Resource Fairness and Composability of Cryptographic Protocols

Juan Garay[1], Philip MacKenzie[2], Manoj Prabhakaran[3], and Ke Yang[2]

[1] Bell Labs – Lucent Technologies. garay@research.bell-labs.com.
[2] Google. philmac@gmail.com, yangke@google.com.
[3] Computer Science Department, University of Illinois at Urbana-Champaign. mmp@uiuc.edu.

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, and has been investigated thoroughly over many years [54, 55, 38, 7, 18, 37]. Defining security is one of the first challenges in achieving this [37, 13, 48, 14, 40, 43, 3, 50]. The *universal composability* (UC) framework of Canetti [14] is among the models that provide perhaps the strongest security guarantees.

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."

In this paper we investigate a less studied aspect of multiparty computation, 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).[1] Clearly, fairness is a very desirable property for secure MPC protocols, and in fact, many of the security definitions cited above imply fairness. (See [40] 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, 18] and in the computational setting [38, 37] (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 [51] and computationally [38, 37].

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 [19, 37]. 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, 12] 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" [42].

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, 27, 30, 39, 4,

[1] This property is also known as "complete fairness," and can be contrasted with "partial fairness," where fairness is achieved only when there are certain restrictions on corruption of parties [40].

23]. 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, 36, 49]. As we discuss below, our constructions also use a gradual release mechanism secure against parallel attacks.

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

[2] Indeed, there have been considerable efforts in finding efficient exponentiation algorithms (e.g., [1, 53]) and still the best methods are sequential.

[3] In the original formulation of the UC framework [14], 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 [15] also has "immediate functionalities" as the default—see Section 2.1.

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.

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]

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

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 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.

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.

Our results. We now 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.

[5] In a previous version of this work [35], 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).

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* [31], 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.* [17] 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.* [21] 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.

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 Section 3 is 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 a composition theorem. In Section 3 we present the $\mathcal{F}_{\text{CPFO}}$ functionality and show a protocol that realizes a wrapped version of it, which we then use to achieve resource-fair MPC. Due to space limitations, proofs,

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

detailed remarks and extensions are omittied from this extended abstract and can be found in the full version of the paper [32].

2 FMPC Framework and Resource Fairness

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 [14, 15]. 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. Refer to [15] for a detailed presentation of the UC framework. The critical features of the FMPC framework are:

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).

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*

[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 $x^2 \bmod M$ on input (x, M)) in constant time.

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 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 [41, 45] 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 [40] (with a slight modification to the ideal abstraction of broadcasting, to allow for the round structure in our synchronous model).

Guaranteed-round message delivery from functionalities. Following the revised formulation of the UC framework [15], 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.*

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

[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.

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.

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 [14], 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 $(\mathsf{fairdeliver}, sid, 0, \{((\mathsf{output}, y), P_i)\}_{1 \leq i \leq n}, s)$, that is, set up a fair delivery of message $(\mathsf{output}, sid, y)$ to all parties for delivery in the sth round.

Fig. 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 $\mathcal{F}_f$

[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.

satisfies the intuitive definition of fairness for secure function evaluation. (This is called "complete fairness" in [40].) 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 [19], 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 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 $F(\mathit{msg\text{-}id})$ denote a $\mathsf{fairdeliver}$ 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 $\mathsf{unopened}$ 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

[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.

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

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 $(\mathsf{fairdeliver}, sid, \textit{msg-id}, \{(\mathsf{msg}_1, P_{i_1}), \ldots, (\mathsf{msg}_m, P_{i_m}), (\mathsf{msg}_{\mathcal{S}}, \mathcal{S})\}, j)$,[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 $(\mathsf{noinvest}, sid, \textit{msg-id})$, 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 $(\mathsf{invest}, sid, \textit{msg-id}, \alpha)$, then
  - the record with ID *msg-id* is marked as opened, and α is stored as $\alpha_{\textit{msg-id}}$. 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 $(\mathsf{dealoffer}, sid, \textit{msg-id}, P_i, \beta)$ 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 $(\mathsf{dealoffer}, sid, \textit{msg-id}, \beta')$ to P_i, where $\beta' = \min(\beta, \alpha_{\textit{msg-id}})$.
- If an honest party P_i responds to $(\mathsf{dealoffer}, sid, \textit{msg-id}, \beta)$ with $(\mathsf{dealaccept}, sid, \textit{msg-id}, \beta)$, 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.

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

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 $\alpha_{\textit{msg-id}}$ 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.

The following fact is easy to verify.

Fact 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.

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 wrapped functionality $\mathcal{W}(\mathcal{F})$), it is for ρ to decide when to grant resource requests made by π. ρ can grant resource requests only using resources it already has (which is either part of its running time, or part of resources granted to it by *its* environment). In the cases we consider, the outer protocol ρ will simply transfer resource requests it receives to *its* environment, and will transfer the resources granted to it back to the subroutine.
- **Resource requests granted by the environment.** We do not impose any restriction on the amount of resources that the environment can grant to the honest parties. In particular, the environment could grant a super-polynomial amount of resources to an honest party. This allows a wider class of environments for which the security guarantee holds. Jumping ahead, we point out that this does not render the system insecure, because of an extra condition that the entire system be simulatable in polynomial time, independent of the amount of resources granted by the environment. This requirement is captured in the definition of security using a device called the *full simulator* (see Definition 1).
- **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 $\mathsf{dealoffer}$ and $\mathsf{dealaccept}$ 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, 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 to $\mathcal{A}$. 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 exist 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 $\mathcal{Z}$.*

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

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

2. $\mathrm{HYB}^{\mathcal{W}_2}_{\rho,\mathcal{A},\mathcal{Z}} \approx \mathrm{FSIM}_{\mathcal{X}^{\mathcal{A}},\mathcal{Z}}$.

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 $\mathcal{F}_{\mathrm{CRS}}$, 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. *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 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 3. *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 [14] 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 1, 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 (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 $\lambda' = \lambda\ell$.

Corollary 1. *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* $\lambda'' = \lambda(\ell(\lambda' + 1))$.

3 Resource-Fair Protocols

3.1 The commit-prove-fair-open functionality

We first present the "commit-prove-fair-open" functionality $\mathcal{F}_{\text{CPFO}}$, and then show how to construct a protocol, 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 $\mathcal{F}^R_{\text{CPFO}}$

$\mathcal{F}^R_{\text{CPFO}}$ 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 deliverat message are received, output $(\mathsf{fairdeliver}, sid, 0, \{((\mathsf{DATA}, (x_1, x_2, ..., x_n)), P_i)\}_{1 \leq i \leq n} \cup \{((\mathsf{DATA}, (x_1, x_2, ..., x_n)), \mathcal{S})\}, s)$.

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

Functionality $\mathcal{F}_{\text{CPFO}}$ is similar to the "commit-and-prove" functionality $\mathcal{F}_{\text{CP}}$ in [17] 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 deliverat message to $\mathcal{F}_{\text{CPFO}}$. (Implicitly we assume that if the round number in the 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" [31] that will play an essential role in the construction of resource-fair protocols. Before doing that, we present the assumptions used by these protocols

Preliminaries for protocol 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. 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.*

The assumptions used in this paper are the *composite decisional Diffie-Hellman* assumption (CDDH) [10], the *decision composite residuosity* assumption (DCRA) [46], and a further refinement of the *generalized Blum-Blum-Shub* assumption (GBBS) [11], which we now state.[14]

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 $\boldsymbol{a}$ 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 $\mathsf{Dist}(x, \boldsymbol{a}) = S$, where $\mathsf{Dist}(x, \boldsymbol{a})$ denotes the minimal absolute difference between x and elements in $\boldsymbol{a}$. (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 $\boldsymbol{u}$ 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 $\mathcal{A}$,

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

In this paper we present protocols that work in the CRS model and in 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

[14] Refer to [32] for remarks on the differences between the version presented here and the original one.

infrastructure (PKI) model is stronger. Upon initial activation, a PKI functionality, $\mathcal{F}_{\rm PKI}$, 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.

Time-lines. We 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. *Let κ be a security parameter. A* decreasing time-line *is a tuple $L = \langle N, g, \boldsymbol{u}\rangle$, 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 $\boldsymbol{u}$ 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 $\boldsymbol{u}$ the* points *in L, and $u[\kappa]$ the* end point *in L.*

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 [31] 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 5. *Let $L = \langle N, g, \boldsymbol{u}\rangle$ and $L' = \langle N, h, \boldsymbol{v}\rangle$ 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. 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 state in the following lemma, assuming CDDH and GBBS, this pseudorandomness remains true if $\mathcal{A}$ knows L, but not the shifting factor α.

Lemma 1 (Strong Pseudorandomness). *Let $L = \langle N, g, \boldsymbol{u}\rangle$ be a randomly generated decreasing time-line and $L' = \langle N, h, \boldsymbol{v}\rangle$ be a time-line derived from L with random shifting factor α. Let κ and δ be as in the GBBS assumption. Let*

$\boldsymbol{w}$ be the vector containing the last $(\ell+1)$ elements in $\boldsymbol{v}$, i.e., $\boldsymbol{w} = (v[\kappa-\ell], v[\kappa-\ell+1], ..., v[\kappa])$. 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 $\mathcal{A}$,

$$\left|\Pr[\mathcal{A}(N, g, \boldsymbol{u}, h, \boldsymbol{w}, v[\kappa-\ell-1]) = 1] - \Pr[\mathcal{A}(N, g, \boldsymbol{u}, h, \boldsymbol{w}, R^2) = 1]\right| \leq \epsilon(\kappa). \tag{3}$$

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 introduced above. We use the multi-session version of the "one-to-many" $\hat{\mathcal{F}}_{\text{ZK}}$ functionality from [17], which is shown in Figure 4.[15] In particular, we need the $\hat{\mathcal{F}}_{\text{ZK}}$ functionality for the following relations.

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

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

Fig. 4. *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: $\hat{R}((M, g, h, w, z, y), \alpha) = (h = g^\alpha \bmod M) \wedge R(y, z/w^\alpha \bmod M)$.

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 $L = \langle N, g, \boldsymbol{u}\rangle$. To commit to a value x_i, party P_i derives a new time-line $L_i = \langle N, g_i, \boldsymbol{v_i}\rangle$, 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 $\hat{\mathcal{F}}_{\text{ZK}}^{\mathsf{DL}}$) 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.

[15] In [17] the framework used is that originally presented in [14]. However, since we are using the modified version from [15], we modify the functionality $\hat{\mathcal{F}}_{\text{ZK}}$ by explicitly allowing the adversary to block messages from the functionality to the parties.

The prove phase is directly handled by the $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\hat{R}}$ 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 $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DH}}$), 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 $\hat{\mathcal{F}}_{\mathrm{ZK}}$ 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 5. 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 $\mathbb{Z}_N^*$. In [32] we discuss 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.

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

Theorem 3. *Assume that GBBS and CDDH hold. Then protocol* GradRel *securely realizes the ideal functionality $\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}}^{R})$ with $O(n)$-investment in the $(\mathcal{F}_{\mathrm{CRS}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DL}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DH}}, \hat{\mathcal{F}}_{\mathrm{ZK}}^{\hat{R}})$-hybrid model, assuming static corruptions.*

Refer to [32] for the proof of this theorem. Here 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 longer 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 $\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}})$ 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 $\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}})$ (and hence in those rounds the simulation is perfect).
- At this point a deal is still not offered by $\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}})$ 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 $\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}})$

Protocol $\mathsf{GradRel}^R$

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

Round 1 **(commit phase)** For each party P_i, $1 \le i \le n$, 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 $L_i = \langle N, g_i, \boldsymbol{v_i}\rangle$.
2. Set $z_i \leftarrow v_i[\kappa] \cdot x_i = (u[\kappa])^{\alpha_i} \cdot x_i \bmod N$ and broadcast message (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 $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DL}}$ functionality.

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

Round 2 **(prove phase)** For each party P_i, $1 \le i \le n$, 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 $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\hat{R}}$ functionality.
2. After receiving messages $(\mathsf{ZK\text{-}PROOF}, sid, 0, P_i, (N, g, g_i, u[\kappa], z_i, y_i))$ from $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\hat{R}}$, 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, $1 \le i \le n$, do:

1. Broadcast (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 $\hat{\mathcal{F}}_{\mathrm{ZK}}^{\mathsf{DH}}$.
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 $1 \le j \le n$, output $(\mathsf{DATA}, sid, x_1, x_2, ..., x_n)$ and terminate.

Panic mode: For each party P_i, $1 \le i \le n$, 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.

Fig. 5. *Protocol* $\mathsf{GradRel}$, *running in the CRS model in* $(\kappa + 2)$ *rounds.*

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 $\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}})$, 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 $\mathcal{W}(\mathcal{F}_{\mathrm{CPFO}})$. 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.

By "plugging in" the UCZK protocol from [17] into protocol GradRel, we have the following corollary.

Corollary 2. *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.*

3.2 Resource-fair multi-party computation

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., [21, 17, 25]) 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.

In the full paper, we present two constructions that convert the MPC protocols of Canetti *et al.* [17] and Cramer *et al.* [21] into resource-fair MPC protocols. Here we state the results.

Theorem 4. *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 $(\mathcal{F}_{\mathrm{CRS}}, \mathcal{W}(\mathcal{F}_{\mathrm{CPFO}}))$-hybrid model in the FMPC framework, assuming static corruptions.*

Corollary 3. *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 $\mathcal{F}_{\mathrm{CRS}}$-hybrid model in the FMPC framework, assuming static corruptions.*

Theorem 5. *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 $(\mathcal{F}_{\mathrm{PKI}}, \mathcal{W}(\mathcal{F}_{\mathrm{CPFO}}))$-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.*

Acknowledgements. We thank Amit Sahai for helpful discussions on the formulation of the notion of resource fairness, and Yehuda Lindell and Jesper Nielsen, as well as the anonymous reviewers for *TCC '06* for their many helpful comments.

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*, LNCS 1880, pp. 236–254, Springer-Verlag, 2000.
12. C. Cachin and J. Camenisch. Optimistic Fair Secure Computation. In *Advances in Cryptology—CRYPTO '00*, LNCS 1880, pp. 93–111, Springer-Verlag, 2000.
13. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143-202, Winter 2000.
14. 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.

15. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2005. Revised version of [14].
16. R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO 2001*, LNCS 2139, pp. 19–40, 2001.
17. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally Composable Two-party and Multi-party Secure Computation. In *34th STOC*, 2002.
18. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th STOC*, pp. 11–19, 1988.
19. 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)*, pp. 364-369, 1986.
20. R. Cramer. Modular Design of Secure yet Practical Cryptographic Protocols. Ph.D. Thesis. CWI and University of Amsterdam, 1997.
21. R. Cramer, I. Damgård, and J. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption In *Advances in Cryptology - EuroCrypt 2001 Proceedings*, LNCS 2045, pp. 280–300, Springer-Verlag, 2001.
22. 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, pp. 174–187, 1994.
23. I. Damgård. Practical and Provably Secure Release of a Secret and Exchange of Signatures. In *Journal of Cryptology* 8(4), pp. 201–222, 1995.
24. 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.
25. I. Damgård, and J. Nielsen. Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption. In *Advances in Cryptology - CRYPTO '03*, 2003.
26. 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.
27. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.
28. M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein and A. Smith. Detectable Byzantine Agreement Tolerating Faulty Majorities (from scratch). In *21st PODC*, pp. 118–126, 2002.
29. P. Fouque, G .Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *Proceedings of Financial Crypto 2000*, 2000.
30. 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.
31. J. Garay and M. Jakobsson. Timed Release of Standard Digital Signatures. In *Financial Cryptography '02*, LNCS 2357, pp. 168–182, Springer-Verlag, 2002.
32. J. Garay, P. MacKenzie, M. Prabhakaran and K. Yang. Resource Fairness and Composability of Cryptographic Protocols. In Cryptology ePrint Archive, `http://eprint.iacr.org/2005/370`.
33. J. Garay, P. MacKenzie and K. Yang. Strengthening Zero-Knowledge Protocols using Signatures. In *Advances in Cryptology – Eurocrypt 2003*, LNCS 2656, pp.177-194, 2003. Full version in Cryptology ePrint Archive, `http://eprint.iacr.org/2003/037`, 2003. To appear in *Journal of Cryptology.*

34. 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.
35. 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`.
36. J. Garay and C. Pomerance. Timed Fair Exchange of Standard Signatures. In *Financial Cryptography 2003*, LNCS 2742, pp. 190–207, Springer-Verlag, 2003.
37. O. Goldreich. Secure Multi-Party Computation (Working Draft, Version 1.2), March 2000. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.
38. 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.
39. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority, In *CRYPTO '90*, pp. 77-93, Springer-Verlag, 1991.
40. S. Goldwasser and Y. Lindell. Secure Computation Without Agreement. In *Journal of Cryptology*, 18(3), pp. 247-287, 2005.
41. 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.
42. M. Lepinski, S. Micali, C. Peikert, and A. Shelat. Completely fair SFE and coalition-safe cheap talk. In *23rd PODC*, pp. 1–10, 2004.
43. Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation.In *FOCS 2003*.
44. P. MacKenzie and K. Yang. On Simulation Sound Trapdoor Commitments. In *Advances in Cryptology–Eurocrypt '04*, pp.382–400, 2004.
45. J. B. Nielsen. On Protocol Security in the Cryptographi Model. Ph.D. Thesis. Aarhus University, 2003.
46. P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *Advances in Cryptology–Eurocrypt '99*, pp.223–238, 1999.
47. T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology – CRYPTO '91*, LNCS 576, 129–140, Springer-Verlag, 1991.
48. 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.
49. B. Pinkas. Fair Secure Two-Party Computation. In *Eurocrypt 2003*, pp. 87–105, 2003.
50. M. Prabhakaran and A. Sahai. New notions of security: Achieving universal composability without trusted setup. Cryptology ePrint Archive, Report 2004/139. Extended abstract in Proc. 36th STOC, pp. 242–251, 2004.
51. T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *21st STOC*, pp. 73–85, 1989.
52. V. Shoup. A Computational Introduction to Number Theory and Algebra. *Preliminary book, available at* `http://shoup.net/ntb/`.
53. J. Sorenson. A Sublinear-Time Parallel Algorithm for Integer Modular Exponentiation. Available from `http://citeseer.nj.nec.com/sorenson99sublineartime.html`.
54. A. Yao. Protocols for Secure Computation. In *FOCS 1982*, pp. 160–164, 1982.
55. A. Yao. How to generate and exchange secrets. In *FOCS 1986*, pp. 162–167, 1986.