

Fair Distributed Computation of Reactive Functions

Juan Garay
Yahoo Labs
garay@yahoo-inc.com

Björn Tackmann *
UC San Diego
btackmann@eng.ucsd.edu

Vassilis Zikas†
ETH Zurich
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. This is a desirable property, as honest parties are more reluctant to participate in an (unfair) protocol in which cheaters learn their outputs while the honest parties waste their time and computation resources. But what makes fairness an even more intriguing topic is Cleve’s seminal result [STOC’86], which proves that it is impossible to achieve in the presence of dishonest majorities.

Cleve’s result ignited a quest for more relaxed, yet meaningful definitions of fairness, with numerous works suggesting such relaxations and protocols satisfying them. A common pattern in these 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 inputs strictly before any output is computed. Yet, many natural cryptographic tasks are of a *reactive* (stateful) nature, where parties provide inputs and receive outputs several times during the course of the computation. This is the case, for example, when computing multi-stage auctions or emulating a virtual stock-exchange market, or even when computing basic cryptographic tasks such as commitments and secret sharing.

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, Katz, Tackmann and Zikas [PODC’15], which, informally, defines the utility of an adversary who wants to break fairness and uses it as a measure of a protocol’s success in satisfying the property. Similarly to the non-reactive notion, our definition enjoys the advantage of offering a comparative notion of fairness for reactive functions, inducing a partial order on protocols with respect to fairness.

We then turn to the question of finding protocols that restrict the adversary’s utility. We provide, for each parameter choice of the adversary’s utility, a protocol for fair and reactive two-party computation, and prove the optimality of this protocol for one (natural) class of parameter values and (non-tight) lower bounds for all remaining values. Our study shows that achieving fairness in the reactive setting is more complex than in the much-studied case of one-shot functions. For example, in contrast to the non-reactive case, (a) increasing the number of rounds used for reconstructing the output can lead to improved fairness, and (b) the minimal number of rounds required in the reconstruction depends on the *exact values* of the adversary’s utility.

Key words: Cryptographic protocols, secure multi-party computation, fairness, game theory.

*Research partly done while at ETH Zurich, and partly supported by the SNF via the Fellowship no. P2EZP2-155566 and by the NSF grant CNS-1228890.

†Research supported in part by the SNF via the Ambizione grant PZ00P-2142549.

1 Introduction

In secure multi-party computation (MPC) [Yao82, GMW87], a set of n parties wishes 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 [Cle86] 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 [Blu84, BG89, Dam95, BN00, Pin03, GMPY06], in which parties take turns in releasing bits of information. More recently, Asharov *et al.* [ACH11] 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 [GK12].

All of these weaker notions of fairness, however, are of an “all-or-nothing” nature, in the sense that either a protocol achieves the respective security definition, or the notion renders the protocol unfair and makes not further statement about it. For example, this is the case for *resource fairness* [GMPY06], which formalizes the intuition of the *gradual release* paradigm [Blu84, BG89, Dam95, BN00, Pin03] 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.* [GKTZ15] changed the state of things, by introducing a *quantitative* approach to fairness. This new notion is based on the idea that one can use an appropriate utility function to express the preferences of an adversary who wants to break fairness, and allows for comparing protocols with respect to how fair they are, placing them in a partial order according to a relative-fairness relation.¹ Previously, the only other notion providing any sort of comparative statement was that of *1/p-security* (aka “partial fairness”), where security is given up with probability $1/p$ for some polynomial p [GK10, BLOO11], but which does not always guarantee privacy

¹We stress that 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.

and correctness (see [GKTZ15] for a detailed comparison).

Technically, the approach in [GKTZ15] builds on machinery developed in the recently proposed *Rational Protocol Design* (RPD) framework, by Garay *et al.* [GKM⁺13]. In more detail, [GKM⁺13] 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, e.g., [Can00, Can01]) 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 [Can01]) in a fair/optimal protocol by a protocol which securely implements it without affecting its fairness/optimality.

Our contributions. In this work 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.

As in [GKTZ15], 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, 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 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, but up to a point; 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 [GKM⁺13] that are needed for understanding and evaluating our results. In addition, we extend Canetti’s simulation-based model of computation and protocol composition [Can00] to support computation of reactive functions. 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 [GKTZ15] to the reactive computation case.) Section 4 is dedicated to the fair reactive protocol (Section 4.2) and lower bounds (Section 4.3). Some detailed expositions, complementary material and proofs appear in the appendix.

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 $[m] := \{1, \dots, m\}$. 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 $\mu : \mathbb{N} \rightarrow \mathbb{R}$ 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 $f \stackrel{\text{negl}}{\approx} g$ to denote that \exists negligible $\mu : |f - g| \leq \mu$, and $f \stackrel{\text{negl}}{\geq} g$ to denote \exists negligible $\mu : f \geq g - \mu$, with $\stackrel{\text{negl}}{\leq}$ defined analogously.

For the model of computation and protocol composition, we follow Canetti’s adaptive simulation-based model for multi-party computation [Can00]. The protocol execution is formalized by collections of interactive Turing machines (ITMs); the set of all *efficient* ITMs is denoted by ITM . We generally denote our protocols by Π and our (ideal) functionalities (which are also referred to as *the trusted party* [Can00]) 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 $\{\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}}(k, z)\}_{k \in \mathbb{N}, z \in \{0, 1\}^*}$, which is more compactly often written as $\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}}$, 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. As mentioned above, we cast our definitions of fairness for the reactive setting in the technically simpler framework in [Can00] (allowing sequential and modular composition), which considers synchronous protocols with guaranteed termination.² The [Can00] framework, however, lacks a formal definition of computation of reactive functions. In this section we describe the corresponding real-world/ideal-world experiments based on this model with adaptive adversaries [Can00, Section 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 [Can00]: 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 [KMTZ13], reactive computation can be seen as an ordered sequence of computations of non-reactive functions (SFE) that can maintain a joint (private) state. More concretely, reactive computation is specified by a vector of (probabilistic) functions $\vec{f} = (f_1, \dots, f_m)$, where each $f_\lambda \in \vec{f}$ 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* $\vec{S}_\lambda \in ((\{0, 1\}^* \cup \{\perp\})^n \times R)^{(\lambda-1)}$, which includes the inputs and random coins used for the evaluation of functions $f_1, \dots, f_{\lambda-1}$. Each $f_\lambda \in \vec{f}$ outputs a vector of strings $\vec{y}_\lambda = (y_{1,\lambda}, \dots, y_{n,\lambda}) \in \{0, 1\}^n$, 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 [Can00], but instead of a single function, the trusted third party (TTP, or “functionality”) $\mathcal{F}_{\text{RC}}^{\vec{f}}$ is parameterized by the vector $\vec{f} = (f_1, \dots, f_m)$ 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

²Our definitions can be extended to Universally Composable (UC) security [Can01] using the approach of Katz *et al.* [KMTZ13] to model terminating synchronous (reactive) computation in UC.

is currently computed. The output of the computation is taken to be the vector of outputs of all functions in \vec{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 are able to choose their input for any $f_\lambda, \lambda \in [m]$, depending on inputs and outputs from the evaluation of $f_1, \dots, f_{\lambda-1}$. 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* $\text{Inp}_i^1, \dots, \text{Inp}_i^m$, where for each $\lambda \in [m]$, $\text{Inp}_i^\lambda : ((\{0, 1\}^*)^{\lambda-1})^2 \rightarrow \{0, 1\}^*$ is a function that on input the inputs and outputs from the evaluation of functions $f_1, \dots, f_{\lambda-1}$ computes the input for the evaluation of f_λ . (Wlog, assume that p_i 's input to f_1 is $\text{Inp}_i^1(0, 0)$.) Note that, unlike the parties, the interaction of the TTP with the simulator allows him to choose his inputs during the (ideal) protocol execution. Refer to Appendix A for the formal description of the above ideal process.

The real-world execution. The real-world experiment is analogous to the corresponding experiment in [Can00], where the input of each party p_i is his input-deciding function vector $\text{Inp}_i^1, \dots, \text{Inp}_i^m$.

Rational Protocol Design. Our results utilize the *Rational Protocol Design* (RPD) framework [GKM⁺13]. Here we review the basic elements that are needed to motivate and express our definitions and results; we refer to [GKM⁺13] 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* D and an *attacker* A . The designer D plays first by specifying a protocol Π for the (honest) participants to run; subsequently, the attacker A , who is informed about 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). Note that it suffices to define the utility u_{A} of the adversary as the game is zero-sum. (The utility u_{D} of the designer is then $-u_{\mathsf{A}}$.)

In RPD, the definition of utilities relies on the simulation paradigm³, 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 the optimal protocol with respect to fairness. Towards that goal, we follow the three-step process described in [GKM⁺13, GKTZ15] 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 described in Section 2 but modify it to allow the simulator

³In RPD the statements are formalized in Canetti's Universal Composition (UC) framework [Can01]; however, one can use any other simulation-based model, in particular the one in [Can00] described above.

\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 [GKTZ15], 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 $\vec{f} = (f_1, \dots, f_m)$ as described in Section 2 and is denoted $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ (or simply $\mathcal{F}_{\text{RC}}^{\perp}$ if the function sequence is clear from the context). A detailed description can be found in Appendix A. We point out that when $\mathcal{F}_{\text{RC}}^{\perp}$ 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 [GKTZ15].

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 any 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_{ij}^{R} 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 $\vec{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \mathbb{R}^4$, corresponding to events $\vec{E}^{\text{R}} = (E_{00}^{\text{R}}, E_{01}^{\text{R}}, E_{10}^{\text{R}}, E_{11}^{\text{R}})$.

Finally, we define the expected payoff of a given simulator \mathcal{S} (for an environment \mathcal{Z}) to be⁴:

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

⁴Refer to [GKM⁺13, Section 2] for the rationale behind this formulation.

Step 3: Defining the attacker’s utility. Given $U_I^{\mathcal{F}_{\text{RC}}^\perp, \vec{\gamma}}(\mathcal{S}, \mathcal{Z})$, the utility $u_{\mathbf{A}}(\Pi, \mathcal{A})$ for a pair (Π, \mathcal{A}) of a protocol Π and an adversary \mathcal{A} is defined following the methodology in [GKM⁺13] as the expected payoff of the *best* simulator⁵ that simulates \mathcal{A} in the $\mathcal{F}_{\text{RC}}^\perp$ -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 $\vec{\gamma}$ explicit, we sometimes denote the above utility as $\hat{U}^{\Pi, \mathcal{F}_{\text{RC}}^\perp, \vec{\gamma}}(\mathcal{A})$ and refer to it as the *payoff of strategy \mathcal{A}* (for attacking Π).

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

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

To complete our formulation, we now describe a natural relation among the values in $\vec{\gamma}$ 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 $\gamma_{01} = \min_{\gamma \in \vec{\gamma}} \{\gamma\}$. 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 $\vec{\gamma}$, 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 $\Gamma_{\text{fair}} \subseteq \mathbb{R}^4$. Summarizing, our fairness-specific payoff (“preference”) vector $\vec{\gamma}$ satisfies

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

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 $u_{\mathbf{A}}(\Pi, \mathcal{A})$) 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 $\vec{\gamma} \in \Gamma_{\text{fair}}$ be a preference vector. We say that Π is *at least as fair as Π' with respect to $\vec{\gamma}$* (i.e., it is at least as $\vec{\gamma}$ -fair), denoted $\Pi \stackrel{\vec{\gamma}}{\succeq} \Pi'$, if

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

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

4 Fair and Reactive 2PC

The optimally fair two-party computation (2PC) protocol in the non-reactive case [GKTZ15] 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

⁵The best simulator is taken to be the one that minimizes his payoff [GKM⁺13].

(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 [GKTZ15].

4.1 Distributing the Output in Rounds

In case the adversary’s utility satisfies $\gamma_{00} > \gamma_{11}$, one can construct protocols with better fairness guarantees if one adds more rounds to the 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. But 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.

In more detail, for a protocol with r rounds and for each round $i = 1, \dots, r$, there is a probability $p_i \in [0, 1]$ 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 giving the same capability to the honest part; this corresponds to the adversary in each round delaying his message until receiving the honest party’s message for the same round, this behavior is possible unless the timing guarantees given by the network are extremely strong. Consequently, in each round $i = 1, \dots, r$, the adversary can trade giving the probability p_i corresponding to the current i th round to the honest party, obtaining the probability p_{i+1} of the next $(i + 1)$ st round in exchange. We now have to determine the values p_1, \dots, p_r such as to keep the adversary in a constant dilemma.

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

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

which implies

$$p_{j+1} = p_j \left(\frac{-\gamma_{01}}{\gamma_{10}}\right).$$

With $\varrho := -\frac{\gamma_{01}}{\gamma_{10}}$, we obtain by induction that $p_j = \varrho^{j-1}p_1$. By the fact that $\sum_{j=1}^r p_j = 1$, this means that

$$\sum_{j=1}^{r-1} p_j = \sum_{j=1}^{r-1} (\varrho^{j-1}p_1) = \left(\sum_{j=1}^{r-1} \varrho^{j-1} \right) \cdot p_1 = 1 - p_r,$$

or $p_1 = (1 - p_r) \left(\sum_{j=1}^{r-1} \varrho^{j-1} \right)^{-1}$. In fact, we show in the remaining part 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. (This is made precise in [GKTZ15].) 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 the description of a fair reactive 2PC protocol, which is optimal when $\gamma_{11} > -\gamma_{10}$, as it follows from our lower bound results (Section 4.3).

4.2 The 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 $f_{\text{sh},f_\lambda,\mathcal{D}}$ (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 $f_{\text{sh},f_\lambda,\mathcal{D}}$ 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 $l \in [r-1] \setminus \{l_i\}$, $y_{1l} = y_{2l} = \text{DummyRound}$, where *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_{-il_i} is *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, $\Pi_{\text{RC}}^{\text{fair}}$, for evaluating the two-party functionality described by $\vec{f} = (f_1, \dots, f_m)$. The protocol is parametrized by the function

vector \vec{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^* .

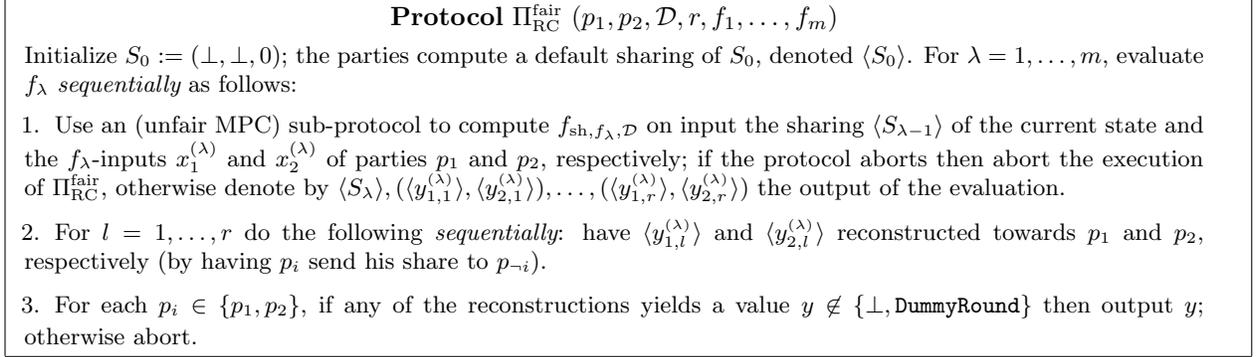


Figure 1: The protocol for fair reactive 2PC.

We give a complete description of the function $f_{\text{sh}, \lambda, \mathcal{D}}$ used by $\Pi_{\text{RC}}^{\text{fair}}$ in Figure 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.

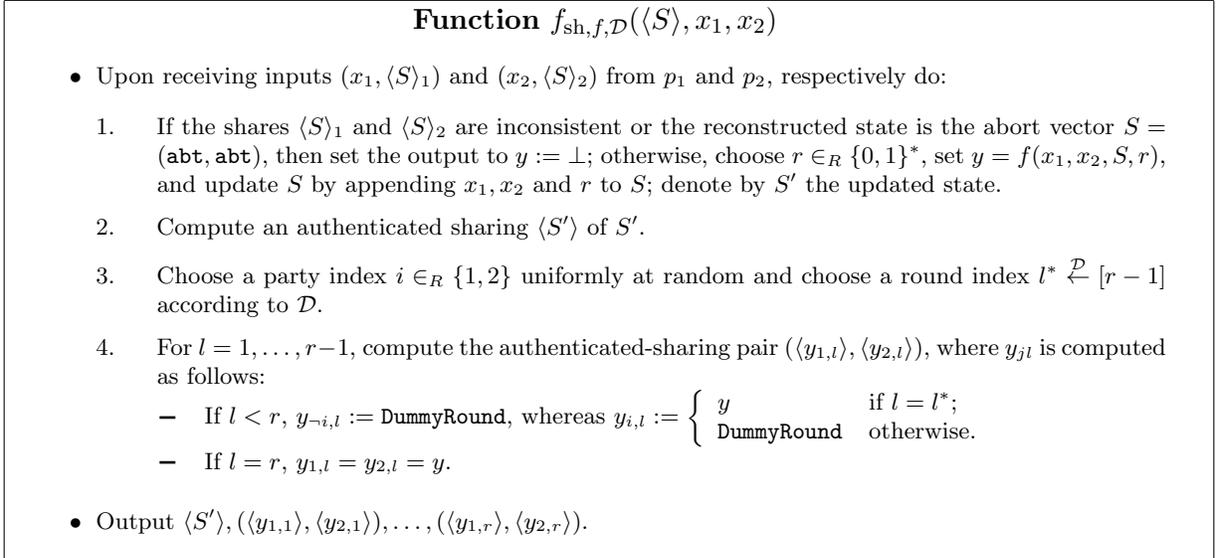


Figure 2: The function to compute the authenticated sharings that are used in protocol $\Pi_{\text{RC}}^{\text{fair}}$.

We now analyze the degree of fairness achieved by $\Pi_{\text{RC}}^{\text{fair}}$, which we later (Section 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 3. *Let $\vec{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \Gamma_{\text{fair}}$. Then*

$$\bar{u}_{\mathbf{A}}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}) \stackrel{\text{negl}}{\leq} \max \left\{ \frac{\gamma_{10}}{2 \sum_{\ell=1}^{r-1} \varrho^{\ell-1}}, \frac{\gamma_{10} + \gamma_{11}}{2} \right\},$$

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

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. 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 protocol $\Pi_{\text{RC}}^{\text{fair}}$ is optimally fair, as the lower bound tightly matches the upper bound from Theorem 3. 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.

To show a lower bound on the adversary’s expected payoff, we use a specific “two-phase exchange” functionality $f_{2\text{Ex}}^\perp$ 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 Figure 3.)

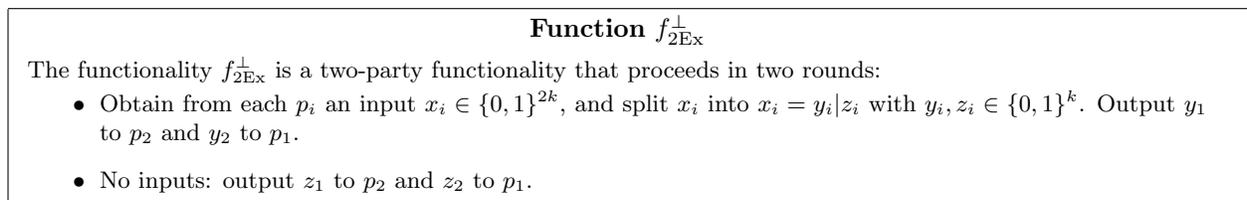


Figure 3: The two-phase exchange functionality.

We begin by considering 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.⁶ The bound proven in the following lemma comes from the fact that if one of the parties gets the output first, then the adversary 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 4. *Let $\vec{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \Gamma_{\text{fair}}$. For every protocol Π which securely implements the (unfair) functionality $f_{2\text{Ex}}^\perp$, there exists an adversary \mathcal{A} with*

$$\bar{u}_{\mathcal{A}}(\Pi, \mathcal{A}) \stackrel{\text{negl}}{\geq} \frac{\gamma_{10} + \gamma_{11}}{2}.$$

As mentioned above, if the adversary aborts in a later phase of a reactive MPC protocol, the honest party cannot simply start over because the adversary has obtained output already in an earlier phase. For this reason, the optimally fair protocol in the SFE case is not applicable here. In fact, in the case of reactive MPC, we can prove that the maximum utility of the adversary also depends on the number of rounds. The following lemma shows 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. In the following statements, we use \mathcal{A}_{gen} to denote the adversarial strategy

⁶Note that 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.

that uses either \mathcal{A}_1 or \mathcal{A}_2 with probability 1/2 each. The proof of the following lemma is in the full version.

Lemma 5. *Let $\vec{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \Gamma_{\text{fair}}$, and Π be an r -round-reconstruction protocol that securely implements the (unfair) functionality $f_{2\text{Ex}}^\perp$, such that*

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

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

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

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

Now, by increasing the number of rounds of 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 $\Pi_{\text{RC}}^{\text{fair}}$ is optimally fair in certain cases. The number of rounds necessary for the optimal result depends on the exact values of the adversary’s utility vector, and can be computed from the formulae in Section 4.1; details are in the full version.

Corollary 6. *Let $\vec{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \Gamma_{\text{fair}}$ and $\gamma_{11} > -\gamma_{01}$. Then protocol $\Pi_{\text{RC}}^{\text{fair}}$ from Figure 1 is optimally $\vec{\gamma}$ -fair.*

References

- [ACH11] Gilad Asharov, Ran Canetti, and Carmit Hazay. Towards a game theoretic view of secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 426–445, Heidelberg, 2011. Springer.
- [BG89] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In *Proceedings of the 30th Symposium on Foundations of Computer Science*, pages 468–473. IEEE, 1989.
- [BLOO11] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. $1/p$ -secure multiparty computation without honest majority and the best of both worlds. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 277–296, Heidelberg, 2011. Springer.
- [Blu84] Manuel Blum. How to exchange (secret) keys. *ACM Transactions on Computer Science*, 1:175–193, 1984.
- [BN00] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254, Heidelberg, 2000. Springer.
- [Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13:143–202, April 2000.
- [Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.
- [Cle86] Richard E. 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*, pages 364–369, Berkeley, 1986. ACM.

- [Dam95] Ivan Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8(4):201–222, 1995.
- [GK10] Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In Henry Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 157–176. Springer, 2010.
- [GK12] Adam Groce and Jonathan Katz. Fair computation with rational players. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 81–98. Springer, 2012.
- [GKM⁺13] Juan A. Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *54th Annual Symposium on Foundations of Computer Science*. IEEE, 2013.
- [GKTZ15] Juan A. Garay, Jonathan Katz, Björn Tackmann, and Vassilis Zikas. How fair is your protocol? A utility-based approach to protocol optimality. In Paul Spirakis, editor, *Proceedings of the 2015 ACM symposium on Principles of distributed computing*. ACM Press, 2015.
- [GMPY06] Juan A. Garay, Philip MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 404–428. Springer, 2006.
- [GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game—A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 218–229. ACM, 1987.
- [KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498, Heidelberg, 2013. Springer.
- [Pin03] Benny Pinkas. Fair secure two-party computation. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 87–105, Heidelberg, 2003. Springer.
- [Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91. IEEE, 1982.

A Secure Multi-Party Computation of Reactive Functions

In this section we provide formal definitions of the ideal world execution corresponding to reactive MPC and its relaxation used in our fairness definition. Although we will design protocols only for the two-party case (2PC), since this is the first formal treatment of the reactive case we provide our definitions for the more general case of n parties. (See the figures below.)

The events E_{ij}^R can now be described more precisely as follows:

E_{00}^R : The simulator does not ask $\mathcal{F}_{RC}^{f,\perp}$ for the all the corrupted party’s outputs, i.e., it sends $o = \text{NoOut}$ during the evaluation of at least one f_i , and it instructs $\mathcal{F}_{RC}^{f,\perp}$ to abort by sending $s = \text{abort}$.

E_{01}^R : The simulator does not ask $\mathcal{F}_{RC}^{f,\perp}$ for all the corrupted party’s outputs and does *not* instruct it to abort, i.e., it does not send $s = \text{abort}$.

E_{10}^R : The simulator asks $\mathcal{F}_{RC}^{f,\perp}$ for all his outputs (i.e., always sends $o = \text{YesOut}$) and instructs it to abort before the honest party receives all its outputs (i.e., in the last invocation of Step B.5).

E_{11}^R : The simulator asks the functionality for all his outputs, and it does not sent $s = \text{abort}$.

Reactive MPC – Ideal Process

The trusted party $\mathcal{F}_{\text{RC}}^{\vec{f}}$ is parameterized by a vector $\vec{f} = (f_1, \dots, f_m)$ of multiparty functions, where $f_\lambda : (\{0, 1\}^* \cup \{\perp\})^n \times ((\{0, 1\}^* \cup \{\perp\})^n \times R)^{(\lambda-1)} \times R \rightarrow (\{0, 1\}^*)^n$, for $\lambda = 1, \dots, m$. That are to be evaluated sequentially. Each $p_i \in \mathcal{P}$ has m local input deciding function $\text{Inp}_i^1, \dots, \text{Inp}_i^m$, where $\text{Inp}_i^\lambda : ((\{0, 1\}^*)^{\lambda-1})^2 \rightarrow \{0, 1\}^*$, for $\lambda \in [m]$. The simulator \mathcal{S} chooses the parties to corrupt adaptively (at any point, denote by \mathcal{I} the set of corrupted parties) and gets to see their input-deciding function and all outputs they have received so far, and possibly replace the inputs they hand to functions in \vec{f} that have not yet been evaluated.

A. Every $p_i \in \mathcal{P} \setminus \mathcal{I}$ hands his vector of input deciding functions to the functionality $\mathcal{F}_{\text{RC}}^{\vec{f}}$.

B. For $\lambda = 1, \dots, n$ the following steps are executed sequentially:

1. For every $p_i \in \mathcal{P} \setminus \mathcal{I}$, $\mathcal{F}_{\text{RC}}^{\vec{f}}$ computes p_i input $x_{i,\lambda}$ for the λ th function $f_\lambda \in \vec{f}$ as follows:
 - If $\lambda = 1$ then $x_{i,1} = \text{Inp}_i(0^*, 0^*)$,
 - Otherwise, let $x_{i,1}, \dots, x_{i,\lambda-1}$ and $y_{i,1}, \dots, y_{i,\lambda-1}$ denote the inputs and outputs from the evaluations of the functions $f_1, \dots, f_{\lambda-1}$, respectively. $\mathcal{F}_{\text{RC}}^{\vec{f}}$ computes $x_{i,\lambda} = \text{Inp}_i((x_{i,1}, \dots, x_{i,\lambda-1}), (y_{i,1}, \dots, y_{i,\lambda-1}))$.

\mathcal{S} gets to corrupt parties adaptively as in [Can00] (upon corruption of some p_i , \mathcal{S} gets to see and possible change p_i 's input and all outputs generated for him so far, i.e., output from the evaluation of $f_1, \dots, f_{\lambda-1}$). For each $p_i \in \mathcal{I}$, \mathcal{S} sends some input $x'_{i,\lambda}$ to $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ (if \mathcal{S} sends no value or an invalid value for some $p_i \in \mathcal{I}$ then the functionality takes a default value d for this input). $\mathcal{F}_{\text{RC}}^{\vec{f}}$ denotes the vector of all inputs for f_λ as $(x'_{1,\lambda}, \dots, x'_{n,\lambda})$.

2. $\mathcal{F}_{\text{RC}}^{\vec{f}}$ chooses $r_\lambda \in R$ uniformly at random and computes

$$(y_{1,\lambda}, \dots, y_{n,\lambda}) = f((x'_{1,\lambda}, \dots, x'_{n,\lambda}, r_\lambda), (x'_{1,\lambda-1}, \dots, x'_{n,\lambda-1}, r_{\lambda-1}), (x'_{1,1}, \dots, x'_{n,1}, r_1)),$$

where R_ℓ , $\ell = 1, \dots, \lambda$ denotes the randomness used by $\mathcal{F}_{\text{RC}}^{\vec{f}}$ for the evaluation of f_ℓ .

3. $\mathcal{F}_{\text{RC}}^{\vec{f}}$ sends to \mathcal{S} the outputs $\{y_{i,\lambda}\}_{i \in \mathcal{I}}$ of corrupted parties p_i . (\mathcal{S} is as usually allowed to corrupt more parties adaptively.)
4. For each $p_j \in \mathcal{P} \setminus \mathcal{I}$: $\mathcal{F}_{\text{RC}}^{\vec{f}}$ records p_j 's output $y_{j,\lambda}$.

C. For each honest p_i , $\mathcal{F}_{\text{RC}}^{\vec{f}}$ sends $\vec{y}_i = (y_{i,1}, \dots, y_{i,m})$ to p_i who outputs \vec{y}_i (corrupted parties output \perp)

The trusted party $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ is parameterized by a vector $\vec{f} = (f_1, \dots, f_m)$ of multiparty functions, where $f_\lambda : (\{0, 1\}^* \cup \{\perp\})^n \times ((\{0, 1\}^* \cup \{\perp\})^n \times R)^{(\lambda-1)} \times R \rightarrow (\{0, 1\}^*)^n$, for $\lambda = 1, \dots, m$. That are to be evaluated sequentially. Each $p_i \in \mathcal{P}$ has m local input deciding function $\text{Inp}_i^1, \dots, \text{Inp}_i^m$, where $\text{Inp}_i^\lambda : ((\{0, 1\}^*)^{\lambda-1})^2 \rightarrow \{0, 1\}^*$, for $\lambda \in [m]$. The simulator \mathcal{S} chooses the parties to corrupt adaptively (at any point, denote by \mathcal{I} the set of corrupted parties) and gets to see their input-deciding function and all outputs they have received so far, and possibly replace the inputs they hand to functions in \vec{f} that have not yet been evaluated. He is also allowed to non-receive his outputs (i.e., outputs of corrupted parties) and to prevent honest parties from receiving outputs (i.e., force and abort) before or after receiving his own outputs.

A. Every $p_i \in \mathcal{P} \setminus \mathcal{I}$ hands his vector of input deciding functions to the functionality $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$.

B. For $\lambda = 1, \dots, n$ the following steps are executed sequentially:

1. For every $p_i \in \mathcal{P} \setminus \mathcal{I}$, $\mathcal{F}_{\text{RC}}^{\vec{f}}$ computes p_i input $x_{i,\lambda}$ for the λ th function $f_\lambda \in \vec{f}$ as follows:
 - If $\lambda = 1$ then $x_{i,1} = \text{Inp}_i(0^*, 0^*)$,
 - Otherwise, let $x_{i,1}, \dots, x_{i,\lambda-1}$ and $y_{i,1}, \dots, y_{i,\lambda-1}$ denote the inputs and outputs from the evaluations of the functions $f_1, \dots, f_{\lambda-1}$, respectively. $\mathcal{F}_{\text{RC}}^{\vec{f}}$ computes $x_{i,\lambda} = \text{Inp}_i((x_{i,1} \dots, x_{i,\lambda-1}), (y_{i,1} \dots, y_{i,\lambda-1}))$.

\mathcal{S} gets to corrupt parties adaptively as in [Can00] (upon corruption of some p_i , \mathcal{S} gets to see and possible change p_i 's input and all outputs generated for him so far, i.e., output from the evaluation of $f_1, \dots, f_{\lambda-1}$). For each $p_i \in \mathcal{I}$, \mathcal{S} sends some input $x'_{i,\lambda}$ to $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ (if \mathcal{S} sends no value or an invalid value for some $p_i \in \mathcal{I}$ then the functionality takes a default value d for this input). $\mathcal{F}_{\text{RC}}^{\vec{f}}$ denotes the vector of all inputs for f_λ as $(x'_{1,\lambda}, \dots, x'_{n,\lambda})$.

2. $\mathcal{F}_{\text{RC}}^{\vec{f}}$ chooses $r_\lambda \in R$ uniformly at random and computes

$$(y_{1,\lambda}, \dots, y_{n,\lambda}) = f((x'_{1,\lambda}, \dots, x'_{n,\lambda}, r_\lambda), (x'_{1,\lambda-1}, \dots, x'_{n,\lambda-1}, r_{\lambda-1}), (x'_{1,1}, \dots, x'_{n,1}, r_1)),$$

where R_ℓ , $\ell = 1, \dots, \lambda$ denotes the randomness used by $\mathcal{F}_{\text{RC}}^{\vec{f}}$ for the evaluation of f_ℓ .

3. Sim sends $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ a message $s = \text{abort}$ (or no message, i.e., $s = \perp$). If \mathcal{S} sends $s = (\text{abort})$, then for every (honest) $p_j \in \mathcal{P} \setminus \mathcal{I}$, $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ updates $y_{j,\lambda} := (\text{abort})$ for all $(j, \lambda) \in [n] \times [m]$ and goes to Step C (i.e, skips the remainder of the computation and produces output).
4. \mathcal{S} sends $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ a message $o \in (\text{YesOut}, \text{NoOut})$. If $o = \text{YesOut}$ then $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ sends to \mathcal{S} the outputs $\{y_{i,\lambda}\}_{i \in \mathcal{I}}$ of corrupted parties p_i . (\mathcal{S} is as usually allowed to corrupt more parties adaptively.)
5. Sim sends $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ a message $s = \text{abort}$ (or no message, i.e., $s = \perp$). If \mathcal{S} sends $s = (\text{abort})$, then for every (honest) $p_j \in \mathcal{P} \setminus \mathcal{I}$, $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ updates $y_{j,\lambda} := (\text{abort})$ for all $(j, \lambda) \in [n] \times [m]$ and goes to Step C (i.e, skips the remainder of the computation and produces output).
6. For each $p_j \in \mathcal{P} \setminus \mathcal{I}$: $\mathcal{F}_{\text{RC}}^{\vec{f}}$ records p_j 's output $y_{j,\lambda}$.

C. For each honest p_i , $\mathcal{F}_{\text{RC}}^{\vec{f}}$ sends $\vec{y}_i = (y_{i,1}, \dots, y_{i,m})$ to p_i who outputs \vec{y}_i (corrupted parties output \perp)

B An Authenticated Secret Sharing Scheme

The sharing of a secret s (field element) is a pair (s_1, s_2) of random field elements (in some larger field) with the property that $s_1 + s_2 = (s, \text{tag}(s, k_1), \text{tag}(s, k_2))$, where k_1 and k_2 are MAC keys

associated with the parties p_1 and p_2 , respectively, and $\text{tag}(x, k)$ denotes a MAC tag for the value x computed with key k . We refer to the values s_1 and s_2 as the *summands*. Each $p_i \in \{p_1, p_2\}$ holds his *share* $(s_i, \text{tag}(s_i, k_{-i}))$ along with the MAC key k_i which is used for the generation of the MAC tags he is supposed to verify. We denote by $\langle s \rangle$ a sharing of s and by $\langle s \rangle_i$ party p_i 's share. The above sharing can be reconstructed towards any of the parties p_i as follows: p_{-i} sends his share $\langle s \rangle_{-i} = (s_{-i}, t_{-i})$ to p_i who, using k_i , verifies that t_{-i} is a valid MAC for s_{-i} . Subsequently, p_i reconstructs the authenticated secret s by computing $s_1 + s_2 := (s, t'_1, t'_2)$ and verifying, using key k_i , that t'_i is a valid MAC for s . If any of the MAC verifications fails then p_i aborts and outputs \perp .

C Fair and Reactive 2PC (cont'd)

C.1 Upper Bound Proofs

Theorem 3. *Let $\vec{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \Gamma_{\text{fair}}$. Then there exists an r -reconstruction-round protocol Π such that*

$$\bar{u}_{\mathbf{A}}(\Pi, \mathcal{A}) \stackrel{\text{negl}}{\leq} \max \left\{ \frac{\gamma_{10}}{2 \sum_{\ell=1}^{r-1} \varrho^{\ell-1}}, \frac{\gamma_{10} + \gamma_{11}}{2} \right\},$$

with $\varrho = \left| \frac{\gamma_{01}}{\gamma_{10}} \right|$. In particular, if $\gamma_{11} > -\gamma_{10}$, then: $\bar{u}_{\mathbf{A}}(\Pi, \mathcal{A}) \stackrel{\text{negl}}{\geq} \frac{\gamma_{10} + \gamma_{11}}{2}$.

Proof. The case $\gamma_{11} \geq 0$ follows as in [GKTZ15], and the main statement immediately follows from Lemma 7. The particular statement for $0 > \gamma_{11} > -\gamma_{10}$ follows by additionally using Lemma 8 to obtain the necessary number of rounds. \square

The following lemma analyzes the protocol and is hence crucial in the above theorem.

Lemma 7. *Let $\vec{\gamma} = (\gamma_{00}, \gamma_{01}, \gamma_{10}, \gamma_{11}) \in \Gamma_{\text{fair}}$ and $\varrho = \left| \frac{\gamma_{01}}{\gamma_{10}} \right|$, and let \mathcal{D} denote the following probability distribution on $[r-1]$: $\Pr_{\varrho}(1) = \left(\sum_{\ell=1}^{r-1} \varrho^{\ell-1} \right)^{-1}$, and for $\ell \in \{2, \dots, r-1\}$: $\Pr_{\varrho}(\ell) = \varrho^{\ell-1} \Pr_{\varrho}(1)$. Then, for any vector of functions $\vec{f} = (f_1, \dots, f_m)$, the protocol $\Pi_{\text{RC}}^{\text{fair}}(p_1, p_2, \mathcal{D}, r, f_1, \dots, f_m)$ securely computes \vec{f} , while for the utility of adversary \mathcal{A} it holds that*

$$\bar{u}_{\mathbf{A}}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}) \stackrel{\text{negl}}{\leq} \max \left\{ \frac{\gamma_{10}}{2 \sum_{\ell=1}^{r-1} \varrho^{\ell-1}}, \frac{\gamma_{10} + \gamma_{11}}{2} \right\}.$$

For simplicity, the following proof is phrased with respect to static adversaries. The generalization to the adaptive case is straightforward because the simulator never commits to any output (it can always simulate the necessary shares should a party become corrupted), and the transformations of the adversarial strategies in the second part of the proof apply equally to static and adaptive strategies, the described adversarial strategies simply corrupt either p_1 or p_2 with some probability.

Proof (sketch). We first construct a simulator $\mathcal{S}_{\mathcal{A}}$ for proving security with respect to $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$. Then, we show that $\mathcal{S}_{\mathcal{A}}$ achieves the upper bound claimed in the lemma. In case either no party or both parties are corrupted, the lemma follows from the non-triviality of the protocol and the assumption that $\vec{\gamma} \in \Gamma_{\text{fair}}$. For the remainder of the proof we consider an adversary who corrupts exactly one party.

We describe the simulator $\mathcal{S}_{\mathcal{A}}$ for an adversary \mathcal{A} corrupting p_1 (the case where p_2 is corrupted is handled analogously). The simulator uses \mathcal{A} in a straight-line black-box manner, where all

messages sent from \mathcal{Z} to \mathcal{A} (and vice-versa) are forwarded to their respective recipient. The following description applies to the computation of each f_λ for $\lambda = 1, \dots, m$.

- At the point where the adversary would send p_1 's input $x'_{1,\lambda}$ to the functionality $\mathcal{F}_{\text{SFE}}^{f_{\text{sh}}, f_\lambda, \mathcal{D}, \perp}$, the simulator $\mathcal{S}_\mathcal{A}$ sends $x'_{1,\lambda}$ to $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$. When the adversary aborts $\mathcal{F}_{\text{SFE}}^{f_{\text{sh}}, f_\lambda, \mathcal{D}, \perp}$, then $\mathcal{S}_\mathcal{A}$ sends (**abort**) to $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$.
- For simulating each of the reconstruction phases, $\mathcal{S}_\mathcal{A}$ selects an index $i \in_R \{1, 2\}$ uniformly at random. We consider the cases $i = 1$ and $i = 2$ separately. In case $i = 2$ (i.e., the index of the honest party is chosen): For each round $l \in [r - 1]$: $\mathcal{S}_\mathcal{A}$ selects a summand $s_2^{(\lambda)}$ that together with the summand of \mathcal{A} reconstructs to **DummyRound**, sends $s_2^{(\lambda)}$ to \mathcal{A} ; in the r -th round, $\mathcal{S}_\mathcal{A}$ requests p_1 's f_λ -output from $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$ by sending **YesOut**, chooses a summand $s_2^{(\lambda)}$ that together with the summand of \mathcal{A} reconstructs to this output, and sends $s_2^{(\lambda)}$ to \mathcal{A} . If, in any one of the rounds, \mathcal{A} sends inconsistent shares, then $\mathcal{S}_\mathcal{A}$ sends **NoOut** and (**abort**) to $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$. In case $i = 1$ (i.e., the index of the corrupted party is chosen), $\mathcal{S}_\mathcal{A}$ samples a round l^* according to the distribution \mathcal{D} and the same as before, with the only difference that in round l^* , $\mathcal{S}_\mathcal{A}$ requests p_1 's f_λ -output from $\mathcal{F}_{\text{RC}}^{\vec{f}, \perp}$, and reconstructs this output (instead of **DummyRound**) to \mathcal{A} .

The fact that the above is a good simulation for protocol Π follows directly in the $\mathcal{F}_{\text{SFE}}^{f_{\text{sh}}, f_\lambda, \mathcal{D}, \perp}$ -hybrid model from the fact that the party which receives the “early output” as well as the corresponding round are chosen according to the same distribution as the function $f_{\text{sh}, f_\lambda, \mathcal{D}}$.

To complete the proof we need to show that the above simulator achieves the bound stated in the lemma. An important observation is that $\mathcal{S}_\mathcal{A}$ has the following property: It starts simulating the evaluation of the λ -th function f_λ of \vec{f} only after asking for the output of each of the previous functions $f_1, \dots, f_{\lambda-1}$ and only if it has not (yet) sent the **abort** symbol to the functionality.

For the next claim, we describe the following adversaries $\mathcal{A}_1^{(i)}, \dots, \mathcal{A}_r^{(i)}$: $\mathcal{A}_\ell^{(i)}$ corrupts p_i in the onset of the execution and (in principle) behaves honestly up to round ℓ in the reconstruction of the outputs of f_m . For the first $\ell - 1$ rounds, $\mathcal{A}_\ell^{(i)}$ first checks whether it obtained the output already. In this case, it aborts, otherwise it follows the protocol for this round. In round ℓ , however, $\mathcal{A}_\ell^{(i)}$ aborts independently of whether it obtained the message.

Claim 1. *For some $i \in \{1, 2\}$ and some $\ell \in [r]$,*

$$\bar{u}_\mathbf{A}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}_\ell^{(i)}) \stackrel{\text{negl}}{\geq} \bar{u}_\mathbf{A}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}),$$

for all adversaries \mathcal{A} .

Proof (sketch). Let \mathcal{A} be an arbitrary adversary. We denote by \mathcal{Z} a specific environment that maximizes the adversary's payoff. We transform \mathcal{A} in several steps and show that each of the transformations only preserves or increases the payoff.

First, we transform \mathcal{A} into an adversary \mathcal{A}' that behaves similarly to \mathcal{A} , but whenever \mathcal{A} (after corrupting a party) sends to the honest party a message that is not the correct share according to the output of the functionality, then \mathcal{A}' simply aborts. Note that—except with the negligible probability of forging a share of the authenticated sharing—sending a value different from the original share will make the honest party abort the protocol. Hence,

$$\bar{u}_\mathbf{A}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}') \stackrel{\text{negl}}{\geq} \bar{u}_\mathbf{A}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}).$$

Second, we transform \mathcal{A}' into an adversary \mathcal{A}'' that, for each message obtained from the honest party while evaluating the last function f_m of \vec{f} , checks whether it obtained the output already.⁷ In this case, \mathcal{A}'' aborts immediately. Clearly, \mathcal{A}'' obtains all outputs whenever \mathcal{A}' does, and the honest parties only obtain all outputs in an execution with \mathcal{A}'' if they would have obtained them with \mathcal{A}' as well. By the natural assumptions on the payoff vector ($\gamma_{10} \geq \max\{\gamma_{00}, \gamma_{11}\}$ and $\min\{\gamma_{00}, \gamma_{11}\} \geq \gamma_{01}$),

$$\bar{u}_{\mathbf{A}}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}'') \stackrel{\text{negl}}{\geq} \bar{u}_{\mathbf{A}}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}').$$

The adversary \mathcal{A}'' can be written as a convex combination of the adversaries $\mathcal{A}_1, \dots, \mathcal{A}_r$. Note that the values that \mathcal{A}'' obtained from the execution of $\Pi_{\text{RC}}^{\text{fair}}$ before the reconstruction are statistically independent of the round and party chosen in the evaluation of $f_{\text{sh},f,\mathcal{D}}$. Even together with all shares received from p_2 up to round ℓ , all values have the same distribution in all cases where either the chosen round is larger than ℓ or the chosen party is p_2 . Consequently, the distribution of rounds in which \mathcal{A}'' aborts prematurely (i.e., without receiving the output) and the choice of the party to corrupt are independent of the actual values chosen in $f_{\text{sh},f,\mathcal{D}}$. As \mathcal{A}'' can be written as a convex combination of the \mathcal{A}_ℓ , there is necessarily one such strategy which is at least as good as \mathcal{A}'' . \square

It remains to compute the payoff of the adversarial strategies $\mathcal{A}_1, \dots, \mathcal{A}_r$. For \mathcal{A}_1 , the payoff is

$$\bar{u}_{\mathbf{A}}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}_1) \stackrel{\text{negl}}{\approx} \frac{1}{2} \Pr_{\ell}(1) \gamma_{10} = \frac{\gamma_{10}}{2 \sum_{\ell=1}^{r-1} \varrho^{\ell-1}},$$

since \mathcal{A}_1 obtains the value only if $l^* = 1$ and $i^* = 1$, and p_2 never obtains the value. Analogously,

$$\begin{aligned} \bar{u}_{\mathbf{A}}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}_\ell) &\stackrel{\text{negl}}{\approx} \frac{1}{2} \left(\sum_{j=1}^{\ell} \Pr_{\ell}(j) \right) \cdot \gamma_{10} + \frac{1}{2} \left(\sum_{j=1}^{\ell-1} \Pr_{\ell}(j) \right) \cdot \gamma_{01} \\ &= \frac{1}{2} \sum_{j=1}^{\ell} \varrho^{j-1} \Pr_{\ell}(1) \cdot \gamma_{10} + \frac{1}{2} \sum_{j=1}^{\ell-1} \varrho^{j-1} \Pr_{\ell}(1) \cdot \gamma_{01} \\ &= \frac{1}{2} \sum_{j=1}^{\ell} \varrho^{j-1} \Pr_{\ell}(1) \cdot \gamma_{10} - \frac{1}{2} \sum_{j=2}^{\ell} \varrho^{j-1} \Pr_{\ell}(1) \cdot \gamma_{10} = \frac{\gamma_{10}}{2 \sum_{\ell=1}^{r-1} \varrho^{\ell-1}}, \end{aligned}$$

for all $l < r$. The adversary \mathcal{A}_r is exactly the strategy considered in the non-reactive setting, so

$$\bar{u}_{\mathbf{A}}(\Pi_{\text{RC}}^{\text{fair}}, \mathcal{A}_r) \stackrel{\text{negl}}{\approx} \frac{\gamma_{10} + \gamma_{11}}{2}.$$

Altogether, we obtain the claimed bound. \square

Lemma 8. *For a preference vector $\vec{\gamma} \in \Gamma$ with $\gamma_{11} > -\gamma_{10}$. The number of rounds required to make the generic strategy \mathcal{A}_{gen} optimal is:*

– if $-\gamma_{01} = \gamma_{10}$, then

$$r \geq \frac{2\gamma_{10} + \gamma_{11}}{\gamma_{10} + \gamma_{11}};$$

– else,

$$r \geq \left\lceil \log_{-\gamma_{01}/\gamma_{10}} \left(\frac{\gamma_{11} - \gamma_{01}}{\gamma_{10} + \gamma_{11}} \right) \right\rceil + 1.$$

⁷Note that \mathcal{A}'' can perform this check with at most negligible error probability, since $\Pi_{\text{RC}}^{\text{fair}}$ is secure with abort and during the evaluation of the last function f_m , the only possible outputs of the protocol are \perp or the correct value.

Proof. We can compute how many rounds a protocol must *at least* have to achieve the optimal bound, namely such that $\frac{1}{2} \left(\sum_{i=1}^{r-1} (-\gamma_{01})^{i-1} / \gamma_{10}^i \right)^{-1} \leq \frac{\gamma_{10} + \gamma_{11}}{2}$, so after multiplying by 2:

$$\left(\sum_{i=1}^{r-1} (-\gamma_{01})^{i-1} / \gamma_{10}^i \right)^{-1} = \gamma_{10} \left(\sum_{i=1}^{r-1} (-\gamma_{01} / \gamma_{10})^{i-1} \right)^{-1} \leq \gamma_{10} + \gamma_{11}.$$

First, if $-\gamma_{01} = \gamma_{10}$, then the above equation means that

$$r \geq \frac{2\gamma_{10} + \gamma_{11}}{\gamma_{10} + \gamma_{11}}.$$

Otherwise, if $-\gamma_{01} \neq \gamma_{10}$, we compute the partial sums and obtain (after inverting)

$$\begin{aligned} \gamma_{10}^{-1} \sum_{i=0}^{r-2} (-\gamma_{01} / \gamma_{10})^i &= \gamma_{10}^{-1} \frac{(-\gamma_{01} / \gamma_{10})^{r-1} - 1}{(-\gamma_{01} / \gamma_{10}) - 1} \\ &\geq \frac{1}{\gamma_{10} + \gamma_{11}}, \text{ or} \\ \frac{(-\gamma_{01} / \gamma_{10})^{r-1} - 1}{(-\gamma_{01} / \gamma_{10}) - 1} &\geq \frac{\gamma_{10}}{\gamma_{10} + \gamma_{11}}. \end{aligned}$$

If $-\gamma_{01} > \gamma_{10}$, then

$$\begin{aligned} (-\gamma_{01} / \gamma_{10})^{r-1} &\geq \frac{-\gamma_{01} - \gamma_{10}}{\gamma_{10} + \gamma_{11}} + 1 \\ &= \frac{\gamma_{11} - \gamma_{01}}{\gamma_{10} + \gamma_{11}}, \end{aligned}$$

which means

$$r - 1 \geq \log_{-\gamma_{01} / \gamma_{10}} \left(\frac{\gamma_{11} - \gamma_{01}}{\gamma_{10} + \gamma_{11}} \right).$$

(The same inequality arises for $-\gamma_{01} < \gamma_{10}$ because the inequality is first a “ \leq ,” but then the base in the logarithm is < 1 .) As the number of rounds must be an integer, we obtain

$$r \geq \left\lceil \log_{-\gamma_{01} / \gamma_{10}} \left(\frac{\gamma_{11} - \gamma_{01}}{\gamma_{10} + \gamma_{11}} \right) \right\rceil + 1,$$

as we have to round up the number of rounds to make the “generic” adversarial strategies optimal. \square

C.2 Lower Bound Proofs

Lemma 4. *For every protocol Π which securely implements the (unfair) functionality $f_{2\text{Ex}}^\perp$, there exists an adversary \mathcal{A} with*

$$\bar{u}_{\mathcal{A}}(\Pi, \mathcal{A}) \stackrel{\text{negl}}{\geq} \frac{\gamma_{10} + \gamma_{11}}{2}.$$

Proof. As in [GKTZ15, Theorem 5], we consider the adversary strategies \mathcal{A}_1 and \mathcal{A}_2 , where \mathcal{A}_i (statically) corrupts p_i and follows the protocol until the first output phase is finished. In each round of the second output phase, \mathcal{A}_i receives all messages from p_{-i} , checks whether the protocol

for p_1 provides the output,⁸ and in that case records the output and abort the execution before sending p_{-i} 's message for that round.⁹ Otherwise, let p_1 correctly execute its instructions for that round.

We use the following claim, which is proven analogously to Lemma [GKTZ15, Lemma 4].

Claim 2. *Let $\vec{\gamma} \in \Gamma$ be a preference vector. For every protocol Π which securely implements the (unfair) functionality $f_{2\text{Ex}}^\perp$ the following inequality holds:*

$$\bar{u}_A(\Pi, \mathcal{A}_1) + \bar{u}_A(\Pi, \mathcal{A}_2) \stackrel{\text{negl}}{\geq} \gamma_{10} + \gamma_{11}.$$

The adversary \mathcal{A} chooses one of the strategies \mathcal{A}_1 or \mathcal{A}_2 uniformly at random. As a result, we obtain

$$\bar{u}_A(\Pi, \mathcal{A}) = \frac{1}{2} \cdot \bar{u}_A(\Pi, \mathcal{A}_1) + \frac{1}{2} \cdot \bar{u}_A(\Pi, \mathcal{A}_2) \stackrel{\text{negl}}{\geq} \frac{1}{2}(\gamma_{10} + \gamma_{11}).$$

□

Lemma 5. *Let Π be an r -round-reconstruction protocol that securely implements the (unfair) functionality $f_{2\text{Ex}}^\perp$, such that*

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

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

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

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

Proof. We start with a rough outline of the proof. First, we measure certain parameters of the protocol in a “benign” environment. Second, we construct a (generalized) “probabilistic release” protocol that is *at least as fair as* the original protocol. Third, we prove a lower bound on the adversary’s payoff for *any* protocol of this “probabilistic release” type.

Measuring the reconstruction phase. Consider the dummy adversary \mathcal{D} (which simply forwards all communication from and to the environment) and the environment \mathcal{Z} that admits the execution of the protocol up to the round where both parties provide as output the first half of the other party’s input. Note that, from this point on, if the adversary aborts, the only choice for honest party is to also abort (and provide output only if it already obtained the correct value). This is due to the fact that the adversary already obtained output and the honest party cannot simply restart the protocol with a default input for the adversary.

We fix this particular state of the execution (including the random tapes, so the execution is deterministic), and we measure, for both p_1 and p_2 , the “first round in which p_i would provide output, given that p_{-i} would abort in that round.” (Formally, we copy the state of the complete system and execute it with adversaries $\mathcal{D}^{(1)}$ and $\mathcal{D}^{(2)}$ where $\mathcal{D}^{(i)}$ corrupts p_{-i} and, in each round, checks whether p_{-i} would output the second half of p_i 's input before actually sending the message of the round.) For such a fixed state, we obtain a pair $(\ell_1, \ell_2) \in [r]^2$ of round numbers. Note that, except for a negligible fraction of the cases, both parties will indeed provide output in some

⁸This is possible since as a second output in the reactive computation, the protocol may output only either the correct value or an “abort” symbol.

⁹This attack is possible because the adversary is rushing.

round, as the protocol is secure with abort and the other party follows the protocol. This procedure determines a random distribution over pairs of rounds, which we denote by D .

Constructing the protocol. Given such a distribution, we construct a “canonical” protocol $\bar{\pi}$ that is at least as fair as Π . In particular, the payoff of a “benign” strategy with respect to Π is at least as large (up to a negligible difference) as the payoff with respect to $\bar{\pi}_D$, and the best strategies against $\bar{\pi}_D$ are indeed the “benign” ones.

Protocol $\bar{\pi}_D$ for 2 Parties.

The following $\mathcal{F}_{\text{RC}}^{\bar{f}, \perp}$ -hybrid^a multi-party protocol (with D a distribution on $[r] \times [r]$) computes the functionality $f_{2\text{Ex}}^\perp$. Define the function \bar{f} as

$$\begin{aligned} \bar{f} : (\{0, 1\}^{2k})^2 \times (\mathcal{R} \times [r]^2) &\rightarrow \{0, 1\}^k \times \{0, 1\}^k \times (S^2 \times S^2)^r \\ (y_1|z_1, y_2|z_2, R, \ell_1, \ell_2) &\mapsto (y_2, y_1, \langle \delta_{1\ell_1}(z_2) \rangle, \langle \delta_{1\ell_2}(z_1) \rangle, \dots, \\ &\quad \langle \delta_{r\ell_1}(z_2) \rangle, \langle \delta_{r\ell_2}(z_1) \rangle), \end{aligned}$$

where the pair (ℓ_1, ℓ_2) is chosen according to the distribution D .

Computation: Use $\mathcal{F}_{\text{SFE}}^{\bar{f}, \perp}$ to compute the respective outputs of \bar{f} . A party not obtaining its output aborts.

Reconstruction: In each round, each party sends the respective share of the other party for the current round. A party that does not receive an expected (correct) share aborts (but provides output if applicable).

Output: A party that obtained a value $\neq \perp$ in one of the previous rounds outputs this value (and \perp otherwise).

^aRecall that when $\mathcal{F}_{\text{RC}}^\perp$ is parameterized with a single function—as is the case with $\mathcal{F}_{\text{RC}}^{\bar{f}, \perp}$ —then it corresponds to the standard SFE functionality $\mathcal{F}_{\text{SFE}}^{\bar{f}, \perp}$ (i.e., computation of non-reactive functions) with unfair abort as in [GKTZ15].

Of course, we have to show that the protocol $\bar{\pi}$ indeed implements the functionality $f_{2\text{Ex}}^\perp$.

Claim 3. *The protocol $\bar{\pi}$ implements the functionality $f_{2\text{Ex}}^\perp$.*

Proof. This proof is similar to the “security with abort” part in the proof of Theorem 3; the correctness follows from the security of the authenticated sharing. \square

Let $\mathcal{A}_\ell^{(i)}$ and $\mathcal{Z}_\ell^{(i)}$ for $i \in \{1, 2\}$ and $\ell \in [r]$ be the pair of adversary and environment that corrupts party i and begins to run the protocol faithfully. Indeed, $\mathcal{A}_\ell^{(i)}$ behaves exactly as \mathcal{A}_{gen} up to round ℓ , but aborts after obtaining and checking the message in this round ℓ .

Claim 4. *For the above described adversarial strategies $\mathcal{A}_\ell^{(i)}$ and environments $\mathcal{Z}_\ell^{(i)}$,*

$$\inf_{S \in \text{SIM}_{\mathcal{A}_\ell^{(i)}}} \{U^{f_{2\text{Ex}}^\perp, \vec{\gamma}}(S, \mathcal{Z}_\ell^{(i)})\} \stackrel{\text{negl}}{\geq} \inf_{S \in \text{SIM}_{\mathcal{A}_\ell^{(i)}}} \{U^{f_{2\text{Ex}}^\perp, \vec{\gamma}}(S, \mathcal{Z}_\ell^{(i)})\}.$$

Proof. The probability that the adversary obtains the output is the same in both cases by the definition of the protocol $\bar{\pi}_D$. Yet, the probability that the honest party outputs the value in the case of $\bar{\pi}_D$ is *exactly* according to the distribution D , whereas for Π it is only *upper bounded* by D (as the honest party in Π might have output the value in a previous round, but output \perp again in later rounds). Assuming that $\gamma_{10} \geq \gamma_{11}$ and $\gamma_{00} \geq \gamma_{01}$, the statement of the claim follows (the negligible slackness appears as the real payoff is defined via the ideal payoff). \square

The above claim only makes a statement about a specific type of adversary, and of course

$$\sup_{\mathcal{A}} \bar{u}_{\mathbf{A}}(\Pi, \mathcal{A}) \stackrel{\text{negl}}{\geq} \bar{u}_{\mathbf{A}}(\Pi, \mathcal{A}_{\ell}^{(i)}) \stackrel{\text{negl}}{\geq} \inf_{\mathcal{S} \in \text{SIM}_{\mathcal{A}_{\ell}^{(i)}}} \{U^{f_{2\text{Ex}}, \vec{\gamma}}(\mathcal{S}, \mathcal{Z}_{\ell}^{(i)})\}.$$

For general protocols such as Π there might be more successful adversarial strategies than $\mathcal{A}_{\ell}^{(i)}$. In the particular case of $\bar{\pi}_D$, however, these strategies are indeed optimal.

Claim 5. *For some $i \in \{1, 2\}$ and $\ell \in [r]$,*

$$\inf_{\mathcal{S} \in \text{SIM}_{\mathcal{A}_{\ell}^{(i)}}} \{U^{f_{2\text{Ex}}, \vec{\gamma}}(\mathcal{S}, \mathcal{Z}_{\ell}^{(i)})\} \stackrel{\text{negl}}{\geq} \bar{u}_{\mathbf{A}}(\bar{\pi}_D, \mathcal{A}),$$

for all adversaries \mathcal{A} , where $\mathcal{A}_r = \mathcal{A}_{\text{gen}}$.

Proof (sketch). Let \mathcal{A} be an arbitrary adversary. By the definition of the payoff,

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_D, \mathcal{A}) \stackrel{\text{negl}}{\approx} \sup_{\mathcal{Z}} \left\{ \inf_{\mathcal{S} \in \text{SIM}_{\mathcal{A}}} \{U^{f_{2\text{Ex}}, \vec{\gamma}}(\mathcal{S}, \mathcal{Z})\} \right\},$$

and we denote by \mathcal{Z} a specific environment that fulfills the condition. The remainder of the claim is proven exactly as the first claim in Theorem 3. \square

To simplify the lower bound proof for the payoff of the best adversary $\mathcal{A}_{\ell}^{(i)}$ and the protocol $\bar{\pi}_D$, we transform the distribution D in several steps. For each step, however, we show that the transformation can only lead to a “more secure” protocol. The first observation is that the protocol should never provide the output to both p_1 and p_2 in the same round: The “rushing” adversary will receive its message for the round first and abort before sending the response. Hence, define the distribution D' by setting $D'(\ell, \ell) = 0$, $D'(\ell, \ell + 1) = D(\ell, \ell + 1) + \frac{1}{2}D(\ell, \ell)$, and $D'(\ell + 1, \ell) = D(\ell + 1, \ell) + \frac{1}{2}D(\ell, \ell)$ for $\ell \in [r - 1]$, and $D'(\ell, \ell') = D(\ell, \ell')$ everywhere else.

Claim 6. *For D and D' as described above, we have*

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_D, \mathcal{A}_{\ell}^{(i)}) \stackrel{\text{negl}}{\geq} \bar{u}_{\mathbf{A}}(\bar{\pi}_{D'}, \mathcal{A}_{\ell}^{(i)}),$$

for $i \in \{1, 2\}$ and all $\ell \in [r]$.

Proof. We only have to argue that the adversary’s payoff in the case (ℓ, ℓ) is at least as large as in the case $(\ell, \ell + 1)$ (respectively $(\ell + 1, \ell)$). For adversaries $\mathcal{A}_{\ell'}^{(i)}$ with $\ell' < \ell$, the payoff remains constant. If $\ell' \geq \ell$ and $\mathcal{A}_{\ell'}^{(i)}$ obtains the value in round ℓ , the payoff is also the same as for (ℓ, ℓ) . If, however, $\mathcal{A}_{\ell'}^{(i)}$ obtains the value only in round $\ell + 1$, the payoff decreases from γ_{10} to γ_{11} . \square

From the distribution D' we define a distribution D'' such that the party that obtains the output later will receive it only in the last round. Formally, $D''(\ell, \ell') = 0$ for $\ell < r$ and $\ell' < r$, and $D''(\ell, r) = \sum_{\ell' > \ell} D'(\ell, \ell')$.

Claim 7. For D' and D'' as described above, we have

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{D'}, \mathcal{A}_\ell^{(i)}) \stackrel{\text{negl}}{\geq} \bar{u}_{\mathbf{A}}(\bar{\pi}_{D''), \mathcal{A}_\ell^{(i)}).$$

Proof. In cases where the adversary obtains the output first, the payoff remains constant (either the adversary gets the output in both cases or in none; the honest party does not get the output). If the honest party gets the value first, then either the payoff remains constant (for $\mathcal{A}_r^{(i)}$ or if the adversary aborts before obtaining the output also in $\bar{\pi}_{D'}$), or the payoff decreases if the adversary aborts before the final round (as $\gamma_{00} \geq \gamma_{01}$). \square

For the remaining part of the proof, we consider the adversaries $\mathcal{A}_1, \dots, \mathcal{A}_{r-1}$ that behave as follows: In the beginning, \mathcal{A}_ℓ chooses a party p_i to corrupt uniformly at random and then behaves as $\mathcal{A}_\ell^{(i)}$. In this setting, it makes sense to consider the protocol $\bar{\pi}$ as being parametrized by a vector

$$\vec{q} \in \text{dist}_{r-1}(q_r) \quad := \quad \left\{ (q_1, \dots, q_{r-1}) \in [0, 1]^{r-1} \mid \sum_{\ell=1}^r q_\ell = 1 \right\}$$

that we obtain essentially as a symmetric version of D'' . Roughly speaking, each party obtains the output in round ℓ with probability exactly q_ℓ , but these probabilities are of course not independent— at least one party will always obtain the value only in the last round (so $q_r \geq 1/2$). For (the best of) these adversaries, we can explicitly prove a lower bound on the utility when attacking the protocol $\bar{\pi}_{\vec{q}}$.

Claim 8 (tradeoff). Consider the adversaries $\mathcal{A}_1, \dots, \mathcal{A}_r$ as above, and $q_r \in [1/2, 1]$. Then,

$$\min_{\vec{q} \in \text{dist}_{r-1}(q_r)} \max_{1 \leq \ell \leq r-1} \bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_\ell) \stackrel{\text{negl}}{\geq} \frac{(1 - q_r)\gamma_{10}}{\sum_{\ell=1}^{r-1} \rho^{\ell-1}}.$$

Proof. We can easily compute the adversary's payoff for these fixed strategies. For \mathcal{A}_ℓ , the probability that the adversary obtains the output is $\sum_{j=1}^{\ell} q_j$, and the payoff in this case is γ_{10} . The probability that the honest party obtains the output is $\sum_{j=1}^{\ell-1} q_j$. As the adversary aborts prematurely, the payoff in this case is γ_{01} , which results in

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_\ell) \stackrel{\text{negl}}{\approx} \left(\sum_{j=1}^{\ell} q_j \right) \cdot \gamma_{10} + \left(\sum_{j=1}^{\ell-1} q_j \right) \cdot \gamma_{01}.$$

Our goal is to find the optimal vector \vec{q} , that is, the payoff of the *best* adversary is minimized. We encounter a trade-off: By changing the vector \vec{q} to reduce the payoff of a certain adversary, we inevitably increase the payoff for different adversary. Consequently, for an optimal \vec{q} all adversaries have essentially the same payoff, which is formalized in the following claim (if this is not the case, we can provably reduce the maximum payoff).

Claim 9. If \vec{q} is optimal, then

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_\ell) \stackrel{\text{negl}}{\approx} \bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_{\ell'})$$

for all $\ell, \ell' \in [r - 1]$.

Proof. Assume that \vec{q} is optimal but there exists a ℓ such that

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_{\ell}) \stackrel{\text{negl}}{\approx} \max_{\ell < \ell' < r} \bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_{\ell'})$$

and

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_{\ell}) = \bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_{\ell-1}) + \delta$$

with noticeable δ . For $\varepsilon = \delta/(\gamma_{10} - \gamma_{01})$, define the vector \vec{q}' with $q'_u = q_u$ for $\ell \neq u \neq \ell - 1$, $q'_\ell = q_\ell - \varepsilon$, and $q'_{\ell-1} = q_{\ell-1} + \varepsilon$. Then,

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}'}, \mathcal{A}_u) = \bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_u)$$

for all $\ell \neq u \neq \ell - 1$,

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}'}, \mathcal{A}_{\ell-1}) \stackrel{\text{negl}}{\approx} \bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_{\ell-1}) + \gamma_{10}/(\gamma_{10} - \gamma_{01})\delta,$$

and

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}'}, \mathcal{A}_{\ell}) \stackrel{\text{negl}}{\approx} \bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_{\ell}) + \gamma_{01}/(\gamma_{10} - \gamma_{01})\delta.$$

As a result, the payoff of the adversary \mathcal{A}_{ℓ} decreases by a noticeable amount. If \mathcal{A}_{ℓ} was the only strategy with maximal payoff, we already reached a contradiction. If there were more than one strategy with maximal payoff, then this procedure must be applied repeatedly. (Of course, the analogous argument can also be made for the other direction, i.e., by shifting “probability mass” from ℓ to $\ell + 1$.) \square

Following the claim, we can now compute an optimal vector \vec{q} and the payoff of the individual adversaries. In particular, the claim implies that $q_1\gamma_{10} = (q_1 + q_2)\gamma_{10} + q_1\gamma_{01} = \dots$, which means

$$\left(\sum_{u=1}^{j+1} q_u\right) \gamma_{10} + \left(\sum_{u=1}^j q_u\right) \gamma_{01} = \left(\sum_{u=1}^j q_u\right) \gamma_{10} + \left(\sum_{u=1}^{j-1} q_u\right) \gamma_{01},$$

which can be reformulated as

$$q_{j+1} = q_j \left(\frac{-\gamma_{01}}{\gamma_{10}}\right).$$

With $\varrho := -\frac{\gamma_{01}}{\gamma_{10}}$, we obtain by induction that $q_j = \varrho^{j-1}q_1$. By the fact that $\sum_{j=1}^r q_j = 1$, this means that

$$\sum_{j=1}^{r-1} q_j = \sum_{j=1}^{r-1} (\varrho^{j-1}q_1) = \left(\sum_{j=1}^{r-1} \varrho^{j-1}\right) \cdot q_1 = 1 - q_r,$$

or $q_1 = (1 - q_r) \left(\sum_{j=1}^{r-1} \varrho^{j-1}\right)^{-1}$. By the above formula, we compute the payoff of \mathcal{A}_1 as

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}_1) \stackrel{\text{negl}}{\approx} q_1\gamma_{10} = \frac{(1 - q_r) \cdot \gamma_{10}}{\sum_{j=1}^{r-1} \varrho^{j-1}},$$

which is equal to the payoff of all adversaries $\mathcal{A}_2, \dots, \mathcal{A}_{r-1}$. \square

Note that if $q_r = \frac{1}{2} + x$, then

$$\bar{u}_{\mathbf{A}}(\bar{\pi}_{\mathbf{D}}, \mathcal{A}_{\text{gen}}) \stackrel{\text{negl}}{\approx} (1/2 + x)\gamma_{10} + (1/2 - x)\gamma_{11} = \frac{\gamma_{10} + \gamma_{11}}{2} + x(\gamma_{10} - \gamma_{11}),$$

together with the statement about \mathcal{A}_{gen} we obtain $x \leq \frac{\omega}{\gamma_{10} - \gamma_{11}}$. Hence, we obtain by the trade-off claim that there exists an adversary \mathcal{A} with

$$\bar{u}_{\mathcal{A}}(\bar{\pi}_{\vec{q}}, \mathcal{A}) \stackrel{\text{negl}}{\geq} \frac{(1 - q_r)\gamma_{10}}{\sum_{\ell=1}^{r-1} \varrho^{\ell-1}} = \frac{(1/2 - x)\gamma_{10}}{\sum_{\ell=1}^{r-1} \varrho^{\ell-1}} \geq \frac{\left(\frac{1}{2} - \frac{\omega}{\gamma_{10} - \gamma_{11}}\right)\gamma_{10}}{\sum_{\ell=1}^{r-1} \varrho^{\ell-1}},$$

and by all claims proven before, there is an adversary for protocol Π which is at least as good. \square

D Asymptotic Behavior of the Number of Rounds

As Figure 4 illustrates, the number of rounds grows infinitely for $\varrho \rightarrow 2$. The following corollary, however, states that this growth is moderate in the sense that whenever ϱ is bounded away from 2 by a noticeable distance, the necessary number of rounds remains linear in the security parameter. The main reason for the function to diverge at this point is that $\gamma_{11} \rightarrow -\gamma_{10}$, which means that overall we approach the situation where the adversary prefers to always abort and obtain $\gamma_{00} = 0$ (as $(\gamma_{10} + \gamma_{11})/2 \rightarrow 0$ as well).

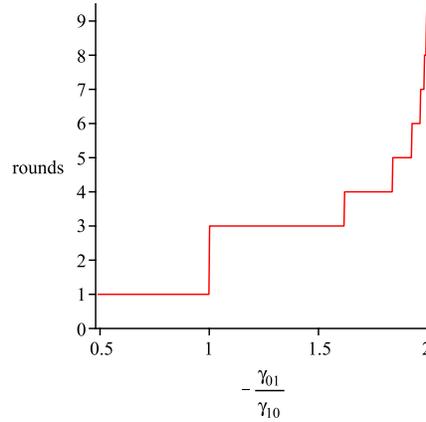


Figure 4: The optimal number of rounds, given $\varrho = -\frac{\gamma_{01}}{\gamma_{10}}$

Corollary 9. *Let $\vec{\gamma} \in \Gamma$ be a preference vector, and assume that $\gamma_{11} = \gamma_{10} + \gamma_{01}$, and define $\varrho = -\frac{\gamma_{01}}{\gamma_{10}}$. If there exists a polynomial $q(k)$ such that $\varrho(k) + 1/q(k) \leq 2$, then the optimal number of rounds in Lemma 8 is linear in k .*

Proof. We compute

$$\begin{aligned} \hat{r}(k) - 1 &= \left\lceil \log_{\varrho} \left(\frac{\gamma_{11} - \gamma_{01}}{\gamma_{10} + \gamma_{11}} \right) \right\rceil = \left\lceil \log_{\varrho} \left(\frac{\gamma_{10}}{2\gamma_{10} + \gamma_{01}} \right) \right\rceil \\ &= \left\lceil \log_{\varrho} \left(\frac{1}{2 - \varrho(k)} \right) \right\rceil = \left\lceil -\frac{\log(2 - \varrho(k))}{\log \varrho(k)} \right\rceil, \end{aligned}$$

and realize that the enumerator diverges for $\varrho(k) \rightarrow 2$. In particular,

$$\hat{r}(k) - 1 = \left\lceil -\frac{\log(2 - \varrho(k))}{\log \varrho(k)} \right\rceil \leq \left\lceil -\frac{\log 1/q(k)}{\log \varrho(k)} \right\rceil = \left\lceil \frac{\log q(k)}{\log \varrho(k)} \right\rceil,$$

where for the denominator we have $\log \varrho(k) \rightarrow \log 2$. Hence, $\hat{r}(k) \leq c \cdot \log q(k)$ for some $c > 0$ and if the ratio is bounded away from 2 by a noticeable term, then the necessary number of rounds is linear. \square