}$ is detailed in the following. The protocol ID is sid. The function to be computed is $f(x_1, \ldots, x_n)$. The protocol participants are $\mathcal{P} = \{P_1, \ldots, P_n\}$. We assume that all parties have registered with the clock functionality in advance and are therefore synchronized once the following steps start.

- **Setup Generation**
  Time $\tau_{-2} = \mathtt{Round2Time}(1) - 2\mathtt{T} - 2$*:*

  The parties invoke the sampling functionality, i.e., every party $P_i \in \mathcal{P}$ starts off by sending the sampling functionality a message (REQUEST); the sampling functionality returns $(R_i^{\mathtt{priv}}, R^{\mathtt{pub}})$ to P_i where $R_i^{\mathtt{priv}}$ is P_i's private component (including all random coins he needs to run the protocol, along his signing key sk_i) of the setup and $R^{\mathtt{pub}}$ is the public component (the same for every party P_j) which includes the vector of UC commitments $(\mathsf{Com}_1, \ldots, \mathsf{Com}_n)$, where for $j \in [n]$, Com_j is a commitment to $R_j^{\mathtt{priv}}$, along with a vector of public (verification) keys $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ corresponding to the signing keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and a common reference string CRS. Every party outputs its own public key, as its wallet address for the protocol, i.e., $\mathtt{address}_i = \mathsf{vk}_i$.
- **Check Availability of Funds**
  Time $\tau_{-1} = \mathtt{Round2Time}(1) - \mathtt{T} - 2$:

  Every party $P_i \in \mathcal{P}$ does the following: P_i reads the current state from the ledger. If the state does not include for each $(i, j, \rho) \in [n]^2 \times [\rho_{\mathtt{m}}]$ a transaction $\mathbb{B}_{c,\mathtt{address},\mathtt{address}_i,\Sigma^0_{i,j,\rho},\mathtt{aux}^0_{i,j,\rho},\sigma,\tau}$, where $\Sigma^0_{i,j,\rho} = ((0,\infty),(\mathsf{sid}, i, j, \rho), \mathcal{R}_\emptyset)$ then P_i broadcasts $\perp$ and every party aborts the protocol execution.
- **Input and Committing Transactions**
  Time $\tau_0 = \mathtt{Round2Time}(1) - \mathtt{T}$:

  Every party P_i receives its input x_i ($x_i = 0$ if no input is received in the first activation of P_i for time $\mathtt{Round2Time}(1) - \mathtt{T}$) and submits to the ledger the following "commitment" transactions:[19]

  1. For each $P_j \in \mathcal{P}$: $\mathbb{B}_{c,\mathtt{address}_i,\mathtt{address}_j,\Sigma_{i,j,1},\mathtt{aux}_{i,j,1},\sigma,\tau}$, where $\mathtt{aux}_{i,j,1} = R^{\mathtt{pub}}$ and $\Sigma_{i,j,1} = (\mathtt{arg}_1{}^{i,j,1}, \mathtt{arg}_2{}^{i,j,1}, \mathtt{arg}_3{}^{i,j,1})$ with

[18] Throughout the following description, we say that the ledger does some check to refer to the process of checking a corresponding relation, as part of validating a special transaction.

[19] Recall that, by definition of the clock, every party has as much time as it needs to complete all the steps below before the clock advances time.

- $\mathtt{arg_1}^{i,j,1} = (\mathtt{Round2Time}(1), \mathtt{Round2Time}(1)+1)$
- $\mathtt{arg_2}^{i,j,1} = (\mathsf{sid}, i, j, 1)$
- $\mathtt{arg_3}^{i,j,1} = \mathcal{R}_{i,j,1}$ defined as follows: Let $\mathcal{P}' = \mathcal{P} \cup \{P_{n+1}\}$, where P_{n+1} denotes the judge, be the player set implicit in $R^{\mathtt{pub}}$, [20] and let $\mathcal{P}'_i \subseteq \mathcal{P}'$ denote the set of parties (wallets), such that in the first block posted after time $\mathtt{Round2Time}(1) - \mathtt{T}$ all parties $P_k \in \mathcal{P}'_i$ had exactly one transaction for every $P_j \in \mathcal{P}$ with $\mathtt{arg_1}^{k,j,1} = (\mathtt{Round2Time}(1), \mathtt{Round2Time}(1)+1)$, $\mathtt{arg_2}^{k,j,1} = (\mathsf{sid}, k, j, 1)$, and $\mathtt{aux}^1_{k,j,1} = R^{\mathtt{pub}}$. Furthermore, let $\mathcal{P}''_i = \mathcal{P}'_i \cup \{P_{n+1}\}$, and let $\pi'_{\mathtt{m}}|_{\mathcal{P}''_i}$ be the protocol with public identifiability for computing $F^{+1}|_{\mathcal{P}'_i}$, described above and denote by $R^{\mathtt{pub}}|_{\mathcal{P}''_i}$ the restriction of the public setup to the parties in $\mathcal{P}''_i$. Then $\mathcal{R}_{i,j,1}(\mathsf{state}, \mathsf{buffer}, \mathtt{tx}) = 1$ if and only if the protocol of the judge with public setup $R^{\mathtt{pub}}|_{\mathcal{P}''_i}$ accepts the auxiliary string $\mathtt{aux_{tx}}$ in $\mathtt{tx}$ as P_i's next message in $\pi'_{\mathtt{m}}|_{\mathcal{P}''_i}$ (and does not abort).

2. For each round $\rho = 2, \ldots, \rho_{\mathtt{m}}$ and each $P_j \in \mathcal{P}$: $\mathbb{B}_{c,\mathtt{address}_i,\mathtt{address}_j,\Sigma_{i,j,\rho},\mathtt{aux}^1_{i,j,\rho},\sigma,\tau}$, where $\mathtt{aux}^1_{i,j,\rho} = R^{\mathtt{pub}}$ and $\Sigma_{i,j,\rho} = (\mathtt{arg_1}, \mathtt{arg_2}, \mathtt{arg_3})$ with
   - $\mathtt{arg_1} = (\mathtt{Round2Time}(\rho), \mathtt{Round2Time}(\rho+1)+1)$
   - $\mathtt{arg_2} = (\mathsf{sid}, i, j, \rho)$.
   - $\mathtt{arg_3} = \mathcal{R}_{i,j,\rho}$ defined as follows: Let $\mathcal{P}'_i, \mathcal{P}''_i, \pi'_{\mathtt{m}}|_{\mathcal{P}''_i}$ be defined as above (and denote $\mathcal{P}'_i = \{P_{i_1}, \ldots, P_{i_m}\}$. Then $\mathcal{R}_{i,j,\rho}(\mathsf{state}, \mathsf{buffer}, \mathtt{tx}) = 1$ if and only if, for each $r = 1, \ldots, \rho-1$ and each party $P_{i_k} \in \mathcal{P}'_i$, the state state includes transactions in which the auxiliary input is $\mathtt{aux}_{i_k,r}$ such that the protocol of the judge with public setup $R^{\mathtt{pub}}|_{\mathcal{P}''_i}$, and transcript $(\mathtt{aux}_{i_1,1}, \ldots, \mathtt{aux}_{i_m,1}), \ldots, (\mathtt{aux}_{i_1,\rho-1}, \ldots, \mathtt{aux}_{i_m,\rho-1})$, accepts the auxiliary string $\mathtt{aux_{tx}}$ in $\mathtt{tx}$ as P_i's next (ρ-round) message in $\pi'_{\mathtt{m}}|_{\mathcal{P}''_i}$ (and does not abort).

- **Claiming Committed Transactions/Executing the Protocol**
  Time $\tau \geq \mathtt{Round2Time}(1)$:

  For each $\rho = 1, \ldots, \rho_{\mathtt{m}}+1$, every P_i does the following at time $\mathtt{Round2Time}(\rho)$,:

  1. If $\tau = \mathtt{Round2Time}(\rho_{\mathtt{m}}+1)$ then go to Step 4; otherwise do the following:
  2. Read the ledger's state, and compute $\mathcal{P}'_i, \mathcal{P}''_i, \pi'_{\mathtt{m}}|_{\mathcal{P}''_i}$ as above.
  3. If the state state is not aborting, i.e, it includes for each $r = 1, \ldots, \rho-1$ and each party $P_{i_k} \in \mathcal{P}'_i$ in which the auxiliary input is $\mathtt{aux}_{i_k,r}$ such that P_i executing $\pi'_{\mathtt{m}}|_{\mathcal{P}''_i}$ with public setup $R^{\mathtt{pub}}|_{\mathcal{P}''_i}$, private setup $R^{\mathtt{priv}}_i$, and transcript $(\mathtt{aux}_{i_1,1}, \ldots, \mathtt{aux}_{i_m,1}), \ldots, (\mathtt{aux}_{i_1,\rho-1}, \ldots, \mathtt{aux}_{i_m,\rho-1})$ for the first $r-1$ rounds does not abort, then compute P_i's message for round ρ, denoted as m_ρ, and submit to the ledger for each $P_k \in \mathcal{P}''_i$ a transaction $\mathbb{B}_{c,\mathtt{address}_k,\mathtt{address}_i,\Sigma'_{k,i,\rho},\mathtt{aux}^\rho_{k,i,\rho},\sigma,\tau}$, where $\mathtt{aux}^\rho_{k,i,1} = m_\rho$ and $\Sigma'_{k,i,\rho} = (\mathtt{arg_1}, \mathtt{arg_2}, \mathtt{arg_3})$ with
     - $\mathtt{arg_1} = (0, \infty)$
     - $\mathtt{arg_2} = (\mathsf{sid}, k, i, \rho)$
     - $\mathtt{arg_3} = \mathcal{R}_\emptyset$.
  4. Otherwise, i.e., if the state state is aborting, then submit to the ledger for each round $r = 1, \ldots, \rho-1$, and each $P_k \in \mathcal{P}$ a transaction by which the committed trans-

[20] Recall that $R^{\mathtt{pub}}$ includes commitments to all parties' private randomness (including the judge's P_d) used for running the protocol, which is an implicit representation of the player set.

action towards P_k corresponding to round r is claimed back to $\mathtt{address}_i$, [21] i.e., $\mathbb{B}_{c,\mathtt{address}_k,\mathtt{address}_i,\Sigma,\mathtt{aux},\sigma,\tau}$, where $\mathtt{aux} = \perp$ and $\Sigma = (\mathtt{arg}_1, \mathtt{arg}_2, \mathtt{arg}_3)$ with

- $\mathtt{arg}_1 = (0, \infty)$
- $\mathtt{arg}_2 = (\mathsf{sid}, i, k, r)$
- $\mathtt{arg}_3 = \mathcal{R}_\emptyset$.

This completes the description of the protocol. The protocol terminates in $O(\rho_{\mathtt{m}})$ (Bitcoin) rounds; thus by using a constant-round protocol $\pi_{\mathtt{m}}$ [IOZ14], we obtain a protocol with constantly many Bitcoin rounds. Furthermore, as soon as an honest party posts a protocol-related transaction, he is guaranteed to either receive his output or have a positive balance (of at least c coins) after $O(\rho_{\mathtt{m}})$ Bitcoin rounds. The following theorem states the achieved security.

Theorem 3.2. *Let $\bar{\mathcal{G}} = (\bar{\mathcal{G}}_{\text{LEDGER}}, \bar{\mathcal{G}}_{\text{CLOCK}})$. The above protocol in the $(\bar{\mathcal{G}}, \mathcal{F}_{\text{CORR}}^{\mathcal{D}})$-hybrid world realizes $\mathcal{F}$ with fair compensation.*

Proof (sketch). We first prove that the above protocol is simulatable, by sketching the corresponding simulator $\mathcal{S}$. If the protocol aborts already before the parties make their transactions, then the simulator can trivially simulate such an abort, as he needs to just receive the state of the ledger and see if all wallets corresponding to honest parties have sufficient funds to play the protocol. In the following we show that the rest of the protocol (including the ledger's contents) can be simulated so that if there is an abort, honest parties' wallets have a positive balance as required by Q fairness. First we observe that the simulator $\mathcal{S}$ can easily decide the islands in which the parties are split, as he internally simulates the sampling functionality. Any island other than the one of honest parties (all honest parties will be in the same island because they will post transactions including the same public setup-component) is trivially simulatable as it only consists of adversarial parties and no guarantee is given about their wallets by Q-fairness. Therefore, it suffices to provide a simulator for the honest parties' island. To this direction, the simulator uses the simulator $\mathcal{S}_{\pi'_{\mathtt{m}}}$ which is guaranteed to exist from the security of $\pi'_{\mathtt{m}}$ to decide which messages to embed in the transactions of honest parties (the messages corresponding to corrupted parties are provided by the adversary). If $\mathcal{S}_{\pi'_{\mathtt{m}}}$ would abort, then $\mathcal{S}$ interacts the ideal functionality to abort and continues by claiming back all the committed transactions to the honest parties' wallets, as the protocol would. The soundness of the simulation of $\mathcal{S}_{\pi'_{\mathtt{m}}}$ ensure that the output of the parties and the contents of the ledger in the real and the ideal world are indistinguishable.

To complete the proof, we argue that when the protocol aborts, then honest parties have a positive balance of at least c coins as required by predicate Q. This is argued as follows: The parties that are not in the honest parties' islands cannot claim any transaction that honest parties make towards them as the ledger will see they as not in the island and reject them. Thus by the last round every honest party will have re-claimed all transactions towards parties not in his island. As fas as parties in the honest island are concerned, if no abort occurs then every party will have posted all his transactions, and therefore the balance will be 0. Otherwise, assume that the protocol aborts because some (corrupted) P_i broadcasts an inconsistent message in some round ρ. By inspection of the protocol one can verify that honest parties will be able to claim all transaction-commitments done to them up to round ρ (as they honestly execute their protocol) plus all committed transactions that they made for rounds $\rho + 1 \ldots, \rho_{\mathtt{m}}$. Additionally, because P_i broadcasts an inconsistent message in rounds ρ, he will be unable to claim transactions of

[21] For simplicity in case of an abort we have the parties claim-back all committed transactions up to that round; the second time these claims are made, they will be of course rejected.

honest parties done for round ρ; these bitcoins will be reclaimed by the honest parties, thus giving their wallets a positive balance of at least c. □

4 Using Ethereum contracts

In this section we comment on the feasibility of implementing our construction using Ethereum contracts. Ethereum is a type of virtual machine that operates over a blockchain protocol and enables the execution of complex transactions, [Woo14]. Transactions in Ethereum contain the recipient of the message, a signature identifying the sender, the amount of *ether* and the data to send, as well as two values called startgas and gasprice. These two values signify that in order for transactions to be processed "gas" needs to be spent that will be collected by the miner running the transaction. Gas can be funded with ether.

Transaction recipients are regular accounts as well as "smart" contracts. A contract is a special account that has its own code that is executed whenever it receives a transaction or a message. Contracts are stateful in the sense that they can maintain data in local (virtual) memory that has 2^{256} entries. A contract when executed can change its local state as well as generate new transactions.

The contract is executed by the miner that processes an incoming transaction for the contract. The decision to execute the contract depends also on the investment made by the transaction that is incoming; contract code may be expensive to run and thus a miner may refuse to execute the code of the contract if it is not sufficiently funded.

When a contract is executed by a miner, the code of the contract has access to various contextual information such as the current block timestamp, the current block number and so on. Using the current block timestamp, in particular, the contract is able to make time-sensitive decisions. For instance, in this way, a party may generate a contract that conditionally transfers some funds to someone that provides a specific type of data in a transaction. The funds may be locked in the contract while after a certain time the funds in the contract may be withdrawn back by the entity that initiated the contract.

Unfortunately Ethereum contracts are not able to inspect the blockchain when they are executed. It follows that they cannot directly implement the type of conditional transactions that our protocol requires as single contracts.

Nevertheless, it is feasible to obtain an implementation of our protocol within Ethereum as follows. First recall the type of transactions required by our protocol.

$$(\mathbb{B}_{v,\mathtt{address}_i,\mathtt{address}_j,\Sigma=(\tau_-,\tau_+),\alpha,\mathcal{R}),\mathtt{aux},\sigma)}$$

Instead of handling those as transactions, the parties initiate a contract for each protocol instance that accepts messages that encode the above transactions. In more detail, in order to implement the above transaction, an Ethereum transaction is generated from the sender directed to the account of the contract that includes v ether and passes as values tx.data$[i]$, $i = 0, 1, 2, \ldots$ the transaction elements $v, \mathtt{address}_i, \mathtt{address}_j, \tau_-, \tau_+, \alpha, \mathcal{R}, \mathtt{aux}, \sigma$.

The contract when executed with such transaction it uses its contract.storage[.] local state to store the data of the transaction. The contract is credited itself the value v that is put on hold by the transaction.

When a transaction is stored, the contract can accept withdrawing transactions as specified in our protocol description. Given any such withdrawing transaction the corresponding transaction via the anchor α will be recovered and it will be marked as spent if all the accompanying information is acceptable. Note that the contract has access to the current time via the block.timestamp and thus can make decisions with respect to the time parameters τ_- and τ_+.

Note that in order for a smart contract to process a transaction some additional ether is needed to be spent for using the Ethereum virtual machine (such ether is transformed to gas and is spent when the contract is executed by the miners). In our treatment such costs are considered negligible (while the amount of minimum compensation c is not negligible).

References

[ADMM14a] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via the bitcoin deposits. In *1st Workshop on Bitcoin Research 2014 (in Assocation with Financial Crypto)*, 2014. http://eprint.iacr.org/2013/837.

[ADMM14b] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458. IEEE Computer Society Press, May 2014.

[ALZ13] Gilad Asharov, Yehuda Lindell, and Hila Zarosim. Fair and efficient secure multiparty computation with reputation systems. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 201–220. Springer, December 2013.

[ASW97] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *ACM CCS 97*, pages 7–17. ACM Press, April 1997.

[ASW98] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures (extended abstract). In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 591–606. Springer, May / June 1998.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, August 2005.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439. Springer, August 2014.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, August 2000.

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

[CC00] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 93–111. Springer, August 2000.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, February 2007.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 597–608. ACM Press, November 2014.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In Juris Hartmanis, editor, *STOC*, pages 364–369. ACM, 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[GGJ+15] Juan A. Garay, Ran Gelles, David S. Johnson, Aggelos Kiayias, and Moti Yung. A little honesty goes a long way - the two-tier model for secure multiparty computation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 134–158. Springer, March 2015.

[GK09] S. Dov Gordon and Jonathan Katz. Complete fairness in multi-party computation without an honest majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 19–35. Springer, March 2009.

[GKM+13] Juan A. Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *54th FOCS*, pages 648–657. IEEE Computer Society Press, October 2013.

[GMPY06] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 404–428. Springer, March 2006.

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

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

[KB14] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 30–41. ACM Press, November 2014.

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

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, 2008.

[Pin03] Benny Pinkas. Fair secure two-party computation. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 87–105. Springer, May 2003.

[Woo14] Gavin Wood. Ethereum: A secure decentralized transaction ledger. 2014. http://gavwood.com/paper.pdf.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

A Proofs

Proof of Theorem 2.4

Proof. The proof idea is similar to that for composition theorem in [Can01]. Here we need to show that for all PPT real world adversary $\mathcal{A}$ there exists PPT simulator $\mathcal{S}$ so that for all PPT environment $\mathcal{Z}$, the following holds:

$$\text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{H},\mathsf{Q}'_{\bar{\mathcal{G}}})}_{\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{EXEC}^{\bar{\mathcal{G}}}_{\sigma^\pi,\mathcal{A},\mathcal{Z}} \tag{2}$$

By the condition that π realizes $\mathcal{F}$ with $\mathsf{Q}_{\bar{\mathcal{G}}}$-fairness with respect to global functionality $\bar{\mathcal{G}}$, we have: $\forall\mathcal{A}''$ $\exists\mathcal{S}''$ so that $\forall\mathcal{Z}''$

$$\text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})}_{\mathcal{S}'',\mathcal{Z}''} \overset{c}{\approx} \text{EXEC}^{\bar{\mathcal{G}}}_{\pi,\mathcal{A}'',\mathcal{Z}''} \tag{3}$$

By the condition that σ realizes $\mathcal{H}$ with $\mathsf{Q}'_{\bar{\mathcal{G}}}$-fairness with respect to global functionality $\bar{\mathcal{G}}$, in the $\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})$-hybrid world, we have: $\forall\mathcal{A}'$ $\exists\mathcal{S}'$ so that $\forall\mathcal{Z}'$,

$$\text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{H},\mathsf{Q}'_{\bar{\mathcal{G}}})}_{\mathcal{S}',\mathcal{Z}'} \overset{c}{\approx} \text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})}_{\sigma,\mathcal{A}',\mathcal{Z}'} \tag{4}$$

We next prove the theorem.

We first describe the real execution $\text{EXEC}^{\bar{\mathcal{G}}}_{\sigma^\pi,\mathcal{A},\mathcal{Z}}$. Let K be a polynomial upper bound on the number of instances of π that are invoked by σ, and let $\pi[k]$ denote the k-th copy of protocol π. Here let the adversary $\mathcal{A} = \left(\mathcal{A}^\sigma, \mathcal{A}^{\pi[1]}, \mathcal{A}^{\pi[2]}, \ldots, \mathcal{A}^{\pi[K]}\right)$, where each $\mathcal{A}^{\pi[k]}$ is interacting with the k-th instance of π. Note that, as in the UC framework, the environment provides inputs to (and receives outputs from) the "father" protocol σ, and the protocol σ provides inputs to (and receives outputs from) its own subroutines $\pi[k]$'s. We remark that, different from the UC framework, here all protocol instances (including the father protocol and the subroutines) are allowed to access to the global functionality $\bar{\mathcal{G}}$.

We then describe the $\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})$-hybrid execution $\text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})}_{\sigma,\widetilde{\mathcal{A}},\mathcal{Z}}$. Now we let $\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})[k]$ denote the k-th copy of functionality $\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})$ that invoked by protocol σ. Similarly, we define the adversary $\widetilde{\mathcal{A}} = \left(\mathcal{A}^\sigma, \mathcal{S}^{\pi[1]}, \mathcal{S}^{\pi[2]}, \ldots, \mathcal{S}^{\pi[K]}\right)$, where each $\mathcal{S}^{\pi[k]}$ is interacting with the k-th instance of $\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})$. As mentioned before, here the environment provides inputs to protocol σ, and protocol σ provides input to its own subroutines, functionality copies $\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})[k]$'s; all protocol instances are allowed to access to the global functionality $\bar{\mathcal{G}}$.

Based on the above description, we next show the two worlds are indistinguishable through a hybrid argument, as follows:

Lemma A.1. *For all PPT $\mathcal{A}$, there exists PPT $\widetilde{\mathcal{A}}$ so that for all PPT $\mathcal{Z}$,* $\text{EXEC}^{\bar{\mathcal{G}}}_{\sigma^\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{F},\mathsf{Q}_{\bar{\mathcal{G}}})}_{\sigma,\widetilde{\mathcal{A}},\mathcal{Z}}$.

Proof. To prove the lemma, we define hybrids $\mathbf{Hyb}^k$ as follows:

- Let σ^k denote the following protocol instances:
  - an instance of σ;
  - $k-1$ instances of π, denoted $\pi[1], \ldots, \pi[k-1]$;
  - $K-k+1$ instances of $\mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})$, denoted $\mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})[k], \ldots, \mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})[K]$;
- Let $\mathcal{A}^k$ denote the following adversary copies:
  - $\mathcal{A}^\sigma$;
  - $k-1$ instances of $\mathcal{A}^\pi$, denoted $\mathcal{A}^{\pi[1]}, \ldots, \mathcal{A}^{\pi[k-1]}$;
  - $K-k+1$ instances of $\mathcal{S}^\pi$, denoted $\mathcal{S}^{\pi[k]}, \ldots, \mathcal{S}^{\pi[K]}$;

Define $\mathcal{B}$, which consists of K copies of adversaries, and K copies of protocol/functionality instances. Define $\mathcal{D}$, which consists of the k-th copy of adversary $\mathcal{A}$, and the k-th copy of protocol/functionality instance π. If $\mathcal{A} = \mathcal{S}^{\pi[k]}$ and $\pi = \mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})[k]$, then $\mathcal{B}$ is identical to $\mathbf{Hyb}^k$. If $\mathcal{A} = \mathcal{A}^{\pi[k]}$ and $\pi = \pi[k]$, then $\mathcal{B}$ is identical to $\mathbf{Hyb}^{k+1}$. We next show that adjacent hybrids are indistinguishable.

Claim A.2. *For $k \in \{1, \ldots, K\}$, the hybrids $\mathbf{Hyb}^k$ and $\mathbf{Hyb}^{k+1}$ are indistinguishable to any PPT $\mathcal{Z}$.*

Proof of Claim A.2. By contradiction, assume there is a PPT $\mathcal{Z}$ who can tell the difference between $\mathbf{Hyb}^k$ and $\mathbf{Hyb}^{k+1}$. That means, $\textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^k, \mathcal{A}^k, \mathcal{Z}} \not\overset{\text{c}}{\approx} \textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^{k+1}, \mathcal{A}^{k+1}, \mathcal{Z}}$. Based on such $\mathcal{Z}$, we can define $\mathcal{Z}^k$ to simulate the interaction of all the rest of the network except the k-th place of the subroutine.

Based on the definition of coercion hybrid $\mathbf{Hyb}^k$ above, we can easily see that $\textsc{Exec}^{\bar{\mathcal{G}}, \mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})}_{\mathcal{S}^{\pi[k]}, \mathcal{Z}^k}$ is the representation of $\textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^k, \mathcal{A}^k, \mathcal{Z}}$. Similarly, we can easily see that $\textsc{Exec}^{\bar{\mathcal{G}}}_{\pi, \mathcal{A}^{\pi[k]}, \mathcal{Z}^k}$ is the representation of $\textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^{k+1}, \mathcal{A}^{k+1}, \mathcal{Z}}$.

Based on the assumption that $\textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^k, \mathcal{A}^k, \mathcal{Z}} \not\overset{\text{c}}{\approx} \textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^{k+1}, \mathcal{A}^{k+1}, \mathcal{Z}}$, we immediately have $\textsc{Exec}^{\bar{\mathcal{G}}, \mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})}_{\mathcal{S}^{\pi[k]}, \mathcal{Z}^k} \not\overset{\text{c}}{\approx} \textsc{Exec}^{\bar{\mathcal{G}}}_{\pi, \mathcal{A}^{\pi[k]}, \mathcal{Z}^k}$. However, this contradicts to the premise in Equation 3. That means, our assumption that $\textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^k, \mathcal{A}^k, \mathcal{Z}} \not\overset{\text{c}}{\approx} \textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^{k+1}, \mathcal{A}^{k+1}, \mathcal{Z}}$ is not true. This completes our proof of the claim that $\mathbf{Hyb}^k$ and $\mathbf{Hyb}^{k+1}$ are indistinguishable for $k \in \{1, \ldots, K\}$. □

Finally, we note that hybrid $\mathbf{Hyb}^1$ is identical to $\textsc{Exec}^{\bar{\mathcal{G}}, \mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})}_{\sigma, \widetilde{\mathcal{A}}, \mathcal{Z}}$, and the hybrid $\mathbf{Hyb}^{K+1}$ is identical to $\textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^\pi, \mathcal{A}, \mathcal{Z}}$. Based on the claim above, we can see that $\mathbf{Hyb}^1 \overset{\text{c}}{\approx} \mathbf{Hyb}^2 \overset{\text{c}}{\approx} \cdots \overset{\text{c}}{\approx} \mathbf{Hyb}^K \overset{\text{c}}{\approx} \mathbf{Hyb}^{K+1}$. This implies that $\textsc{Exec}^{\bar{\mathcal{G}}}_{\sigma^\pi, \mathcal{A}, \mathcal{Z}} \overset{\text{c}}{\approx} \textsc{Exec}^{\bar{\mathcal{G}}, \mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})}_{\sigma, \widetilde{\mathcal{A}}, \mathcal{Z}}$, which completes the proof of the lemma. □

We then consider the ideal world and the $\mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})$-hybrid world. Recall the fact that σ realizes $\mathcal{W}(\mathcal{H}, \mathsf{Q}'_{\bar{\mathcal{G}}})$ with $\mathsf{Q}'_{\bar{\mathcal{G}}}$-fairness in the $\mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})$-hybrid world. We immediately have Equation 4 and the following lemma.

Lemma A.3. *For all PPT $\widetilde{\mathcal{A}}$, there exists PPT $\mathcal{S}$ so that for all PPT $\mathcal{Z}$,* $\textsc{Exec}^{\bar{\mathcal{G}}, \mathcal{W}(\mathcal{H}, \mathsf{Q}'_{\bar{\mathcal{G}}})}_{\mathcal{S}, \mathcal{Z}} \overset{c}{\approx} \textsc{Exec}^{\bar{\mathcal{G}}, \mathcal{W}(\mathcal{F}, \mathsf{Q}_{\bar{\mathcal{G}}})}_{\sigma, \widetilde{\mathcal{A}}, \mathcal{Z}}$

Based on the two lemmas above, we now have: for all $\mathcal{A}$, there exists $\mathcal{S}$ so that for all $\mathcal{Z}$, $\text{EXEC}^{\bar{\mathcal{G}}}_{\sigma^\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{EXEC}^{\bar{\mathcal{G}},\mathcal{W}(\mathcal{H},\mathsf{Q}'_{\bar{\mathcal{G}}})}_{\mathcal{S},\mathcal{Z}}$. This completes the first part of the proof.

□