Fair Distributed Computation of Reactive Functions[⋆]

Juan Garay[1], Björn Tackmann[2,⋆⋆], and Vassilis Zikas[3,⋆⋆⋆]

[1] Yahoo Labs, Sunnyvale, CA, USA
garay@yahoo-inc.com
[2] UC San Diego, Computer Science and Engineering, San Diego, CA, USA
btackmann@eng.ucsd.edu
[3] ETH Zurich, Department of Computer Science, Zurich, Switzerland
vzikas@inf.ethz.ch

Abstract. A *fair* distributed protocol ensures that dishonest parties have no advantage over honest parties in learning their protocol's output. What makes fairness a particularly intriguing research topic is Cleve's seminal result [STOC'86], which proved that fairness is impossible to achieve in the presence of dishonest majorities and ignited a quest for more relaxed, yet meaningful definitions of fairness. A common pattern in existing works, however, is that they only treat the case of *non-reactive* computation—i.e., distributed computation of "one-shot" (stateless) functions, in which parties give all inputs strictly before any output is computed. Yet, many natural cryptographic tasks are of a *reactive* (stateful) nature.

In this work, we introduce the first notion of fairness tailored to reactive distributed computation, which can be realized in the presence of dishonest majorities. Our definition builds on the recently suggested utility-based fairness notion (for non-reactive functions) by Garay *et al.* [PODC'15], which, informally, measures the protocol's fairness by means of the utility of an adversary who aims to break it. As in the [PODC'15] work, our approach enjoys the advantage of offering a comparative notion, inducing a partial order on protocols with respect to fairness.

We investigate protocols that restrict the adversary's utility and provide, for each choice of parameters specifying this utility, a protocol for fair and reactive two-party computation, which is optimal for a (natural) range of parameters. Our study shows that achieving fairness in the reactive setting is more complex than in the much-studied case of one-shot functions, as increasing the number of rounds used for reconstructing the output can lead to improved fairness, and the minimal required number of rounds depends on the *exact values* of the adversary's utility.

Keywords: Cryptographic protocols · secure multi-party computation · fairness · game theory

⋆ The full version of this paper can be found in [13].

⋆⋆ Research done in part while at ETH Zurich, and partly supported by the SNF through Fellowship no. P2EZP2-155566 and by NSF grant CNS-1228890.

⋆⋆⋆ Research supported in part by the SNF through Ambizione grant PZ00P-2142549.

1 Introduction

In secure multi-party computation (MPC) [19, 14], a set of n parties wish to perform some joint computation on their inputs in a secure manner, despite the arbitrary behavior of some of them. The basic security requirements are *privacy* (cheating parties learn only their output of the computation) and *correctness* (cheaters cannot distort the outcome of the computation). An additional desired property is *fairness*, which, roughly speaking, requires that the protocol does not give a cheating party any advantage in learning the output of the computation over the honest parties.

In traditional cryptographic definitions, the worst-case scenario of collaborative cheating is captured by the notion of a (central) adversary. Informally, the adversary is an entity which takes control of ("corrupts") parties and then uses them to attack the computation. Unfortunately, an early impossibility result by Cleve [8] established that with such an adversary it is impossible to achieve all three properties—correctness, privacy and fairness—simultaneously, unless there is a majority of *honest* (i.e., uncorrupted) parties.

Following Cleve's impossibility, much work has focused on achieving meaningful weaker notions of fairness. One main example of this are *gradual release*-type approaches [4, 2, 9, 5, 18, 12], in which parties take turns in releasing bits of information. More recently, Asharov *et al.* [1] suggested a definition of fairness for the case of two parties using ideas from so-called "rational cryptography," where all the protocol participants are modeled as rational players aiming to maximize a given utility function, and presented a gradual-release-based protocol satisfying their definition. This rational model for fairness was later enhanced and extended in various ways (e.g., arbitrary instead of fail-stop misbehavior, ideal-world/real-world definition) by Groce and Katz [16].

All of these weaker notions of fairness, however, are formalized in an "all-or-nothing" manner, in the sense that either a protocol achieves the respective security definition, or the notion renders the protocol unfair and makes no further statement about it. For example, this is the case for *resource fairness* [12], which formalizes the intuition of the *gradual release* paradigm [4, 2, 9, 5, 18] in a simulation-based framework. Indeed, a resource-fair protocol should ensure that, upon abort by the adversary, the amount of computation that the honest party needs for producing the output is comparable to the adversary's for the same task; yet, a protocol that achieves a worse ratio between the amount of work required by the honest party and the adversary is not distinguished from a fully unfair one. The same holds for the above fairness definitions in rational cryptography, which require the protocol to be an equilibrium strategy with respect to a preference/utility function for *curious-but-exclusive* agents, where each agent prefers learning the output to not learning it, but would rather be the only one that learns it. We remark, though, that some of these frameworks do offer completeness results, in the sense that they show that one *can* construct protocols that are fair in the respective notions; nevertheless, none of them provides a comparative statement for protocols which do not fully satisfy their property.

Recent work by Garay *et al.* [11] introduced a *quantitative* approach to fairness, based on the idea that one can use an appropriate utility function to express the preferences of an adversary who wants to break fairness.[4] The approach allows for comparing protocols with respect to how fair they are, placing them in a partial order according to a relative-fairness relation. Previously, the only other notion providing any sort of comparative statement was that of $1/p$-*security* (aka. "partial fairness") [15, 3], where security is given up with probability $1/p$ for some polynomial p, but which does not always guarantee privacy and correctness (see [11] for a detailed comparison).

Technically, the approach of [11] builds on machinery developed in the recently proposed *Rational Protocol Design* (RPD) framework of Garay *et al.* [10]. In more detail, the framework describes how to design protocols which keep the utility of an attacker aiming at provoking certain security breaches as low as possible. At a high level, this is then used as follows: first, one specifies the class of utility functions that naturally capture an adversary attacking a protocol's fairness, and then one interprets the actual utility that the best attacker (i.e., the one maximizing its utility) obtains against a given protocol as a measure of the protocol's success in satisfying the property. The more a protocol limits its best attacker with respect to the fairness-specific utility function, the fairer the protocol is. We remark that, in addition, this quantitative fairness approach preserves the composability of the underlying security model (such as when using, e.g., [6, 7]) with respect to standard secure protocols, in the sense that it allows the replacement of an ideal component (a "hybrid" or ideal functionality in the language of [7]) in a fair/optimal protocol by a protocol which securely implements it without affecting its fairness/optimality.

Our contributions. We present the first notion of fairness tailored to *reactive* distributed computation, where parties provide inputs and receive outputs multiple times during the course of the computation; the notion can be realized in the presence of dishonest majorities.

We specify the utility function characterizing the incentives of an attacker who aims at breaking fairness of a two-party MPC protocol, deriving the natural quantitative notions of fairness and of protocol optimality. However, and as expected, formulation and analysis are quite more complex here than in the non-reactive case [11], where for example the honest parties can simply restart the protocol after an "early abort" where no party received outputs, using default inputs for the parties that caused the abort. In contrast, in the reactive case earlier rounds in the computation may already have leaked information to the adversary, which makes a restart potentially unsafe. As a result, the protocol we present bounds the adversary's utility by the maximum of two terms, one of which is the same as in the non-reactive case and corresponds to the adversary's strategy of aborting right after obtaining its output, and the other one stems from the potential "early aborts" and depends on the number of rounds used

[4] This approach is incomparable to the one in rational cryptography, as the honest parties are *not* rational and follow whichever protocol is designed for them.

in the reconstruction of the protocol output as well as the exact values of the adversary's utility.

We then derive lower bounds, showing the protocol optimally fair for a natural class of parameter values—at a high level, those expressing that the adversary prefers that the honest party does not get the output, to the extent that he is willing to have negative utility when all parties receive the output (otherwise previous results apply), but only up to a point, after which the adversary's aversion toward giving the output to the honest parties is so large that he will abort any protocol prematurely. Besides being optimally fair, the protocol is also optimal with respect to the number of *reconstruction rounds*. For the remaining values, the lower bound we derive is close to the bound achieved by our protocol but not tight; we leave the closing of this gap as an open problem.

Organization of the paper. The remainder of the paper is organized as follows. In Section 2 we describe notation and the very basics of the RPD framework [10] that are needed for understanding and evaluating our results. In Section 3 we define the utility function of attackers who aim at violating fairness, which enables the relative assessment of protocols as well as the notions of "optimal" fairness which we use in this work. This section is a generalization of the approach in [11] to the reactive computation case. Section 4 is dedicated to the fair reactive protocol, starting with a general outline in Section 4.1, and Section 4.2 explaining the protocol in detail; lower bounds are shown in Section 4.3.

2 Preliminaries and Model

We first establish some notational conventions. For an integer $m \in \mathbb{N}$, the set of positive numbers smaller or equal to m is $[m] := \{1, \ldots, m\}$. In the context of two-party protocols, we will always refer to the parties as p_1 and p_2, and for $i \in \{1, 2\}$ the symbol $\neg i$ refers to the value $3 - i$ (so $p_{\neg i} \neq p_i$). Most statements in this paper are actually asymptotic with respect to an (often implicit) security parameter $k \in \mathbb{N}$. Hence, $f \leq g$ means that $\exists k_0 \; \forall k \geq k_0 : f(k) \leq g(k)$, and a function $\mu : \mathbb{N} \to \mathbb{R}$ is *negligible* if for all polynomials p, $\mu \leq 1/p$, and *noticeable* if there exists a polynomial p with $\mu \geq 1/p$. We further introduce the symbol $f \overset{\text{negl}}{\approx} g$ to denote that $\exists$ negligible μ : $|f - g| \leq \mu$, and $f \overset{\text{negl}}{\geq} g$ to denote $\exists$ negligible μ : $f \geq g - \mu$, with $\overset{\text{negl}}{\leq}$ defined analogously.

For the model of computation and protocol composition, we follow Canetti's adaptive simulation-based model for multi-party computation [6]. The protocol execution is formalized by collections of interactive Turing machines (ITMs); the set of all *efficient* ITMs is denoted by $\texttt{ITM}$. We generally denote our protocols by Π and our (ideal) functionalities (which are also referred to as *the trusted party* [6]) by $\mathcal{F}$ both with descriptive super- or subscripts, the adversary by $\mathcal{A}$, the simulator (aka the ideal-world adversary) by $\mathcal{S}$, and the environment by $\mathcal{Z}$. The random variable ensemble $\{\textsc{exec}_{\Pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k \in \mathbb{N}, z \in \{0,1\}^*}$, which is more compactly often written as $\textsc{exec}_{\Pi,\mathcal{A},\mathcal{Z}}$, describes the contents of $\mathcal{Z}$'s output tape after an execution with Π, $\mathcal{F}$, and $\mathcal{A}$, on auxiliary input $z \in \{0, 1\}^*$.

Secure computation of reactive functions. The framework in [6] considers synchronous protocols with guaranteed termination and allows for sequential and modular composition, but lacks a formal definition of computation of reactive functions.[5] We describe here the real-world/ideal-world experiments for reactive functions based on the model with adaptive adversaries [6, Sect. 5]. Although we will be designing protocols only for two-party computation (2PC), since this is the first formal treatment of the reactive setting with respect to fairness, we provide definitions for the more general case of n parties. The resulting model allows for modular composition in a similar sense as in [6]: in each round of a protocol, the parties can make use of a sub-protocol computing another functionality. For the reduction to work, it is important that the higher-level protocol does not continue—apart from interacting with the sub-protocol—until the sub-protocol has terminated.

As discussed in [17], reactive computation can be seen as an ordered sequence of computations of non-reactive functions that can maintain a joint (private) state. More concretely, reactive computation is specified by a vector of (probabilistic) functions $\boldsymbol{f} = (f_1, \ldots, f_m)$, where each $f_\lambda \in \boldsymbol{f}$ takes as input a vector of values from $\{0,1\}^* \cup \{\perp\}$ (corresponding to the parties inputs to f_λ), a uniformly random value r from a known domain R (corresponding to the random coins used for evaluating f_λ), and a *state vector* $\boldsymbol{S}_\lambda \in ((\{0,1\}^* \cup \{\perp\})^n \times R)^{(\lambda-1)}$, which includes the inputs and random coins used for the evaluation of functions $f_1, \ldots, f_{\lambda-1}$. Each $f_\lambda \in \boldsymbol{f}$ outputs a vector of strings $\boldsymbol{y}_\lambda = (y_{1,\lambda}, \ldots, y_{n,\lambda}) \in \{0,1\}^n$, where $y_{i,\lambda}$ is p_i's output.

The ideal process. At a high level, execution in the ideal world is similar to the corresponding experiment in [6], but instead of a single function, the trusted third party (TTP, or "functionality") $\mathcal{F}_{\text{RC}}^{\boldsymbol{f}}$ is parameterized by the vector $\boldsymbol{f} = (f_1, \ldots, f_m)$ of functions to be sequentially evaluated, with each of these functions receiving as input the state vector (consisting of all inputs received so far as well as the used randomness) along with parties' inputs to the function which is currently computed. The output of the computation is taken to be the vector of outputs of all functions in $\boldsymbol{f}$.

The ability to maintain a joint state, however, is not the only difference between reactive and non-reactive computation. Rather, we need to ensure that parties be able to choose their input for any $f_\lambda, \lambda \in [m]$, depending on inputs and outputs from the evaluation of $f_1, \ldots, f_{\lambda-1}$. Thus, we cannot fix the input sequence of the parties at the beginning of the protocol execution as is the case with the ideal-evaluation experiment of non-reactive functions. Instead, we assume that every party $p_i \in \mathcal{P}$ gives as input to the trusted party a sequence of m *input-deciding functions* $\text{Inp}_i^1, \ldots, \text{Inp}_i^m$, where for each $\lambda \in [m]$, $\text{Inp}_i^\lambda : ((\{0,1\}^*)^{\lambda-1})^2 \to \{0,1\}^*$ is a function that on input the inputs and outputs from the evaluation of functions $f_1, \ldots, f_{\lambda-1}$ computes the input for the evaluation of

[5] Our definitions can be extended to Universally Composable (UC) security [7] using the approach of Katz *et al.* [17] to model terminating synchronous (reactive) computation in UC.

f_λ. (Without loss of generality, assume that p_i's input to f_1 is $\mathrm{Inp}_i^1(0,0)$.) Unlike the parties, the simulator is allowed to to choose his inputs during his ongoing interaction with the TTP.

The real-world execution. The real-world experiment in analogous to the corresponding experiment in [6], where the input of each party p_i is his input-deciding function vector $\mathrm{Inp}_i{}^1, \ldots, \mathrm{Inp}_i^m$.

Rational Protocol Design. Our results utilize the *Rational Protocol Design* (RPD) framework [10]. Here we review the basic elements that are needed to motivate and express our definitions and results; we refer to the framework paper [10] for further details. In RPD, security is defined via a two-party sequential zero-sum game with perfect information, called the *attack game*, between a protocol *designer* $\mathtt{D}$ and an *attacker* $\mathtt{A}$. The designer $\mathtt{D}$ plays first by specifying a protocol Π for the (honest) participants to run; subsequently, the attacker $\mathtt{A}$, who is informed about $\mathtt{D}$'s move (i.e., learns the protocol) plays by specifying a polynomial-time attack strategy $\mathcal{A}$ by which it may corrupt parties and try to subvert the execution of the protocol (uncorrupted parties follow Π as prescribed). It suffices to define the utility $u_{\mathtt{A}}$ of the adversary as the game is zero-sum; the utility $u_{\mathtt{D}}$ of the designer is then $-u_{\mathtt{A}}$.

In RPD, the definition of utilities relies on the simulation paradigm[6], with the caveat that the real-world execution is compared to an ideal process in which $\mathcal{S}$ gets to interact with a *relaxed* version of the functionality which, in addition to implementing the task as $\mathcal{F}$ would, also allows the simulator to perform the attacks we are interested in capturing. For example, an attack to the protocol's correctness is modeled by the functionality allowing the simulator to modify the outputs (even of honest parties). Given such a functionality, the utility of any given adversary is defined as the expected utility of the best simulator for this adversary, where the simulator's utility is defined according to which weaknesses of the ideal functionality the simulator is forced to exploit.

3 Utility-based Fairness and Protocol Optimality

In this section, we utilize the RPD machinery to introduce a natural fairness relation (partial order) to the space of efficient protocols for secure reactive two-party computation (2PC) and define maximal elements in this order to be optimal protocols with respect to fairness. Towards that goal, we follow the three-step process described in [10, 11] for specifying an adversary's utility, instantiating this process with parameters that capture a fairness-targeted attacker.

Step 1: Relaxing the ideal experiment to allow attacks on fairness. First, we relax the ideal world to allow the simulator to perform fairness-related attacks. In particular, we consider the ideal-world experiment for reactive MPC

[6] In RPD the statements are formalized in Canetti's Universal Composition (UC) framework [7]; however, one can use any other simulation-based model, in particular the one in [6] described above.

described in Sect. 2 but modify it to allow the simulator $\mathcal{S}$ to (1) refuse receiving his inputs from the functionality and/or (2) refuse the functionality to deliver outputs to the parties (i.e., instruct it to abort); analogously to [11], the simulator is allowed to choose when to abort, i.e., before or after receiving his inputs if he chooses to. The reactive MPC ideal functionality is parameterized by the (sequence of) functions $\boldsymbol{f} = (f_1, \ldots, f_m)$ as described in Sect. 2 and is denoted $\mathcal{F}_{\text{RC}}^{\boldsymbol{f},\perp}$ (or simply $\mathcal{F}_{\text{RC}}^{\perp}$ if the function sequence is clear from the context). We point out that when $\mathcal{F}_{\text{RC}}^{\cdot,\perp}$ is parameterized with a single function (as in $\mathcal{F}_{\text{RC}}^{f,\perp}$) then it corresponds to the standard SFE functionality $\mathcal{F}_{\text{SFE}}^{f,\perp}$ (i.e., computation of non-reactive functions) with unfair abort as in [11].

Step 2: Events and payoffs. Next, we specify a set of events in the experiment corresponding to the ideal evaluation of $\mathcal{F}_{\text{RC}}^{\perp}$ which capture whether or not a fairness breach occurs, and assign to each such event a "payoff" value capturing the severity of provoking the event. The relevant questions to ask with respect to fairness are:

1. Does the adversary learn "noticeable" information about the output of the corrupted parties?
2. Do honest parties learn their output?

In comparison to the non-reactive case, there are *a priori* different ways to define the events, based on whether one asks for the adversary to receive *any* output or *all* the outputs. Since the reactive computation proceeds round by round, a natural choice is to ask for the honest parties to receive *all* outputs, or otherwise to ask for the adversary to also not receive information about *some* output. The corresponding events (which we use to describe fairness) correspond to the four possible combinations of answers to the above questions. In particular, we define the events indexed by a string $ij \in \{0,1\}^2$, where i (resp., j) equals 1 if the answer to the first (resp., second) question is yes and 0 otherwise. The events are then as follows:

E_{00}^{R}: The simulator does not ask $\mathcal{F}_{\text{RC}}^{f,\perp}$ for the all of the corrupted party's outputs and instructs $\mathcal{F}_{\text{RC}}^{f,\perp}$ to abort. This corresponds to neither the honest party nor the adversary receiving all their outputs.

E_{01}^{R}: The simulator does not ask $\mathcal{F}_{\text{RC}}^{f,\perp}$ for all of the corrupted party's outputs and does not instruct it to abort. This corresponds to the honest party receiving all its outputs and the adversary not receiving some of its outputs. This accounts also for the case where no party is corrupted.

E_{10}^{R}: The simulator asks $\mathcal{F}_{\text{RC}}^{f,\perp}$ for all his outputs and instructs it to abort before the honest party receives all its outputs. This corresponds to the adversary receiving all its outputs and the honest party not receiving some of its outputs.

E_{11}^{R}: The simulator asks the functionality for all his outputs, and allows the honest party to receive all its outputs (i.e., it does not abort). This accounts also for the case where all parties are corrupted.

We remark that our definition does not give full advantage to an adversary corrupting both parties. This is consistent with the intuitive notion of fairness, as when there is no honest party, the adversary has nobody to gain an unfair advantage over.

To each of the events E^{R}_{ij} we associate a real-valued *payoff* γ_{ij} which captures the adversary's utility when provoking this event. Thus, the adversary's payoff is specified by vector $\boldsymbol{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \mathbb{R}^4$, corresponding to events $\boldsymbol{E}^{\mathrm{R}} = (E^{\mathrm{R}}_{00}, E^{\mathrm{R}}_{01}, E^{\mathrm{R}}_{10}, E^{\mathrm{R}}_{11})$.

Finally, we define the expected payoff of a given simulator $\mathcal{S}$ (for an environment $\mathcal{Z}$) to be[7]:

$$U_I^{\mathcal{F}^{\perp}_{\mathrm{RC}},\boldsymbol{\gamma}}(\mathcal{S},\mathcal{Z}) \quad := \quad \sum_{i,j\in\{0,1\}} \gamma_{ij} \Pr[E^{\mathrm{R}}_{ij}]. \tag{1}$$

Step 3: Defining the attacker's utility. Given $U_I^{\mathcal{F}^{\perp}_{\mathrm{RC}},\boldsymbol{\gamma}}(\mathcal{S},\mathcal{Z})$, the utility $u_{\mathtt{A}}(\Pi,\mathcal{A})$ for a pair $(\Pi,\mathcal{A})$ of a protocol Π and an adversary $\mathcal{A}$ is defined following the methodology in [10] as the expected payoff of the *best* simulator[8] that simulates $\mathcal{A}$ in the $\mathcal{F}^{\perp}_{\mathrm{RC}}$-ideal world in presence of the least favorable environment—i.e., the one that is *most* favorable to the attacker. To make the payoff vector $\boldsymbol{\gamma}$ explicit, we sometimes denote the above utility as $\hat{U}^{\Pi,\mathcal{F}^{\perp}_{\mathrm{RC}},\boldsymbol{\gamma}}(\mathcal{A})$ and refer to it as the *payoff of strategy* $\mathcal{A}$ (for attacking Π).

More formally, for a protocol Π, denote by $\mathtt{SIM}_{\mathcal{A}}$ the class of simulators for $\mathcal{A}$, i.e, $\mathtt{SIM}_{\mathcal{A}} = \{\mathcal{S} \in \mathtt{ITM} \mid \forall \mathcal{Z} : \textsc{exec}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \textsc{exec}_{\mathcal{F}^{\perp}_{\mathrm{RC}},\mathcal{S},\mathcal{Z}}\}$. The payoff of strategy $\mathcal{A}$ (for attacking Π) is then defined as:

$$u_{\mathtt{A}}(\Pi,\mathcal{A}) \quad := \quad \hat{U}^{\Pi,\mathcal{F}^{\perp}_{\mathrm{RC}},\boldsymbol{\gamma}}(\mathcal{A}) \quad := \quad \sup_{\mathcal{Z}\in\mathtt{ITM}} \inf_{\mathcal{S}\in\mathtt{SIM}_{\mathcal{A}}} \{U_I^{\mathcal{F}^{\perp}_{\mathrm{RC}},\boldsymbol{\gamma}}(\mathcal{S},\mathcal{Z})\}. \tag{2}$$

To complete our formulation, we now describe a natural relation among the values in $\boldsymbol{\gamma}$ which is both intuitive and consistent with existing approaches to fairness, and which we will assume to hold for the remainder of the paper. Specifically, we will consider attackers whose least preferred event is that the honest parties receive their output while the attacker does not, i.e., we assume that $\gamma_{01} = \min_{\gamma\in\boldsymbol{\gamma}}\{\gamma\}$. Furthermore, we will assume that the attacker's favorite choice is that he receives the output and the honest parties do not, i.e., $\gamma_{10} = \max_{ij\in\{0,1\}^2}\{\gamma_{ij}\}$. Lastly, we point out that for an arbitrary payoff vector $\boldsymbol{\gamma}$, one can assume without loss of generality that any one of its values equals zero, and, therefore, we can set $\gamma_{00} = 0$. This can be seen immediately by setting $\gamma'_{ij} = \gamma_{ij} - \gamma_{01}$. We denote the set of all payoff vectors adhering to the above restrictions by $\Gamma_{\mathrm{fair}} \subseteq \mathbb{R}^4$. Summarizing, our fairness-specific payoff ("preference") vector $\boldsymbol{\gamma}$ satisfies

$$0 = \gamma_{01} \leq \min\{\gamma_{00},\gamma_{11}\} \quad \text{and} \quad \max\{\gamma_{00},\gamma_{11}\} < \gamma_{10}.$$

[7] Refer to [10, Sect. 2] for the rationale behind this formulation.

[8] The best simulator is taken to be the one that minimizes his payoff [10].

Optimally fair protocols. We are now ready to define our partial order relation for protocols with respect to fairness. Informally, a protocol Π will be *at least as fair* as another protocol Π' if the utility of the best adversary $\mathcal{A}$ attacking Π (i.e, the adversary which maximizes $u_{\mathtt{A}}(\Pi, \mathcal{A})$) is no larger than the utility of the best adversary attacking Π' (except for some negligible quantity).

Definition 1. *Let Π and Π' be protocols, and $\boldsymbol{\gamma} \in \Gamma_{\text{fair}}$ be a preference vector. We say that Π is* at least as fair as Π' with respect to $\boldsymbol{\gamma}$ *(i.e., it is at least as $\boldsymbol{\gamma}$-fair), denoted $\Pi \overset{\boldsymbol{\gamma}}{\succeq} \Pi'$, if*

$$\sup_{\mathcal{A} \in \mathtt{ITM}} u_{\mathtt{A}}(\Pi, \mathcal{A}) \quad \overset{\text{negl}}{\leq} \quad \sup_{\mathcal{A} \in \mathtt{ITM}} u_{\mathtt{A}}(\Pi', \mathcal{A}). \tag{3}$$

We will refer to a protocol which is a maximal element according to the above fairness relation as an *optimally fair* protocol.

Definition 2. *Let $\boldsymbol{\gamma} \in \Gamma_{\text{fair}}$. A protocol Π is* optimally $\boldsymbol{\gamma}$-fair *if it is at least as $\boldsymbol{\gamma}$-fair as any other protocol Π'.*

4 Fair and Reactive 2PC

The optimally fair two-party computation (2PC) protocol in the non-reactive case [11] can be described as follows: the protocol chooses one party uniformly at random, and the output is reconstructed toward this party first. If one party aborts the protocol early (that is, before the reconstruction phase), the other party can restart the protocol with a default input for the (corrupted) party that aborted. Intuitively, this means that the only way for a corrupted party to prevent the honest party from receiving output is to run the protocol until the reconstruction phase, hope to be the one that is chosen to receive the output first, and then abort the protocol. The result is that the adversary's expected payoff is bounded by $(\gamma_{10} + \gamma_{11})/2$, where γ_{10} is the payoff for an unfair abort, and γ_{11} is the payoff for a fair execution.

The most intuitive idea for solving the same problem for reactive computation is to apply the same reconstruction protocol for distributing the outputs in each round of the reactive computation. Unfortunately, the resulting protocol is not optimal: if the adversary aborts prior to the reconstruction phase in *some* round of the reactive computation, but it already achieved outputs in previous rounds, the honest party *cannot* safely restart the protocol with a default input for the corrupted party. Hence, adversaries with a utility satisfying $\gamma_{00} > \gamma_{11}$ may be better off by aborting the protocol early and thus definitely preventing the honest party from obtaining output—the simple adversarial strategy of choosing one party to corrupt at random and aborting as soon as an output is received is, in contrast to the non-reactive case, no longer optimal. In fact, even in the reactive case, if $\gamma_{11} \geq \gamma_{00} = 0$, then the adversary has no incentive to stop the protocol before obtaining output, so we can use the same protocol as in [11].

4.1 Better Fairness through More Rounds

In case the adversary's utility satisfies $\gamma_{00} > \gamma_{11}$, one can improve fairness guarantees by adding more rounds to a protocol's reconstruction phase. The reason is that if the adversary puts more emphasis on keeping the honest party from learning the output than on him learning the output himself, he might be tempted to abort the protocol even without obtaining output. We use the assumption that E_{01}^{R} is the adversary's least preferred event to threaten him with potentially only obtaining payoff γ_{01} in case of an early abort—and $\gamma_{01} < \gamma_{11}$. By carefully adapting the probabilities with which we output the value in a certain round of a reconstruction protocol, we can consistently keep the adversary in the dilemma between continuing the execution of the protocol or aborting it, maximizing the honest party's probability of obtaining the output.

We describe a protocol with r rounds, where one party—chosen at random—obtains the output during the first $r-1$ rounds, and the other party obtains it only in the last round. In more detail, for each round $l = 1, \ldots, r-1$ there is a probability $p_l \in [0,1]$ for a party to obtain the output in that round. The probabilities are the same for both parties since the setting is symmetric. In each of the rounds, the adversary has the advantage to receive his output before the honest party; this corresponds to the adversary in each round delaying his message until receiving the honest party's message, which is possible unless the timing guarantees given by the network are extremely strong. Consequently, in each round $l = 1, \ldots, r-1$, the adversary can trade giving the probability p_l corresponding to the current ith round to the honest party, obtaining the probability p_{l+1} of the next $(l+1)$st round in exchange. We now have to determine the values $p_1, \ldots, p_{r-1}$ such as to keep the adversary in a constant dilemma.

The payoff for the adversary aborting in round $l \in [1, \ldots, r-1]$ can be computed by the probabilities for the honest party $(p_1 + \cdots + p_{l-1})$ and the adversary $(p_1 + \cdots + p_l)$ to have received the value and the respective payoff values γ_{01} and γ_{10}. The condition is then described by the equation

$$\left(\sum_{u=1}^{l+1} p_u\right)\gamma_{10} + \left(\sum_{u=1}^{l} p_u\right)\gamma_{01} \quad = \quad \left(\sum_{u=1}^{l} p_u\right)\gamma_{10} + \left(\sum_{u=1}^{l-1} p_u\right)\gamma_{01},$$

which corresponds to, for all $l = 1, \ldots, r-2$,

$$p_{l+1} \quad = \quad p_l \left(\frac{-\gamma_{01}}{\gamma_{10}}\right).$$

With $\varrho := -\frac{\gamma_{01}}{\gamma_{10}}$, we obtain by induction that $p_l = \varrho^{l-1} p_1$. Providing the output to the other party only in the last round means that

$$\sum_{l=1}^{r-1} p_l \quad = \quad \sum_{l=1}^{r-1} (\varrho^{l-1} p_1) \quad = \quad \left(\sum_{l=1}^{r-1} \varrho^{l-1}\right) \cdot p_1 \quad = \quad 1/2,$$

or $p_1 = 1/2 \left(\sum_{l=1}^{r-1} \varrho^{l-1}\right)^{-1}$. In fact, we show in the remaining of the paper that the protocol achieving this distribution of probabilities is optimal.

As only the rounds of the reconstruction phase are relevant for the achieved fairness, we call a protocol an *r-round-reconstruction protocol* if it requires only r rounds of interaction to reconstruct the outputs after the computation has taken place. For simplicity, we only consider functionalities in which all parties receive the same output; the extension to the general case can be achieved using standard techniques. We now turn to a more detailed description of a fair reactive 2PC protocol, which is optimal when $\gamma_{11} > -\gamma_{10}$, as it follows from our lower bound results (Sect. 4.3).

4.2 The Fair Reactive Protocol

At a high level, the protocol works as follows: The functionality is sequentially evaluating the functions $f_1, \dots, f_m$; the invariant of the computation is that at any point, the state of the computation (i.e., the inputs and randomness used so far) is shared according to a two-out-of-two authenticated secret sharing. Each function f_λ, for $1 \leq \lambda \leq m$, is evaluated by having the two parties evaluate the function $f_{\mathrm{sh},f_\lambda,\mathcal{D}}$ (formally specified in Figure 2) which on input a sharing $\langle S_{\lambda-1} \rangle$ of the current state along with the parties' inputs $x_{1,\lambda}$ and $x_{2,\lambda}$, outputs a sharing $\langle S_\lambda \rangle$ of the updated state S_λ along with a sharing $\langle f_\lambda \rangle$ of the outputs of f_λ evaluated on $S_{\lambda-1}$, $x_{1\lambda}$ and $x_{2,\lambda}$. Next, the sharing $\langle f_\lambda \rangle$ is reconstructed in an r-round-reconstruction protocol as follows:

– The index of some party $i \in_R \{0,1\}$ is chosen uniformly at random (this will be the party that will receive the output during some *early output round*, i.e., before the last round r);
– for this party p_i, a round $l^* \in [r-1]$ is chosen according to the probability distribution described in Section 4.1;
– in each round $l \in [r-1] \setminus \{l^*\}$ of the reconstruction protocol, party p_i learns only that this round was not chosen;
– in round l^*, p_i learns the complete output;
– in the last round r, the sharing is reconstructed to both parties.

The idea behind the above construction is to have the adversary, in each round, face the following conundrum: To increase the expected payoff, that is, the probability of obtaining the output, it has to proceed to the next round. This means, however, that it first has to finish the current round by sending a message to the honest party, which will of course increase the honest party's probability of receiving the value (and hence reduce the adversary's payoff). For this technique to work, however, we need to make sure that no information about the chosen party and round leaks before the actually chosen party obtains the message in the chosen round.

To achieve the above properties, we use the function $f_{\mathrm{sh},f_\lambda,\mathcal{D}}$ to compute (and output) r pairs of sharings $(\langle y_{11} \rangle, \langle y_{21} \rangle), \dots, (\langle y_{1r} \rangle, \langle y_{2r} \rangle)$ as follows: for each round $l \in [r-1] \setminus \{l_i\}$, $y_{1l} = y_{2l} = \mathtt{DummyRound}$, where $\mathtt{DummyRound}$ is a default value signifying that this is not the output; for round l_i, y_{il_i} is set to the output of the function, whereas $y_{\neg il_i}$ is $\mathtt{DummyRound}$ as before. Finally, for the last round $l = r$, both y_{0r} and y_{1r} are set to the output of the function.

We are now ready to describe our reactive computation protocol, $\Pi_{\mathrm{RC}}^{\mathrm{fair}}$, for evaluating the two-party functionality described by $\boldsymbol{f} = (f_1, \ldots, f_m)$. The protocol is parametrized by the function vector $\boldsymbol{f}$, the number r of reconstruction rounds used for each output, and the probability distribution $\mathcal{D}$ on $[r-1]$ of the early output round l^*.

Protocol $\Pi_{\mathrm{RC}}^{\mathrm{fair}}$ $(p_1, p_2, \mathcal{D}, r, f_1, \ldots, f_m)$

Initialize $S_0 := (\bot, \bot, 0)$; the parties compute a default sharing of S_0, denoted $\langle S_0 \rangle$. For $\lambda = 1, \ldots, m$, evaluate f_λ *sequentially* as follows:

1. Use an (unfair MPC) sub-protocol to compute $f_{\mathrm{sh}, f_\lambda, \mathcal{D}}$ on input the sharing $\langle S_{\lambda-1} \rangle$ of the current state and the f_λ-inputs $x_1^{(\lambda)}$ and $x_2^{(\lambda)}$ of parties p_1 and p_2, respectively; if the protocol aborts then abort the execution of $\Pi_{\mathrm{RC}}^{\mathrm{fair}}$, otherwise denote by $\langle S_\lambda \rangle, (\langle y_{1,1}^{(\lambda)} \rangle, \langle y_{2,1}^{(\lambda)} \rangle), \ldots, (\langle y_{1,r}^{(\lambda)} \rangle, \langle y_{2,r}^{(\lambda)} \rangle)$ the output of the evaluation.
2. For $l = 1, \ldots, r$ do the following *sequentially*: have $\langle y_{1,l}^{(\lambda)} \rangle$ and $\langle y_{2,l}^{(\lambda)} \rangle$ reconstructed towards p_1 and p_2, respectively (by having p_i send his share to $p_{\neg i}$).
3. For each $p_i \in \{p_1, p_2\}$, if any of the reconstructions yields a value $y \notin \{\bot, \texttt{DummyRound}\}$ then output y; otherwise abort.

Fig. 1. The protocol for fair reactive 2PC.

We give a complete description of the function $f_{\mathrm{sh}, \lambda, \mathcal{D}}$ used by $\Pi_{\mathrm{RC}}^{\mathrm{fair}}$ in Fig. 2. The function is parameterized by the function f whose output is to be computed, and further by a probability distribution $\mathcal{D}$ on the set $[r-1]$ according to which the round l^* is chosen.

We now analyze the degree of fairness achieved by $\Pi_{\mathrm{RC}}^{\mathrm{fair}}$, which we later (Sect. 4.3 show optimal for certain parameters by proving a lower bound on the adversary's payoff. The proof of the theorem appears in the full version.

Theorem 1. *Let* $\boldsymbol{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \Gamma_{\mathrm{fair}}$. *Then*

$$\bar{u}_{\mathtt{A}}(\Pi_{\mathrm{RC}}^{\mathrm{fair}}, \mathcal{A}) \quad \overset{\mathrm{negl}}{\leq} \quad \max\left\{ \frac{\gamma_{10}}{2 \sum_{l=1}^{r-1} \varrho^{l-1}}, \frac{\gamma_{10} + \gamma_{11}}{2} \right\},$$

with $\varrho = \left| \frac{\gamma_{01}}{\gamma_{10}} \right|$. *In particular, if* $\gamma_{11} > -\gamma_{10}$, *then* $\bar{u}_{\mathtt{A}}(\Pi_{\mathrm{RC}}^{\mathrm{fair}}, \mathcal{A}) \overset{\mathrm{negl}}{\leq} \frac{\gamma_{10}+\gamma_{11}}{2}$.

The adversary's payoff depends on the number of rounds used in the reconstruction, and the optimal number of rounds depends on the exact values of the adversary's utility. As long as $\gamma_{11} > -\gamma_{10}$, we can adapt the probabilities such that the adversary is incentivized to continue with the protocol, if $\gamma_{11} \leq -\gamma_{10}$, then an abort during the first (non-trivial) reconstruction round is always preferable. We provide more details on this relation in the full version.

4.3 Lower Bounds

In this section, we prove lower bounds on the adversary's payoff that hold with respect to arbitrary protocols. In the case $\gamma_{11} > -\gamma_{10}$, this actually shows that

Function $f_{\mathrm{sh},f,\mathcal{D}}(\langle S \rangle, x_1, x_2)$

- Upon receiving inputs $(x_1, \langle S \rangle_1)$ and $(x_2, \langle S \rangle_2)$ from p_1 and p_2, do:
  1. If the shares $\langle S \rangle_1$ and $\langle S \rangle_2$ are inconsistent or the reconstructed state is the abort vector $S = (\mathtt{abt}, \mathtt{abt})$, then set the output to $y := \bot$; otherwise, choose $r \in_R \{0,1\}^*$, set $y = f(x_1, x_2, S, r)$, and update S by appending x_1, x_2 and r to S; denote by S' the updated state.
  2. Compute an authenticated sharing $\langle S' \rangle$ of S'.
  3. Choose a party index $i \in_R \{1,2\}$ uniformly at random and choose a round index $l^* \overset{\mathcal{D}}{\leftarrow} [r-1]$ according to $\mathcal{D}$.
  4. For $l = 1, \ldots, r-1$, compute the authenticated-sharing pair $(\langle y_{1,l} \rangle, \langle y_{2,l} \rangle)$, where y_{jl} is computed as follows:
     - If $l < r$, $y_{\neg i,l} := \mathtt{DummyRound}$, whereas $y_{i,l} := \begin{cases} y & \text{if } l = l^*; \\ \mathtt{DummyRound} & \text{otherwise.} \end{cases}$
     - If $l = r$, $y_{1,l} = y_{2,l} = y$.
- Output $\langle S' \rangle, (\langle y_{1,1} \rangle, \langle y_{2,1} \rangle), \ldots, (\langle y_{1,r} \rangle, \langle y_{2,r} \rangle)$.

Fig. 2. The function computing the authenticated sharings used in protocol $\Pi_{\mathrm{RC}}^{\mathrm{fair}}$.

protocol $\Pi_{\mathrm{RC}}^{\mathrm{fair}}$ is optimally fair, as the lower bound tightly matches the upper bound from Theorem 1. In the other case, i.e., $\gamma_{11} \leq -\gamma_{10}$, we still give a lower bound which is close to the upper bound we proved.

We show the lower bounds on the adversary's expected payoff using a specific "two-phase exchange" functionality $f_{2\mathrm{Ex}}^{\perp}$ that works as follows: Both parties input a $2k$-bit string, and in the first phase, both obtain the first k bits of the other party's input. In the second phase, they both obtain the remaining k bits of the other party's input. (See Fig. 3.)

Function $f_{2\mathrm{Ex}}^{\perp}$

The functionality $f_{2\mathrm{Ex}}^{\perp}$ is a two-party functionality that proceeds in two rounds:

- Obtain from each p_i an input $x_i \in \{0,1\}^{2k}$, and split x_i into $x_i = y_i | z_i$ with $y_i, z_i \in \{0,1\}^k$. Output y_1 to p_2 and y_2 to p_1.
- No inputs: output z_1 to p_2 and z_2 to p_1.

Fig. 3. The two-phase exchange functionality.

There are simple and generic adversarial strategies $\mathcal{A}_i$ that corrupt party p_i in the beginning but follow the protocol honestly until the last output phase of the protocol. Then, it aborts as soon as it obtained the output—that is, in each round $\mathcal{A}_i$ checks whether the protocol *would* already provide the output if the other (honest) party *would* abort; in this case, $\mathcal{A}_i$ aborts the protocol without sending the messages for that round.[9] The bound proven in the following lemma comes from the fact that if one of the parties gets the output first, then the adversary $\mathcal{A}_{\text{gen}}$ corrupting one party at random has a 1/2 chance to corrupt this party and be the only one to get the output. The payoff of this strategy is the same as for the SFE (non-reactive) case, and the proof of the lemma also resembles the proof of the simpler case.

Lemma 1. *Let $\boldsymbol{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \mathit{\Gamma}_{\text{fair}}$. For every protocol $\mathit{\Pi}$ which securely implements the functionality $f^{\perp}_{\text{2Ex}}$, there exists an adversary $\mathcal{A}$ with*

$$\bar{u}_{\mathtt{A}}(\mathit{\Pi}, \mathcal{A}) \quad \overset{\text{negl}}{\geq} \quad \frac{\gamma_{10} + \gamma_{11}}{2}. \tag{4}$$

The lemma shows the optimality of the protocol described in Sect. 4.2 for all cases where $\gamma_{11} > -\gamma_{10}$. The next lemma provides a lower bound that is relevant in the more general case without this restriction, and is also interesting for protocols for which the bound from Equation (4) is not tight because, e.g., they use too few rounds. In fact, in the case of reactive MPC, the maximum utility of the adversary generally depends on the number of protocol rounds, and in particular we show a trade-off between the payoff of the generic adversary and the payoff of adversaries that potentially abort during the protocol without receiving their output. The proof is in the full version.

Lemma 2. *Let $\boldsymbol{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \mathit{\Gamma}_{\text{fair}}$, and $\mathit{\Pi}$ be an r-round-reconstruction protocol that securely implements the functionality $f^{\perp}_{\text{2Ex}}$, such that*

$$\bar{u}_{\mathtt{A}}(\mathit{\Pi}, \mathcal{A}_{\text{gen}}) \quad \leq \quad \frac{\gamma_{10} + \gamma_{11}}{2} + \omega.$$

Then, there exists an adversary $\mathcal{A}$ with

$$\bar{u}_{\mathtt{A}}(\mathit{\Pi}, \mathcal{A}) \quad \overset{\text{negl}}{\geq} \quad \frac{\left(\frac{1}{2} - \frac{\omega}{\gamma_{10} - \gamma_{11}}\right)\gamma_{10}}{\sum_{\ell=1}^{r-1} \varrho^{\ell-1}},$$

where $\varrho = -\gamma_{01}/\gamma_{10}$.

By increasing the number of rounds in the reconstruction phase and choosing a suitable distribution of probabilities over the rounds, we can decrease the payoff of the "aborting" adversaries below the bound of the generic adversary, thus establishing that protocol $\mathit{\Pi}^{\text{fair}}_{\text{RC}}$ is optimally fair. The necessary number of rounds for the optimal result depends on the exact values of the adversary's utility, and can be computed as in Sect. 4.1; details appear in the full version.

Corollary 1. *Let $\boldsymbol{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \mathit{\Gamma}_{\text{fair}}$ and $\gamma_{11} > -\gamma_{01}$. Then protocol $\mathit{\Pi}^{\text{fair}}_{\text{RC}}$ from Fig. 1 is optimally $\boldsymbol{\gamma}$-fair.*

[9] In the case of (reactive) MPC the protocol may output only either the correct value or an "abort" symbol, as an honest party cannot restart the protocol with a default input because the adversary already obtained output in the previous rounds.

References

1. Asharov, G., Canetti, R., Hazay, C.: Towards a game theoretic view of secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 426–445. Springer, Heidelberg (2011)
2. Beaver, D., Goldwasser, S.: Multiparty computation with faulty majority. In: FOCS '89. pp. 468–473. IEEE (1989)
3. Beimel, A., Lindell, Y., Omri, E., Orlov, I.: $1/p$-secure multiparty computation without honest majority and the best of both worlds. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 277–296. Springer, Heidelberg (2011)
4. Blum, M.: How to exchange (secret) keys. ACM Transactions on Computer Science 1, 175–193 (1984)
5. Boneh, D., Naor, M.: Timed commitments. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 236–254. Springer, Heidelberg (2000)
6. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13, 143–202 (April 2000),
7. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In FOCS 2001. pp. 136–145. IEEE (2001)
8. Cleve, R.E.: Limits on the security of coin flips when half the processors are faulty. In STOC '86. pp. 364–369. ACM, Berkeley (1986)
9. Damgård, I.: Practical and provably secure release of a secret and exchange of signatures. Journal of Cryptology 8(4), 201–222 (1995)
10. Garay, J.A., Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Rational protocol design: Cryptography against incentive-driven adversaries. In FOCS 2013. IEEE (2013)
11. Garay, J.A., Katz, J., Tackmann, B., Zikas, V.: How fair is your protocol? A utility-based approach to protocol optimality. In: Spirakis, P. (ed.) PODC 2015. ACM Press (2015)
12. Garay, J.A., MacKenzie, P., Prabhakaran, M., Yang, K.: Resource fairness and composability of cryptographic protocols. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 404–428. Springer (2006)
13. Garay, J.A., Tackmann, B., Zikas, V.: Fair distributed computation of reactive functions. Cryptology ePrint Archive, Report 2015/807 (August 2015)
14. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game—A completeness theorem for protocols with honest majority. In STOC '87. pp. 218–229. ACM (1987)
15. Gordon, D., Katz, J.: Partial fairness in secure two-party computation. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 157–176. Springer (2010)
16. Groce, A., Katz, J.: Fair computation with rational players. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 81–98. Springer (2012)
17. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Heidelberg (2013)
18. Pinkas, B.: Fair secure two-party computation. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 87–105. Springer, Heidelberg (2003)
19. Yao, A.C.: Theory and applications of trapdoor functions. In FOCS '82. pp. 80–91. IEEE (1982)