Resource Fairness and Composability of Cryptographic Protocols

Juan A. Garay* Philip MacKenzie† Manoj Prabhakaran‡ Ke Yang§

October 1, 2005

Abstract

We introduce the notion of *resource-fair* protocols. Informally, this property states that if one party learns the output of the protocol, then so can all other parties, as long as they expend roughly the same amount of resources. As opposed to similar previously proposed definitions, our definition follows the standard simulation paradigm and enjoys strong composability properties. In particular, our definition is similar to the security definition in the universal composability (UC) framework, but works in a model that allows any party to request additional resources from the environment to deal with dishonest parties that may prematurely abort.

In this model we specify the ideally fair functionality as allowing parties to "invest resources" in return for outputs, but in such an event offering all other parties a fair deal. (The formulation of fair dealings is kept independent of any particular functionality, by defining it using a "wrapper.") Thus, by relaxing the notion of fairness, we avoid a well-known impossibility result for fair multi-party computation with corrupted majority; in particular, our definition admits constructions that tolerate arbitrary number of corruptions. We also show that, as in the UC framework, protocols in our framework may be arbitrarily and concurrently composed.

Turning to constructions, we define a "commit-prove-fair-open" functionality and design an efficient resource-fair protocol that securely realizes it, using a new variant of a cryptographic primitive known as "time-lines." With (the fairly wrapped version of) this functionality we show that some of the existing secure multi-party computation protocols can be easily transformed into resource-fair protocols while preserving their security.

1 Introduction

Secure multi-party computation (MPC) is one of the most fundamental problems in cryptography. At a high level, the problem is concerned with n parties, each holding a private input x_i, that want to compute a function $(y_1, y_2, \ldots, y_n) \leftarrow f(x_1, x_2, ..., x_n)$ so that each party learns its own output y_i, but no other information is revealed, even in the presence of malicious parties that may deviate arbitrarily from the protocol [59, 60, 40, 7, 19, 39].

It is standard to define the security of an MPC protocol using a *simulation paradigm*, where two experiments are presented: one *real world* experiment that models the actual setting in which a protocol takes place, and one *ideal process* where an *ideal functionality* performs the desired computation. The security of a protocol is defined (informally) as the existence of an *ideal adversary* in the ideal process

*Bell Labs – Lucent Technologies. E-mail: garay@research.bell-labs.com.

†DoCoMo USA Labs. philmac@docomolabs-usa.cpm

‡Computer Science Department, University of Illinois at Urbana-Champaign. mmp@uiuc.edu

§Google. yangke@google.com..

that *simulates* the real-world experiment for any given real world adversary. Many simulation-based security definitions in various models have been proposed [39, 14, 52, 15, 42, 46, 3]. The *universal composability* (UC) framework of Canetti [15] is among the models that provide perhaps the strongest security guarantee in the following sense: a protocol π that is secure in this framework is guaranteed to remain secure when arbitrarily composed with other protocols, by means of a "composition theorem."

1.1 Fair multi-party computation

This paper focuses on a particular issue in MPC, namely, *fairness*. Informally, a protocol is fair if either all the parties learn the output of the function, or no party learns anything (about the output). This property is also known as "complete fairness," and can be contrasted with "partial fairness," where fairness is achieved only when some conditions are satisfied [42][1]; see also [31].

Clearly, fairness is a very desirable property for secure MPC protocols, and in fact, many of the security definitions cited above imply fairness. (See [42] for an overview of different types of fairness, along with their corresponding histories.) Here we briefly describe some known results about (complete) fairness. Let n be the total number of participating parties and t be the number of corrupted parties. It is known that if $t < n/3$, then fairness can be achieved without any set-up assumptions, both in the information-theoretic setting [7, 19] and in the computational setting [40, 39] (assuming the existence of trapdoor permutations). If $t < n/2$, one can still achieve fairness if all parties have access to a broadcast channel; this also holds both information theoretically [54] and computationally [40, 39].

Unfortunately, the above fairness results no longer hold when $t \geq n/2$, i.e., when a majority of the parties are corrupted. In fact, it was proved that there do not exist fair MPC protocols in this case, even when parties have access to a broadcast channel [20, 39]. Intuitively, this is because the adversary, controlling a majority of the corrupted parties, can abort the protocol prematurely and always gain some unfair advantage. This impossibility result easily extends to the *common reference string* (CRS) model (where there is a common string drawn from a prescribed distribution available to all the parties).

Nevertheless, fairness is still important (and necessary) in many applications in which at least half the parties may be corrupted. One such application is contract signing (or more generally, the fair exchange of signatures) by two parties [8]. To achieve some form of fairness, various approaches have been explored. One such approach adds to the model a trusted third party, who is essentially a judge that can be called in to resolve disputes between the parties. (There is a large body of work following this approach; see, e.g., [2, 13] and references therein.) This approach requires a trusted external party that is constantly available. Another recent approach adds an interesting physical communication assumption called an "envelope channel," which might be described as a "trusted postman" [45].

A different approach that avoids the available trusted party requirement uses a mechanism known as "gradual release," where parties take turns to release their secrets in a "bit by bit" fashion. Therefore, if a corrupted party aborts prematurely, it is only a little "ahead" of the honest party, and the honest party can "catch up" by investing an amount of time that is comparable to (and maybe greater than) the time spent by the adversary. (Note that this is basically an *ad hoc* notion of fairness.) Early works in this category include [8, 29, 33, 41, 4, 24]. More recent work has focused on making sure — under the assumption that there exist problems, such as modular exponentiation, that are not well suited for parallelization[2] — that this "unfairness" factor is bounded by a small constant [11, 38, 53]. As we discuss below, our constructions also use a gradual release mechanism secure against parallel

[1] For example, in [42] there exists a specific party P_1 such that the protocol is fair as long as P_1 is uncorrupted; but when P_1 is corrupted, then the protocol may become completely unfair.

[2] Indeed, there have been considerable efforts in finding efficient exponentiation algorithms (e.g., [1, 58]) and still the best methods are sequential.

attacks.

1.2 Resource fairness

In this paper we propose a new notion of fairness with a rigorous simulation-based security definition (without a trusted third party), that allows circumvention of the impossibility result discussed above in the case of corrupted majorities. We call this new notion *resource fairness*. In a nutshell, resource fairness means that if any party learns the output of a function, then all parties *will be able to learn the output of the function by expending roughly the same amount of resources*. (In our case, the resource will be time.) In order to model this, we allow honest parties in our framework (both in the real world and in the ideal process) to request resources from the environment, and our definition of resource fairness relates the amount of requested resources to the amount of resources available to corrupted parties.

Slightly more formally, a resource-fair functionality can be described in two steps. We start with the most natural notion for a fair functionality $\mathcal{F}$. A critical feature of a fair functionality is the following:

- There are certain messages that $\mathcal{F}$ sends to multiple parties such that all of them must receive the message in the same round of communication. (For this it is necessary that the adversary in the ideal process cannot block messages from $\mathcal{F}$ to the honest parties.[3])

Then we modify it using a "wrapper" to obtain a functionality $\mathcal{W}(\mathcal{F})$. The wrapper allows the adversary to make "deals" of roughly the following kind:

- Even if $\mathcal{F}$ requires a message to be simultaneously delivered to all parties, the adversary can "invest" computational resources and obtain the message from $\mathcal{W}(\mathcal{F})$ in an earlier communication round.

- However, in this case, $\mathcal{W}(\mathcal{F})$ will offer a "fair deal" to the honest parties: each of them will be given the option of obtaining its message by investing (at most) the same amount of computational resources as invested by the adversary.

Once we define $\mathcal{W}(\mathcal{F})$ as our ideal notion of a fair functionality, we need to define when a *protocol* is considered to be as fair as $\mathcal{W}(\mathcal{F})$. We follow the same paradigm as used in the UC framework for defining security: A protocol π is said to be as fair as $\mathcal{W}(\mathcal{F})$ if for every real adversary $\mathcal{A}$ there exists an ideal adversary (simulator) $\mathcal{S}$ such that no environment can distinguish between interacting with $\mathcal{A}$ and parties running a protocol π (the real world), and interacting with $\mathcal{S}$ and parties talking to $\mathcal{W}(\mathcal{F})$ (the ideal world). But in addition we require that $\mathcal{S}$ cannot invest much more resources than $\mathcal{A}$ has.

This last condition is crucial for the notion of resource fairness. To see this, note the following:

- In the ideal world, in the event of the adversary $\mathcal{S}$ obtaining a message by investing some amount of resources, an honest party can be required to invest the same amount of resources to get its message.

- By the indistinguishability condition, this is the same as the amount of resources required by the honest parties in the real world. Thus, the resources required by the honest parties in the real world can be as much as that invested by the adversary $\mathcal{S}$ in the ideal world.

[3] In the original formulation of the UC framework [15], the adversary in the ideal process could block the outputs from the ideal functionality to all the parties. Thus, the ideal process itself is already completely unfair, and therefore discussing fair protocols is not possible. The new version [16] also has "immediate functionalities" as the default—see Section 2.1.

Recall that the (intuitive) notion of resource fairness requires that the resources required by an honest party in the real world should be comparable to what the adversary $\mathcal{A}$ (in the real world) expends, to obtain its output. Thus, to achieve the notion, we must insist that the amount of resources invested by the ideal world adversary $\mathcal{S}$ is comparable to what the real world adversary $\mathcal{A}$ expends.

Note that for these comparisons, *the resources in the ideal world must be measured using the same units as in the real world.* However, these invested resources do not have a physical meaning in the ideal world: it is just a "currency" used to ensure that the fairness notion is correctly reflected in the ideal world process.

The only resource we shall consider in this work is computation time.

Fairness through gradual release. Our definition is designed to capture the fairness guarantees offered by the method of *gradual release*. The gradual release method by itself is not new, but our simulation-based definition of fairness is.

Typical protocols using gradual release consist of a "computation" phase, where some computation is carried out, followed by a "revealing" phase, where the parties gradually release their private information towards learning a result y. Our simulation-based definition requires one to be able to simulate both the computation phase and the release phase. In contrast, previous *ad hoc* security definitions did not require this, and consisted, explicitly or implicitly, of the following three conditions:

1. The protocol must be completely simulatable up to the revealing phase.
2. The revealing phase must be completely simulatable *if the simulator knows* y.
3. If the adversary aborts in the revealing phase and computes y by brute force in time t, then all the honest parties can compute y in time comparable to t.[4]

While carrying some intuition about security and fairness, we note that these definitions are not fully simulation-based. To see this, consider a situation where an adversary $\mathcal{A}$ aborts (with, say, probability 1/2) early on in the revealing phase, such that it is still infeasible for $\mathcal{A}$ to find y by brute force. At this time, it is also infeasible for the honest parties to find y by brute force. Now, how does one simulate $\mathcal{A}$'s view in the revealing phase? Notice that the revealing phase is simulatable *only if the ideal adversary* $\mathcal{S}$ *knows* y. However, since nobody learns y in the real world, they should not learn y in the ideal world, and, in particular, $\mathcal{S}$ should not learn y. Thus, the above approach gives no guarantee that $\mathcal{S}$ can successfully simulate $\mathcal{A}$'s view. In other words, by aborting early in the revealing phase, $\mathcal{A}$ might gain some unfair advantage. This can become an even more serious security problem when protocols are composed.

Environment's role. In our formulation of fairness, if a protocol is aborted, the honest parties get the *option* of investing resources and recovering a message from the functionality. However, the decision of whether to exercise this option is not specified by the protocol itself, but left to the environment. Just being provided with this option is considered fair.[5] The fairness guarantee is that the amount of resources that need to be invested by the adversary to recover the message will be comparable to what the honest party requires. Whether the adversary actually makes that investment or not is not known to the honest parties.

[4] As we discussed before, an honest party typically will spend *more* time than the adversary in this case.

[5] In a previous version of this work [37], we insisted that the protocol itself must decide whether or not to invest computational resources and recover a message from an aborted protocol. Further, for being fair, we required that if the adversary could have obtained its part of the message, then the protocol *must* carry out the recovery. This leads to the unnatural requirement that the protocol must be aware of the computational power of the adversary (up to a constant).

Leaving the recovery decision to the environment has the consequence that our notion of fairness becomes a robust "relative" notion. In some environments the execution might be (intuitively) unfair if, for instance, the environment refuses to grant any requests for resources. However, this is analogous to the situation in the case of security: Some environments can choose to reveal all the honest parties' inputs to the adversary. The protocol's guarantee is limited to mimicking the ideal functionality (which by definition is secure and fair). We do not seek to incorporate absolute guarantees of fairness (or security) into the protocol, as they are dependent on the environment.

1.3 Our results

We summarize the main results presented in this paper.

1. **A fair multi-party computation framework.** We start with a framework for fair multi-party computation (FMPC), which is a variation of the UC framework, but with modifications so that it is possible to design functionalities such that the ideal process is (intuitively) fair.We then present a generic *wrapper* functionality, denoted $\mathcal{W}(\cdot)$, that converts a fair functionality into one that allows for a resource-fair realization in the real world. We then present definitions for resource-fair protocols that securely realize functionalities in this framework. We emphasize that these definitions are in the (standard) simulation paradigm[6] and admit protocols that tolerate an arbitrary number of corruptions. Finally, we prove a *composition theorem* similar to the one in the UC framework.

2. **The "commit, prove and fair-open" functionality.** We define a *commit-prove-fair-open* functionality $\mathcal{F}_{\text{CPFO}}$ in the FMPC framework. This functionality allows all parties to each commit to a value, prove relations about the committed value, and more importantly, open all committed values simultaneously to all parties. This functionality (more specifically, a wrapped version of it) lies at the heart of our constructions of resource-fair MPC protocols. We then construct an efficient resource-fair protocol GradRel that securely realizes $\mathcal{F}_{\text{CPFO}}$, assuming static corruptions. Our protocol uses a new variant of a cryptographic primitive known as *time-lines* [34], which enjoys a property that we call *strong pseudorandomness*. In turn, the construction of time-lines hinges on a refinement of the *generalized BBS* assumption [11], which has broader applicability.

3. **Efficient and resource-fair MPC protocols.** By using the $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ functionality, many existing secure MPC protocols can be easily transformed into resource-fair protocols while preserving their security. In particular, we present two such constructions. The first construction converts the universally composable MPC protocol by Canetti *et al.* [18] into a resource-fair MPC protocol that is secure against static corruptions in the CRS model in the FMPC framework. Essentially, the only thing we need to do here is to replace an invocation of a functionality in the protocol called "commit-and-prove" by our $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ functionality.

   The second construction turns the efficient MPC protocol by Cramer *et al.* [22] into a resource-fair one in the "public key infrastructure" (PKI) model in a similar fashion. The resulting protocol becomes secure and resource fair (assuming static corruptions) in the FMPC framework, while preserving the efficiency of the original protocol — an additive overhead of only $O(\kappa^2 n)$ bits of communication and an additional $O(\kappa)$ rounds, for κ the security parameter.

1.4 Organization of the paper

The paper has two main components: the formalization of the notion of resource-fairness, and protocol constructions satisfying this notion. In Section 2 we present the new notion, and the subsequent

[6] Indeed, as explained in Section 2.4, our definition of resource fairness subsumes the UC definition of security.

sections are dedicated to explaining the protocol constructions. Within Section 2, we describe the FMPC framework, describe "wrapped" functionalities, give security and fairness definitions and finally state and prove a composition theorem. In Section 3 we present some definitions and number-theoretic assumptions used by our constructions. In Section 4 we present the $\mathcal{F}_{\text{CPFO}}$ functionality and a protocol that realizes a wrapped version of it, which we use in Section 5 to achieve resource-fair MPC. For the sake of readability, some of the proofs and extensions are given in the Appendix.

2 FMPC Framework and Resource Fairness

In this section first we describe the FMPC framework. Then we define our new fairness notion, and prove its universal composability.

2.1 The FMPC framework

We now define the new framework used in our paper, which we call the *fair multi-party computation* (FMPC) framework. It is similar to the universal composability (UC) framework [15, 16]. In particular, there are n parties, $P_1, P_2, ..., P_n$, a real-world adversary $\mathcal{A}$, an ideal adversary $\mathcal{S}$, an ideal functionality $\mathcal{F}$, and an environment $\mathcal{Z}$. However, FMPC contains some modifications so that fairness becomes possible. We stress that the FMPC framework still inherits the strong security of UC, and we shall prove a composition theorem in the FMPC framework similar to UC.

Instead of describing the FMPC framework from scratch, we only discuss its most relevant features and differences from the UC framework. We present a brief overview of the UC framework in Appendix A; refer to [16] for a detailed presentation. The critical features of the FMPC framework are:

1. **Interactive circuits/PRAMs.** Instead of interactive Turing machines, we assume the computation models in the FMPC framework are non-uniform interactive PRAMs (IPRAMs).[7] This is a non-trivial distinction, since we will work with exact time bounds in our security definition, and the "equivalence" between various computation models does not carry over there. The reason to make this modification is that, we will need to model machines that allow for simulation and subroutine access with no significant overhead. Thus, if we have two protocols, and one calls the other as a black-box, then the total running time of the two protocols together will be simply the sum of their running times. Obviously, Turing machines are not suitable here.

   We say an IPRAM is *t-bounded* if it runs for a total of at most t steps.[8] We always assume that t is a polynomial of the security parameter κ, though for simplicity we do not explicitly write $t(\kappa)$. We can view a t-bounded IPRAM as a "normal" IPRAM with an explicit "clock" attached to it that terminates the execution after a total number of t cumulative steps (notice that an IPRAM is reactive: i.e., it maintains state across activations).

2. **Synchronous communication with rounds.** In the UC framework, the communication is asynchronous, and controlled by the adversary, and further there is no notion of time. This makes fair MPC impossible, since the adversary may, for example, choose not to deliver the final protocol message to an uncorrupted party P_i. In this case, P_i will never obtain the final result *because it is never activated again.* What is needed is to let parties be able to *time out* if they do not receive an expected message within some time bound. However, instead of incorporating a

[7] IPRAMs are simply extensions to the PRAM machines with special read-only and write-only memories for interacting with each other.

[8] For simplicity, we assume that an IPRAM can compute a modular squaring operation (i.e., compute x^2 mod M on input (x, M)) in constant time.

full-fledged notion of time into the model, for simplicity we shall work in a "synchronous model." Specifically, in the FMPC framework there will be synchronous rounds of communication in both the real world and the ideal process. (See [43, 49] for other synchronous versions of the UC framework.)

In each round we allow the adversary to see the messages sent by other parties in that round, before generating its messages (i.e., we use a *rushing adversary* model.

Note that this model of communication is used in both the real *and* ideal worlds used for defining security. (As we shall see later, a resource-fair ideal functionality is designed to be aware of this round structure. This is necessary because the amount of resources required by an honest party to retrieve messages that the adversary blocks, is directly related to the number of communication rounds in the protocol that pass prior to that.) This allows also the environment to be aware of the round structure.

We stress that in our protocols, we use the synchronous communication model only as a substitute for having time-outs on messages (which are sequentially numbered). Our use of the synchronous model is only that if a message does not arrive in a communication round in which it is expected, then the protocol can specify an action to take.

For simplifying our protocols, we also incorporate an authenticated broadcast capability into our communication model. (This is not essential for the definitions and composition theorem.) The broadcast can be used to ensure that all parties receive the same message; however no fairness guarantee is assumed: some parties may not receive a message broadcast to them. Indeed, such a broadcast mechanism can be replaced by resorting to, for instance, the broadcast protocol from [42] (with a slight modification to the ideal abstraction of broadcasting, to allow for the round structure in our synchronous model).

3. **Guaranteed-round message delivery from functionalities.** Following the revised formulation of the UC framework [16], in our model the messages from an ideal functionality $\mathcal{F}$ are forwarded directly to the uncorrupted parties and cannot be blocked by $\mathcal{S}$.[9] (Note that this is not guaranteed by the previous specification regarding synchronous communication.) Specifically, $\mathcal{F}$ may output $(\mathsf{fairdeliver}, sid, msg\text{-}id, \{(\mathsf{msg}_1, P_{i_1}), \ldots, (\mathsf{msg}_m, P_{i_m})\}, j)$, meaning that each message msg_i will be delivered to the appropriate party P_i at round j. We will call this feature *guaranteed-round message delivery.*

4. **Resource requests.** Typically, an honest party's execution time (per activation) is bounded *a priori* by a polynomial in the security parameter. But in our model, an honest party can "request" the environment to allow it extra computation time. If the request is granted, then the party can run for longer in its activations, for as many computation steps as granted by the environment. More formally, an honest party in the real-world execution can send a message of the form $(\mathsf{dealoffer}, sid, msg\text{-}id, \beta)$ to the environment; if the environment responds to this with $(\mathsf{dealaccept}, sid, msg\text{-}id)$, then the party gets a "credit" of β extra computational steps (which gets added to the credits it accumulated before). In a hybrid model, these credits may also be used to accept deals offered by sub-functionality instances. Note that the environment can decide to grant a request or not, depending on the situation.

[9] In the original UC formulation, messages from the ideal functionality $\mathcal{F}$ were forwarded to the uncorrupted parties by the ideal adversary $\mathcal{S}$, who may block these messages and never actually deliver them. The ability of $\mathcal{S}$ to block messages from $\mathcal{F}$ makes the ideal process inherently unfair.

2.2 A fair SFE functionality

Before we introduce the notion of "wrapped functionalities," it is useful to note that in the model described above, we can construct a functionality that can be considered a fair *secure function evaluation* functionality $\mathcal{F}_f$. This functionality is similar to the homonymous functionality in the UC framework [15], except for (1) the fact that there is no reference to the number of corrupted parties, as in our case it may be arbitrary, (2) the output is a single public value, instead of private outputs to each party[10], (3) the added round structure—in particular, the adversary specifies the round at which the outputs are to be produced (deliverat message)[11], and (4) the use of the fair delivery mechanism of the FMPC framework.

Functionality $\mathcal{F}_f$

$\mathcal{F}_f$ proceeds as follows, running with security parameter κ, parties $P_1, \ldots, P_n$, and an adversary $\mathcal{S}$.

- Upon receiving a value (input, sid, v) from P_i, set $x_i \leftarrow v$.
- As soon as inputs have been received from all parties, compute $y \leftarrow f(x_1, \ldots, x_n)$.
- Wait to receive message $(\mathsf{deliverat}, sid, s)$ from $\mathcal{S}$. As soon as the message is received, output $(\mathsf{fairdeliver}, sid, 0, \{((\mathsf{OUTPUT}, y), P_i)\}_{1 \le i \le n}, s)$, that is, set up a fair delivery of message $(\mathsf{OUTPUT}, sid, y)$ to all parties for delivery in the sth round.

Figure 1: *The SFE functionality for evaluating an n party function f.*

We emphasize that in the FMPC framework, and because $\mathcal{F}_f$ uses the fair delivery mechanism, it is easy to see that in the ideal model, the functionality $\mathcal{F}_f$ satisfies the intuitive definition of fairness for secure function evaluation. (This is called "complete fairness" in [42].) Specifically, if one party receives the output, all parties receive the output.

2.3 Wrapped functionalities

As we have stated previously, according to the result of Cleve [20], it is impossible to construct fair protocols, and thus there is no protocol that could realize the functionality $\mathcal{F}_f$ describe above. Therefore we will create a relaxation of $\mathcal{F}_f$ that can be realized, and that will be amenable to analysis in terms of resource fairness. To do this, we will actually construct a more general *wrapper functionality* which provides an interface to any functionality and will be crucial to defining resource fairness. We denote the wrapper functionality as $\mathcal{W}()$, and a wrapped functionality as $\mathcal{W}(\mathcal{F})$.[12]

The wrapper operates as follows. For ease of explanation, assume the functionality $\mathcal{F}$ schedules a single fair delivery to all parties with the same message. Basically, the wrapper handles this fair delivery by storing the message internally until the specified round for delivery, and then outputing the message to be delivered immediately to each party. It also allows the adversary $\mathcal{S}$ to *invest* resources and obtain the message in advance. (Of course, in the ideal process, this investment is

[10] This can be easily extended to the case where each party receives a different private output, since y may contain information for each individual party, encrypted using a one-time pad. In fact, the framework developed here accommodates interactive functionalities with even more general fairness requirements, where different messages from the functionality can be fairly delivered to different sets of parties at multiple points in the execution.

[11] Alternatively, the functionality could take the number of rounds as a parameter.

[12] Assuming $\mathcal{F}$ is a fair functionality, one could say that $\mathcal{W}(\mathcal{F})$ is a "resource-fair" functionality. However, there is an important distinction: a protocol that securely realizes F would be called a "fair" protocol, while a protocol that securely realizes F would not be called a "resource-fair" protocol unless it satisfies an additional requirement, as is discussed below.

simply notational - the adversary does not actually expend any resources.) It will still deliver the message to each party at the specified round unless $\mathcal{S}$ offers a deal to a party to "expend" a certain amount of resources. If that party does not take the deal, then the wrapper will not deliver the message at any round. The wrapper enforces the condition that it only allows $\mathcal{S}$ to offer a deal for at most the amount of resources that $\mathcal{S}$ itself invested. Except for the messages discussed above, all communication to and from $\mathcal{F}$ are simply forwarded directly to and from $\mathcal{F}$.

The formal definition of $\mathcal{W}(\mathcal{F})$ is given in Figure 2. Here we provide some intuition behind some of the labels and variables. Let $F(msg\text{-}id)$ denote a fairdeliver message record (containing message-destination pairs (msg_i, P_i) and $(\mathsf{msg}_{\mathcal{S}}, \mathcal{S})$), with identifier *msg-id*. Associated with any such record is a round number, which specifies the communication round in which the messages in that record will be delivered to all the parties and $\mathcal{S}$. Initially each such record is marked unopened to signify that no party has received any of the messages yet. At any round the adversary $\mathcal{S}$ has the option of obtaining its messages (i.e., messages for the corrupt players and $\mathcal{S}$) by investing $\alpha_{msg\text{-}id}$ amount of resources.[13] If it does so, then the record is marked opened. Once a message is marked opened, $\mathcal{W}(\mathcal{F})$ will ensure that each honest party is offered a fair deal. For each honest party P_i this can happen in one of two ways: either the adversary offers a deal to the honest party to obtain its message msg_i by investing at most $\alpha_{msg\text{-}id}$ amount of resources (in which case the pair (msg_i, P_i) is marked dealt), or if the adversary makes no such offer, then P_i receives the message at the specified round without having to make any investment at all. Refer to Figure 2 for the complete specification of $\mathcal{W}(\mathcal{F})$.

Fact 2.1 If the adversary obtains a message that was set for fair delivery with message ID *msg-id*, every honest party that is set to receive a message in the fair delivery with message ID *msg-id* will either receive it at the specified round, or will be offered a deal for at most the amount invested by the adversary.

To see why this is true, consider a set of messages that is set for fair delivery, as $(\mathsf{fairdeliver}, sid, msg\text{-}id, \{(\mathsf{msg}_1, P_{i_1}), \ldots, (\mathsf{msg}_m, P_{i_m}), (\mathsf{msg}_{\mathcal{S}}, P_{\mathcal{S}})\}, j)$. The adversary can receive the messages (for the corrupt parties or for itself) in two ways only. The first way is if it does not send any invest or noinvest messages to the functionality. In this case the record is never marked opened, no deal messages are sent, and all parties receive their messages in the specified round j. The second way is if it invests a value α. Then the record is marked opened. After this, if the adversary does not send a dealoffer message until round j, then again all honest parties receive their messages in round j. If the adversary does send a dealoffer message, the $\mathcal{W}(\mathcal{F})$ offers a deal to each honest party to provide their messages if they invest $\beta' \leq \alpha$. Note that we do allow the adversary to specify a value β for the deal,[14] but if $\beta > \alpha$, then $\mathcal{W}(\mathcal{F})$ ignores this value, and makes the offer at the value α. Note that the adversary can prevent all the messages in a record from being delivered (by sending a noinvest message), but this can be done only as long as the message is marked unopened.

Conventions. Below we clarify some of the conventions in the new framework.

- **Using resource-requesting subroutines.** A protocol interfaces with a resource-requesting subroutine in a natural way. When a protocol ρ uses a subroutine π which makes resource requests (for instance, if π accesses a wrapped functionality $\mathcal{W}(\mathcal{F})$, or if π securely realizes a

[13] This simply means that the adversary sends a message $(\mathsf{invest}, sid, msg\text{-}id, \alpha_{msg\text{-}id})$ to $\mathcal{W}(\mathcal{F})$, and the amount $\alpha_{msg\text{-}id}$ is counted towards the total amount of resources invested by $\mathcal{S}$.

[14] The exact amount of resources that the honest party will need to request from the environment will depend on the specifics of the real world protocol. We would like to keep the ideal functionality specification independent of this. Hence we allow this quantity to be specified by the simulator we design. Our simulators will always use $\beta \leq \alpha$, but for a general adversary this is enforced by $\mathcal{W}(\mathcal{F})$ by using $\min \beta, \alpha$ instead of β.

Wrapper functionality $\mathcal{W}(\mathcal{F})$

$\mathcal{W}(\mathcal{F})$ proceeds as follows, running with parties $P_1, \ldots, P_n$, and an adversary $\mathcal{S}$: It internally runs a copy of $\mathcal{F}$.

- Whenever it receives an incoming communication, which is not one of the special messages (invest, noinvest, dealoffer and dealaccept), it immediately passes this message on to $\mathcal{F}$.
- Whenever $\mathcal{F}$ outputs any message *not* marked for fair delivery, output this message (i.e., pass it on to its destination, allowing the adversary to block this message[a]).
- Whenever $\mathcal{F}$ outputs a record (fairdeliver, *sid*, *msg-id*, $\{(\mathsf{msg}_1, P_{i_1}), \ldots, (\mathsf{msg}_m, P_{i_m}), (\mathsf{msg}_{\mathcal{S}}, \mathcal{S})\}, j$),[b] $\mathcal{W}(\mathcal{F})$ stores this for future delivery (in communication round j). The message record is marked unopened to indicate that the adversary has not yet obtained this message. Also all the pairs (msg_i, P_i) in the record are marked undealt to indicate that no deal has been offered to the party P_i for obtaining this message.
- If a record with ID *msg-id* is marked as unopened and the adversary sends a message (noinvest, *sid*, *msg-id*), then that record is erased (and the messages in it will not be delivered to any party).
- If *msg-id* is marked as unopened and the adversary $\mathcal{S}$ sends a message (invest, *sid*, *msg-id*, α), then
  - the record with ID *msg-id* is marked as opened, and α is stored as $\alpha_{msg\text{-}id}$. For each corrupt party P_i, if the record contains the message (msg, P_i), that message is delivered to $\mathcal{S}$ immediately (even if the round j has not yet been reached). If the record contains $(\mathsf{msg}_{\mathcal{S}}, \mathcal{S})$ then that message is also delivered to $\mathcal{S}$ at this point.
- At any round in which a fairdeliver record (marked unopened or opened) is stored for delivery at that round, for every pair (msg, P) in that record marked undealt, msg is output for immediate delivery to P (i.e., using the fair delivery mechanism). Then that record is erased.
- If a record *msg-id* is marked as opened and the adversary sends (dealoffer, *sid*, *msg-id*, P_i, β) for some honest party P_i, then
  - $\mathcal{W}(\mathcal{F})$ marks the pair (msg_i, P_i) in the record *msg-id* as dealt, and sends (dealoffer, *sid*, *msg-id*, β') to P_i, where $\beta' = \min(\beta, \alpha_{msg\text{-}id})$.
- If an honest party P_i responds to (dealoffer, *sid*, *msg-id*, β) with (dealaccept, *sid*, *msg-id*, β), then the stored message msg_i is immediately delivered to P_i, and erased from the stored record.

[a] In a typical fair functionality, all messages from $\mathcal{F}$ could be marked for fair delivery. However we allow for non-fair message delivery also in the model.

[b] A message record is identified using the ID *msg-id*, which $\mathcal{F}$ will ensure is unique for each record.

Figure 2: *The wrapper functionality $\mathcal{W}(\mathcal{F})$.*

wrapped functionality $\mathcal{W}(\mathcal{F})$), it is for ρ to decide when to grant resource requests made by π. In the cases we consider, the outer protocol ρ will simply transfer such requests to *its* environment.

- **Dummy honest parties in the ideal world.** An honest party in the ideal world is typically a "dummy" party. In the original UC framework this means that it acts as a transparent mediator in the communication between the environment and the ideal functionality. In our framework too this is true, but now the interaction also involves dealoffer and dealaccept messages.
- **$\mathcal{A}$'s resources in a hybrid model.** When working in $\mathcal{W}(\mathcal{F})$-hybrid model, the convention regarding bounding the resources of the adversary $\mathcal{A}$ needs special attention: any amount of resources that $\mathcal{A}$ sends as investment to $\mathcal{W}(\mathcal{F})$ gets counted towards its running time. That is, if $\mathcal{A}$ is a t-bounded IPRAM, then the total amount invested by it plus the total number of steps it runs is at most t.

2.4 Security and fairness definitions

So far, we have described the ideal world notion of fairness. As mentioned in Section 1.2, for a protocol to be resource-fair, for each real world adversary $\mathcal{A}$, the ideal world adversary $\mathcal{S}$ built to simulate the protocol should be such that the amount of resources $\mathcal{S}$ invests is not much more than that available $\mathcal{A}$. Below we shall quantify the resource fairness of a protocol by the ratio of the amount of resources that $\mathcal{S}$ invests to the actual resources available to $\mathcal{A}$ (which technically also includes those available to the environment).

The typical order of quantifiers in the simulation-based security definitions allows the ideal-world adversary to depend on the real-world adversary that it simulates, but it should be independent of the environment (i.e., $\forall\mathcal{A}\exists\mathcal{S}\forall\mathcal{Z}$). A stronger definition of security (which all current constructions in the UC framework satisfy) could require the ideal-world adversary to be a "black-box" simulator which depends on $\mathcal{A}$ only by making black-box invocations of $\mathcal{A}$. We employ a slight weakening of this definition: we pass $\mathcal{S}$ a bound t on the running times of $\mathcal{A}$ and $\mathcal{Z}$, as an input parameter. More formally we model $\mathcal{A}$ and $\mathcal{Z}$ as bounded IPRAMs. Our security definition will use the order of quantifiers $\exists\mathcal{S}$ $\forall t$-bounded $\mathcal{A}$ and $\mathcal{Z}$, and it will refer to $\mathcal{S}^{\mathcal{A}}(t)$. Now recall that we allow the ideal-world adversary to invest resources with an ideal functionality. An ideal-world adversary $\mathcal{S}$ with input parameter t (see above) is said to be λ-*restricted* if there is a polynomial $\zeta(\kappa)$ such that the sum of all investments sent by $\mathcal{S}$ to the ideal functionality is bounded by $\lambda t + \zeta(\kappa)$.

The definition of security and fairness using the simulator captures the intuitive requirements of these notions. However, this by itself does not give us universal composability. We shall strengthen the definition as described below to guarantee universal composition as well.

The full simulator. The strengthening is by requiring that (in addition to the security requirement above) there should a "full simulator" which can replace $\mathcal{A}$ *and the honest parties* running the protocol in the real world, without an environment being able to detect the change. We call it a full simulator because it simulates all of the execution of a session to the environment, in contrast to a simulator which does not control the honest parties. In this new scenario, since there are no more honest parties involved in the execution, there is no ideal functionality involved. Such a full simulation would be trivial, because the full simulator has access to all the inputs of $\mathcal{A}$ as well as of the honest parties, and it can simply execute the code of these parties in its simulation. The non-triviality comes from another requirement: the running time of full simulator should be bounded by a fixed polynomial, *independent of the resource-requests granted by* $\mathcal{Z}$.

To see why this is important for us, consider how the universal composition theorem is proven [14]. Out of the multiple sessions of a protocol in the composed setting, we can single out one session and consider all other sessions as part of the environment. The final simulator in the composed setting is obtained by considering (uncomposed) security for each session singled out in this way. However, in our case we must demonstrate an $O(\lambda)$-restricted simulator at the end, and for this we will have to ensure that *the environments considered are all* $O(t)$*-bounded.* (Here t and λ are the parameters from the security of a single session.) This becomes problematic when the honest parties in various sessions may spend more than $O(t)$ time in the panic mode (and hence their running time is *not* a polynomial in κ independent of t).[15]

If a full simulator exists, then the environment can use it to internally simulate a session. Then the time taken by the environment will indeed be independent of the amount of resources granted to

[15] One possible way around this problem that one might immediately think of (but is unsatisfactory) is to restrict the environment to granting resources comparable to t, but no more. However, such a restriction would severely restrict the usability of our framework: even when the adversary can afford to obtain its outputs (using less than t amount of resources), such a restriction will not allow the honest players to spend enough resources (which is more than t, in all our protocols) to get their outputs.

the honest parties being simulated. (See proof of Theorem 2.5 for more details.)

As we shall see, a full simulator is often easy to build. Since the environment has access to all the inputs to this internally simulated session, we can allow the full simulator also such access. Then, a full simulator can be built to simulate the protocol execution faithfully except for the panic mode. In simulating the panic mode, it will be able to directly obtain the results of the panic mode execution (without carrying out the extraction computation) since it knows all the inputs to all the parties.

We shall denote the random variable corresponding to the output produced by $\mathcal{Z}$ on interaction with a full simulator $\mathcal{X}$ by $\text{FSIM}_{\mathcal{X}^{\mathcal{A}},\mathcal{Z}}$.

Definition 2.2 (Securely Realizing Functionalities) *Let $\mathcal{W}_1$ and $\mathcal{W}_2$ be two functionalities. We say a protocol π* securely realizes *the functionality $\mathcal{W}_1$ in the $\mathcal{W}_2$-hybrid model if there exist an ideal world adversary $\mathcal{S}$ and a full simulator $\mathcal{X}$, such that for all t-bounded $\mathcal{A}$ and $\mathcal{Z}$*

1. $\text{HYB}^{\mathcal{W}_2}_{\rho,\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{W}_1,\mathcal{S}^{\mathcal{A}}(t),\mathcal{Z}}$, *and*
2. $\text{HYB}^{\mathcal{W}_2}_{\rho,\mathcal{A},\mathcal{Z}} \approx \text{FSIM}_{\mathcal{X}^{\mathcal{A}},\mathcal{Z}}$.

Furthermore, if $\mathcal{S}$ is λ-restricted, then π securely realizes $\mathcal{W}_1$ with λ-investment *(in the $\mathcal{W}_2$-hybrid model).*

Although the definition above is stated with respect to general functionalities (and this will be useful in proving our composition theorem), this notion of realizing a functionality with λ-investment will be particularly relevant in the case when $\mathcal{W}_1$ is a wrapped functionality, and specifically a wrapped "fair" functionality. To elaborate, let us consider the case where $\mathcal{W}_1$ is $\mathcal{W}(\mathcal{F})$ for some $\mathcal{F}$. (The functionality $\mathcal{W}_2$ can be a wrapped or non-wrapped functionality, i.e., $\mathcal{W}_2$ above can be a non-wrapped functionality like $\mathcal{F}_{\text{CRS}}$, or it can be a wrapped functionality which we use as a module in a larger protocol.) Then we make the following definition.

Definition 2.3 *Let π be a protocol that securely realizes $\mathcal{W}(\mathcal{F})$ with λ-investment. Then π* λ-fairly realizes *$\mathcal{F}$.*

Let us give some intuition behind this definition. First, by Fact 2.1, $\mathcal{W}$ guarantees that any time a corrupted party (or in particular, the ideal adversary that has corrupted that party) receives its fairdeliver message, then every honest party is at least offered a deal to receive its fairdeliver message, and this deal is bounded by the amount that the ideal adversary invests. Second, by the definition above, the ideal adversary invests an amount within a factor of λ to the resources available to the real adversary. Thus, by expending resources at most a factor λ more than the amount available to the real adversary, an honest party in the ideal world may obtain its message. Since the ideal world is indistinguishable from the real world, the honest party in the real world may also obtain the message expending that amount of resources.

To summarize, we use the term λ-fairly to denote "resource fairness" where an honest party may need to spend at most a factor of λ more resources (i.e., time) than an adversary in order to keep the fair deliveries "fair." Now we consider the case where $\mathcal{F}$ is in fact the fair SFE functionality $\mathcal{F}_f$, and formally define resource fairness and (standard) fairness.

Definition 2.4 *Let π be a protocol that securely realizes $\mathcal{W}(\mathcal{F}_f)$ with λ-investment. Then we say π is* λ-fair. *If $\lambda = O(n)$, then we say π is* resource fair, *and if $\lambda = 0$, then we say π is* fair.

Note that in a "fair" protocol, only a fixed polynomial investment is made by the ideal adversary, and thus all deals are bounded by a fixed polynomial. This could simply be incorporated into the protocol, and thus no deals would need to be made. Thus the protocol would actually securely realize $\mathcal{F}_f$. (Of course, as discussed above, if the adversary may corrupt more than a strict minority of parties, then no such protocol exists.)

On choosing $\lambda = O(n)$. The intuition behind the choice of $\lambda = O(n)$ for resource-fair protocols is as follows. As discussed before, since corrupted parties can abort and gain unfair advantage, an honest party needs more time to catch up. In the worst case, there can be $(n-1)$ corrupted parties against one honest party. Since the honest party may need to invest a certain amount of work against every corrupted party, we expect that the honest party would run about $(n-1)$ times as long as the adversary. Thus, we believe that $O(nt)$ is the "necessary" amount of time an honest party needs for a t-bounded adversary. On the other hand, as we show in the sequel, there exist $O(n)$-fair protocols in the FMPC framework, and thus $\lambda = O(n)$ is also sufficient.

Security of resource-fair protocols. Our definition of resource fairness subsumes the UC definition of security. First of all, if a protocol π λ-fairly realizes $\mathcal{F}$, then, by definition it is also a secure realization of $\mathcal{W}(\mathcal{F})$. However it is not a secure realization of $\mathcal{F}$ itself, because $\mathcal{W}(\mathcal{F})$ offers extra features. But note that for adversaries which never use the feature of sending an invest message, $\mathcal{F}$ and $\mathcal{W}(\mathcal{F})$ behave identically. In fact, $\mathcal{F}$ in the original (unfair) UC model of [15] can be modeled using a rigged wrapper: consider $\mathcal{W}'(\mathcal{F})$ which behaves like $\mathcal{W}(\mathcal{F})$ except that it does not offer any deals to the honest parties (but interacts with the adversary in the same way: in particular, it allows the adversary to obtain its outputs by "investing" any amount of resources). Except for the round structure we use, $\mathcal{W}'(\mathcal{F})$ is an exact modeling of $\mathcal{F}$ in the original UC framework. Clearly $\mathcal{W}(\mathcal{F})$, is intuitively as secure as $\mathcal{W}'(\mathcal{F})$ (but is also fair).

2.5 A composition theorem

We now examine the composition of protocols. It turns out that the composition theorem of the UC framework does not automatically imply an analog in the FMPC framework. The main reason for this is that the running time of a resource-requesting protocol is not bounded *a priori*, as there is no bound on the amount of time the environment may decide to grant it in response to a request. This is the reason we introduced the full simulator, whose running time *is* bounded by a polynomial, independent of the environment, and added the extra requirement concerning the full simulator in our definition of security. Using this extra requirement, we are able to prove the composition theorem below.

For simplicity, we shall modify Definition 2.2, so that the simulator $\mathcal{S}$ is passed t which is a bound on the *sum* of the running times of the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ (rather than on the maximum of these two). We state the composition theorem accordingly. This makes a difference of at most a constant factor in the parameters below.

Theorem 2.5 (Universal Composition of Resource-Fair Protocols) *Let $\mathcal{W}_2$ be an ideal functionality. Let π be a protocol in the $\mathcal{W}_2$-hybrid model, which uses atmost ℓ sessions of $\mathcal{W}_2$. Let ρ be a protocol that securely and λ-fairly realizes $\mathcal{W}_2$. Then there exists a λ'-restricted black-box hybrid-mode adversary $\mathcal{H}$, such that for all t, for any t_1-bounded real-world adversary $\mathcal{A}$ and t_2-bounded environment $\mathcal{Z}$ such that $t_1 + t_2 \leq t$, we have*

$$\mathrm{REAL}_{\pi^\rho,\mathcal{A},\mathcal{Z}} \approx \mathrm{HYB}^{\mathcal{W}_2}_{\pi,\mathcal{H}^{\mathcal{A}}(t),\mathcal{Z}}, \tag{1}$$

where $\lambda' = \lambda\ell$.

Proof sketch: The proof follows the same outline as the proof of the composition theorem in the UC framework. Consider any environment $\mathcal{Z}$ and adversary $\mathcal{A}$ (t_1, t_2-bounded, such that $t_1 + t_2 \leq t$). We start from the real-world scenario in which π invokes (up to ℓ) copies of the protocol ρ. Through a series of hybrids, one by one, we replace the ℓ protocol executions by copies of the functionality $\mathcal{W}_2$.

As we move through the hybrids we modify the adversary; the final one will be the adversary $\mathcal{H}$. At every step we shall show that the output of the environment remains essentially unchanged.

In the first scenario which we begin with, we shall consider a "dummy adversary" $\tilde{\mathcal{A}}$ which transparently acts between the environment and the parties in a protocol. This amounts to considering the real-world adversary as essentially a part of the environment $\mathcal{Z}$. Further, we shall consider π also as part of the environment; then the parties running ρ interact directly with the environment, and not with the protocol π. We shall denote this larger environment (which contains the original environment $\mathcal{Z}$, the adversary $\mathcal{A}$, and π) by $\mathcal{Z}^*$. Restricting to $\tilde{\mathcal{A}}$ is w.l.o.g, except that $\mathcal{Z}^*$ is not t-bounded, but t'-bounded where $t' = t + |\pi|$ (here we use the IPRAM model, to bound the total running time of $\mathcal{Z}^*$ by the sum of the running times of its components).

For this proof sketch, we introduce an informal shorthand to denote parts of the entire system, obtained by grouping together various components within $\langle \ \cdot \ \rangle$. For example, for the above description of $\mathcal{Z}^*$ we can write $\mathcal{Z}^* = \langle \mathcal{Z}, \mathcal{A}, \pi \rangle$ (which is not very precise, as a part of $\mathcal{A}$—namely, $\tilde{\mathcal{A}}$—is left out of $\mathcal{Z}^*$). Then, the entire real-world scenario consisting of the environment $\mathcal{Z}^*$ and the ℓ copies of ρ can be represented by $\langle \mathcal{Z}^*, [\rho|\tilde{\mathcal{A}}]_1, \cdots, [\rho|\tilde{\mathcal{A}}]_\ell \rangle$. Here $[\rho|\tilde{\mathcal{A}}]_i$ denotes the i-th session of the protocol ρ, and the dummy adversary $\tilde{\mathcal{A}}$ which is the intermediary between $\mathcal{Z}^*$ and that session. Note that the above notation describes the same system as in $\text{REAL}_{\pi^\rho, \mathcal{A}, \mathcal{Z}}$.

Let $\mathcal{S}$ and $\mathcal{X}$ denote, respectively, the simulator and the full simulator guaranteed by the definition of security of ρ. Analogous to $[\rho|\tilde{\mathcal{A}}]$, we shall denote by $[\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]$ the functionality $\mathcal{W}_2$ and the simulator $\mathcal{S}^{\tilde{\mathcal{A}}}(t')$. Then, if $\ell = 1$, by definition of $\mathcal{S}$, we would have that the outputs (by $\mathcal{Z}^*$) of the two systems $\langle \mathcal{Z}^*, [\rho|\tilde{\mathcal{A}}]_1 \rangle$ and $\langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1 \rangle$ are indistinguishable. For larger ℓ we shall argue below that $\langle \mathcal{Z}^*, [\rho|\tilde{\mathcal{A}}]_1, \cdots, [\rho|\tilde{\mathcal{A}}]_\ell \rangle$ and $\langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1, \cdots, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_\ell \rangle$ are indistinguishable. Then, the adversary $\mathcal{H}$ is simply the collection of all ℓ copies of $\mathcal{S}$ (with some simple syntactic changes so that $\mathcal{A}$ is attached to $\mathcal{H}$ instead of to $\mathcal{Z}^*$). Note that then the system $\langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1, \cdots, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_\ell \rangle$ is identical to the system in $\text{HYB}^{\mathcal{W}_2}_{\pi, \mathcal{H}^{\mathcal{A}}(t), \mathcal{Z}}$.

The description of the proof so far follows the proof in the UC framework exactly. However, the argument to establish $\langle \mathcal{Z}^*, [\rho|\tilde{\mathcal{A}}]_1, \cdots, [\rho|\tilde{\mathcal{A}}]_\ell \rangle \approx \langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1, \cdots, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_\ell \rangle$ will be different for us (and will require the use of the full simulator $\mathcal{X}$). First we describe how the argument in the UC framework breaks down in our case, with $\ell = 2$. As the first step in a hybrid argument, we try to go from $\langle \mathcal{Z}^*, [\rho|\tilde{\mathcal{A}}]_1, [\rho|\tilde{\mathcal{A}}]_2 \rangle$ to $\langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1, [\rho|\tilde{\mathcal{A}}]_2 \rangle$, using the guarantee on $\mathcal{S}$. For this we create an environment $\mathcal{Z}'$ consisting of $\mathcal{Z}^*$ and $[\rho|\tilde{\mathcal{A}}]_2$ (denoted as $\mathcal{Z}' = \langle \mathcal{Z}^*, [\rho|\tilde{\mathcal{A}}]_2 \rangle$). Then we would have $\langle \mathcal{Z}^*, [\rho|\tilde{\mathcal{A}}]_1, [\rho|\tilde{\mathcal{A}}]_2 \rangle = \langle \mathcal{Z}', [\rho|\tilde{\mathcal{A}}]_1 \rangle$ and $\langle \mathcal{Z}', [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1 \rangle = \langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1, [\rho|\tilde{\mathcal{A}}]_2 \rangle$. So to complete the step we just need to argue that $\langle \mathcal{Z}', [\rho|\tilde{\mathcal{A}}]_1 \rangle \approx \langle \mathcal{Z}', [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1 \rangle$. This should follow from the definition of $\mathcal{S}$. But in fact, at this point the proof breaks down. This is because the new environment that we created, $\mathcal{Z}'$ is not necessarily $O(t)$-bounded: the part $[\rho|\tilde{\mathcal{A}}]_2$ inside $\mathcal{Z}'$ may run for much longer than t if $\mathcal{Z}^*$ grants it a request to that effect. This will prevent the resulting simulator from being $O(\lambda)$ restricted. (To remedy the situation, one might consider restricting $\mathcal{Z}^*$ to grant only requests comparable to t, but this would seriously restrict the usability of the FMPC framework.)

It is here that the full simulator is useful. Simply stated, the plan will be to have $[\rho|\tilde{\mathcal{A}}]_2$ above first substituted by a copy of $\mathcal{X}$, $[\mathcal{X}^{\tilde{\mathcal{A}}}]_2$. The more formal line of argument is informally sketched below.

$$
\begin{aligned}
\langle \mathcal{Z}^*, [\rho|\tilde{\mathcal{A}}]_1, \cdots, [\rho|\tilde{\mathcal{A}}]_\ell \rangle &= \langle \mathcal{Z}'_1, [\rho|\tilde{\mathcal{A}}]_1 \rangle && \mathcal{Z}'_1 = \langle \mathcal{Z}^*, [\rho|\tilde{\mathcal{A}}]_2, \cdots, [\rho|\tilde{\mathcal{A}}]_\ell \rangle && (2)\\
&\approx \langle \mathcal{Z}'_1, [\mathcal{X}^{\tilde{\mathcal{A}}}]_1 \rangle && \text{because } \mathcal{X} \text{ is a full-simulator} && (3)\\
&= \langle \mathcal{Z}^*, [\mathcal{X}^{\tilde{\mathcal{A}}}]_1, [\rho|\tilde{\mathcal{A}}]_2, \cdots, [\rho|\tilde{\mathcal{A}}]_\ell \rangle && \text{expanding } \mathcal{Z}'_1 && (4)\\
&\approx \langle \mathcal{Z}^*, [\mathcal{X}^{\tilde{\mathcal{A}}}]_1, \cdots, [\mathcal{X}^{\tilde{\mathcal{A}}}]_\ell \rangle && \text{similarly, in } \ell \text{ steps} && (5)\\
&= \langle \mathcal{Z}''_1, [\mathcal{X}^{\tilde{\mathcal{A}}}]_1 \rangle && \mathcal{Z}''_1 = \langle \mathcal{Z}^*, [\mathcal{X}^{\tilde{\mathcal{A}}}]_2, \cdots, [\mathcal{X}^{\tilde{\mathcal{A}}}]_\ell \rangle && (6)\\
&\approx \langle \mathcal{Z}''_1, [\rho|\tilde{\mathcal{A}}]_1 \rangle && \text{because } \mathcal{X} \text{ is a full-simulator} && (7)\\
&\approx \langle \mathcal{Z}''_1, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1 \rangle && \text{because } \mathcal{S} \text{ is a simulator} && (8)\\
&= \langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1, [\mathcal{X}^{\tilde{\mathcal{A}}}]_2, \cdots, [\mathcal{X}^{\tilde{\mathcal{A}}}]_\ell \rangle && \text{expanding } \mathcal{Z}''_1 && (9)\\
&\approx \langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1, \cdots, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_\ell \rangle && \text{similarly, in } \ell \text{ steps} && (10)
\end{aligned}
$$

Above in step (5), we repeat the steps (2), (3) and (4), $\ell-1$ more times. In the i-th repetition $[\rho|\tilde{\mathcal{A}}]_i$ is replaced by $[\mathcal{X}^{\tilde{\mathcal{A}}}]_i$. The argument is the same in all the iterations, except that in the i-th iteration, we use $\mathcal{Z}'_i = \langle \mathcal{Z}^*, [\mathcal{X}^{\tilde{\mathcal{A}}}]_1, \cdots, [\mathcal{X}^{\tilde{\mathcal{A}}}]_{1-1}, [\rho|\tilde{\mathcal{A}}]_{i+1}, \cdots, [\rho|\tilde{\mathcal{A}}]_\ell \rangle$. Also, step (10) repeats steps (6)-(9), for $i = 2, \ldots, \ell$; at the i-th iteration, we use $\mathcal{Z}''_i = \langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1, \cdots, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_{1-1}, [\mathcal{X}^{\tilde{\mathcal{A}}}]_{i+1}, \cdots, [\mathcal{X}^{\tilde{\mathcal{A}}}]_\ell \rangle$.

What we gain from using the full simulator is that in step (8), the environment $\mathcal{Z}''_i$ is $O(t)$-bounded. On the other hand, note that in step (3), though we use an environment $\mathcal{Z}'_i$ which may not be $O(t)$-bounded, $\mathcal{X}^{\tilde{\mathcal{A}}}$ is independent of $\mathcal{Z}'_i$.

As mentioned earlier, setting $\mathcal{H} = \langle [\mathcal{S}^{\tilde{\mathcal{A}}}]_1, \cdots, [\mathcal{S}^{\tilde{\mathcal{A}}}]_\ell$, and expanding $\mathcal{Z}^*$ as $\langle \mathcal{Z}, \mathcal{A}, \pi \rangle$, we obtain that $\langle \mathcal{Z}^*, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_1, \cdots, [\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_\ell \rangle$ is identical to the system in $\text{HYB}^{\mathcal{W}_2}_{\pi, \mathcal{H}^{\mathcal{A}}(t), \mathcal{Z}}$ (t being $|\mathcal{Z}^*| - |\pi|$). This completes the argument that $\text{REAL}_{\pi^\rho, \mathcal{A}, \mathcal{Z}} \approx \text{HYB}^{\mathcal{W}_2}_{\pi, \mathcal{H}^{\mathcal{A}}(t), \mathcal{Z}}$.

To complete the proof we will need to show that $\mathcal{H}$ is λ'-restricted for $\lambda' = \lambda\ell$. Note that total investment made by $\mathcal{H}$ is the sum of investments made in $\{[\mathcal{W}_2|\mathcal{S}_t^{\tilde{\mathcal{A}}}]_i\}_{i=1}^{\ell}$, which is bounded by $\sum_{i=1}^{\ell} \lambda|\mathcal{Z}''_i|$. Further, $|\mathcal{Z}''_i| \leq |\mathcal{Z}^*| + \ell(|\mathcal{S}| + |\tilde{\mathcal{A}}| + |\mathcal{W}_2|) \leq |\mathcal{Z}| + poly(\kappa)$. The last inequality above follows from the fact that $|\pi|$, ℓ, $|\mathcal{S}|$, $|\tilde{\mathcal{A}}|$, $|\mathcal{W}_2|$ are all bounded by polynomials in κ (independent of $\mathcal{A}$ and $\mathcal{Z}$). Thus, the total investments made by $\mathcal{H}$ is at most $\ell\lambda|\mathcal{Z}| + poly'(\kappa)$ (where $poly'$ is independent of $\mathcal{Z}$ and $\mathcal{A}$). Thus $\mathcal{H}$ is indeed $\ell\lambda$-restricted. ▌

Corollary 2.6 *Let $\mathcal{W}_1$ and $\mathcal{W}_2$ be ideal functionalities. Let π be a protocol that securely realizes $\mathcal{W}_1$ with λ-investment in the $\mathcal{W}_2$-hybrid model. Let ρ be a protocol that securely realizes $\mathcal{W}_2$ with λ'-investment. Then the protocol π^ρ securely realizes $\mathcal{W}_1$ with λ''-investment. Here, if ℓ is an upperbound on the number of sessions of $\mathcal{W}_2$ used by π, then $\lambda'' = \lambda(\ell(\lambda' + 1))$.*

Proof sketch: As in the UC framework, this corollary follows from the composition theorem using a simple hybrid argument. However, now we keep track of the total running time of the simulator as well as the total amount of resources it invests, and also ensure that the condition of the full simulator is satisfied.

To show the first condition in the security definition, we need to show a λ''-restricted simulator $\mathcal{S}$ for the protocol π^ρ. We can use Theorem 2.5 to obtain $\mathcal{H}$ such that $\text{REAL}_{\pi^\rho, \mathcal{A}, \mathcal{Z}} \approx \text{HYB}^{\mathcal{W}_2}_{\pi, \mathcal{H}^{\mathcal{A}}(t), \mathcal{Z}}$, where $|\mathcal{Z}| + |\mathcal{A}| \leq t$. The running time of $\mathcal{H}^{\mathcal{A}}(t)$ is at most $\ell|\mathcal{A}| + c_1$ (where c_1 is some constant independent of t). Further, $\mathcal{H}$ is $\lambda'\ell$-restricted, which means that it invests at most $\lambda'\ell t + c_2$ units. Thus $|\mathcal{Z}| + |\mathcal{H}^{\mathcal{A}}(t)| \leq \ell t + \lambda'\ell t + c_3$ (counting the total running time *and* the total amount of resource

invested). Now, we apply the security guarantee of π in the $\mathcal{W}_2$-hybrid model to obtain $\mathcal{S}$ such that $\text{HYB}^{\mathcal{W}_2}_{\pi,\mathcal{H}^{\mathcal{A}}(t),\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{W}_1,\mathcal{S}^{\mathcal{H}^{\mathcal{A}}(t)}(t'),\mathcal{Z}}$, where $t' = \ell t + \lambda'\ell t + c_3$. Further, the total investment made by $\mathcal{S}^{\mathcal{H}^{\mathcal{A}}(t)}(t')$ is bounded by $\lambda t' = \lambda(\ell(\lambda'+1))t + c_4$.

The only remaining point to prove is the existence of a full simulator for π^ρ. This is done as follows: first one by one each copy of ρ is replaced by a full simulator, by considering everything else as the environment (and using a dummy adversary). Then, in the resulting system everything except π and the (original) adversary is considered as the environment, and the existence of a full simulator for π in the $\mathcal{W}_2$-hybrid model is invoked. Note that at every step of this sequence of operations we ensure that the constructed adversary is of constant size, except in the last step, when it contains the original adversary. This guarantees that the full simulator obtained in the end is a constant size one with black-box access to the original adversary. ▌

3 Preliminaries for Protocol Constructions

In the following sections we present various protocols which are shown to be secure and resource-fair realizations of different functionalities. In this section we detail a few preliminary definitions and the number-theoretic assumptions used in these constructions.

Let κ be the cryptographic security parameter. A function $f : \mathbb{Z} \to [0,1]$ is negligible if for all $\alpha > 0$ there exists an $\kappa_\alpha > 0$ such that for all $\kappa > \kappa_\alpha$, $f(\kappa) < |\kappa|^{-\alpha}$. All functions we use in this paper will include a security parameter as input, either implicitly or explicitly, and we say that these functions are negligible if they are negligible in the security parameter. (They will be polynomial in all other parameters.) Furthermore, we assume that n, the number of parties, is polynomially bounded by κ as well.

A prime p is *safe* if $p' = (p-1)/2$ is also a prime (in number theory, p' is also known as a Sophie-Germain prime). A Blum integer is a product of two primes, each equivalent to 3 modulo 4. We will be working with a special class of Blum integers $N = p_1 p_2$ where p_1 and p_2 are both safe primes. We call such numbers *safe Blum integers*.[16]

We now state the assumptions used in this paper: the *composite decisional Diffie-Hellman* assumption (CDDH), the *decision composite residuosity* assumption (DCRA), and the *generalized Blum-Blum-Shub* assumption (GBBS) (in fact, a refined version of it); readers familiar with these assumptions are invited to proceed to Section 4.

The CDDH assumption. We briefly review the *composite decisional Diffie-Hellman* (CDDH) assumption. (We refer the reader to [10] for more in-depth discussions.) Let $N = p_1 p_2$ where p_1, p_2 are κ-bit safe primes. Let g be a random element from $\mathbb{Z}_N^*$, a, b, c random elements in $\mathbb{Z}_N$, and $\mathcal{A}$ a polynomial-time adversary. There exists a negligible function $\epsilon(\cdot)$, such that

$$\left|\Pr[\mathcal{A}(N, g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(N, g, g^a, g^b, g^c) = 1]\right| \leq \epsilon(\kappa)$$

where the randomness is taken over the random choices of N, g, a, b and c. In this paper, we will use a slight variation of this assumption, where instead of being a random element in $\mathbb{Z}_N^*$, g is a random quadratic residue in $\mathbb{Z}_N^*$. We call the new assumption CDDH-QR. Notice that CDDH-QR easily reduces to CDDH. To see this, given a random tuple (N, g, x, y, z) from the CDDH assumption, (N, g^2, x^2, y^2, z^2) is (statistically close to) a random tuple in the CDDH-QR assumption.

[16] Integers that are the product of two equally-sized safe primes have also been called *rigid integers* [5].

The DCRA assumption. The Paillier encryption scheme [50] is defined as follows, where $\lambda(N)$ is the Carmichael function of N, and L is a function that takes input elements from the set $\{u < N^2 | u \equiv 1 \bmod N\}$ and returns $L(u) = \frac{u-1}{N}$. This definition differs from that in [50] only in that we define the message space for public key $pk = \langle N, g\rangle$ as $[-(N-1)/2, (N-1)/2]$ (versus $\mathbb{Z}_N$ in [50]), and we restrict h to be $1+N$. The security of this cryptosystem relies on the *Decision Composite Residuosity Assumption*, DCRA.

For key generation, choose random $k/2$-bit primes p, q, set $N = pq$, and set $h \leftarrow 1+N$. The public key is $\langle N, h\rangle$ and the private key is $\langle N, h, \lambda(N)\rangle$. To encrypt a message m with public key $\langle N, h\rangle$, select a random $\alpha \in \mathbb{Z}_N^*$ and compute $c \leftarrow g^m \alpha^N \bmod N^2$. To decrypt a ciphertext c with secret key $\langle N, h, \lambda(N)\rangle$, compute $m = \frac{L(c^{\lambda(N)} \bmod N^2)}{L(g^{\lambda(N)} \bmod N^2)} \bmod N$, and the decryption is m if $m \leq (N-1)/2$, and otherwise the decryption is $m - N$. Paillier [50] shows that both $c^{\lambda(N)} \bmod N^2$ and $g^{\lambda(N)} \bmod N^2$ are elements of the form $(1+N)^d \equiv_{N^2} 1 + dN$, and thus the L function can be easily computed for decryption.

The (new) generalized BBS assumption. In this paper we use a further refinement of the *generalized BBS* assumption (GBBS), introduced by Boneh and Naor [11]; see Appendix B for remarks on the differences between the current formulation and the original one.

Given security parameter κ, let $N = p_1 p_2$ be a safe Blum integer with $|p_1| = |p_2| = \kappa$, and let k be an integer bounded from below by κ^c for some positive c. Let $\vec{a}$ be an arbitrary ℓ-dimensional vector where $0 = a[1] < a[2] < \cdots < a[\ell] < 2^k$, and x be an integer between 0 and 2^k such that $\mathsf{Dist}(x, \vec{a}) = S$, where $\mathsf{Dist}(x, \vec{a})$ denotes the minimal absolute difference between x and elements in $\vec{a}$. (Note that, in particular, we have $x \geq S$, since $a[1] = 0$.) Let g be a random element in $\mathbb{Z}_N^*$; define the "repeated squaring" function as $\mathsf{RepSq}_{N,g}(x) = g^{2^x} \bmod N$. Let $\vec{u}$ be an ℓ-dimensional vector such that $u[i] = \mathsf{RepSq}_{N,g}(a[i])$, for $i = 1, ..., \ell$.

Now let $\mathcal{A}$ be a PRAM algorithm whose running time is bounded by $\delta \cdot S$ for some constant δ, and let R be a random element in $\mathbb{Z}_N^*$. The GBBS assumption states that there exists a negligible function $\epsilon(\kappa)$ such that for any $\mathcal{A}$,

$$\left|\Pr[\mathcal{A}(N, g, \vec{a}, \vec{u}, x, \mathsf{RepSq}_{N,g}(x)) = 1] - \Pr[\mathcal{A}(N, g, \vec{a}, \vec{u}, x, R^2) = 1]\right| \leq \epsilon(\kappa). \tag{11}$$

Intuitively, the assumption says that for any adversary $\mathcal{A}$ whose running time is bounded by $\delta \cdot S$, and who sees a collection of ℓ points on a "time-line" (formal definition in Section 4.1) with an arbitrary distribution, a point at distance S away from this collection is still not only unknown, but appears pseudorandom.

4 The Commit-Prove-Fair-Open Functionality

In this section we first present the "commit-prove-fair-open" functionality $\mathcal{F}_{\text{CPFO}}$, and then show how to construct a protocol, $\mathsf{GradRel}$, that securely realizes $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ with $O(n)$-investment using "time-lines." Functionality $\mathcal{F}_{\text{CPFO}}$ is described below.

Functionality $\mathcal{F}_{\text{CPFO}}$ is similar to the "commit-and-prove" functionality $\mathcal{F}_{\text{CP}}$ in [18] in that both functionalities allow a party to commit to a value v and prove relations about v. Note that although $\mathcal{F}_{\text{CP}}$ does not provide an explicit "opening" phase, the opening of v can be achieved by proving an "equality" relation. However, while $\mathcal{F}_{\text{CP}}$ is not concerned with fairness, $\mathcal{F}_{\text{CPFO}}$ is specifically designed to enforce fairness in the opening. In the open phase, $\mathcal{F}_{\text{CPFO}}$ does not require the outputs to be handed over to the parties as soon as the parties request an opening. Instead, it specifies (to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$) a round s in the future when the outputs are to be handed over. We allow the adversary to determine this round by sending a $\mathsf{deliverat}$ message to $\mathcal{F}_{\text{CPFO}}$. (Implicitly we assume that if the

Functionality $\mathcal{F}_{\text{CPFO}}^{R}$

$\mathcal{F}_{\text{CPFO}}^{R}$ is parameterized by a polynomial-time computable binary relation R. It proceeds as follows, running with parties P_1, P_2, ..., P_n and an adversary $\mathcal{S}$.

Round 1 – **commit phase:** Receive message $(\mathsf{commit}, sid, x_i)$ from every party P_i and broadcast $(\mathsf{RECEIPT}, sid, P_i)$ to all parties and $\mathcal{S}$.

Round 2 – **prove phase:** Receive message $(\mathsf{prove}, sid, y_i)$ from every party P_i, and if $R(y_i, x_i) = 1$, broadcast $(\mathsf{PROOF}, sid, P_i, y_i)$ to all parties and $\mathcal{S}$.

Oopen phase: Wait to receive message (open, sid) from party P_i, $1 \leq i \leq n$, and a message $(\mathsf{deliverat}, sid, s)$ from $\mathcal{S}$. As soon as all n open messages and the $\mathsf{deliverat}$ message are received, output $(\mathsf{fairdeliver}, sid, 0, \{((\mathsf{DATA}, (x_1, x_2, ..., x_n)), P_i)\}_{1 \leq i \leq n} \cup \{((\mathsf{DATA}, (x_1, x_2, ..., x_n)), \mathcal{S})\}, s)$.

Figure 3: *The commit-prove-fair-open functionality $\mathcal{F}_{\text{CPFO}}$ with relation R.*

round number in the $\mathsf{deliverat}$ message is less than the current round number, then the functionality will ignore it.)

Later in the paper, we shall see that by replacing some invocations to the $\mathcal{F}_{\text{CP}}$ functionality by invocations to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, we can convert the MPC protocol by Canetti *et al.* (which is completely unfair) into a resource-fair protocol.

Before showing a protocol that securely realizes $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, we present a variant of a cryptographic primitive known as "time-lines" [34] that will play an essential role in the construction of resource-fair protocols.

4.1 Time-lines

We start with some additional notation. We use QR_N to denote the quadratic residues modulo N. In other words, $\mathsf{QR}_N = \{x^2 \mid x \in \mathbb{Z}_N\}$. For a vector $\vec{a}$, we use $a[i]$ to denote the ith element in A. As defined in Section 3, the *distance* between a number x and a vector $\vec{a}$ is the minimal absolute difference between x and elements in $\vec{a}$; we denote this as $\mathsf{Dist}(x, \vec{a})$. More formally, assuming that d is the dimension of $\vec{a}$, we have $\mathsf{Dist}(x, \vec{a}) = \min_{i=1}^{d}\{|x - a[i]|\}$. Also, recall the "repeated squaring" function $\mathsf{RepSq}_{N,g}(x) = g^{2^x} \bmod N$.

We now present a definition of a time-line suitable for our purposes, followed by an efficient way to generate them (according to this definition), the security of which relies on GBBS and CDDH-QR.

Definition 4.1 *Let κ be a security parameter. A* decreasing time-line *is a tuple $L = \langle N, g, \vec{u}\rangle$, where $N = p_1 p_2$ is a safe Blum integer where both p_1 and p_2 are κ-bit safe primes, g is an element in $\mathbb{Z}_N^*$, and $\vec{u}$ is a κ-dimensional vector defined as $u[i] = \mathsf{RepSq}_{N,g}(2^\kappa - 2^{\kappa - i})$ for $i = 1, 2, ..., \kappa$. We call N the time-line* modulus*, g the* seed*, the elements of $\vec{u}$ the* points *in L, and $u[\kappa]$ the* end point *in L.*

In the rest of the paper, we will sometimes call a decreasing time-line simply a "time-line."

To randomly generate a time-line, one picks a random safe Blum integer N along with $g \xleftarrow{R} \mathbb{Z}_N^*$ as the seed, and then produces the points. Naturally, one can compute the points by repeated squaring: By squaring the seed g $2^{\kappa-1}$ times, we get $u[1]$, and from then on, we can compute $u[i]$ by squaring $u[i-1]$; it is not hard to verify that $u[i] = \mathsf{RepSq}_{N,u[i-1]}(2^{\kappa-i})$, for $i = 2, ..., \kappa$. Obviously, using this method to compute all the points would take exponential time. However, if one knows the factorization of N, then the time-line can be efficiently computed [11].

Alternatively, and assuming one time-line is already known, Garay and Jakobsson [34] suggested the following way to efficiently generate additional time-lines. Given a time-line L, one can easily *derive* a new time-line from L, by raising the seed and every point in L to a fixed power α. Clearly, the result is a time-line with the same modulus.

Definition 4.2 *Let $L = \langle N, g, \vec{u}\rangle$ and $L' = \langle N, h, \vec{v}\rangle$ be two lines of identical modulus. We say that time-line L' is* derived from L with shifting factor α *if there exists an $\alpha \in \mathbb{Z}_{[1,\frac{N-1}{2}]}$ such that $h = g^\alpha \bmod N$. We call L the* master *time-line.*

Note that the cost of derivation is just one exponentiation per point, and there is no need to know the factorization of N.

Given a master time-line $L = \langle N, g, \vec{u}\rangle$, there are two methods for computing the "next point" in the derived time-line L'. First, if one knows the shifting factor α, one can simply raise the corresponding point in L to the α-th power — we call this the "deriving method." If α is not known, since L' is a time-line, one can still compute $v[i]$ by repeated squaring $v[i-1]$ for $2^{\kappa-i}$ times — we call this the "squaring method." This is illustrated in Figure 4. Clearly, the deriving method is more efficient than the squaring method, especially at the beginning of the time-line, where the squaring method would take exponential time.

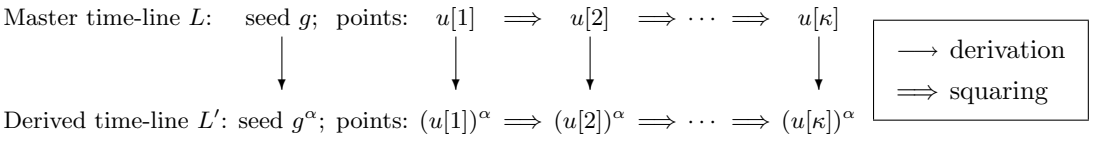

Figure 4: *Computing the next point of a derived time-line.*

In fact, without knowing the master time-line L, if an adversary $\mathcal{A}$ of running time $\delta \cdot 2^\ell$ sees only the seed and the *last* $(\ell+1)$ points of a derived time-line L', the previous point (which is at distance 2^ℓ away) appears pseudorandom to $\mathcal{A}$, assuming that the GBBS assumption holds. Obviously, this pseudorandomness is no longer true if $\mathcal{A}$ also knows the entire master time-line L and the shifting factor α, since it can then use the deriving method to find the previous point (in fact, any point) on L' efficiently. Nevertheless, as we show in the following lemma, assuming CDDH and GBBS, this pseudorandomness remains true if $\mathcal{A}$ knows L, but not the shifting factor α. We call this property the *strong pseudorandomness* of time-lines.[17]

Lemma 4.3 (Strong Pseudorandomness) *Let $L = \langle N, g, \vec{u}\rangle$ be a randomly generated decreasing time-line and $L' = \langle N, h, \vec{v}\rangle$ be a time-line derived from L with random shifting factor α. Let κ and δ be as in the GBBS assumption. Let $\vec{w}$ be the vector containing the last $(\ell+1)$ elements in $\vec{v}$, i.e., $\vec{w} = (v[\kappa-\ell], v[\kappa-\ell+1], ..., v[\kappa])$. Let $\mathcal{A}$ be a PRAM algorithm whose running time is bounded by $\delta \cdot 2^\ell$ for some constant δ. Let R be a random element in $\mathbb{Z}_N^*$. Then, assuming CDDH and GBBS hold, there exists a negligible function $\epsilon(\cdot)$ such that, for any $\mathcal{A}$,*

$$\left|\Pr[\mathcal{A}(N, g, \vec{u}, h, \vec{w}, v[\kappa-\ell-1]) = 1] - \Pr[\mathcal{A}(N, g, \vec{u}, h, \vec{w}, R^2) = 1]\right| \leq \epsilon(\kappa). \tag{12}$$

The proof of this lemma appears in Appendix C.

[17] For convenience, we state the lemma for the specific case of decreasing time-lines. But it is easy to see that it holds for other distributions of points.

4.2 Realizing $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$: Protocol GradRel

Now we construct a protocol, GradRel, that securely realizes wrapped functionality $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ in the $(\mathcal{F}_{\text{CRS}}, \hat{\mathcal{F}}_{\text{ZK}})$-hybrid model using the time-lines from the previous section.[18] We use the multi-session version of the "one-to-many" $\hat{\mathcal{F}}_{\text{ZK}}$ functionality from [18], which is shown in Figure 5.[19] In particular, we need the $\hat{\mathcal{F}}_{\text{ZK}}$ functionality for the following relations.

Functionality $\hat{\mathcal{F}}^{R}_{\text{ZK}}$

$\hat{\mathcal{F}}^{R}_{\text{ZK}}$ proceeds as follows, running parties $P_1, \ldots, P_n$, and an adversary $\mathcal{S}$:

- Upon receiving $(\mathsf{zk\text{-}prove}, sid, ssid, x, w)$ from P_i: If $R(x, w)$ does not hold, ignore. Otherwise, request $\mathcal{S}$ for permission to send $(\mathsf{ZK\text{-}PROOF}, sid, ssid, P_i, x)$ to each of P_j $(j \neq i)$. Send the messages as permissions are granted.

Figure 5: *The (multi-session) zero-knowledge functionality for relation R.*

Discrete log: $\mathsf{DL} = \{((M, g, h), \alpha) \mid h = g^{\alpha} \bmod M\}$

Diffie-Hellman quadruple: $\mathsf{DH} = \{((M, g, h, x, y), \alpha) \mid h = g^{\alpha} \bmod M \ \wedge \ y = x^{\alpha} \bmod M\}$

Blinded relation: Given a binary relation $R(y, x)$, we define a "blinded" relation $\hat{R}$ as follows:

$$\hat{R}((M, g, h, w, z, y), \alpha) = (h = g^{\alpha} \bmod M) \ \wedge \ R(y, z/w^{\alpha} \bmod M)$$

Intuitively, $\hat{R}$ "blinds" the witness x using the Diffie-Hellman tuple $(g, h, w, z/x)$. Obviously $\hat{R}$ is an NP relation if R is.

We now describe protocol GradRel informally. The CRS in GradRel consists of a master time-line $L = \langle N, g, \vec{u} \rangle$. To commit to a value x_i, party P_i derives a new time-line $L_i = \langle N, g_i, \vec{v_i} \rangle$, and uses the tail of L_i to "blind" x_i. More precisely, P_i sends $z_i = v_i[\kappa] \cdot x_i$ as a "timeline-commitment" to x_i together with a zero-knowledge proof of knowledge (through $\hat{\mathcal{F}}^{\mathsf{DL}}_{\text{ZK}}$) that it knows L_i's shifting factor, and thus, x_i. Note that any party can *force-open* the commitment by performing repeated squaring from points in the time-line. However, forced opening can take a long time, and in particular, since $v_i[\kappa]$ is $(2^{\kappa} - 1)$ steps away from the seed g_i, it appears pseudorandom to the adversary.

The prove phase is directly handled by the $\hat{\mathcal{F}}^{\hat{R}}_{\text{ZK}}$ functionality. The opening phase consists of κ rounds. In the i-th round, all parties reveal the ith point in their derived time-lines, followed by a zero-knowledge proof that this point is valid (through $\hat{\mathcal{F}}^{\mathsf{DH}}_{\text{ZK}}$), for $i = 1, 2, ...\kappa$. If at any time in the gradual opening stage, an uncorrupted party does not receive a ZK-PROOF message in a round when it is expected (possibly because the adversary blocked it, or a corrupted party did not send a proper zk-prove message to an $\hat{\mathcal{F}}_{\text{ZK}}$ functionality) then it enters the *panic mode*. In this mode, an uncorrupted party requests time from the environment to force-open the commitments of all other parties. If the environment accepts, the party forces-open the commitment; otherwise it aborts.

The detailed description of the protocol is given in Figure 6. The security of this protocol is based on CDDH, DCRA, and GBBS. The δ in the protocol is the constant δ from the GBBS assumption. As a technical note, GradRel assumes that all the committed values are quadratic residues in $\mathbb{Z}^*_N$. We discuss in Appendix D how this assumption can be removed. Clearly, protocol GradRel uses $O(\kappa^2 n)$ bits of communication. As mentioned in Section 2.1, the protocol employs a broadcast channel for convenience.

[18] See Appendix A for a description of the $\mathcal{F}_{\text{CRS}}$ functionality.

[19] In [18] the framework used is that originally presented in [15]. However, since we are using the modified version from [16], we modify the functionality $\hat{\mathcal{F}}_{\text{ZK}}$ by explicitly allowing the adversary to block messages from the functionality to the parties.

Protocol $\mathsf{GradRel}^R$

Set-up: The CRS consists of a master time-line $L = \langle N, g, \vec{u} \rangle$.

Round 1 **(commit phase)** For each party P_i, $1 \leq i \leq n$, upon receiving input $(\mathsf{commit}, sid, x_i)$, do:

1. Pick $\alpha_i \xleftarrow{R} [1, \frac{N-1}{2}]$, set $g_i \leftarrow g^{\alpha_i} \bmod N$, and compute from L a derived time-line $L_i = \langle N, g_i, \vec{v_i} \rangle$.
2. Set $z_i \leftarrow v_i[\kappa] \cdot x_i = (u[\kappa])^{\alpha_i} \cdot x_i \bmod N$ and broadcast message $(\text{COMMIT}, sid, P_i, g_i, z_i)$.
3. Send message $(\mathsf{zk\text{-}prove}, sid, 0, (N, g, g_i), \alpha_i)$ to the $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}$ functionality.

All parties output $(\mathsf{RECEIPT}, sid, P_i)$ after receiving $(\mathsf{ZK\text{-}PROOF}, sid, 0, P_i, (N, g, g_i))$ from $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}$.

Round 2 **(prove phase)** For each party P_i, $1 \leq i \leq n$, upon receiving input $(\mathsf{prove}, sid, y_i)$, do:

1. Send message $(\mathsf{zk\text{-}prove}, sid, 0, (N, g, g_i, u[\kappa], z_i, y_i), \alpha)$ to the $\hat{\mathcal{F}}_{\text{ZK}}^{\hat{R}}$ functionality.
2. After receiving messages $(\mathsf{ZK\text{-}PROOF}, sid, 0, P_i, (N, g, g_i, u[\kappa], z_i, y_i))$ from $\hat{\mathcal{F}}_{\text{ZK}}^{\hat{R}}$, all parties output $(\mathsf{PROOF}, sid, P_i, y_i)$.

Round $r = 3, \ldots, (\kappa + 2)$ **(open phase)** Let $\ell = r - 2$. For each party P_i, $1 \leq i \leq n$, do:

1. Broadcast $(\text{RELEASE}, sid, v_i[\ell])$ and send message $(\mathsf{zk\text{-}prove}, sid, r, (N, g, g_i, u[\ell], v_i[\ell]), \alpha_i)$ to ideal functionality $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$.
2. After receiving all n RELEASE and ZK-PROOF messages, proceed to the next round. Otherwise, if any of the broadcast messages is missing, go to panic mode.

At the end of round $(\kappa+2)$, compute $x_j = z_j \cdot (v_j[\kappa])^{-1} \bmod N$, for $1 \leq j \leq n$, output $(\mathsf{DATA}, sid, x_1, x_2, ..., x_n)$ and terminate.

Panic mode: For each party P_i, $1 \leq i \leq n$, do:

- Send $(\mathsf{dealoffer}, sid, \emptyset, n\delta \cdot 2^{\kappa-\ell+1})$ to the environment.
- If the environment responds with $(\mathsf{dealaccept}, sid, \emptyset)$, for $j = 1, 2, ..., n$, and use $v_j[\ell - 1]$ from the previous round to directly compute x_j committed by P_j as $x_j = z_j \cdot \left(\mathsf{RepSq}_{N, v_j[\ell-1]}(2^{\kappa-\ell+1} - 1)\right)^{-1} \bmod N$. Then output $(\mathsf{DATA}, sid, x_1, x_2, ..., x_n)$ in round $(\kappa + 2)$ and terminate.
- Otherwise, output $\perp$ in round $(\kappa + 2)$ and terminate.

Figure 6: *Protocol* GradRel*, running in the CRS model in* $(\kappa + 2)$ *rounds.*

We can show an ideal adversary for $\mathcal{W}(\mathcal{F}_{\text{CPFO}}^R)$ that invests nt/δ and produces a simulation indistinguishable from GradRel. Therefore, GradRel securely realizes $\mathcal{W}(\mathcal{F}_{\text{CPFO}}^R)$ with n/δ-investment.

Theorem 4.4 *Assume that GBBS and CDDH hold. Then protocol* GradRel *securely realizes the ideal functionality* $\mathcal{W}(\mathcal{F}_{\text{CPFO}}^R)$ *with* $O(n)$*-investment in the* $(\mathcal{F}_{\text{CRS}}, \hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}, \hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}, \hat{\mathcal{F}}_{\text{ZK}}^{\hat{R}})$*-hybrid model, assuming static corruptions.*

Before proving this theorem we sketch the essential new elements involving the wrapper. In constructing a simulator $\mathcal{S}$, the most interesting aspect is the simulation of the fair-open phase. Note that the opening takes place in rounds, with the value released in each round being "closer" to the value to be revealed.

- $\mathcal{S}$ internally runs the adversary $\mathcal{A}$, and simulates to it the protocol messages from the honest parties. Initially $\mathcal{S}$ uses random values to simulate the values released by the honest parties in each round.

- However, once the released value gets sufficiently close to the final value, $\mathcal{S}$ can no more use random values, because even a t-bounded adversary and environment can distinguish between that and the values released by the honest party in an actual execution. So, before that point, $\mathcal{S}$ will *invest* sufficient amount of time with $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ and obtain the value to be opened. (The "sufficient" amount is the same as what an honest party entering the panic mode at this point would have requested the environment.) Further rounds in the simulation are carried out using the value obtained from $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ (and hence in those rounds the simulation is perfect).

- At this point a deal is still not offered by $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ to any honest party. But if in a future round, the adversary $\mathcal{A}$ causes a RELEASE or a ZK-PROOF message not to reach an honest party P (which in the real execution would prompt P to enter the panic mode), at that point $\mathcal{S}$ would request $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ to send a deal to P, with investment required from P being the actual time that the protocol would request the environment then. This amount will be no more than what $\mathcal{S}$ invested.

- In the ideal world protocol, if P receives a deal offer from $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, then it would pass it on to the environment, and if the deal is accepted by the environment, then P will invest the amount of time specified in the deal, and obtain the committed value from $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$. In the real world protocol, if P enters the panic mode it will send the deal offer to the environment, and if the deal is accepted by the environment, then P will use the amount of time specified in the deal offer to force-open the computed value. In either, case the environment sees the same behavior from P.

To show that this simulation is good, we depend on the fact that the values released in the initial rounds of the actual execution are pseudorandom, and that in the simulation $\mathcal{S}$ switches to the actual values before this pseudorandomness ceases to hold. The $O(n)$ factor in the amount invested by $\mathcal{S}$ is because of the fact that $\mathcal{S}$ has to make the advance investment for commitments by all honest parties (at most n), whereas the adversary $\mathcal{A}$ might choose to attack any one of them. The $O(n)$ factor also includes (in the constant) the factor δ from the GBBS assumption.

To prove the theorem we must also show a full simulator. A full simulator is essentially a faithful execution of the adversary and the honest parties. The only non-triviality resides in that its running time should not depend on the amount of resources granted by the environment. This is not a problem, since the full simulator will *know* the committed values and need not extract it as the honest parties do in the protocol.

Proof: Let $\mathcal{A}$ be a t-bounded adversary that operates against protocol GradRel. We construct an ideal adversary (i.e., simulator) $\mathcal{S}$ and a full simulator $\mathcal{X}$ so that for all t-bounded environment $\mathcal{Z}$ and adversary $\mathcal{A}$, we have

$$\text{HYB}^{(\mathcal{F}_{\text{CRS}},\hat{\mathcal{F}}^{\mathsf{DL}}_{\text{ZK}},\hat{\mathcal{F}}^{\mathsf{DH}}_{\text{ZK}},\hat{\mathcal{F}}^{\hat{R}}_{\text{ZK}})}_{\mathsf{GradRel},\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{W}(\mathcal{F}_{\text{CPFO}}),\mathcal{S}^{\mathcal{A}}(t),\mathcal{Z}}$$

$$\text{HYB}^{(\mathcal{F}_{\text{CRS}},\hat{\mathcal{F}}^{\mathsf{DL}}_{\text{ZK}},\hat{\mathcal{F}}^{\mathsf{DH}}_{\text{ZK}},\hat{\mathcal{F}}^{\hat{R}}_{\text{ZK}})}_{\mathsf{GradRel},\mathcal{A},\mathcal{Z}} \approx \text{FSIM}_{\mathcal{X}^{\mathcal{A}},\mathcal{Z}}.$$

First we describe the construction of $\mathcal{S}$ and prove the first of the above relations. The construction of the full simulator is straightforward and will be explained after that.

Simulator $\mathcal{S}$. At the beginning of the protocol, $\mathcal{S}$ simulates the $\mathcal{F}_{\text{CRS}}$ functionality by generating a master time-line $L = \langle N, g, \vec{u}\rangle$ just as in the real protocol. Note that since $\mathcal{S}$ generates N, it knows the factorization of N. Assume that $N = p_1 p_2$; $\mathcal{S}$ sets $\Lambda = (p_1 - 1)(p_2 - 1)/4$. Then, during the ideal process, $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$. Messages received from $\mathcal{Z}$ are forwarded to the simulated $\mathcal{A}$,

and messages sent by the simulated $\mathcal{A}$ to its environment are forwarded to $\mathcal{Z}$. Furthermore, $\mathcal{S}$ also plays the roles of the various ideal ZK functionalities.

We describe the behavior of $\mathcal{S}$ as responses to other parties' actions.

Commitment by an uncorrupted party: When $\mathcal{S}$ sees a broadcast message $(\mathsf{RECEIPT}, sid, P_i)$ from $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, it means that an uncorrupted party P_i has committed to a value. $\mathcal{S}$ then generates random elements $g_i \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$, $z_i \stackrel{R}{\leftarrow} \mathsf{QR}_N$ and simulates the two broadcast messages in the real world: $(\text{COMMIT}, sid, P_i, g_i, z_i)$ from P_i, and $(\mathsf{ZK\text{-}PROOF}, sid, 0, P_i, (N, g, g_i))$ from $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}$. Effectively, $\mathcal{S}$ "fakes" a derived time-line with g_i being the seed and z_i being the fake timeline-committed value for P_i.

Commitment by a corrupted party: When $\mathcal{S}$ sees a broadcast message $(\text{COMMIT}, sid, P_i, g_i, z_i)$ from a corrupted party P_i (controlled by $\mathcal{A}$), it means that P_i is committing to a value. Then $\mathcal{S}$ acts as the $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}$ functionality and expects the message $(\mathsf{zk\text{-}prove}, sid, 0, (N, g, g_i), \alpha_i)$ from P_i. Assuming that $g_i = g^{\alpha_i} \bmod N$ (otherwise $\mathcal{S}$ ignores this message), $\mathcal{S}$ can then find out the value P_i commits to by $x_i \leftarrow z_i \cdot (u[\kappa])^{-\alpha_i} \bmod N$. $\mathcal{S}$ then sends message $(\mathsf{commit}, sid, x_i)$ to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ on behalf of P_i.

Proof by an uncorrupted party: When $\mathcal{S}$ sees a broadcast message $(\mathsf{PROOF}, sid, P_i, y_i)$ from $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, it means that an uncorrupted party P_i has succeeded in a proof to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$. $\mathcal{S}$ then simulates this by faking a broadcast message $(\mathsf{ZK\text{-}PROOF}, sid, 0, P_i, (N, g, g_i, u[\kappa], z_i, y_i))$ from the $\hat{\mathcal{F}}_{\text{ZK}}^{\hat{R}}$ functionality.

Proof by a corrupted party: When $\mathcal{S}$ sees a message $(\mathsf{zk\text{-}prove}, sid, 0, (N, g, g_i, u[\kappa], z_i, y_i), \alpha)$ from a corrupted party P_i (controlled by $\mathcal{A}$) to $\hat{\mathcal{F}}_{\text{ZK}}^{\hat{R}}$, it means that P_i is attempting a proof. $\mathcal{S}$ then verifies the witness, and if the verification succeeds, it sends message $(\mathsf{prove}, sid, y_i)$ to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ on behalf of P_i.

Simulating the open phase: In the open phase, $\mathcal{S}$ simulates the gradual opening of the uncorrupted parties. Naturally, the simulation proceeds in rounds. First, it sends message $(\mathsf{deliverat}, sid, (\kappa + 2))$ to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$. Let $m = \kappa - \lfloor \log_2 \left(\frac{t}{\delta}\right) \rfloor - 1$. $\mathcal{S}$ behaves differently in the first $m-1$ rounds of the Open phase from the last $\kappa - m + 1$ rounds; the difference lies in the release value used in simulating the uncorrupted parties (i.e., the value x in the message $(\text{RELEASE}, sid, P_i, x)$ sent by the uncorrupted parties).

- In the first $m-1$ rounds of the open phase, $\mathcal{S}$ simply uses a random value each time for the value being released. That is, in round ℓ, $1 \leq \ell \leq (m-1)$, for each uncorrupted party P_i, $\mathcal{S}$ randomly generates $v_{i,\ell} \stackrel{R}{\leftarrow} \mathsf{QR}_N$ and fakes two broadcast messages: $(\text{RELEASE}, sid, P_i, v_{i,\ell})$ from P_i and $(\mathsf{ZK\text{-}PROOF}, sid, \ell, P_i, (N, g, g_i, u[\ell], v_{i,\ell}))$ from $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$.

  Then, $\mathcal{S}$ waits to receive the RELEASE messages from all the corrupted parties, as well as their $\mathsf{zk\text{-}prove}$ messages to the $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$ functionality. $\mathcal{S}$ proceeds to the next round if all the anticipated messages are received and verified. If any of the messages is missing, or any of the proofs are incorrect, $\mathcal{S}$ sends $(\mathsf{noinvest}, sid, 0)$ to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ and goes to panic mode.
- At round m of the open phase, $\mathcal{S}$ switches its strategy. Notice that from this round on, the adversary may be able to force open the commitment so $\mathcal{S}$ needs to produce a correct commitment. So $\mathcal{S}$ will find the openings in this round. First, $\mathcal{S}$ sends the messages (open, sid) to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ on behalf of every corrupted party P_j, and then sends $(\mathsf{invest}, sid, 0, n2^{\kappa-m+1})$ to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$. It then immediately receives the opening of all the committed values in the message $(\mathsf{DATA}, sid, x_1, x_2, ..., x_n)$ from $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$.

Once $\mathcal{S}$ knows the committed value x_i from uncorrupted P_i, $\mathcal{S}$ can now generated a "real" derived time-line for P_i that is consistent with x_i. This is done by producing the time-line *backward:* We know that the end point of the time-line must be z_i/x_i, and thus the other points should be the roots of z_i/x_i. More precisely, for each uncorrupted party P_i, $\mathcal{S}$ computes $w_i = (z_i/x_i)^{(2^{1-2^{\kappa-m}}) \bmod \Lambda} \bmod N$, which is the $2^{2^{\kappa-m}-1}$th root of (z_i/x_i). Then $\mathcal{S}$ fakes broadcast messages $(\text{RELEASE}, sid, P_i, w_i)$ from P_i and $(\mathsf{ZK\text{-}PROOF}, sid, m, P_i, (N, g, g_i, u[i], w_i))$ from $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$.

Then, $\mathcal{S}$ waits to receive the open messages from all the corrupted parties, as well as their zk-prove messages. As in the previous rounds, it proceeds to the next round if all the messages are received and verified. Otherwise $\mathcal{S}$ goes to the panic round.

- From round m on, $\mathcal{S}$ simulates the gradual opening using the time-line generated in round m. More precisely, in the ℓth round, $\mathcal{S}$ sends the message $(\text{RELEASE}, sid, P_i, \mathsf{RepSq}_{N,w_i}(2^{\kappa-m} - 2^{\kappa-\ell}))$ from P_i and fakes the corresponding messages from $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$.

  Then, as in the previous case, $\mathcal{S}$ waits to receive the broadcast messages and the messages to the $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DH}}$ functionality from all the corrupted parties, and goes to the panic round if these messages are not received or incorrect.
- Panic mode: Here $\mathcal{S}$ simulates each uncorrupted party asking for a deal. For an uncorrupted party P_i, simulator sends $(\mathsf{dealoffer}, sid, P_i, 0, n2^{\kappa-\ell+1} - 1)$ to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$. (Note that this gets sent to P_i, and hence to the environment from P_i.) After this, the simulator simply forwards messages appropriately between $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, P_i, and the environment.

Finally, $\mathcal{S}$ outputs what the simulated $\mathcal{A}$ outputs.

This finishes the description of the ideal adversary $\mathcal{S}$. Now we prove that no t-bounded environment $\mathcal{Z}$ can distinguish the hybrid experiment with a t-bounded $\mathcal{A}$ and GradRel from the ideal process with $\mathcal{S}^{\mathcal{A}}(t)$ and $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, or equivalently,

$$\text{HYB}^{\mathcal{F}_{\text{CRS}},\hat{\mathcal{F}}_{\text{ZK}}}_{\mathsf{GradRel},\mathcal{A},\mathcal{Z}}(\kappa, z) \approx \text{IDEAL}_{\mathcal{W}(\mathcal{F}_{\text{CPFO}}),\mathcal{S}^{\mathcal{A}}[t],\mathcal{Z}}(\kappa, z)$$

To do so, we construct two experiments $\mathsf{Mix}^{\mathcal{F}_{\text{CRS}}}_{\mathcal{S}_H^{\mathcal{A}}[t],\mathcal{Z}}$ and $\mathsf{Mix}^{\mathcal{F}_{\text{CRS}}}_{\mathcal{S}_I^{\mathcal{A}}[t],\mathcal{Z}}$, which only differ in their adversaries $\mathcal{S}_H$ and $\mathcal{S}_I$. These differ only slightly, so we will describe $\mathcal{S}_H$, and then how $\mathcal{S}_I$ differs from $\mathcal{S}_H$. $\mathcal{S}_H$ behaves like $\mathcal{S}$ in that it runs $\mathcal{A}$ (forwarding messages between $\mathcal{Z}$ and $\mathcal{A}$) and simulates uncorrupted parties and the ideal ZK functionalities for $\mathcal{A}$. However, it does not simulate the CRS functionality (i.e., it does not generate the master time-line, and thus does not know the factorization of N), but instead has access to $\mathcal{F}_{\text{CRS}}$. ($\mathcal{S}_H$ forwards any messages between the CRS functionality and $\mathcal{A}$.) $\mathcal{S}_H$ also emulates the $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ functionality. Finally, $\mathcal{S}_H$ behaves differently from $\mathcal{S}$ when simulating the commit and open phases for an uncorrupted party, as follows.

When an uncorrupted party P_i sends a message $(\mathsf{commit}, sid, x_i)$ on to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, $\mathcal{S}_H$ simulates the actual GradRel protocol for P_i (instead of sending a random element z_i). In the open phase, $\mathcal{S}_H$ sends RELEASE messages as in the real-world experiment (instead of revealing random values).

$\mathcal{S}_I$ differs from $\mathcal{S}_H$ in that it behaves like $\mathcal{S}$ in the commit phase, and also in the open phase until step m. That is, it commits to random numbers until step m, and then commits to the "correct" values as in the real-world experiment

First, we show that $\text{HYB}^{\mathcal{F}_{\text{CRS}},\hat{\mathcal{F}}_{\text{ZK}}}_{\mathsf{GradRel},\mathcal{A},\mathcal{Z}}(\kappa, z) \approx \mathsf{Mix}^{\mathcal{F}_{\text{CRS}}}_{\mathcal{S}_H^{\mathcal{A}}[t],\mathcal{Z}}(\kappa, z)$. Actually, the distributions produced by these two experiments are identical. This follows from the fact that the simulated ZK functionalities behave exactly the same as the actual ZK functionalities, and given perfect ZK functionalities, the

outputs of the honest parties in Mix are exactly what they would be in HYB, since the committed values that are opened in HYB must be exactly the same as what was extracted from the ZK functionalities Mix. Also, by inspection, the behavior of the parties on deals are indistinguishable.

Next, we show that $\mathsf{Mix}^{\mathcal{F}_{\mathrm{CRS}}}_{\mathcal{S}_I^{\mathcal{A}}[t],\mathcal{Z}}(\kappa, z) \approx \mathrm{IDEAL}_{\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}}),\mathcal{S}^{\mathcal{A}}[t],\mathcal{Z}}(\kappa, z)$. Again, the distributions produced by these two experiments are identical. This follows from the fact that the distributions of $\mathcal{F}_{\mathrm{CRS}}$ and the simulated CRS functionality are the same, the distribution of z_i values produced by simulated honest parties is the same, and that the values produced in the open phase are the same (even though the values computed at step m and after are computed in different ways).

Finally, we show that $\mathsf{Mix}^{\mathcal{F}_{\mathrm{CRS}}}_{\mathcal{S}_H^{\mathcal{A}}[t],\mathcal{Z}}(\kappa, z) \approx \mathsf{Mix}^{\mathcal{F}_{\mathrm{CRS}}}_{\mathcal{S}_I^{\mathcal{A}}[t],\mathcal{Z}}(\kappa, z)$ The difference between these experiments is as follows. In the Mix experiment with $\mathcal{S}_H$, each uncorrupted party P_i produces a consistent time-line L_i — or a prefix of it (in the case of premature abort). In the Mix experiment with $\mathcal{S}_I$, however, the first $(m-1)$ points of each uncorrupted party's time-line are replaced by random quadratic residues. Nevertheless, the last $(\kappa - m + 1)$ points of each time-line are real and consistent with the committed value of each uncorrupted party. So the difference lies in the prefixes of the time-lines of the uncorrupted parties. Then one can easily reduce the indistinguishability between the two experiments reduces to the strong pseudorandomness of time-lines (Lemma 4.3) via a standard hybrid argument.

Full simulator $\mathcal{X}$. Recall that the full simulator $\mathcal{X}$ replaces not just the adversary, but also the honest parties running the protocol. It gets access to all the inputs to all the parties. So the simulation is almost trivial: it can, for the most part, faithfully execute the code of the honest parties and the adversary. However, the running time of $\mathcal{X}$ must be independent of the amount of resources granted to it (i.e., granted to the honest parties simulated by it) by the environment. Note that a faithful simulation of the honest parties will result in $\mathcal{X}$'s running time depend on what is granted by $\mathcal{Z}$. So, when a deal is accepted by $\mathcal{Z}$, $\mathcal{X}$ cannot carry out the panic mode computations. Nevertheless, it can find the outcome of such a computation. This is because $\mathcal{X}$ knows all the values committed by all the parties that it is simulating (as these values are available from the communication with the zero-knowledge functionalities). Thus, we can construct a full simulator $\mathcal{X}$ as required by the security and fairness definition, by following the protocol exactly, except for the panic mode computation (which can be avoided as described above). ∎

By "plugging in" the UCZK protocol from [18] into protocol GradRel, we have the following corollary.

Corollary 4.5 *Assume GBBS and CDDH hold, and that enhanced trapdoor permutations exist. Then there exists a protocol that securely realizes $\mathcal{W}(\mathcal{F}^R_{\mathrm{CPFO}})$ with $O(n)$-investment in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid model, assuming static corruptions.*

5 Resource-Fair Secure Multi-Party Computation

In this section we show how to construct resource-fair protocols that securely realize the (wrapped) SFE functionality in the FMPC framework. At a high level, our strategy is very simple. Typical secure multi-party protocols (e.g., [22, 18, 26]) contain an "output" phase, in which every party reveals a secret value, and once all secret values are revealed, every party computes the output of the function. We modify the output phase to have the parties invoke the $\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}})$ functionality. A bit more concretely, assuming each party P_i holds a secret value v_i to reveal, each P_i first commits to v_i and then proves its correctness. Finally $\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}})$ opens all the commitments simultaneously.

We present two constructions that convert the MPC protocols of Canetti *et al.* [18] and Cramer *et al.* [22] into resource-fair MPC protocols.

5.1 Resource-fair MPC in the CRS model

Theorem 5.1 *Assuming the existence of enhanced trapdoor permutations, for any polynomial-time computable function f, there exists a polynomial-time protocol that securely realizes $\mathcal{W}(\mathcal{F}_f)$ with $O(n)$-investment in the $(\mathcal{F}_{\text{CRS}}, \mathcal{W}(\mathcal{F}_{\text{CPFO}}))$-hybrid model in the FMPC framework, assuming static corruptions.*

Proof sketch: Consider the MPC protocol secure against malicious adversaries by Canetti *et al.* [18]. We denote it by π_f. Recall that π_f is "compiled" from another protocol $\hat{\pi}_f$ that is only secure against "honest-but-curious" adversaries. In the compilation process, P_i commits to its initial values using a commit-and-prove functionality called $\mathcal{F}_{\text{CP}}$, and then for every message m that P_i sends in protocol $\hat{\pi}_f$, the compiler makes P_i send a zk-prove message to the $\mathcal{F}_{\text{CP}}$ ideal functionality in protocol π_f to prove that message m was computed correctly. The protocol $\hat{\pi}_f$ itself consists of three *stages* — the input preparation stage, the circuit evaluation stage, and the output stage. In particular, the output stage of $\hat{\pi}_f$ consists of each party P_i broadcasting its share m_i of the output. After the compilation, the output stage in π_f consists of each party P_i broadcasting m_i along with a proof that m_i is valid.

We modify protocol π_f to make it secure in the FMPC framework. Notice that π_f assumes a broadcast channel, which is built into the FMPC framework, and it is rather straightforward to fit π_f into the round structure of FMPC—we omit these technical details. The non-trivial modification comes at the output stage, where instead of broadcasting m_i, each party P_i commits to m_i by sending message $(\mathsf{commit}, sid, m_i)$ to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$. In the next round, each party P_i then sends message $(\mathsf{prove}, sid, y_i)$ to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ to prove the correctness of m_i. Here y_i is the appropriate string so that the proof to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ is equivalent to the proof to $\mathcal{F}_{\text{CP}}$. Finally, all parties send (open, sid) to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, which causes it to send the messages m_i to all parties using the fair-delivery mechanism described in Figure 2. We denote this modified protocol by $\tilde{\pi}_f$.

$\tilde{\pi}_f$ also incorporates the following modification: if it receives a dealoffer message from $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, then it passes on this message to the environment, as an offer for the output from $\mathcal{F}_f$; if it gets a response form the environment then that will be passed on to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, and the response from $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ will be used to compute the output from $\mathcal{F}_f$ (which will be returned to the environment).

Next, we describe an ideal adversary $\tilde{\mathcal{S}}$ for an adversary $\tilde{\mathcal{A}}$ in protocol $\tilde{\pi}_f$. $\tilde{\mathcal{S}}$ is adapted from the ideal adversary $\mathcal{S}$ for some adversary $\mathcal{A}$ for protocol π_f. Below we sketch how $\mathcal{A}$ is constructed from $\tilde{\mathcal{A}}$, and $\tilde{\mathcal{S}}$ from $\mathcal{S}$.[20]

$\mathcal{A}$ internally runs $\tilde{\mathcal{A}}$. Recall that π_f is in the $\mathcal{F}_{\text{CP}}$-hybrid model, whereas $\tilde{\pi}_f$ is in the $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$-hybrid model.[21] So $\tilde{\mathcal{A}}$ may interact with the wrapper (sending messages invest, noinvest, dealoffer), whereas such messages from $\mathcal{A}$ will not be entertained by $\mathcal{F}_{\text{CP}}$. The adversary $\mathcal{A}$ is constructed from $\tilde{\mathcal{A}}$ by incorporating the wrapper into it, as follows: the messages that $\tilde{\mathcal{A}}$ sends to the wrapper are internally handled by $\mathcal{A}$. When $\tilde{\mathcal{A}}$ sends an invest message (addressed to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$), $\mathcal{A}$ will send a deliverat message to $\mathcal{F}_{\text{CP}}$, directing it to send the output at that round.

$\mathcal{S}$ is obtained by applying the security guarantee of π_f from [18], but is naturally modified to deal with the round structure and the deliverat messages from $\mathcal{A}$. When $\mathcal{A}$ sends $(\mathsf{deliverat}, sid, s)$ message

[20] One can think of this as a general adaptation of the simulation of an unfair protocol to the simulation of one that is resource-fair; in fact, we will be using it again in the construction of Theorem 5.3.

[21] To be a little more precise, we cosmetically modify $\mathcal{F}_{\text{CP}}$ of [18] as follows, so that it resembles the $\mathcal{F}_{\text{CPFO}}$ functionality in Figure 3: we add an explicit open phase (to distinguish it from other proof phases); then in the open phase $\mathcal{F}_{\text{CP}}$ sends the messages to the parties in round s if the adversary instructs it do so using a $(\mathsf{deliverat}, sid, s)$ message. It is easily verified that this modification does not change the arguments in [18].

to $\mathcal{F}_{\text{CP}}$ then $\mathcal{S}$ will send the same message to $\mathcal{F}_f$. Then it uses the output obtained from $\mathcal{F}_f$ to continue the simulation as in [18].

$\tilde{\mathcal{S}}$ internally simulates $\mathcal{S}$, and externally it interacts with $\mathcal{W}(\mathcal{F}_f)$. If $\mathcal{S}$ sends $(\mathsf{deliverat}, sid, s)$ to (simulated) $\mathcal{F}_f$, then $\tilde{\mathcal{S}}$ will send an invest message to $\mathcal{W}(\mathcal{F}_f)$ to obtain the output. The amount of resources invested is the same as what $\tilde{\mathcal{A}}$ (which is internally simulated by $\mathcal{A}$, which in turn is internally simulated by $\mathcal{S}$ and by $\tilde{\mathcal{S}}$) invests (to the wrapper simulated by $\mathcal{A}$). $\tilde{\mathcal{S}}$ interacts with the environment through $\tilde{\mathcal{A}}$.

We point out the following chain of actions in the simulation: when $\tilde{\mathcal{A}}$ sends an invest message to $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, $\mathcal{A}$ sends a $\mathsf{deliverat}$ message to $\mathcal{F}_{\text{CP}}$, which is forwarded by $\mathcal{S}$ to $\mathcal{F}_f$, and then $\tilde{\mathcal{S}}$ sends an invest message to $\mathcal{W}(\mathcal{F}_f)$. Then, the message received from $\mathcal{W}(\mathcal{F}_f)$ is used by $\tilde{\mathcal{S}}$ to respond to $\mathcal{S}$'s $\mathsf{deliverat}$ message, which in turn responds to $\mathcal{A}$'s $\mathsf{deliverat}$ message, and then $\mathcal{A}$ can respond to $\tilde{\mathcal{A}}$'s invest message. Also note that from the environment's point of view, during the simulation the honest parties in the $\mathcal{W}(\mathcal{F}_f)$-hybrid model behave like the honest parties in the actual execution of $\tilde{\pi}_f$, in particular passing to and fro the $\mathsf{dealoffer}$ messages corresponding to the $\mathcal{W}(\mathcal{F}_f)$ functionality.

It is straightforward (but somewhat tedious in the details) to show that from the point of view of the environment, the above simulation by $\tilde{\mathcal{S}}$ is a *perfect* simulation of the actual execution of $\tilde{\pi}_f$ with the adversary $\tilde{\mathcal{A}}$. ∎

Corollary 5.2 *Assuming GBBS, CDDH, and the existence of enhanced trapdoor permutations, for any polynomial-time computable function f, there exists a resource-fair protocol that securely realizes $\mathcal{W}(\mathcal{F}_f)$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model in the FMPC framework, assuming static corruptions.*

Proof: It directly follows from Theorem 2.5, Corollary 4.5, and Theorem 5.1. ∎

5.2 Efficient resource-fair MPC in the PKI model

We now show an efficient and resource-fair MPC protocol in the PKI model. (See Appendix A for discussions on the PKI model.)

Theorem 5.3 *Assuming GBBS, CDDH, DCRA, and strong RSA, for any polynomial-time computable function f, there exists a resource-fair protocol that securely realizes $\mathcal{W}(\mathcal{F}_f)$ in the $(\mathcal{F}_{\text{PKI}}, \mathcal{W}(\mathcal{F}_{\text{CPFO}}))$-hybrid model in the FMPC framework, assuming static corruptions. Furthermore, this protocol has communication complexity $O(\kappa n|C| + \kappa^2 n)$ bits and consists of $O(d + \kappa)$ rounds.*

Proof sketch: Cramer *et al.* [22] proved that for any polynomial-time computable function f (represented as an arithmetic circuit C), there exists a protocol, call it CDN_f, that securely realizes $\mathcal{F}_f$ in the PKI model, assuming DCRA and DDH. Furthermore, the protocol uses $O(\kappa n|C|)$ bits of communication and proceeds in $O(d)$ rounds, where κ is the security parameter, $|C|$ is the number of gates in C, and d the depth of C.[22]

We note that the protocol is only secure against less than $n/2$ corruptions. A crucial ingredient in the construction is a threshold homomorphic cryptosystem (e.g., the threshold version of the Paillier cryptosystem). Values on the wires of C are encrypted, and the parties share the decryption key using a (n, t)-threshold system, so that any $(t+1)$ parties can jointly decrypt, but any t parties cannot. By avoiding the sharing of values, but instead only sharing the decryption key, the resulting construction is very efficient in terms of communication complexity.

[22] In fact, the theorem in [22] is more general. Here we just state a special case of their result, using a threshold version of the Paillier encryption scheme [25, 32] and Pedersen commitment [51] as the trapdoor commitment scheme.

Protocol CDN_f is proved secure in [22] in a "modular composition" framework [14], which is somewhat weaker than the UC framework and our FMPC framework. We now show a series of transformations that converts CDN_f into a resource-fair protocol in the FMPC framework.

1. Given that security is in the modular composition framework, CDN_f does not remain secure under general composition. Specifically, the reason for this is the use in the protocol of *standard* trapdoor commitments (TC) to construct the zero-knowledge protocols (more precisely, to convert the Σ-protocols [23, 21] into "normal" zero-knowledge protocols), and such commitment schemes may be malleable. Our first transformation consists in replacing the TC schemes by *simulation-sound trapdoor commitment* (SSTC) schemes [35, 48]. MacKenzie and Yang [48] have shown that this change will make a zero-knowledge protocol universally composable if the underlying protocol has a non-rewinding knowledge extractor, and the zero-knowledge protocols from [22] can be easily modified to accommodate a non-rewinding extractor, using techniques from [35]. As a result, after the zero-knowledge protocols are strengthened to be universally composable, it is not hard to verify that protocol CDN_f thus modified becomes secure in the UC framework. Note also that there exist very efficient constructions of SSTC schemes, assuming strong RSA [35, 48].[23]

2. Next, we modify the threshold of the cryptosystem so that only when *all* parties participate can they decrypt an encrypted message. In [22], two homomorphic cryptosystems are proposed: a threshold version of the Paillier cryptosystem and a system based on the quadratic residuosity assumption and DDH. Both systems admit efficient zero-knowledge proofs and joint decryption protocols. Furthermore, in both systems, the joint decryption phase consists of each party P_i broadcasting a single value v_i along with a zero-knowledge proof that v_i is "correct." After all parties broadcast the correct values, every party can then perform the decryption on its own. We change the cryptosystem to have an $(n, n-1)$-threshold. (Of course, by doing this, this intermediate protocol becomes unfair.)

3. Finally, we further modify the joint decryption phase by having all parties invoke the $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ functionality to release their secret information simultaneously. That is, instead of directly outputting their values, each party P_i commits to its value v_i, proves the correctness of v_i, and then has the $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ functionality open all the values simultaneously. In more detail, this is done in two steps: first, the protocol is modified to invoke the version of functionality $\mathcal{F}_{\text{CP}}$ discussed in the proof of Theorem 5.1, and then the transformation and simulation argument presented there are applied.

After all these modifications, and assuming that the homomorphic threshold encryption scheme being used is Paillier, the resulting protocol becomes resource-fair in the $(\mathcal{F}_{\text{PKI}}, \mathcal{W}(\mathcal{F}_{\text{CPFO}}))$-hybrid model in the FMPC framework, under the assumptions and complexities stated in the theorem. ∎

Acknowledgements

The authors thank Amit Sahai for helpful discussions on the formulation of the notion of resource fairness, and Yehuda Lindell and Jesper Nielsen for useful comments.

[23] An alternative approach is to use, instead of SSTC schemes, universally composable commitment (UCC) schemes. This is in fact the approach Damgård and Nielsen [26] use. In fact, with this transformation they manage to prove that the resulting protocol is secure against adaptive corruptions in the *non-erasing* model, while the constructions in [35] and subsequently in [48] using SSTCs only achieve security against adaptive corruptions in the *erasing* model. On the other hand, SSTC admits simpler, more efficient constructions than UCC, and thus allows more efficient constructions. Since the current paper is only concerned with static corruptions, the difference between the erasing and non-erasing models is irrelevant.

References

[1] L. Adleman and K. Kompella. Using smoothness to achieve parallelism. In *20th STOC*, pp. 528–538, 1988.

[2] N. Asokan, V. Shoup, and M. Waidner. Optimistic Fair Exchange of Digital Signatures (Extended Abstract). In *EUROCRYPT 1998*, pp. 591–606, 1998.

[3] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *1st Theory of Cryptography Conference (TCC)* (LNCS 2951), pp. 336-354, 2004.

[4] D. Beaver and S. Goldwasser. Multiparty Computation with Faulty Majority. In *30th FOCS*, pages 503–513, 1990.

[5] J. Benaloh and M. de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In *EUROCRYPT 1993* (LNCS 765), pp. 274–285, 1994.

[6] M. Ben-Or, O. Goldreich, S. Micali and R. Rivest. A Fair Protocol for Signing Contracts. *IEEE Transactions on Information Theory* 36(1):40–46, 1990.

[7] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th STOC*, pp. 1–10, 1988.

[8] M. Blum. How to exchange (secret) keys. In *ACM Transactions on Computer Systems*, 1(2):175–193, May 1983.

[9] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.

[10] D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium* (LNCS 1423), pp. 48–63, 1998.

[11] D. Boneh and M. Naor. Timed commitments (extended abstract). In *Advances in Cryptology—CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254, Springer-Verlag, 2000.

[12] F. Boudot, B. Schoenmakers and J. Traoré. A fair and efficient solution to the socialist millionaires' problem. In *Discrete Applied Mathematics*, 111(1-2): 23-36 2001.

[13] C. Cachin and J. Camenisch. Optimistic Fair Secure Computation. In *Advances in Cryptology—CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111, Springer-Verlag, 2000.

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

[15] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Electronic Colloquium on Computational Complexity (ECCC) TR01-016, 2001. Previous version "A unified framework for analyzing security of protocols" availabe at the ECCC archive TR01-016. Extended abstract in FOCS 2001.

[16] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Revised version of [15].

[17] R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO 2001* (LNCS 2139), pp. 19–40, 2001.

[18] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally Composable Two-party and Multi-party Secure Computation. In *34th ACM Symposium on the Theory of Computing*, 2002.

[19] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th STOC*, pp. 11–19, 1988.

[20] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC 1986)*, pages 364-369 (1986)

[21] R. Cramer. Modular Design of Secure yet Practical Cryptographic Protocols. Ph.D. Thesis. CWI and University of Amsterdam, 1997.

[22] R. Cramer, I. Damgård, and J. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption In *Advances in Cryptology - EuroCrypt 2001 Proceedings*, volume 2045 of *Lecture Notes in Computer Science*, pp. 280–300, Springer-Verlag, 2001.

[23] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology - CRYPTO '94* (LNCS 839), pages 174–187, 1994.

[24] I. Damgård. Practical and Provably Secure Release of a Secret and Exchange of Signatures. In *Journal of Cryptology* 8(4), pp. 201–222, 1995.

[25] I. Damgård and M .Jurik. Efficient protocols based probabilistic encryptions using composite degree residue classes. In *Research Series RS-00-5*, BRICS, Department of Computer Science, University of Aarhus, 2000.

[26] I. Damgård, and J. Nielsen. Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In *Advances in Cryptology - CRYPTO '03*, 2003.

[27] D. Dolev, C. Dwork and M. Naor. Non-malleable cryptography. *SIAM J. on Comput.*, 30(2):391–437, 2000. An earlier version appeared in *23rd ACM Symp. on Theory of Computing*, pp. 542–552, 1991.

[28] C. Dwork, M. Naor and A. Sahai. Concurrent zero-knowledge. In *30th ACM Symposium on the Theory of Computing*, pp. 409–418, 1998.

[29] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.

[30] R. Fagin, M. Naor and P. Winkler. Comparing information without leaking it. *Communications of the ACM*, 38(5):77-85, 1996.

[31] M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein and A. Smith. Detectable Byzantine Agreement Tolerating Faulty Majorities (from scratch). In *21st PODC*, pages 118–126, 2002.

[32] P. Fouque, G .Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *Proceedings of Financial Crypto 2000*, 2000.

[33] Z. Galil, S. Haber, and M. Yung. Cryptographic Computation: Secure Fault-tolerant Protocols and the Public-Key Model. In *CRYPTO'87*, pp. 135–155, 1988.

[34] J. Garay and M. Jakobsson. Timed Release of Standard Digital Signatures. In *Financial Cryptography '02*, volume 2357 of *Lecture Notes in Computer Science*, pages 168–182, Springer-Verlag, pp. 168–182, 2002.

[35] J. Garay, P. MacKenzie and K. Yang. Strengthening Zero-Knowledge Protocols using Signatures. In *Advances in Cryptology – Eurocrypt 2003*, Warsaw, Poland, LNCS 2656, pp.177-194, 2003. Full version available at *ePrint Archive*, report 2003/037, `http://eprint.iacr.org/2003/037`, 2003.

[36] J. Garay, P. MacKenzie and K. Yang. Efficient and Universally Composable Committed Oblivious Transfer and Applications. In *1st Theory of Cryptography Conference (TCC)*, (LNCS 2951), pp. 297-316, 2004.

[37] J. Garay, P. MacKenzie and K. Yang. Efficient and Secure Multi-Party Computation with Faulty Majority and Complete Fairness. In Cryptology ePrint Archive, `http://eprint.iacr.org/2004/019`.

[38] J. Garay and C. Pomerance. Timed Fair Exchange of Standard Signatures. In *Financial Cryptography 2003*, volume 2742 of *Lecture Notes in Computer Science*, pp. 190–207, Springer-Verlag, 2003.

[39] O. Goldreich. Secure Multi-Party Computation (Working Draft, Version 1.2), March 2000. Available from `http://www.wisdom.weizmann.ac.il/ oded/pp.html`.

[40] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing*, pp. 218–229, 1987.

[41] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority, In *CRYPTO '90*, pp. 77-93, Springer-Verlag, 1991.

[42] S. Goldwasser and Y. Lindell. Secure Computation Without Agreement. In *Journal of Cryptology*, 18(3), pp. 247-287, 2005.

[43] D. Hofheinz and J. Müller-Quade. A Synchronous Model for Multi-Party Computation and Incompleteness of Oblivious Transfer. In Cryptology ePrint Archive, `http://eprint.iacr.org/2004/016`, 2004.

[44] M. Jakobsson and M. Yung. Proving without knowing: on oblivious, agnostic and blindfolded provers. In *CRYPTO '96*, pp. 186–200, LNCS 1109, 1996.

[45] M. Lepinski, S. Micali, C. Peikert, and A. Shelat. Completely fair SFE and coalition-safe cheap talk. In *23rd PODC*, pages 1–10, 2004.

[46] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation.In *FOCS 2003*.

[47] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. In *CRYPTO 2002* (LNCS 2442), pp. 385–400, 2002.

[48] P. MacKenzie and K. Yang. On Simulation Sound Trapdoor Commitments. Manuscript.

[49] J. B. Nielsen. On Protocol Security in the Cryptographi Model. Ph.D. Thesis. Aarhus University, 2003.

[50] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *Advances in Cryptology–Eurocrypt '99*, pp.223–238, 1999.

[51] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology – CRYPTO '91* (LNCS 576), 129–140, 1991.

[52] B. Pfitzmann and M. Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In *ACM Conference on Computer and Communications Security (CSS)*, pp. 245–254, 2000.

[53] B. Pinkas. Fair Secure Two-Party Computation. In *Eurocrypt 2003*, pages 87–105, 2003.

[54] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *21st STOC*, pp. 73–85, 1989.

[55] A. Solomaa. Public-key cryptography. *Springer-Verlag*, 1990.

[56] B. Schneier. Applied cryptography. *John Wiley & Sons*, 1996.

[57] V. Shoup. A Computational Introduction to Number Theory and Algebra. *Preliminary book, available at* `http://shoup.net/ntb/`.

[58] J. Sorenson. A Sublinear-Time Parallel Algorithm for Integer Modular Exponentiation. Available from `http://citeseer.nj.nec.com/sorenson99sublineartime.html`.

[59] A. Yao. Protocols for Secure Computation. In *FOCS 1982*, pages 160–164, 1982.

[60] A. Yao. How to generate and exchange secrets. In *FOCS 1986*, pages 162–167, 1986.

A The Universal Composability Framework

We describe the universal composability (UC) framework briefly.

The execution in the real-life model and the ideal process proceeds basically as follows. The environment $\mathcal{Z}$ drives the execution. It can provide input to a party P_i or to the adversary, $\mathcal{A}$ or $\mathcal{S}$. If P_i is given an input, P_i is activated. In the ideal process P_i simply forwards the input directly to $\mathcal{F}$, which is then activated, possibly writing messages on its outgoing communication tape, and then handing activation back to P_i.[24] In the real-life model, P_i follows its protocol, either writing messages on its outgoing communication tape or giving an output to $\mathcal{Z}$. Once P_i is finished, $\mathcal{Z}$ is activated again. If the adversary is activated, it follows its protocol, possibly giving output to $\mathcal{Z}$, and also either corrupting a party, or performing one of the following activities. If the adversary is $\mathcal{A}$ in the real-life model, it may deliver a message from the output communication tape of one honest party to another, or send a message on behalf of a corrupted party. If the adversary is $\mathcal{S}$ in the ideal process, it may deliver a message from $\mathcal{F}$ to a party, or send a message to $\mathcal{F}$. If a party or $\mathcal{F}$ receives a message, it is activated, and once it finishes, $\mathcal{Z}$ is activated

At the beginning of the execution, all participating entities are given the security parameter $k \in \mathbb{N}$ and random bits. The environment is also given an auxiliary input $z \in \{0,1\}^*$. At the end of the execution, the environment outputs a single bit. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble of random variables describing $\mathcal{Z}$'s output when interacting in the real-life model with adversary $\mathcal{A}$ and players running protocol π, with input z, security parameter k, and uniformly-chosen random tapes for all participating entities. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the distribution ensemble of random variables describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$, with input z, security parameter k, and uniformly-chosen random tapes for all participating entities.

A protocol π *securely realizes* an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any auxiliary input, can tell with non-negligible advantage whether it is interacting with $\mathcal{A}$ and players running π in the real-life model, or with $\mathcal{S}$ and $\mathcal{F}$ in the ideal-process. More precisely, $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, where denotes *computational indistinguishability*. (In particular, this means that for any $d \in \mathbb{N}$ there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and for all inputs z, $|\Pr[\,\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\,] - \Pr[\,\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\,]| < k^{-d}$).

To formulate the composition theorem, one must introduce a hybrid model, a real-life model with access to an ideal functionality $\mathcal{F}$. In particular, this $\mathcal{F}$-hybrid model functions like the real-life model, but where the parties may also exchange messages with an unbounded number of copies of $\mathcal{F}$, each copy identified via a unique *session identifier* (*sid*). The communication between the parties and each one of these copies mimics the ideal process, and in particular the hybrid adversary does not have access to the contents of the messages. Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble of random variables describing the output of $\mathcal{Z}$, after interacting in the $\mathcal{F}$-hybrid model with protocol π. Let π be a protocol in the $\mathcal{F}$-hybrid model, and ρ a protocol that secures realizes $\mathcal{F}$. The composed protocol π^ρ is now constructed by replacing the first message to $\mathcal{F}$ in π by an invocation of a new copy of ρ, with fresh random input, the same *sid*, and with the contents of that message as input; each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of ρ, with the contents of that message as new input to ρ.

Canetti [15] proves the following composition theorem.

Theorem A.1 [[15]] Let $\mathcal{F}$, $\mathcal{G}$ be ideal functionalities. Let π be an n-party protocol that securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, and let ρ be an n-party protocol that securely realizes $\mathcal{F}$. Then

[24] In [18], the behavior is modified. The inputs are forwarded to the functionality by the ideal adversary, which sees the *public header* of the inputs and may choose not to forward them. See [18] for more detailed discussions. We choose to use the old formulation since it is slightly simpler and the distinction does not make a difference when one only deals with static corruption.

protocol π^ρ securely realizes $\mathcal{G}$.

The CRS model and the PKI model. In this paper we work with both the CRS model and the PKI model. In the CRS model, there is a common reference string (CRS) generated from a prescribed distribution accessible to all parties at the beginning of the protocol. The $\mathcal{F}_{\text{CRS}}$ functionality simply returns the CRS. The public key infrastructure (PKI) model is stronger. Upon initial activation, a PKI functionality, $\mathcal{F}_{\text{PKI}}$, generates a public string as well as a private string for each party. We note that both models can be defined in the UC and the FMPC frameworks.

B Remarks on the New Generalized BBS Assumption

We first present the generalized BBS assumption as formulated by Boneh and Naor [11], and then point out the differences with our formulation of Section 3.

Let n be a positive integer representing a certain security parameter. Let N be a Blum integer, i.e., $N = p_1 p_2$, with p_1 and p_2 as above, $|p_1| = |p_2| = n$. For g and N as above, and a positive integer $k > n'$, let

$$\vec{w}_{g,k} \;=\; < g^2, g^4, g^{16}, \ldots, g^{2^{2^i}}, \ldots, g^{2^{2^{k-1}}}, g^{2^{2^k}} > (\mathrm{mod} N). \tag{13}$$

The $(n', n, \delta, \epsilon)$ *generalized BBS assumption*, as presented in [11], states that for any integer $n' < k < n$ and any PRAM algorithm $\mathcal{A}$ whose running time is less than $\delta \cdot 2^k$,

$$\left| \Pr[\mathcal{A}(N, g, k, \vec{w}_{g,k}, g^{2^{2^{k+1}}}) = 1] - \Pr[\mathcal{A}(N, g, k, \vec{w}_{g,k}, R^2) = 1] \right| < \epsilon \tag{14}$$

where the probability is taken over the random choice of an n-bit Blum integer as above, an element $g \stackrel{R}{\leftarrow} \mathbb{Z}_N$ and $R \stackrel{R}{\leftarrow} \mathbb{Z}_N$.

We remark the following differences:

1. In the original GBBS assumption the vector $\vec{a}$ is fixed to a specific pattern. Subsequent work [38, 53] has considered different (fixed) patterns, but without modifying the assumption. The pattern needed by the constructions in this paper is similar to the one in [38, 53]; however, instead of proposing an assumption for another fixed pattern, we propose the current formulation as a reasonable generalization.

2. Note that in the new formulation, the running time of the adversary is no longer necessarily (proportional to) a power of two, as in in the original one. Additionally, the split of parameters in the new formulation — κ and k — *is needed* since, as stated, the original formulation would allow an adversary to run in time $2^{\Omega(\kappa)}$. Using this much time, however, the adversary could factor N and easily distinguish points on the time-line from a random point — indeed, there exist factoring algorithms of running time $2^{O(\kappa^{1/3}(\log \kappa)^{2/3})}$ (see, e.g., [57] for details). The new formulation fixes this problem by setting an upper bound function of k so that the adversary's running time would not be sufficient to factor N.

C Proofs

Proof: (of Lemma 4.3 — Strong Pseudorandomness) We consider four different distributions and prove that they are all computationally indistinguishable to each other, which shall imply our lemma.

At a high level, all four distributions consists of a master time-line L and part of a derived time-line L'. However, in two of the distributions, the master time-line is real, while in the other two the master time-line is "faked." Similarly, in two of the distributions, the derived time-line is "faithful," while in the other two the derived time-line is "unfaithful." We now describe these four distributions in more detail.

$\mathsf{Dist}^{\mathsf{R,F}}$**: Real master time-line, faithful derived time-line.** Let $L = \langle N, g, \vec{u}\rangle$ be a master time-line, $\alpha \xleftarrow{R} \mathbb{Z}_{[1,\frac{N-1}{2}]}$, and $h = g^\alpha \bmod N$. Let $\vec{w}$ be the last $(\ell+1)$ elements of the time-line derived from L with shifting factor α; i.e., $\vec{w} = \langle (u[\kappa-\ell])^\alpha, (u[\kappa-\ell+1])^\alpha, ...(u[\kappa])^\alpha\rangle$. Let $X = (u[\kappa-\ell-1])^\alpha$. $\mathsf{Dist}^{\mathsf{R,F}} = \langle N, g, \vec{u}, h, \vec{w}, X\rangle$.

$\mathsf{Dist}^{\mathsf{R,U}}$**: Real master time-line, unfaithful derived time-line.** Same as in $\mathsf{Dist}^{\mathsf{R,F}}$, except that $X = R^2$ is a random quadratic residue in $\mathbb{Z}_N$, instead of $(u[\kappa-\ell-1])^\alpha$.

$\mathsf{Dist}^{\mathsf{F,F}}$**: Fake master time-line, faithful derived time-line.** Same as in $\mathsf{Dist}^{\mathsf{R,F}}$, except that the first $(\kappa-\ell-1)$ points in the master time-line are fake. More precisely, let $L^* = \langle N, g, \vec{u^*}\rangle$, where $u^*[i] = \mathsf{RepSq}_{N,g}(2^\kappa - 2^{\kappa-i})$ for $i = \kappa-\ell, ..., \kappa$, and $u^*[\kappa-i] = S_i^2$ is a random quadratic residue in $\mathbb{Z}_N$, for $i = 1, ..., \kappa-\ell-1$. The derived time-line is constructed in the same way as in $\mathsf{Dist}^{\mathsf{R,F}}$. $\mathsf{Dist}^{\mathsf{F,F}} = \langle N, g, \vec{u}^*, h, \vec{w}, X\rangle$.

$\mathsf{Dist}^{\mathsf{F,U}}$**: Fake master time-line, unfaithful derived time-line.** Same as in $\mathsf{Dist}^{\mathsf{F,F}}$, except that $X = Q^2$ is a random quadratic residue in $\mathbb{Z}_N$, instead of $(u[\kappa-\ell-1])^\alpha$.

Note that $\mathsf{Dist}^{\mathsf{R,F}}$ and $\mathsf{Dist}^{\mathsf{R,U}}$ are the two distributions that $\mathcal{A}$ tries to distinguish in the lemma. So if we can prove that all the four distributions are indistinguishable, the lemma is proved. We do this in three steps, as follows:

$$\mathsf{Dist}^{\mathsf{R,F}} \overset{\text{(YMG-BBS)}}{\approx} \mathsf{Dist}^{\mathsf{F,F}} \overset{\text{(CDDH)}}{\approx} \mathsf{Dist}^{\mathsf{F,U}} \overset{\text{(YMG-BBS)}}{\approx} \mathsf{Dist}^{\mathsf{R,U}}$$

$\mathsf{Dist}^{\mathsf{R,F}}$ **and** $\mathsf{Dist}^{\mathsf{F,F}}$ **are indistinguishable to an** $\mathcal{A}$ **of running time** $\delta \cdot 2^\ell$**:** The difference between $\mathsf{Dist}^{\mathsf{R,F}}$ and $\mathsf{Dist}^{\mathsf{F,F}}$ is in the "early" points of the master time-line, namely, points $u[1]$, ..., $u[\kappa-\ell-1]$. Notice that each of these points are at distance at least 2^ℓ away from each other, and from the rest of the points in the time-line. A standard hybrid-argument reduces the indistinguishability between $\mathsf{Dist}^{\mathsf{R,F}}$ and $\mathsf{Dist}^{\mathsf{F,F}}$ to the YMG-BBS assumption.

$\mathsf{Dist}^{\mathsf{F,F}}$ **and** $\mathsf{Dist}^{\mathsf{F,U}}$ **are indistinguishable to any polynomial-time** $\mathcal{A}$**:** The only difference between $\mathsf{Dist}^{\mathsf{F,F}}$ and $\mathsf{Dist}^{\mathsf{F,U}}$ is the element X. In $\mathsf{Dist}^{\mathsf{F,F}}$, $X = (u^*[\kappa-\ell-1])^\alpha = S_{\kappa-\ell-1}^{2\alpha}$, while in $\mathsf{Dist}^{\mathsf{F,U}}$ $X = Q^2$ is a random quadratic residue. Notice that $S_{\kappa-t}$ and Q are both independent from the rest of the elements in their distributions, thus, one can reduce the indistinguishability between $\mathsf{Dist}^{\mathsf{F,F}}$ and $\mathsf{Dist}^{\mathsf{F,U}}$ to the indistinguishability between the tuples $(g, h, S_{\kappa-\ell-1}^2, S_{\kappa-\ell-1}^{2\alpha})$ from $\mathsf{Dist}^{\mathsf{F,F}}$ and $(g, h, S_{\kappa-\ell-1}^2, Q)$ from $\mathsf{Dist}^{\mathsf{F,U}}$, which in turn reduces to the CDDH-QR assumption.[25]

$\mathsf{Dist}^{\mathsf{F,U}}$ **and** $\mathsf{Dist}^{\mathsf{R,U}}$ **are indistinguishable to an** $\mathcal{A}$ **of running time** $\delta \cdot 2^\ell$**:** Same as in the first step, the indistinguishability is reduced to YMG-BBS via a hybrid argument. ∎

D Extending Protocol GradRel to the General Case

The GradRel protocol only works if all committed values are quadratic residues. To fake the commitment to a value x, $\mathcal{S}$ needs to fake a time-commitment z before knowing x. Then, after seeing x, $\mathcal{S}$

[25] We assume that Q and $S_{\kappa-\ell-1}^2$ are powers of g, which only fails with negligible probability.

needs to generate a time-line in the "reverse" direction. In particular, $\mathcal{S}$ needs to find the 2^{2^t-1}th roots of z/x for various values of t. So z/x needs to be a quadratic residue. In the simulation, z is chosen to be a quadratic residue, and thus x needs to be a quadratic residue as well.

We can fix the problem in the following way. Observe that -1 has Jacobi symbol $+1$, but is not a quadratic residue modulo N. Let V be an arbitrary element in $\mathbb{Z}_N^*$ with Jacobi symbol -1. Then for any $x \in \mathbb{Z}_N^*$, exactly one of the four elements $\{x, -x, xV, -xV\}$ is a quadratic residue. Consider a modified version of protocol GradRel that additionally contains V in the common reference string. When a party commits to a value x, it makes five timed commitments to x_1, x_2, x_3, x_4, y, where $\{x_1, x_2, x_3, x_4\}$ is a random permutation of $\{x, -x, xV, -xV\}$ and $y \in \{1, 4, 9, 16\}$ indicates which x_i is the x ($y = i^2$ means that $x_i = x$). Naturally, the party also needs to provide zero-knowledge proofs that these commitments are consistent. In the open phase, all five values are opened. Obviously, in the case of premature abort, the uncorrupted parties can still force-open all five commitments and recover x. Furthermore, $\mathcal{S}$ can simulate the commitments and the opening for any x. For the commitments to $\{x_1, x_2, x_3, x_4\}$, $\mathcal{S}$ generates random $\{z_1, z_2, z_3, z_4\}$ whose quadratic residuosity modulo p_1 and p_2 are a random permutation of $\{(+1, +1), (+1, -1), (-1, +1), (-1, -1)\}$. Then $\mathcal{S}$ generates a random quadratic residue w and the time-commitment for y, since y is always a quadratic residue. When receiving the actual value x, $\mathcal{S}$ can find out its quadratic residuosity modulo p_1 and p_2, and thus can find the correct permutation and fake the opening of $\{z_1, z_2, z_3, z_4\}$ to $\{x, -x, xV, -xV\}$, as well as the fake opening of w to one of the values in $\{1, 4, 9, 16\}$.

The modified GradRel protocol works for any inputs in $\mathbb{Z}_N^*$, and its communication complexity is only a constant times that of GradRel.