Complete Fairness in Secure Two-Party Computation

S. Dov Gordon* Carmit Hazay† Jonathan Katz‡ Yehuda Lindell§

Abstract

In the setting of secure two-party computation, two mutually distrusting parties wish to compute some function of their inputs while preserving, to the extent possible, various security properties such as privacy, correctness, and more. One desirable property is *fairness* which guarantees, informally, that if one party receives its output, then the other party does too. Cleve (STOC 1986) showed that complete fairness cannot be achieved *in general* without an honest majority. Since then, the accepted folklore has been that *nothing* non-trivial can be computed with complete fairness in the two-party setting.

We demonstrate that this folklore belief is *false* by showing completely fair protocols for various non-trivial functions in the two-party setting based on standard cryptographic assumptions. We first show feasibility of obtaining complete fairness when computing any function over polynomial-size domains that does not contain an "embedded XOR"; this class of functions includes boolean AND/OR as well as Yao's "millionaires' problem". We also demonstrate feasibility for certain functions that do contain an embedded XOR, and prove a lower bound showing that any completely fair protocol for such functions must have round complexity super-logarithmic in the security parameter. Our results demonstrate that the question of completely fair secure computation without an honest majority is far from closed.

Keywords: cryptography, secure computation, fairness, distributed computing

*Dept. of Computer Science, Columbia University. Work done while at the University of Maryland.

†Dept. of Computer Science, Aarhus University. Work done while at Bar-Ilan University.

‡Dept. of Computer Science, University of Maryland. Work supported by NSF grants #0447075 and #0830464, and US-Israel Binational Science Foundation grant #2004240.

§Dept. of Computer Science, Bar-Ilan University. Work supported by US-Israel Binational Science Foundation grant #2004240.

1 Introduction

In the setting of secure computation, a set of parties wish to run some protocol for computing a function of their inputs while preserving, to the extent possible, security properties such as privacy, correctness, input independence, etc. These requirements, and more, are formalized by comparing a real-world execution of the protocol to an *ideal world* where there is a trusted entity who performs the computation on behalf of the parties. Informally, a protocol is "secure" if for any real-world adversary $\mathcal{A}$ there exists a corresponding ideal-world adversary $\mathcal{S}$ (corrupting the same parties as $\mathcal{A}$) such that the result of executing the protocol in the real world with $\mathcal{A}$ is computationally indistinguishable from the result of computing the function in the ideal world with $\mathcal{S}$.

One desirable property is *fairness* which, intuitively, means that either *everyone* receives the output, or else *no one* does. Unfortunately, it has been shown by Cleve [11] that complete fairness[1] is impossible to achieve *in general* when a majority of parties is not honest (which, in particular, includes the two-party setting); specifically, Cleve rules out completely fair coin tossing, which implies the impossibility of computing boolean XOR with complete fairness. Since Cleve's work, the accepted folklore has been that *nothing* non-trivial can be computed with complete fairness without an honest majority, and researchers have simply resigned themselves to being unable to achieve this goal. Indeed, the standard formulation of secure computation (see [18]) posits *two* ideal worlds, and two corresponding definitions of security: one that incorporates fairness and is used when a majority of the parties are assumed to be honest (we refer to the corresponding definition as "security with complete fairness"), and one that does *not* incorporate fairness and is used when an arbitrary number of parties may be corrupted (we refer to the corresponding definition as "security with abort", since the adversary in this case may abort the protocol once it receives its output).

Protocols achieving security with complete fairness when a majority of parties are honest, for arbitrary functionalities, are known (assuming a broadcast channel) [19, 5, 9, 1, 30], as are protocols achieving security with abort for any number of corrupted parties (under suitable cryptographic assumptions) [19, 18]. Since the work of Cleve, however, there has been no progress toward a better understanding of complete fairness *without* an honest majority. No further impossibility results have been shown (i.e., other than those that follow trivially from Cleve's result), nor have any completely fair protocols for any non-trivial[2] functions been constructed. In short, the question of fairness without an honest majority has been treated as closed for over two decades.

1.1 Our Results

Cleve's work shows that *certain functions* cannot be computed with complete fairness without an honest majority. The "folklore" interpretation of this result seems to have been that *nothing* (non-trivial) can be computed with complete fairness without an honest majority. Surprisingly, we show that this folklore is *false* by demonstrating that many interesting and non-trivial functions *can* be computed with complete fairness in the two-party setting. Our positive results can be based on standard cryptographic assumptions such as the existence of enhanced trapdoor permutations. (Actually, our results can be based on the minimal assumption that oblivious transfer is possible.)

[1] Various notions of *partial* fairness have also been considered; see Section 1.2 for a brief discussion.

[2] It is not hard to see that some trivial functions (e.g., the constant function) can be computed with complete fairness. Furthermore, any function *that depends on only one party's input* can be computed with complete fairness, as can any function where only one party receives output. We consider such functions "trivial" in this context.

Our first result concerns functions without an *embedded XOR*, where a function f is said to have an embedded XOR if there exist inputs x_0, x_1, y_0, y_1 such that $f(x_i, y_j) = i \oplus j$. We show:

Theorem *Let f be a two-input boolean function defined over polynomial-size domains that does not contain an embedded XOR. Then, under suitable cryptographic assumptions, there exists a protocol for securely computing f with complete fairness.*

This result is described in Section 3. The round complexity of our protocol in this case is linear in the size of the domains, hence the restriction that the domains be of polynomial size.

Examples of functions without an embedded XOR include boolean OR and AND, as well as Yao's "millionaires' problem" [31] (i.e., the greater-than function). We remark that even "simple" functions such as OR/AND are non-trivial in the context of secure two-party computation since they cannot be computed with information-theoretic privacy [10] and are in fact complete for two-party secure computation with abort [24].

Recall that Cleve's result rules out completely fair computation of boolean XOR. Given this and the fact that our first result applies only to functions without an embedded XOR, a natural conjecture is that the presence of an embedded XOR serves as a barrier to completely fair computation of a given function. Our next result shows that this conjecture is false:

Theorem *Under suitable cryptographic assumptions, there exist two-input boolean functions containing an embedded XOR that can be securely computed with complete fairness.*

This result is described in Section 4. The round complexity of the protocol here is super-logarithmic in the security parameter. We show that this is, in fact, inherent:

Theorem *Let f be a two-party function containing an embedded XOR. Then any protocol securely computing f with complete fairness (assuming one exists) requires $\omega(\log n)$ rounds.*

Our proof of the above is reminiscent of Cleve's proof [11], except that Cleve only needed to consider the adversary's ability to *bias* a coin toss, whereas we must jointly consider both *bias* and *privacy* (since, for certain functions containing an embedded XOR, it is possible for an adversary to bias the output even in the ideal world). This makes the proof considerably more complex.

1.2 Related Work

Questions of fairness have been studied since the early days of secure computation. Previous work has been dedicated to achieving various *relaxations* of fairness (i.e., "partial fairness"), both for the case of specific functionalities like coin tossing [11, 12, 28] and contract signing/exchanging secrets [6, 26, 14, 4, 13], as well as for the case of general functionalities [32, 16, 3, 20, 15, 7, 29, 17, 22]. While relevant, such work is tangential to our own: here, rather than try to achieve *partial* fairness for *all* functionalities, we are interested in obtaining *complete* fairness and then ask for which functionalities this is possible.

1.3 Open Questions

We have shown the first positive results for completely-fair secure computation of non-trivial functionalities without an honest majority. This re-opens an area of research that was previously thought to be closed, and leaves many tantalizing open directions to explore. The most pressing question left open by this work is to provide a tight characterization of which boolean functions can be computed with complete fairness in the two-party setting. More generally, the positive results

shown here apply only to deterministic, single-output,[3] boolean functions defined over polynomial-size domains. Relaxing any of these restrictions in a non-trivial way (or proving the impossibility of doing so) would be an interesting next step. Finally, what can be said with regard to complete fairness in the *multi-party* setting without honest majority? (This question is interesting both with and without the assumption of a broadcast channel.) Initial feasibility results have been shown [21], but much work remains to be done.

2 Definitions

We let n denote the security parameter. A function $\mu(\cdot)$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large n it holds that $\mu(n) < 1/p(n)$. A *distribution ensemble* $X = \{X(a,n)\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$, where $\mathcal{D}_n$ is a set that may depend on n. (Looking ahead, n will be the security parameter and $\mathcal{D}_n$ will denote the domain of the parties' inputs.) Two distribution ensembles $X = \{X(a,n)\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ and $Y = \{Y(a,n)\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ are *computationally indistinguishable*, denoted $X \stackrel{\text{c}}{\equiv} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every n and every $a \in \mathcal{D}_n$

$$\left|\Pr[D(X(a,n)) = 1] - \Pr[D(Y(a,n)) = 1]\right| \leq \mu(n).$$

The statistical difference between two distributions $X(a,n)$ and $Y(a,n)$ is defined as

$$\mathsf{SD}\left(X(a,n),\, Y(a,n)\right) = \frac{1}{2} \cdot \sum_s \left|\Pr[X(a,n) = s] - \Pr[Y(a,n) = s]\right|,$$

where the sum ranges over s in the support of either $X(a,n)$ or $Y(a,n)$. Two distribution ensembles $X = \{X(a,n)\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ and $Y = \{Y(a,n)\}_{a\in\mathcal{D}_n, n\in\mathbb{N}}$ are *statistically close*, denoted $X \stackrel{\text{s}}{\equiv} Y$, if there is a negligible function $\mu(\cdot)$ such that for every n and every $a \in \mathcal{D}_n$, it holds that $\mathsf{SD}\left(X(a,n), Y(a,n)\right) \leq \mu(n)$.

Functionalities. In the two-party setting, a *functionality* $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ is a sequence of randomized processes, where each f_n maps pairs of inputs to pairs of outputs (one for each party). We write $f_n = (f_n^1, f_n^2)$ if we wish to emphasize the two outputs of f_n, but stress that if f_n^1 and f_n^2 are randomized then the outputs of f_n^1 and f_n^2 are correlated random variables. The domain of f_n is $X_n \times Y_n$, where X_n (resp., Y_n) denotes the possible inputs of the first (resp., second) party.[4] If $|X_n|$ and $|Y_n|$ are polynomial in n, then we say that $\mathcal{F}$ is defined over *polynomial-size domains*. If each f_n is deterministic we will refer to each f_n as well as the collection $\mathcal{F}$, as a *function*.

2.1 Secure Two-Party Computation with Complete Fairness

In what follows, we define what we mean by a *secure* protocol. Our definition follows the standard definition of [18] (based on [20, 27, 2, 8]) except that *we require complete fairness even though we are in the two-party setting*. (Thus, our definition is equivalent to the one in [18] for the case of an honest majority, even though we do *not* have an honest majority.) We consider *active* (i.e., malicious) adversaries, who may deviate from the protocol arbitrarily, and static corruptions.

[3] I.e., where both parties receive the same output.

[4] The typical convention in secure computation is to let $f_n = f$ and $X_n = Y_n = \{0,1\}^*$ for all n. We will be dealing with functions defined over polynomial-size domains, which is why we introduce this notation.

Two-party computation. A two-party protocol for computing a functionality $\mathcal{F} = \{(f_n^1, f_n^2)\}$ is a protocol running in polynomial time and satisfying the following functional requirement: if party P_1 begins by holding 1^n and input $x \in X_n$, and party P_2 holds 1^n and input $y \in Y_n$, then the joint distribution of the outputs of the parties is statistically close to $(f_n^1(x,y), f_n^2(x,y))$.

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in a real protocol execution to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted party exists) can do no more harm than if it were involved in the above-described ideal computation.

We assume an adversary who corrupts one of the parties. It is also meaningful to consider an eavesdropping adversary who corrupts neither of the parties (and should learn nothing from the execution), but such an adversary is easily handled and is not very interesting in our setting.

Execution in the ideal model. The parties are P_1 and P_2, and there is an adversary $\mathcal{A}$ who has corrupted one of them. An ideal execution for the computation of $\mathcal{F} = \{f_n\}$ proceeds as follows:

Inputs: P_1 and P_2 hold the same value 1^n, and their inputs $x \in X_n$ and $y \in Y_n$, respectively; the adversary $\mathcal{A}$ receives an auxiliary input z.

Send inputs to trusted party: The honest party sends its input to the trusted party. The corrupted party controlled by $\mathcal{A}$ may send any value of its choice. Denote the pair of inputs sent to the trusted party by (x', y').

Trusted party sends outputs: If $x' \notin X_n$ the trusted party sets x' to some default input in X_n; likewise if $y' \notin Y_n$ the trusted party sets y' equal to some default input in Y_n. Then the trusted party chooses r uniformly at random and sends $f_n^1(x', y'; r)$ to party P_1 and $f_n^2(x', y'; r)$ to party P_2.

Outputs: The honest party outputs whatever it was sent by the trusted party, the corrupted party outputs nothing, and $\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view.

We let $\text{IDEAL}_{\mathcal{F},\mathcal{A}(z)}(x,y,n)$ be the random variable consisting of the output of the adversary and the output of the honest party following an execution in the ideal model as described above.

Execution in the real model. We next consider the real model in which a two-party protocol π is executed by P_1 and P_2 (and there is no trusted party). In this case, the adversary $\mathcal{A}$ gets the inputs of the corrupted party and sends all messages on behalf of this party, using an arbitrary polynomial-time strategy. The honest party follows the instructions of π.

Let $\mathcal{F}$ be as above and let π be a two-party protocol computing $\mathcal{F}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input z. We let $\text{REAL}_{\pi,\mathcal{A}(z)}(x,y,n)$ be the random variable consisting of the view of the adversary and the output of the honest party, following an execution of π where P_1 begins by holding 1^n and input x and P_2 begins by holding 1^n and y.

Security as emulation of an ideal execution in the real model. Having defined the ideal and real models, we can now define security of a protocol. Loosely speaking, the definition asserts that a secure protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated as follows:

Definition 2.1 *Protocol π is said to* securely compute $\mathcal{F}$ with complete fairness *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(x,y,n)\right\}_{(x,y)\in X_n\times Y_n,\, z\in\{0,1\}^*,\, n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x,y,n)\right\}_{(x,y)\in X_n\times Y_n,\, z\in\{0,1\}^*,\, n\in\mathbb{N}}.$$

2.2 Secure Two-Party Computation With Abort

This definition is the standard one for secure two-party computation [18] in that it allows *early abort*; i.e., the adversary may receive its own output even though the honest party does not. We again let P_1 and P_2 denote the two parties, and consider an adversary $\mathcal{A}$ who has corrupted one of them. The only change from the definition in Section 2.1 is with regard to the ideal model for computing $\mathcal{F}=\{f_n\}$, which is now defined as follows:

Inputs: As previously.

Send inputs to trusted party: As previously.

Trusted party sends output to corrupted party: If $x'\notin X_n$ the trusted party sets x' to some default input in X_n; likewise if $y'\notin Y_n$ the trusted party sets y' equal to some default input in Y_n. Then the trusted party chooses r uniformly at random, computes $z_1=f_n^1(x',y';r)$ and $z_2=f_n^2(x',y';r)$, and sends z_i to the corrupted party P_i (i.e., to the adversary $\mathcal{A}$).

Adversary decides whether to abort: After receiving its output (as described above), the adversary either sends abort of continue to the trusted party. In the former case the trusted party sends $\perp$ to the honest party P_j, and in the latter case the trusted party sends z_j to P_j.

Outputs: As previously.

We let $\text{IDEAL}^{\mathsf{abort}}_{\mathcal{F},\mathcal{A}(z)}(x,y,n)$ be the random variable consisting of the output of the adversary and the output of the honest party following an execution in the ideal model as described above.

Definition 2.2 *Protocol π is said to* securely compute $\mathcal{F}$ with abort *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\text{IDEAL}^{\mathsf{abort}}_{\mathcal{F},\mathcal{S}(z)}(x,y,n)\right\}_{(x,y)\in X_n\times Y_n,\, z\in\{0,1\}^*,\, n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x,y,n)\right\}_{(x,y)\in X_n\times Y_n,\, z\in\{0,1\}^*,\, n\in\mathbb{N}}.$$

2.3 The Hybrid Model

The hybrid model combines both the real and ideal models. Specifically, an execution of a protocol π in the $\mathcal{G}$-hybrid model, for some functionality $\mathcal{G}$, involves the parties sending normal messages to each other (as in the real model) and, in addition, having access to a trusted party computing $\mathcal{G}$. The parties communicate with this trusted party in exactly the same way as in the ideal models described above; the question of which ideal model is taken (that with or without abort) must be specified. In this paper, we always consider a hybrid model where the functionality $\mathcal{G}$ is computed according to the ideal model *with abort*. In all our protocols in the $\mathcal{G}$-hybrid model there will only be *sequential* calls to $\mathcal{G}$; i.e., there is at most a single call to $\mathcal{G}$ per round, and no other messages are sent during any round in which $\mathcal{G}$ is called.

Let $\mathcal{G}$ be a functionality and let π be a two-party protocol for computing some functionality $\mathcal{F}$, where π includes real messages between the parties as well as calls to $\mathcal{G}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input z. We let $\text{HYBRID}^{\mathcal{G}}_{\pi,\mathcal{A}(z)}(x,y,n)$ be the random variable consisting of the view of the adversary and the output of the honest party, following an execution of π (with ideal calls to $\mathcal{G}$) where P_1 begins by holding 1^n and input x and P_2 begins by holding 1^n and input y. Both security with complete fairness and security with abort can be defined via the natural modifications of Definitions 2.1 and 2.2.

The hybrid model gives a powerful tool for proving the security of protocols. Specifically, we may design a real-world protocol for securely computing some functionality $\mathcal{F}$ by first constructing a protocol for computing $\mathcal{F}$ in the $\mathcal{G}$-hybrid model. Letting π denote the protocol thus constructed (in the $\mathcal{G}$-hybrid model), we denote by π^ρ the real-world protocol in which calls to $\mathcal{G}$ are replaced by sequential execution of a real-world protocol ρ that computes $\mathcal{G}$. ("Sequential" here implies that only one execution of ρ is carried out at any time, and no other π-protocol messages are sent during execution of ρ.) The results of [8] then imply that if π securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, and ρ securely computes $\mathcal{G}$, then the composed protocol π^ρ securely computes $\mathcal{F}$ (in the real world). For completeness, we state this result formally as we will use it in this work:

Proposition 1 *Let ρ be a protocol that securely computes $\mathcal{G}$ with abort, and let π be a protocol that securely computes $\mathcal{F}$ with complete fairness in the $\mathcal{G}$-hybrid model (where $\mathcal{G}$ is computed according to the ideal world with abort). Then protocol π^ρ securely computes $\mathcal{F}$ with complete fairness.*

2.4 Information-Theoretic MACs

We briefly review the standard definition for information-theoretically secure message authentication codes (MACs). (We use such MACs for simplicity, though computationally secure MACs would also suffice.) A *message authentication code* consists of three polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$. The *key-generation algorithm* Gen takes as input the security parameter 1^n in unary and outputs a key k. The *message authentication algorithm* Mac takes as input a key k and a message $M \in \{0,1\}^{\leq n}$, and outputs a tag t; we write this as $t = \mathsf{Mac}_k(M)$. The *verification algorithm* Vrfy takes as input a key k, a message $M \in \{0,1\}^{\leq n}$, and a tag t, and outputs a bit b; we write this as $b = \mathsf{Vrfy}_k(M,t)$. We regard $b = 1$ as acceptance and $b = 0$ as rejection, and require that for all n, all k output by $\mathsf{Gen}(1^n)$, all $M \in \{0,1\}^{\leq n}$, it holds that $\mathsf{Vrfy}_k(M, \mathsf{Mac}_k(M)) = 1$.

We say $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ is a *secure m-time MAC*, where m may be a function of n, if no computationally unbounded adversary can output a valid tag on a new message after seeing valid tags on m other messages. For our purposes, we do not require security against an adversary who adaptively chooses its m messages for which to obtain a valid tag; it suffices to consider a non-adaptive definition where the m messages are fixed in advance. (Nevertheless, known constructions satisfy the stronger requirement.) Formally:

Definition 2.3 *Message authentication code* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is an* information-theoretically secure m-time MAC *if for any sequence of messages $M_1, \ldots, M_m$ and any adversary $\mathcal{A}$, the following is negligible in the security parameter n:*

$$\Pr\left[\begin{array}{l} k \leftarrow \mathsf{Gen}(1^n);\ \forall i : t_i = \mathsf{Mac}_k(M_i); \\ (M', t') \leftarrow \mathcal{A}(M_1, t_1, \ldots, M_m, t_m) \end{array} \ :\ \mathsf{Vrfy}_k(M', t') = 1 \bigwedge M' \notin \{M_1, \ldots, M_m\} \right].$$

3 Fair Computation of the Millionaires' Problem (and More)

In this section, we describe a protocol for securely computing the millionaires' problem (and related functionalities) with complete fairness. (We discuss in Section 3.2 how this generalizes, rather easily, to any function over polynomial-size domains that does not contain an embedded XOR.) Specifically, we look at functions defined by a lower-triangular matrix, as in the following table:

	y_1	y_2	y_3	y_4	y_5	y_6
x_1	0	0	0	0	0	0
x_2	1	0	0	0	0	0
x_3	1	1	0	0	0	0
x_4	1	1	1	0	0	0
x_5	1	1	1	1	0	0
x_6	1	1	1	1	1	0

Let $\mathcal{F} = \{f_{m(n)}\}_{n\in\mathbb{N}}$ denote a function of the above form, where $m = m(n)$ denotes the size of the domains of each input which we assume, for now, have the same size. (In the next section we will consider the case when they are unequal.) Let $X_m = \{x_1, \ldots, x_m\}$ denote the valid inputs for the first party and let $Y_m = \{y_1, \ldots, y_m\}$ denote the valid inputs for the second party. By suitably ordering these elements, we may write f_m as follows:

$$f_m(x_i, y_j) = \begin{cases} 1 & \text{if } i > j \\ 0 & \text{if } i \leq j \end{cases} . \tag{1}$$

Viewed in this way, f_m is exactly the millionaires' problem or, equivalently, the "greater-than" function. The remainder of this section is devoted to a proof of the following theorem:

Theorem *Let $m = \mathsf{poly}(n)$. Assuming the existence of constant-round general secure two-party computation with abort, there exists an $\Theta(m)$-round protocol that securely computes $\mathcal{F} = \{f_m\}$ with complete fairness.*

Constant-round protocols for general secure two-party computation with abort can be constructed based on enhanced trapdoor permutations or any constant-round protocol for oblivious transfer [25]. (The assumption of a constant-round protocol is needed only for the claim regarding round complexity.) The fact that our protocol requires (at least) $\Theta(m)$ rounds explains why we require $m = \mathsf{poly}(n)$. When $m = 2$, we obtain a constant-round protocol for computing boolean AND with complete fairness and, by symmetry, we also obtain a protocol for boolean OR. We remark further that our results extend to variants of f_m such as the "greater-than-or-equal-to" function, or the "greater-than" function where the sizes of the domains X and Y are unequal; see Section 3.2 for a full discussion.

3.1 The Protocol

In this section, we write f in place of f_m, and X and Y in place of X_m and Y_m.

Intuition. At a high level, our protocol works as follows. Say the input of P_1 is x_i, and the input of P_2 is y_j. Following a constant-round "pre-processing" phase, the protocol proceeds in a series of m iterations, where P_1 learns the output — namely, the value $f(x_i, y_j)$ — in iteration i, and P_2 learns the output in iteration j. (That is, in contrast to standard protocols, the iteration in which

a party learns the output *depends on the value of its own input.*) If one party (say, P_1) aborts after receiving its iteration-k message, and the second party (say, P_2) has not yet received its output, then P_2 "assumes" that P_1 learned its output in iteration k, and so computes f on its own using input x_k for P_1. (In this case, that means that P_2 would output $f(x_k, y_j)$.) We stress that a malicious P_1 may, of course, abort in any iteration it likes (and not necessarily in the iteration in which it learns its output); the foregoing is only an intuitive explanation.

The fact that this approach gives complete fairness can be intuitively understood as follows. Say P_1 is malicious and uses x_i as its effective input, and let y denote the (unknown) input of P_2. There are two possibilities: P_1 either aborts in iteration $k < i$, or iteration $k \geq i$. (If P_1 never aborts then fairness is trivially achieved.) In the first case, P_1 never learns the correct output and so fairness is achieved. In the second case, P_1 does obtain the output $f(x_i, y)$ (in iteration i) and then aborts in some iteration $k \geq i$. Here we consider two sub-cases depending on the value of P_2's input $y = y_j$:

- If $j < k$ then P_2 has already received its output in a previous iteration and fairness is achieved.
- If $j \geq k$ then P_2 has not yet received its output. Since P_1 aborts in iteration k, the protocol directs P_2 to output $f(x_k, y) = f(x_k, y_j)$. Since $j \geq k \geq i$, we have $f(x_k, y_j) = 0 = f(x_i, y_j)$ (relying on the specifics of f), and so the output of P_2 is equal to the output obtained by P_1 (and thus fairness is achieved). This is the key observation that enables us to obtain fairness for this function.

We formalize the above intuition in our proof, where we demonstrate an ideal-world simulator corresponding to the actions of any malicious P_1. Of course, we also consider the case of a malicious P_2.

Formal description of the protocol. We use a message authentication code $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$; see Definition 2.3. For convenience, we use an m-time message authentication code (MAC) with information-theoretic security, though a computationally secure MAC would also suffice.

We also rely on a sub-protocol for securely computing a randomized functionality ShareGen defined in Figure 1. In our protocol, the parties will compute ShareGen as a result of which P_1 will obtain shares $a_1^{(1)}, b_1^{(1)}, a_2^{(1)}, b_2^{(1)}, \ldots$ and P_2 will obtain shares $a_1^{(2)}, b_1^{(2)}, a_2^{(2)}, b_2^{(2)}, \ldots$. (The functionality ShareGen also provides the parties with MAC keys and tags so that if a malicious party modifies the share it sends to the other party, then the other party will almost certainly detect this. In case such manipulation is detected, it will be treated as an abort.) The parties then exchange their shares one-by-one in a sequence of m iterations. Specifically, in iteration i party P_2 will send $a_i^{(2)}$ to P_1, thus allowing P_1 to reconstruct the value $a_i \stackrel{\text{def}}{=} a_i^{(1)} \oplus a_i^{(2)}$, and then P_1 will send $b_i^{(1)}$ to P_2, thus allowing P_2 to learn the value $b_i \stackrel{\text{def}}{=} b_i^{(2)} \oplus b_i^{(1)}$.

Let π be a protocol that securely computes ShareGen *with abort*. Our protocol for computing f with complete fairness uses π and is given in Figure 2.

Theorem 3.1 *If* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is an information-theoretically secure m-time MAC, and π securely computes* ShareGen *with abort, then the protocol in Figure 2 securely computes $\{f_m\}$ with complete fairness.*

Proof: Let Π denote the protocol in Figure 2. We analyze Π in a hybrid model where there is a trusted party computing ShareGen. (Since π is only guaranteed to securely compute ShareGen with abort, the adversary in the hybrid model is allowed to abort the trusted party computing ShareGen

before output is sent to the honest party.) We prove that an execution of Π in this hybrid model is *statistically close* to an evaluation of f in the ideal model (with complete fairness), where the only difference occurs due to MAC forgeries. Applying Proposition 1 then implies the theorem.

We separately analyze corruption of P_1 and P_2, beginning with P_1:

Claim 2 *For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_1 and running Π in a hybrid model with access to an ideal functionality computing* ShareGen *(with abort), there exists a non-uniform, probabilistic polynomial-time adversary $\mathcal{S}$ corrupting P_1 and running in the ideal world with access to an ideal functionality computing f (with complete fairness), such that*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z)}(x,y,n)\right\}_{(x,y)\in X_m\times Y_m, z\in\{0,1\}^*, n\in\mathbb{N}} \stackrel{\text{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen}}_{\Pi,\mathcal{A}(z)}(x,y,n)\right\}_{(x,y)\in X_m\times Y_m, z\in\{0,1\}^*, n\in\mathbb{N}}.$$

Proof: Let P_1 be corrupted by $\mathcal{A}$. We construct a simulator $\mathcal{S}$ given black-box access to $\mathcal{A}$:

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input x, the auxiliary input z, and the security parameter n.
2. $\mathcal{S}$ receives the input x' of $\mathcal{A}$ to the computation of the functionality ShareGen.
   (a) If $x' \notin X$ (this includes the case when $x' = \perp$ since $\mathcal{A}$ aborts), then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of ShareGen, sends x_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.
   (b) Otherwise, if the input is some $x' \in X$, then $\mathcal{S}$ chooses uniformly distributed shares $a_1^{(1)},\ldots,a_m^{(1)}$ and $b_1^{(1)},\ldots,b_m^{(1)}$. In addition, it generates keys $k_a, k_b \leftarrow \mathsf{Gen}(1^n)$ and computes $t_i^b = \mathsf{Mac}_{k_b}(i\|b_i^{(1)})$ for every i. Finally, it hands $\mathcal{A}$ the strings $a_1^{(1)},\ldots,a_m^{(1)}$, $(b_1^{(1)},t_1^b),\ldots,(b_m^{(1)},t_m^b)$, and k_a as its output from the computation of ShareGen.

ShareGen

Inputs: Let the inputs to ShareGen be x_i and y_j with $1 \le i,j \le m$. (If one of the received inputs is not in the correct domain, then both parties are given output $\perp$.) The security parameter is n.

Computation:

1. Define values $a_1,\ldots,a_m$ and $b_1,\ldots,b_m$ in the following way:
   - Set $a_i = b_j = f(x_i, y_j)$.
   - For $\ell \in \{1,\ldots,m\}$, $\ell \neq i$, set $a_\ell = \text{NULL}$.
   - For $\ell \in \{1,\ldots,m\}$, $\ell \neq j$, set $b_\ell = \text{NULL}$.

   (Technically, a_i, b_i are represented as 2-bit values with, say, 00 interpreted as '0', 11 interpreted as '1', and 01 interpreted as 'NULL'.)
2. For $1 \le i \le m$, choose $(a_i^{(1)}, a_i^{(2)})$ and $(b_i^{(1)}, b_i^{(2)})$ as random secret sharings of a_i and b_i, respectively. (I.e., $a_i^{(1)}$ is random and $a_i^{(1)} \oplus a_i^{(2)} = a_i$.)
3. Compute $k_a, k_b \leftarrow \mathsf{Gen}(1^n)$. For $1 \le i \le m$, let $t_i^a = \mathsf{Mac}_{k_a}(i\|a_i^{(2)})$ and $t_i^b = \mathsf{Mac}_{k_b}(i\|b_i^{(1)})$.

Output:

1. P_1 receives the values $a_1^{(1)},\ldots,a_m^{(1)}$ and $(b_1^{(1)},t_1^b),\ldots,(b_m^{(1)},t_m^b)$, and the MAC-key k_a.
2. P_2 receives the values $(a_1^{(2)},t_1^a),\ldots,(a_m^{(2)},t_m^a)$ and $b_1^{(2)},\ldots,b_m^{(2)}$, and the MAC-key k_b.

Figure 1: Functionality ShareGen.

Protocol 1

Inputs: Party P_1 has input x and party P_2 has input y. The security parameter is n.

The protocol:

1. **Preliminary phase:**
   (a) Parties P_1 and P_2 run protocol π for computing $\mathsf{ShareGen}$, using their respective inputs x, y, and security parameter n.
   (b) If P_1 receives $\perp$ from the above computation (because P_2 aborts the computation or uses an invalid input in π) it outputs $f(x, y_1)$ and halts. Likewise, if P_2 receives $\perp$, it outputs $f(x_1, y)$ and halts. Otherwise, the parties proceed.
   (c) Denote the output of P_1 from π by $a_1^{(1)}, \ldots, a_m^{(1)}$, $(b_1^{(1)}, t_1^b), \ldots, (b_m^{(1)}, t_m^b)$, and k_a.
   (d) Denote the output of P_2 from π by $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$, $b_1^{(2)}, \ldots, b_m^{(2)}$, and k_b.
2. **For $i = 1, \ldots, m$ do:**
   P_2 sends the next share to P_1:
   (a) P_2 sends $(a_i^{(2)}, t_i^a)$ to P_1.
   (b) P_1 receives $(a_i^{(2)}, t_i^a)$ from P_2. If $\mathsf{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 0$ (or if P_1 received an invalid message, or no message), then P_1 halts. If P_1 has already determined its output in some earlier iteration, then it outputs that value. Otherwise, it outputs $f(x, y_{i-1})$ (if $i = 1$, then P_1 outputs $f(x, y_1)$).
   (c) If $\mathsf{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 1$ and $a_i^{(1)} \oplus a_i^{(2)} \neq \text{NULL}$ (i.e., $x = x_i$), then P_1 sets its output to be $a_i^{(1)} \oplus a_i^{(2)}$ (and continues running the protocol).
   P_1 sends the next share to P_2:
   (a) P_1 sends $(b_i^{(1)}, t_i^b)$ to P_2.
   (b) P_2 receives $(b_i^{(1)}, t_i^b)$ from P_1. If $\mathsf{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 0$ (or if P_2 received an invalid message, or no message), then P_2 halts. If P_2 has already determined its output in some earlier iteration, then it outputs that value. Otherwise, it outputs $f(x_i, y)$.
   (c) If $\mathsf{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 1$ and $b_i^{(1)} \oplus b_i^{(2)} \neq \text{NULL}$ (i.e., $y = y_i$), then P_2 sets its output to be $b_i^{(1)} \oplus b_i^{(2)}$ (and continues running the protocol).

Figure 2: Protocol for computing f.

3. If $\mathcal{A}$ sends abort to the trusted party computing $\mathsf{ShareGen}$ (signalling that P_2 should receive $\perp$ as output from $\mathsf{ShareGen}$), then $\mathcal{S}$ sends x_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise (i.e., if $\mathcal{A}$ sends $\mathsf{continue}$), $\mathcal{S}$ proceeds as below.
4. Let i (with $1 \leq i \leq m$) be the index such that $x' = x_i$ (such an i exists since $x' \in X$).
5. To simulate iteration j, for $j < i$, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ chooses $a_j^{(2)}$ such that $a_j^{(1)} \oplus a_j^{(2)} = \text{NULL}$, and computes the tag $t_j^a = \mathsf{Mac}_{k_a}(j \| a_j^{(2)})$. Then $\mathcal{S}$ gives $\mathcal{A}$ the message $(a_j^{(2)}, t_j^a)$.
   (b) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{b}_j^{(1)}, \hat{t}_j^b)$ in the jth iteration:
      i. If $\mathsf{Vrfy}_{k_b}(j \| \hat{b}_j^{(1)}, \hat{t}_j^b) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends x_j to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.

ii. If $\mathsf{Vrfy}_{k_b}(j\|\hat{b}_j^{(1)}, \hat{t}_j^b) = 1$, then $\mathcal{S}$ proceeds to the next iteration.

6. To simulate iteration i, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ sends x_i to the trusted party computing f, and receives back the output $z = f(x_i, y)$.
   (b) $\mathcal{S}$ chooses $a_i^{(2)}$ such that $a_i^{(1)} \oplus a_i^{(2)} = z$, and computes the tag $t_i^a = \mathsf{Mac}_{k_a}(i\|a_i^{(2)})$. Then $\mathcal{S}$ gives $\mathcal{A}$ the message $(a_i^{(2)}, t_i^a)$.
   (c) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{b}_i^{(1)}, \hat{t}_i^b)$. If $\mathsf{Vrfy}_{k_b}(i\|\hat{b}_i^{(1)}, \hat{t}_i^b) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathsf{Vrfy}_{k_b}(j\|\hat{b}_j^{(1)}, \hat{t}_j^b) = 1$, then $\mathcal{S}$ proceeds to the next iteration.

7. To simulate iteration j, for $i < j \leq m$, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ chooses $a_j^{(2)}$ such that $a_j^{(1)} \oplus a_j^{(2)} = \text{NULL}$, and computes the tag $t_j^a = \mathsf{Mac}_{k_a}(j\|a_j^{(2)})$. Then $\mathcal{S}$ gives $\mathcal{A}$ the message $(a_j^{(2)}, t_j^a)$.
   (b) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{b}_j^{(1)}, \hat{t}_j^b)$. If $\mathsf{Vrfy}_{k_b}(j\|\hat{b}_j^{(1)}, \hat{t}_j^b) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathsf{Vrfy}_{k_b}(j\|\hat{b}_j^{(1)}, \hat{t}_j^b) = 1$, then $\mathcal{S}$ proceeds to the next iteration.

8. If $\mathcal{S}$ has not halted yet, at this point it halts and outputs whatever $\mathcal{A}$ outputs.

We analyze the simulator $\mathcal{S}$ described above. In what follows we assume that if $\mathsf{Vrfy}_{k_b}(j\|\hat{b}_j^{(1)}, \hat{t}_j^b) = 1$ then $\hat{b}_j^{(1)} = b_j^{(1)}$ (meaning that $\mathcal{A}$ sent the same share that it received). Under this assumption, we show that the distribution generated by $\mathcal{S}$ is *identical* to the distribution in a hybrid execution between $\mathcal{A}$ and an honest P_2. Since this assumption holds with all but negligible probability (by security of the information-theoretic MAC), this proves statistical closeness as stated in the claim.

Let y denote the input of P_2. It is clear that the view of $\mathcal{A}$ in an execution with $\mathcal{S}$ is identical to the view of $\mathcal{A}$ in a hybrid execution with P_2; the only difference is that the initial shares given to $\mathcal{A}$ are generated by $\mathcal{S}$ without knowledge of $z = f(x', y)$, but since these shares are uniformly distributed the view of $\mathcal{A}$ is unaffected. Therefore, what is left to demonstrate is that the joint distribution of $\mathcal{A}$'s view and P_2's output is identical in the hybrid world and the ideal world. We show this now by separately considering three different cases:

1. *Case 1: $\mathcal{S}$ sends x_1 to the trusted party because $x' \notin X$, or because $\mathcal{A}$ aborted the computation of* ShareGen*:* In the hybrid world, P_2 would have received $\perp$ from ShareGen, and would have then output $f(x_1, y)$ as instructed by protocol Π. This is exactly what P_2 outputs in the ideal execution with $\mathcal{S}$ because, in this case, $\mathcal{S}$ sends x_1 to the trusted party computing f.

   If Case 1 does not occur, let x_i be defined as in the description of the simulator.

2. *Case 2: $\mathcal{S}$ sends x_j to the trusted party, for some $j < i$:* This case occurs when $\mathcal{A}$ aborts the protocol in some iteration $j < i$ (either by refusing to send a message, sending an invalid message, or sending an incorrect share). There are two sub-cases depending on the value of P_2's input y. Let ℓ be the index such that $y = y_\ell$. Then:

   (a) If $\ell \geq j$ then, in the hybrid world, P_2 would not yet have determined its output (since it only determines its output once it receives a valid message from P_1 in iteration ℓ). Thus, as instructed by the protocol, P_2 would output $f(x_j, y)$. This is exactly what P_2 outputs in the ideal world, because $\mathcal{S}$ sends x_j to the trusted party in this case.

(b) If $\ell < j$ then, in the hybrid world, P_2 would have already determined its output $f(x', y) = f(x_i, y_\ell)$ in the ℓth iteration. In the ideal world, P_2 will output $f(x_j, y_\ell)$ since $\mathcal{S}$ sends x_j to the trusted party. Since $j < i$ we have $\ell < j < i$ and so $f(x_j, y_\ell) = f(x_i, y_\ell) = 1$. Thus, P_2's output $f(x_i, y)$ in the hybrid world is equal to its output $f(x_j, y)$ in the ideal execution with $\mathcal{S}$.

3. *Case 3: $\mathcal{S}$ sends x_i to the trusted party:* Here, P_2 outputs $f(x_i, y)$ in the ideal execution. We show that this is identical to what P_2 would have output in the hybrid world. There are two sub-cases depending on P_2's input y. Let ℓ be the index such that $y = y_\ell$. Then:

   (a) If $\ell < i$, then P_2 would have already determined its output $f(x', y) = f(x_i, y)$ in the ℓth iteration. (The fact that we are in Case 3 means that $\mathcal{A}$ could not have sent an incorrect share prior to iteration i.)

   (b) If $\ell \geq i$, then P_2 would not yet have determined its output. There are two sub-cases:

      i. $\mathcal{A}$ sends correct shares in iterations $j = i, \ldots, \ell$ (inclusive). Then P_2 would determine its output as $b_\ell^{(1)} \oplus b_\ell^{(2)} = f(x', y) = f(x_i, y)$, exactly as in the ideal world.

      ii. $\mathcal{A}$ sends an incorrect share in iteration ζ, where $i \leq \zeta \leq \ell$. In this case, by the specification of the protocol, party P_2 would output $f(x_\zeta, y) = f(x_\zeta, y_\ell)$. However, since $i \leq \zeta \leq \ell$ we have $f(x_\zeta, y_\ell) = 0 = f(x_i, y_\ell)$. Thus, P_2 outputs the same value in the hybrid and ideal executions.

This concludes the proof of the claim. ■

The following claim, dealing with a corrupted P_2, completes the proof of the theorem:

Claim 3 *For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_2 and running Π in a hybrid model with access to an ideal functionality computing* ShareGen *(with abort), there exists a non-uniform, probabilistic polynomial-time adversary $\mathcal{S}$ corrupting P_2 and running in the ideal world with access to an ideal functionality computing f (with complete fairness), such that*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z)}(x,y,n)\right\}_{(x,y)\in X_m\times Y_m, z\in\{0,1\}^*, n\in\mathbb{N}} \stackrel{\text{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen}}_{\Pi,\mathcal{A}(z)}(x,y,n)\right\}_{(x,y)\in X_m\times Y_m, z\in\{0,1\}^*, n\in\mathbb{N}}.$$

Proof: Say P_2 is corrupted by $\mathcal{A}$. We construct a simulator $\mathcal{S}$ given black-box access to $\mathcal{A}$:

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input y, the auxiliary input z, and the security parameter n.

2. $\mathcal{S}$ receives the input y' of $\mathcal{A}$ to the computation of the functionality ShareGen.

   (a) If $y' \notin Y$ (this includes the case when $y' = \perp$ since $\mathcal{A}$ aborts), then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of ShareGen, sends y_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.

   (b) Otherwise, if the input is some $y' \in Y$, then $\mathcal{S}$ chooses uniformly distributed shares $a_1^{(2)}, \ldots, a_m^{(2)}$ and $b_1^{(2)}, \ldots, b_m^{(2)}$. In addition, it generates keys $k_a, k_b \leftarrow \mathsf{Gen}(1^n)$ and computes $t_i^a = \mathsf{Mac}_{k_a}(i \| a_i^{(2)})$ for every i. Finally, it hands $\mathcal{A}$ the strings $b_1^{(2)}, \ldots, b_m^{(2)}$, $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$, and k_b as its output from the computation of ShareGen.

3. If $\mathcal{A}$ sends abort to the trusted party computing ShareGen, then $\mathcal{S}$ sends y_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise (i.e., if $\mathcal{A}$ sends continue), $\mathcal{S}$ proceeds as below.

4. Let i (with $1 \leq i \leq m$) be the index such that $y' = y_i$ (such an i exists since $y' \in Y$).

5. To simulate iteration j, for $j < i$, simulator $\mathcal{S}$ works as follows:

   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_j^{(2)}, \hat{t}_j^a)$ in the jth iteration:

      i. If $\mathsf{Vrfy}_{k_a}(j \| \hat{a}_j^{(2)}, \hat{t}_j^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends y_{j-1} to the trusted party computing f (if $j = 1$, then $\mathcal{S}$ sends y_1), outputs whatever $\mathcal{A}$ outputs, and halts.

      ii. If $\mathsf{Vrfy}_{k_a}(j \| \hat{a}_j^{(2)}, \hat{t}_j^a) = 1$, then $\mathcal{S}$ proceeds.

   (b) Choose $b_j^{(1)}$ such that $b_j^{(1)} \oplus b_j^{(2)} = \text{NULL}$, and compute the tag $t_j^b = \mathsf{Mac}_{k_b}(j \| b_j^{(1)})$. Then give to $\mathcal{A}$ the message $(b_j^{(1)}, t_j^b)$.

6. To simulate iteration i, simulator $\mathcal{S}$ works as follows:

   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_i^{(2)}, \hat{t}_i^a)$.

      i. If $\mathsf{Vrfy}_{k_a}(i \| \hat{a}_i^{(2)}, \hat{t}_i^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends y_{i-1} to the trusted party computing f (if $i = 1$ then $\mathcal{S}$ sends y_1), outputs whatever $\mathcal{A}$ outputs, and halts.

      ii. If $\mathsf{Vrfy}_{k_a}(i \| \hat{a}_i^{(2)}, \hat{t}_i^a) = 1$, then $\mathcal{S}$ sends y_i to the trusted party computing f, receives the output $z = f(x, y_i)$, and proceeds.

   (b) Choose $b_i^{(1)}$ such that $b_i^{(1)} \oplus b_i^{(2)} = z$, and compute the tag $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$. Then give to $\mathcal{A}$ the message $(b_i^{(1)}, t_i^b)$.

7. To simulate iteration j, for $i < j \leq m$, simulator $\mathcal{S}$ works as follows:

   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_j^{(2)}, \hat{t}_j^a)$. If $\mathsf{Vrfy}_{k_a}(j \| \hat{a}_j^{(2)}, \hat{t}_j^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathsf{Vrfy}_{k_a}(j \| \hat{a}_j^{(2)}, \hat{t}_j^a) = 1$, then $\mathcal{S}$ proceeds.

   (b) Choose $b_j^{(1)}$ such that $b_j^{(1)} \oplus b_j^{(2)} = \text{NULL}$, and compute the tag $t_j^b = \mathsf{Mac}_{k_b}(j \| b_j^{(1)})$. Then give to $\mathcal{A}$ the message $(b_j^{(1)}, t_j^b)$.

8. If $\mathcal{S}$ has not halted yet, at this point it halts and outputs whatever $\mathcal{A}$ outputs.

As in the proof of the previous claim, we assume in what follows that if $\mathsf{Vrfy}_{k_a}(j \| \hat{a}_j^{(2)}, \hat{t}_j^a) = 1$ then $\hat{a}_j^{(2)} = a_j^{(2)}$ (meaning that $\mathcal{A}$ sent P_1 the same share that it received). Under this assumption, we show that the distribution generated by $\mathcal{S}$ is *identical* to the distribution in a hybrid execution between $\mathcal{A}$ and an honest P_1. Since this assumptions holds with all but negligible probability (by security of the MAC), this proves statistical closeness as stated in the claim.

Let x denote the input of P_1. Again, it is clear that the view of $\mathcal{A}$ in an execution with $\mathcal{S}$ is identical to the view of $\mathcal{A}$ in a hybrid execution with P_1. What is left to demonstrate is that the joint distribution of $\mathcal{A}$'s view and P_1's output is identical. We show this by considering four different cases:

1. *Case 1: $\mathcal{S}$ sends y_1 to the trusted party because $y' \notin Y$, or because $\mathcal{A}$ aborted the computation of* ShareGen*:* In such a case, the protocol instructs P_1 to output $f(x, y_1)$, exactly what P_1 outputs in the ideal world.

2. *Case 2: $\mathcal{S}$ sends y_1 to the trusted party because $\mathcal{A}$ sends an incorrect share in the first iteration:* In this case, the simulator sends y_1 to the trusted party computing f, and so the output of P_1 in the ideal world is $f(x, y_1)$. In the hybrid world, P_1 will also output $f(x, y_1)$ as instructed by the protocol.

   If Cases 1 and 2 do not occur, let y_i be defined as in the description of the simulator.

3. *Case 3: $\mathcal{S}$ sends y_{j-1} to the trusted party, for some $1 \leq j-1 < i$, because $\mathcal{A}$ sends an incorrect share in the jth iteration:* The output of P_1 in the ideal world is $f(x, y_{j-1})$. There are two sub-cases here, depending on the value of P_1's input x. Let ℓ be the index such that $x = x_\ell$. Then:

   (a) If $\ell < j$ then, in the hybrid world, P_1 would have already determined its output $f(x, y') = f(x_\ell, y_i)$. But since $\ell \leq j-1 < i$ we have $f(x_\ell, y_i) = 0 = f(x_\ell, y_{j-1})$, and so P_1's output is identical in both the hybrid and ideal worlds.

   (b) If $\ell \geq j$ then, in the hybrid world, P_1 would not yet have determined its output. Therefore, as instructed by the protocol, P_1 will output $f(x, y_{j-1})$ in the hybrid world, which is exactly what it outputs in the ideal execution with $\mathcal{S}$.

4. *Case 4: $\mathcal{S}$ sends y_i to the trusted party:* This case occurs when $\mathcal{A}$ sends correct shares up through and including iteration i. The output of P_1 in the ideal world is $f(x, y_i)$. There are again two sub-cases here, depending on the value of P_1's input x. Let ℓ be the index such that $x = x_\ell$. Then:

   (a) If $\ell \leq i$, then P_1 would have already determined its output $f(x, y') = f(x_\ell, y_i)$ in the ℓth iteration. This matches what P_1 outputs in the ideal execution with $\mathcal{S}$.

   (b) If $\ell > i$, then P_1 would not have yet have determined its output. There are two sub-cases:

      i. $\mathcal{A}$ sends correct shares in iterations $j = i+1, \ldots, \ell$ (inclusive). This implies that, in the hybrid world, P_1 would determine its output to be $a_\ell^{(1)} \oplus a_\ell^{(2)} = f(x, y') = f(x, y_i)$, exactly as in the ideal execution.

      ii. $\mathcal{A}$ sends an incorrect share in iteration ζ, where $i < \zeta \leq \ell$. In this case, by the specification of the protocol, party P_1 would output $f(x, y_{\zeta-1}) = f(x_\ell, y_{\zeta-1})$ in the hybrid world. But since $i \leq \zeta - 1 < \ell$ we have $f(x_\ell, y_{\zeta-1}) = 1 = f(x_\ell, y_i)$, and so P_1's output is identical in both the hybrid and ideal worlds.

This completes the proof of the claim. ■

The preceding claims along with Proposition 1 imply the theorem. ■

3.2 Handling any Function without an Embedded XOR

The protocol in the previous section, as described, applies only to the "greater-than" function on two equal-size domains X and Y. For the case of the greater-than function with $|X| = |Y| + 1$, the same protocol (with one small change) still works. Specifically, let $X = \{x_1, \ldots, x_{m+1}\}$ and

$Y = \{y_1, \ldots, y_m\}$ with f still defined as in Equation (1). Modify the protocol of Figure 2 so that if the end of the protocol is reached and P_1 holds input x_{m+1}, then P_1 outputs 1. Then the same proof as in the previous section shows that this protocol is also completely fair. (Adapting Claim 3 is immediate: the view of a malicious P_2 is simulated in the same way; as for the output of the honest P_1, the case when P_1 holds input $x = x_i$ with $i < m + 1$ is analyzed identically, and when $x = x_{m+1}$ then P_1 outputs 1 no matter what in both the hybrid and ideal worlds. Adapting Claim 2 requires only a little thought to verify that the analysis in Case 2(b) still holds when $i = m + 1$.)

We now show that the protocol can be applied to any function defined over polynomial-size domains that does not contain an embedded XOR. This is because any such function can be "converted" to the greater-than function as we now describe.

Let $g : X \times Y \to \{0, 1\}$ be a function that does not contain an embedded XOR, and let $X = \{x_1, \ldots, x_{m_1}\}$ and $Y = \{y_1, \ldots, y_{m_2}\}$. It will be convenient to picture g as an $m_1 \times m_2$ matrix, where entry (i, j) contains the value $g(x_i, y_j)$. Similarly, we can view any matrix as a function.

We will apply a sequence of transformations to g that will result in a "functionally equivalent" function g'', where by "functionally equivalent" we mean that g can be computed with perfect security (and complete fairness) in the g''-hybrid model (where g'' is computed by a trusted party with complete fairness). It follows that a secure and completely fair protocol for computing g'' yields a secure and completely fair protocol for computing g. The transformations are as follows:

1. First, remove any duplicate rows or columns in g. (E.g., if there exist i and i' such that $g(x_i, y) = g(x_{i'}, y)$ for all $y \in Y$, then remove either row i or row i'.) Denote the resulting function by g', and say that g' (viewed as a matrix) has dimension $m'_1 \times m'_2$. It is clear that g' is functionally equivalent to g.

2. We observe that no two rows (resp., columns) of g' have the same Hamming weight. To see this, notice that two non-identical rows (resp., columns) with the same Hamming weight would imply the existence of an embedded XOR in g', and hence an embedded XOR in g.

   Since the maximum Hamming weight of any row is m'_2, this implies that $m'_1 \leq m'_2 + 1$. Applying the same argument to the columns shows that $m'_2 \leq m'_1 + 1$, and so the number of rows is within 1 of the number of columns. Assume $m'_1 \geq m'_2$; if not, we may simply take the transpose of g' (which just has the effect of swapping the roles of the parties).

3. Order the rows of g' in increasing order according to their Hamming weight. Order the columns in the same way. Once again this results in a function g'' that is functionally equivalent to g' (and hence to g).

All the above transformations are efficiently computable since we are assuming that the initial domains X and Y are of polynomial size.

Given g'' resulting from the above transformations, there are now three possibilities (recall we assume that the number of rows is at least the number of columns):

1. *Case 1:* $m'_1 = m'_2 + 1$. In this case the first row of g'' is an all-0 row and the last row is an all-1 row, and we exactly have an instance of the greater-than function with $m'_1 = m'_2 + 1$.

2. *Case 2:* $m'_1 = m'_2$ *and the first row of* g'' *is an all-0 row.* Then we again have an instance of the greater-than function, except now with equal-size domains.

3. *Case 3: $m'_1 = m'_2$ and the first row of g'' is not an all-0 row.* In this case, the last row of g'' must be an all-1 row. Taking the complement of every bit in the matrix (and then re-ordering the rows and columns accordingly) gives a function that is still functionally equivalent to g and is exactly an instance of the greater-than function on equal-size domains.

We have thus proved:

Theorem 3.2 *Let f be a two-input function defined over polynomial-size domains that does not contain an embedded XOR. Then, assuming the existence of general secure two-party computation with abort, there exists a protocol for securely computing f with complete fairness.*

The assumption in the theorem is minimal, since the existence of even a secure-with-abort protocol for computing boolean OR implies the existence of oblivious transfer [24], which in turn suffices for constructing a secure-with-abort protocol for any polynomial-time functionality [23].

4 Fair Computation of Functions with an Embedded XOR

Recall that Cleve's result showing impossibility of completely fair coin tossing implies the impossibility of completely fair computation of boolean XOR. (More generally, it implies the impossibility of completely fair computation of any function f that enables coin tossing: i.e., any f such that a completely fair implementation of f suffices for coin tossing.) Given this, along with the fact that our result in the previous section applies only to functions that do not contain an embedded XOR, it is tempting to conjecture that no function containing an embedded XOR can be computed with complete fairness. In this section, we show that this is not the case and that there exist functions with an embedded XOR that *can* be computed with complete fairness. Interestingly, however, such functions appear to be "more difficult" to compute with complete fairness; specifically, we refer the reader to Section 5 where we prove a lower bound of $\omega(\log n)$ on the round complexity of any protocol for completely fair computation of any function having an embedded XOR. (Note that, in general, this bound is incomparable to the result of the previous section, where the round complexity was linear in the domain size.)

It will be instructive to see why Cleve's impossibility result does not immediately rule out complete fairness for all functions containing an embedded XOR. Consider the following function f (which is the example for which we will later prove feasibility):

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

If the parties could be forced to choose their inputs from $\{x_1, x_2\}$ and $\{y_1, y_2\}$, respectively, then it would be easy to generate a fair coin toss from any secure computation of f (with complete fairness) by simply instructing both parties to choose their inputs uniformly from the stated domains. (This results in a fair coin toss since the output is uniform at long as either party chooses their input at random.) Unfortunately, a protocol for securely computing f does *not* restrict the first party to choosing its input in $\{x_1, x_2\}$, and cannot prevent that party from choosing input x_3 and thus biasing the result toward 1 with certainty. (Naive solutions such as requiring the first party to provide a zero-knowledge proof that it chose its input in $\{x_1, x_2\}$ do not work either, since we still

need a way for, e.g., the second party to decide on their output in case the zero-knowledge proof of the first party fails.) Of course, this only shows that Cleve's impossibility result does not apply but does not prove that a completely fair protocol for computing f exists.

4.1 The Protocol

Preliminaries. In this section we present a generic protocol for computing a boolean function $\mathcal{F} = \{f_n : X_n \times Y_n \to \{0,1\}\}$. (For convenience, we write X and Y and drop the explicit dependence on n in what follows.) The protocol is parameterized by a function $\alpha = \alpha(n)$, and the number of rounds is set to $m = \omega(\alpha^{-1} \log n)$ in order for correctness to hold with all but negligible probability. (We thus must have α noticeable to ensure that the number of rounds is polynomial in n.)

We do *not* claim that the protocol is completely fair for arbitrary functions $\mathcal{F}$ and arbitrary settings of α. Rather, we claim that for *some* functions $\mathcal{F}$ there exists a corresponding α for which the protocol is completely fair. In Section 4.2, we prove this for one specific function that contains an embedded XOR. In Appendix A we generalize the proof and show that the protocol can be used for completely fair computation of other functions as well.

Overview and intuition. As in the protocol of the previous section, the parties begin by running a "preliminary" phase during which values $a_1, b_1, \ldots, a_m, b_m$ are generated based on the parties' respective inputs x and y, and shares of the $\{a_i, b_i\}$ are distributed to each of the parties. (As before, this phase will be carried out using a standard protocol for secure two-party computation, where one party can abort the execution and prevent the other party from receiving any output.) As in the previous protocol, following the preliminary phase the parties exchange their shares one-by-one in a sequence of m iterations, with P_1 reconstructing a_i and P_2 reconstructing b_i in iteration i. At the end of the protocol, P_1 outputs a_m and P_2 outputs b_m. If a party (say, P_1) ever aborts, then the other party (P_2 in this case) outputs the last value it successfully reconstructed; i.e., if P_1 aborts before sending its iteration-i message, P_2 outputs b_{i-1}. (This assumes $i > 1$. See the formal description of the protocol for further details.)

In contrast to our earlier protocol, however, the values $a_1, b_1, \ldots, a_m, b_m$ are now generated *probabilistically* in the following way: first, a value $i^* \in \{1, \ldots, m\}$ is chosen according to a geometric distribution with parameter α (see below), in a way such that neither party learns the value of i^*. For $i < i^*$, the value a_i (resp., b_i) is chosen in a manner that is *independent* of P_2's (resp., P_1's) input; specifically, we set $a_i = f(x, \hat{y})$ for randomly chosen $\hat{y} \in Y$ (and analogously for b_i). For all $i \geq i^*$, the values a_i and b_i are set equal to $f(x,y)$. Note that if $m = \omega(\alpha^{-1} \log n)$, we have $a_m = b_m = f(x,y)$ with all but negligible probability and so correctness holds. (The protocol could also be modified so that $a_m = b_m = f(x,y)$ with probability 1, thus giving perfect correctness. But the analysis is easier without this modification.)

Fairness is more difficult to see and, of course, cannot hold for all functions f since some functions cannot be computed fairly. But as intuition for why the protocol achieves fairness for certain functions, we observe that: (1) if a malicious party (say, P_1) aborts in some iteration $i < i^*$, then P_1 has not yet obtained any information about P_2's input and so fairness is trivially achieved. On the other hand, (2) if P_1 aborts in some iteration $i > i^*$ then *both* P_1 and P_2 have received the correct output $f(x,y)$ and fairness is obtained. The worst case, then, occurs when P_1 aborts exactly in iteration i^*, as P_1 has then learned the correct value of $f(x,y)$ while P_2 has not. However, P_1 cannot identify iteration i^* with certainty, even if it knows the other party's input y! This is because P_1 can *randomly* receive the correct output value even in rounds $i < i^*$. Although the

$\mathsf{ShareGen}'$

Inputs: Let the inputs to $\mathsf{ShareGen}'$ be $x \in X$ and $y \in Y$. (If one of the received inputs is not in the correct domain, then both parties are given output $\perp$.) The security parameter is n.

Computation:

1. Define values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$ in the following way:
   - Choose i^* according to a geometric distribution with parameter α (see text).
   - For $i = 1$ to $i^* - 1$ do:
     - Choose $\hat{y} \leftarrow Y$ and set $a_i = f(x, \hat{y})$.
     - Choose $\hat{x} \leftarrow X$ and set $b_i = f(\hat{x}, y)$.
   - For $i = i^*$ to m, set $a_i = b_i = f(x, y)$.
2. For $1 \le i \le m$, choose $(a_i^{(1)}, a_i^{(2)})$ and $(b_i^{(1)}, b_i^{(2)})$ as random secret sharings of a_i and b_i, respectively. (E.g., $a_i^{(1)}$ is random and $a_i^{(1)} \oplus a_i^{(2)} = a_i$.)
3. Compute $k_a, k_b \leftarrow \mathsf{Gen}(1^n)$. For $1 \le i \le m$, let $t_i^a = \mathsf{Mac}_{k_a}(i \| a_i^{(2)})$ and $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$.

Output:

1. Send to P_1 the values $a_1^{(1)}, \ldots, a_m^{(1)}$ and $(b_1^{(1)}, t_1^b), \ldots, (b_m^{(1)}, t_m^b)$, and the MAC-key k_a.
2. Send to P_2 the values $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$ and $b_1^{(2)}, \ldots, b_m^{(2)}$, and the MAC-key k_b.

Figure 3: Functionality $\mathsf{ShareGen}'$, parameterized by a value α.

adversary may happen to guess i^* correctly, the fact that it can never be sure whether its guess is correct is what allows us to prove fairness. (Recall, we define fairness via indistinguishability from an ideal world in which fairness is guaranteed. This intuition provides a way of understanding what is going on, but the formal proof does not exactly follow this intuition.)

Formal description of the protocol. The protocol is parameterized by a value $\alpha = \alpha(n)$ which is assumed to be noticeable. Let $m = \omega(\alpha^{-1} \log n)$. As in the previous section, we use an m-time MAC with information-theoretic security. We also rely on a sub-protocol π computing a functionality $\mathsf{ShareGen}'$ that generates shares (and associated MAC tags) for the parties; see Figure 3. (As before, π securely computes $\mathsf{ShareGen}'$ *with abort*.) We continue to let $a_1^{(1)}, b_1^{(1)}, a_2^{(1)}, b_2^{(1)}, \ldots$ denote the shares obtained by P_1, and let $a_1^{(2)}, b_1^{(2)}, a_2^{(2)}, b_2^{(2)}, \ldots$ denote the shares obtained by P_2.

Functionality $\mathsf{ShareGen}'$ generates a value i^* according to a *geometric distribution* with parameter α. This is the probability distribution on $\mathbb{N} = \{1, 2, \ldots\}$ given by repeating a Bernoulli trial (with parameter α) until the first success. In other words, i^* is determined by tossing a biased coin (that is heads with probability α) until the first head appears, and letting i^* be the number of tosses performed. Note that neither party learns the value of i^*. We use a geometric distribution for i^* because it has the following useful property: for any i, the probability that $i^* = i$ — conditioned on the event that $i^* \ge i$ — is independent of i (namely, $\Pr[i^* = i \mid i^* \ge i] = \alpha$). We remark that, as far as $\mathsf{ShareGen}'$ is concerned, if $i^* > m$ then the exact value of i^* is unimportant, and so $\mathsf{ShareGen}'$ can be implemented in strict (rather than expected) polynomial time. In any case, our choice of m ensures that $i^* \le m$ with all but negligible probability.

Our second protocol calls $\mathsf{ShareGen}'$ as a subroutine and then has the parties exchange their shares as in our first protocol. As discussed above, aborts are handled differently here in that a party also outputs the last value it reconstructed if the other party aborts. A formal description

of the protocol is given in Figure 4.

Protocol 2

Inputs: Party P_1 has input x and party P_2 has input y. The security parameter is n.

The protocol:

1. **Preliminary phase:**
   (a) P_1 chooses $\hat{y} \in Y$ uniformly at random, and sets $a_0 = f(x, \hat{y})$. Similarly, P_2 chooses $\hat{x} \in X$ uniformly at random, and sets $b_0 = f(\hat{x}, y)$.
   (b) Parties P_1 and P_2 run protocol π for computing $\mathsf{ShareGen}'$, using their respective inputs x and y, and security parameter n.
   (c) If P_1 receives $\perp$ from the above computation, it outputs a_0 and halts. Likewise, if P_2 receives $\perp$ then it outputs b_0 and halts. Otherwise, the parties proceed to the next step.
   (d) Denote the output of P_1 from π by $a_1^{(1)}, \ldots, a_m^{(1)}$, $(b_1^{(1)}, t_1^b), \ldots, (b_m^{(1)}, t_m^b)$, and k_a.
   (e) Denote the output of P_2 from π by $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$, $b_1^{(2)}, \ldots, b_m^{(2)}$, and k_b.
2. **For $i = 1, \ldots, m$ do:**
   P_2 sends the next share to P_1:
   (a) P_2 sends $(a_i^{(2)}, t_i^a)$ to P_1.
   (b) P_1 receives $(a_i^{(2)}, t_i^a)$ from P_2. If $\mathsf{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 0$ (or if P_1 received an invalid message, or no message), then P_1 outputs a_{i-1} and halts.
   (c) If $\mathsf{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 1$, then P_1 sets $a_i = a_i^{(1)} \oplus a_i^{(2)}$ (and continues running the protocol).
   P_1 sends the next share to P_2:
   (a) P_1 sends $(b_i^{(1)}, t_i^b)$ to P_2.
   (b) P_2 receives $(b_i^{(1)}, t_i^b)$ from P_1. If $\mathsf{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 0$ (or if P_2 received an invalid message, or no message), then P_2 outputs b_{i-1} and halts.
   (c) If $\mathsf{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 1$, then P_2 sets $b_i = b_i^{(1)} \oplus b_i^{(2)}$ (and continues running the protocol).
3. If all m iterations have been run, party P_1 outputs a_m and party P_2 outputs b_m.

Figure 4: Generic protocol for computing a function f.

4.2 Proof of Security for a Particular Function

Protocol 2 cannot guarantee complete fairness for *all* functions f. Rather, what we claim is that for *certain* functions f and particular associated values of α, the protocol provides complete fairness. In this section, we prove security for the following function f:

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

This function has an embedded XOR, and is defined over a finite domain so that $X_n = X = \{x_1, x_2, x_3\}$ and $Y_n = Y = \{y_1, y_2\}$. For this f, we set $\alpha = 1/5$ in Protocol 2.

Theorem 4.1 *If* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is an information-theoretically secure* m*-time MAC, and* π *securely computes* $\mathsf{ShareGen}'$ *with abort, then the protocol in Figure 4, with* $\alpha = 1/5$*, securely computes* f *with complete fairness.*

Proof: Let Π denote the protocol in Figure 4 with $\alpha = 1/5$. We analyze Π in a hybrid model where there is a trusted party computing $\mathsf{ShareGen}'$. (One again, we stress that since π is only guaranteed to securely compute $\mathsf{ShareGen}'$ with abort, the adversary is allowed to abort the trusted party computing $\mathsf{ShareGen}'$ before it sends output to the honest party.) We will prove that an execution of Protocol 2 in this hybrid model is *statistically close* to an evaluation of f in the ideal model with complete fairness, where the only differences can occur due to MAC forgeries. Applying Proposition 1 then implies the theorem.

In the two claims that follow, we separately analyze corruption of P_2 and P_1. The case of a corrupted P_2 is relatively easy to analyze since P_1 always "gets the output first" (because, in every iteration — and iteration i^* in particular — P_2 sends its share first). The proof of security when P_1 is corrupted is much more challenging, and is given second.

Claim 4 *For every non-uniform, polynomial-time adversary* $\mathcal{A}$ *corrupting* P_2 *and running* Π *in a hybrid model with access to an ideal functionality computing* $\mathsf{ShareGen}'$ *(with abort), there exists a non-uniform, probabilistic polynomial-time adversary* $\mathcal{S}$ *corrupting* P_2 *and running in the ideal world with access to an ideal functionality computing* f *(with complete fairness), such that*

$$\left\{\mathrm{IDEAL}_{f,\mathcal{S}(z)}(x,y,n)\right\}_{(x,y)\in X\times Y, z\in\{0,1\}^*, n\in\mathbb{N}} \stackrel{\mathrm{s}}{\equiv} \left\{\mathrm{HYBRID}^{\mathsf{ShareGen}'}_{\Pi,\mathcal{A}(z)}(x,y,n)\right\}_{(x,y)\in X\times Y, z\in\{0,1\}^*, n\in\mathbb{N}}.$$

Proof: Let P_2 be corrupted by $\mathcal{A}$. We construct a simulator $\mathcal{S}$ given black-box access to $\mathcal{A}$:

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input y, the auxiliary input z, and the security parameter n. The simulator also chooses $\hat{y} \in Y$ uniformly at random. (It will send $\hat{y}$ to the trusted party, if needed.)
2. $\mathcal{S}$ receives the input y' of $\mathcal{A}$ to the computation of the functionality $\mathsf{ShareGen}'$.
   (a) If $y' \notin Y$ (this includes the case when $y' = \perp$ since $\mathcal{A}$ aborts), then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of $\mathsf{ShareGen}'$ and sends $\hat{y}$ to the trusted party computing f. It then halts and outputs whatever $\mathcal{A}$ outputs.
   (b) Otherwise, if the input is some $y' \in Y$, then $\mathcal{S}$ chooses uniformly distributed shares $a_1^{(2)}, \ldots, a_m^{(2)}$ and $b_1^{(2)}, \ldots, b_m^{(2)}$. In addition, it generates keys $k_a, k_b \leftarrow \mathsf{Gen}(1^n)$ and computes $t_i^a = \mathsf{Mac}_{k_a}(i \| a_i^{(2)})$ for every i. Finally, it hands $\mathcal{A}$ the strings $b_1^{(2)}, \ldots, b_m^{(2)}$, $(a_1^{(2)}, t_1^a), \ldots, (a_m^{(2)}, t_m^a)$, and k_b as its output from the computation of $\mathsf{ShareGen}'$.
3. If $\mathcal{A}$ sends abort to the trusted party computing $\mathsf{ShareGen}'$, then $\mathcal{S}$ sends $\hat{y}$ to the trusted party computing f. It then halts and outputs whatever $\mathcal{A}$ outputs. Otherwise (i.e., if $\mathcal{A}$ sends $\mathsf{continue}$), $\mathcal{S}$ proceeds as below.
4. $\mathcal{S}$ chooses i^* according to a geometric distribution with parameter α.
5. For $i = 1$ to $i^* - 1$:
   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_i^{(2)}, \hat{t}_i^a)$ in the ith iteration. If $\mathsf{Vrfy}_{k_a}(i \| \hat{a}_i^{(2)}, \hat{t}_i^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends $\hat{y}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise, $\mathcal{S}$ proceeds.

(b) $\mathcal{S}$ chooses $\hat{x} \in X$ uniformly at random, computes $b_i = f(\hat{x}, y')$, sets $b_i^{(1)} = b_i^{(2)} \oplus b_i$, and computes $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$. It gives $\mathcal{A}$ the message $(b_i^{(1)}, t_i^b)$. (Note that a fresh $\hat{x}$ is chosen in every iteration.)

6. For $i = i^*$:

   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_{i^*}^{(2)}, \hat{t}_{i^*}^a)$. If $\mathsf{Vrfy}_{k_a}(i^* \| \hat{a}_{i^*}^{(2)}, \hat{t}_{i^*}^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ sends $\hat{y}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise, $\mathcal{S}$ sends y' to the trusted party computing f, receives the output $z = f(x, y')$, and proceeds.

   (b) $\mathcal{S}$ sets $b_{i^*}^{(1)} = b_{i^*}^{(2)} \oplus z$, and computes $t_{i^*}^b = \mathsf{Mac}_{k_b}(i^* \| b_{i^*}^{(1)})$. It gives $\mathcal{A}$ the message $(b_{i^*}^{(1)}, t_{i^*}^b)$.

7. For $i = i^* + 1$ to m:

   (a) $\mathcal{S}$ receives $\mathcal{A}$'s message $(\hat{a}_i^{(2)}, \hat{t}_i^a)$ in the ith iteration. If $\mathsf{Vrfy}_{k_a}(i \| \hat{a}_i^{(2)}, \hat{t}_i^a) = 0$ (or the message is invalid, or $\mathcal{A}$ aborts), then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts.

   (b) $\mathcal{S}$ sets $b_i^{(1)} = b_i^{(2)} \oplus z$, and computes $t_i^b = \mathsf{Mac}_{k_b}(i \| b_i^{(1)})$. It gives $\mathcal{A}$ the message $(b_i^{(1)}, t_i^b)$.

8. If $\mathcal{S}$ has not halted yet, at this point it outputs whatever $\mathcal{A}$ outputs and halts.

We assume that if $\mathsf{Vrfy}_{k_a}(i \| \hat{a}_i^{(2)}, \hat{t}_i^a) = 1$, then $\hat{a}_i^{(2)} = a_i^{(2)}$ (meaning that $\mathcal{A}$ sent the same share that it received). It is straightforward to prove that this is the case with all but negligible probability based on the information-theoretic security of the MAC. Under this assumption, the distribution generated by $\mathcal{S}$ in an ideal-world execution with a trusted party computing f is *identical* to the distribution in a hybrid execution between $\mathcal{A}$ and an honest P_1. To see this, first note that the view of $\mathcal{A}$ is identical in both worlds. As for the output of P_1, if $\mathcal{A}$ aborts (or sends an invalid message) before sending its first-iteration message, then P_1 outputs $f(x, \hat{y})$ for a random $\hat{y} \in Y$ in both the hybrid and ideal worlds. If $\mathcal{A}$ aborts after sending a valid iteration-i message then, conditioned on $\mathcal{A}$'s view at that point, the distribution of i^* is identical in the hybrid and ideal worlds. Moreover, in both worlds, P_1 outputs $f(x, \hat{y})$ (for a random $\hat{y} \in Y$) if $i < i^*$ and outputs $f(x, y')$ if $i \geq i^*$. This concludes the proof of this case. ■

We remark that the proof of the preceding claim did not depend on the value of α or the particular function f. The value of α and the specific nature of f will become important when we deal with a malicious P_1 in the proof of the following claim.

Claim 5 *For every non-uniform, polynomial-time adversary $\mathcal{A}$ corrupting P_1 and running Π in a hybrid model with access to an ideal functionality computing $\mathsf{ShareGen}'$ (with abort), there exists a non-uniform, probabilistic polynomial-time adversary $\mathcal{S}$ corrupting P_1 and running in the ideal world with access to an ideal functionality computing f (with complete fairness), such that*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z)}(x, y, n)\right\}_{(x,y)\in X\times Y, z\in\{0,1\}^*, n\in\mathbb{N}} \stackrel{\text{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen}'}_{\Pi,\mathcal{A}(z)}(x, y, n)\right\}_{(x,y)\in X\times Y, z\in\{0,1\}^*, n\in\mathbb{N}}.$$

Proof: Say P_1 is corrupted by an adversary $\mathcal{A}$. We construct a simulator $\mathcal{S}$ that is given black-box access to $\mathcal{A}$. *For readability in what follows, we ignore the presence of the MAC-tags and keys.* That is, we do not mention the fact that $\mathcal{S}$ computes MAC-tags for messages it gives to $\mathcal{A}$, nor do we mention the fact that $\mathcal{S}$ must verify the MAC-tags on the messages sent by $\mathcal{A}$. When we say that $\mathcal{A}$ "aborts", we include in this the event that $\mathcal{A}$ sends an invalid message, or a message whose tag does not pass verification.

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input[5] x', auxiliary input z, and the security parameter n. The simulator also chooses $\hat{x} \in X$ uniformly at random (it will send $\hat{x}$ to the trusted party, if needed).

2. $\mathcal{S}$ receives the input x of $\mathcal{A}$ to the computation of the functionality $\mathsf{ShareGen}'$.

   (a) If $x \notin X$ (this includes the case when $x = \perp$ since $\mathcal{A}$ aborts), then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of $\mathsf{ShareGen}'$, sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.

   (b) Otherwise, if the input is some $x \in X$, then $\mathcal{S}$ chooses uniformly distributed shares $a_1^{(1)}, \ldots, a_m^{(1)}$ and $b_1^{(1)}, \ldots, b_m^{(1)}$. Then, $\mathcal{S}$ gives these shares to $\mathcal{A}$ as its output from the computation of $\mathsf{ShareGen}'$.

3. If $\mathcal{A}$ sends abort to the trusted party computing $\mathsf{ShareGen}'$, then $\mathcal{S}$ sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise (i.e., if $\mathcal{A}$ sends $\mathsf{continue}$), $\mathcal{S}$ proceeds as below.

4. Choose i^* according to a geometric distribution with parameter α. We now branch depending on the value of x.

   If $x = x_3$:

5. For $i = 1$ to m:

   (a) $\mathcal{S}$ sets $a_i^{(2)} = a_i^{(1)} \oplus 1$ and gives $a_i^{(2)}$ to $\mathcal{A}$. (Recall that $f(x_3, y) = 1$ for any y.)

   (b) If $\mathcal{A}$ aborts and $i \leq i^*$, then $\mathcal{S}$ sends $\hat{x}$ to the trusted party computing f. If $\mathcal{A}$ aborts and $i > i^*$ then $\mathcal{S}$ sends $x = x_3$ to the trusted party computing f. In either case, $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts.

   If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds to the next iteration.

6. If $\mathcal{S}$ has not halted yet, then if $i^* \leq m$ it sends x_3 to the trusted party computing f while if $i^* > m$ it sends $\hat{x}$. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

   If $x \in \{x_1, x_2\}$:

7. Let $\bar{x}$ be the "other" value in $\{x_1, x_2\}$; i.e., if $x = x_c$ then $\bar{x} = x_{3-c}$.

8. For $i = 1$ to $i^* - 1$:

   (a) $\mathcal{S}$ chooses $\hat{y} \in Y$ uniformly at random, computes $a_i = f(x, \hat{y})$, and sets $a_i^{(2)} = a_i^{(1)} \oplus a_i$. It gives $a_i^{(2)}$ to $\mathcal{A}$. (Note that a fresh $\hat{y}$ is chosen in every iteration.)

   (b) If $\mathcal{A}$ aborts:

      i. If $a_i = 0$, then with probability $1/3$ send $\bar{x}$ to the trusted party computing f, and with probability $2/3$ send x_3.

      ii. If $a_i = 1$, then with probability $1/3$ send x to the trusted party computing f; with probability $1/2$ send $\bar{x}$; and with probability $1/6$ send x_3.

[5] To simplify readability later, we reserve x for the value input by $\mathcal{A}$ to the computation of $\mathsf{ShareGen}'$.

In either case, $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts.

If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.

9. For $i = i^*$ to m:
   (a) If $i = i^*$ then $\mathcal{S}$ sends x to the trusted party computing f and receives $z = f(x, y)$.
   (b) $\mathcal{S}$ sets $a_i^{(2)} = a_i^{(1)} \oplus z$ and gives $a_i^{(2)}$ to $\mathcal{A}$.
   (c) If $\mathcal{A}$ aborts, then $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.
10. If $\mathcal{S}$ has not yet halted, and has not yet sent anything to the trusted party computing f (this can only happen if $i^* > m$ and $\mathcal{A}$ has never aborted), then it sends $\hat{x}$ to the trusted party. Then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

We will show that the distribution generated by $\mathcal{S}$ in an ideal-world execution with a trusted party computing f is *identical* to the distribution in a hybrid execution between $\mathcal{A}$ and an honest P_2. (As always, we are ignoring here the possibility that $\mathcal{A}$ can forge a valid MAC-tag; once again, this introduces only a negligible statistical difference.) We first observe that the case of $x = x_3$ is straightforward since in this case $\mathcal{S}$ does not need to send anything to the trusted party until *after* $\mathcal{A}$ aborts. (This is because $a_i = 1$ for all i since $f(x_3, y) = 1$ for all $y \in Y$; note that this is the first time in the proof we rely on specific properties of f.) For the remainder of the proof, we therefore focus our attention on the case when $x \in \{x_1, x_2\}$.

Let $\text{VIEW}_{\mathsf{hyb}}(x, y)$ be the random variable denoting the view of $\mathcal{A}$ in the hybrid world (i.e., running Π with a trusted party computing $\mathsf{ShareGen}'$) when P_2 holds input y and $\mathcal{A}$ uses input x in the computation of $\mathsf{ShareGen}'$. Let $\text{VIEW}_{\mathsf{ideal}}(x, y)$ be the random variable denoting the view of $\mathcal{A}$ in the ideal world (i.e., where $\mathcal{S}$ runs $\mathcal{A}$ as a black-box and interacts with a trusted party computing f) with x, y similarly defined. Finally, let $\text{OUT}_{\mathsf{hyb}}(x, y), \text{OUT}_{\mathsf{ideal}}(x, y)$ be random variables denoting the output of the honest player P_2 in the hybrid and ideal worlds, respectively, for the given x and y. We will show that for any $x \in \{x_1, x_2\}$ and $y \in Y$,

$$\left(\text{VIEW}_{\mathsf{hyb}}(x, y), \text{OUT}_{\mathsf{hyb}}(x, y)\right) \equiv \left(\text{VIEW}_{\mathsf{ideal}}(x, y), \text{OUT}_{\mathsf{ideal}}(x, y)\right). \tag{2}$$

(We stress that the above assumes $\mathcal{A}$ never forges a valid MAC-tag, and therefore the security parameter n can be ignored and perfect equivalence obtained. Taking the possibility of a forged MAC-tag into account, the above distributions would then have statistical difference negligible in the security parameter n.) It is immediate from the description of $\mathcal{S}$ that $\text{VIEW}_{\mathsf{hyb}}(x, y) \equiv \text{VIEW}_{\mathsf{ideal}}(x, y)$ for any x, y; the difficulty lies in arguing about the joint distribution of $\mathcal{A}$'s view and P_2's output, as above.

We prove Eq. (2) by showing that for any x, y as above and any view v and bit b, it holds that:

$$\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}(x, y), \text{OUT}_{\mathsf{hyb}}(x, y)\right) = (v, b)\right] = \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}(x, y), \text{OUT}_{\mathsf{ideal}}(x, y)\right) = (v, b)\right]. \tag{3}$$

Clearly, if v represents a view that does not correspond to the actions of $\mathcal{A}$ (e.g., v contains a_i, but given view v the adversary would have aborted prior to iteration i; or v does not contain a_i, but given view v the adversary would not have aborted prior to iteration i), then both probabilities in Eq. (3) are identically 0 (regardless of b). From now on, therefore, we only consider views that correspond to actions of $\mathcal{A}$.

$\mathcal{A}$'s view consists of its initial inputs, the values $a_1^{(1)}, b_1^{(1)}, \ldots, a_m^{(1)}, b_m^{(1)}$ that $\mathcal{A}$ receives from computation of $\mathsf{ShareGen}'$, and — if $\mathcal{A}$ does not abort before the first iteration — a sequence of values $a_1, \ldots, a_i$ where i is the iteration in which $\mathcal{A}$ aborts (if any). (Technically $\mathcal{A}$ receives $a_1^{(2)}, \ldots, a_i^{(2)}$ but we equivalently consider the reconstructed values $a_1, \ldots, a_i$ instead.) Looking at the description of $\mathcal{S}$, it is easy to see that if v represents a view in which $\mathcal{A}$ aborts before the first iteration, or in which $\mathcal{A}$ never aborts (i.e., $\mathcal{A}$ runs the protocol to completion), then Eq. (3) holds for either choice of b. Thus, the "difficult" cases to analyze are exactly those in which $\mathcal{A}$ aborts in some iteration i.

Let v be a view in which $\mathcal{A}$ aborts in iteration i (i.e., after receiving its iteration-i message). We will let $\mathcal{A}$'s initial inputs and its outputs from $\mathsf{ShareGen}'$ be implicit, and focus on the vector of values $\vec{a}_i = (a_1, \ldots, a_i)$ that $\mathcal{A}$ sees before it aborts in iteration i, We will show that for any x, y as above, any $\vec{a}_i$, and any bit b it holds that

$$\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (\vec{a}_i, b)\right] = \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (\vec{a}_i, b)\right]. \quad (4)$$

We stress that we are considering exactly those views $\vec{a}_i = (a_1, \ldots, a_i)$ in which $\mathcal{A}$ aborts after receiving its iteration-i message; there is thus no possibility that $\mathcal{A}$ might abort given the sequence of values $a_1, \ldots, a_j$ (with $j < i$).

Toward proving Eq. (4), we first prove:

Claim 6 *For any $x \in \{x_1, x_2\}$ and $y \in Y$,*

$$\begin{aligned} &\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (\vec{a}_i, b) \bigwedge i^* < i\right] \\ &\quad = \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (\vec{a}_i, b) \bigwedge i^* < i\right]. \end{aligned} \quad (5)$$

Proof: A proof of this claim follows easily from the observation that, conditioned on $i^* < i$, the "true" input of P_1 is used to compute P_2's output in both the hybrid and ideal worlds.

Formally, fix some x, y and let these be implicit in what follows. To prove the claim, note that

$$\begin{aligned} &\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}\right) = (\vec{a}_i, b) \bigwedge i^* < i\right] \\ &\quad = \Pr\left[\text{OUT}_{\mathsf{hyb}} = b \mid \text{VIEW}_{\mathsf{hyb}} = \vec{a}_i \bigwedge i^* < i\right] \cdot \Pr\left[\text{VIEW}_{\mathsf{hyb}} = \vec{a}_i \bigwedge i^* < i\right] \end{aligned}$$

and

$$\begin{aligned} &\Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}\right) = (\vec{a}_i, b) \bigwedge i^* < i\right] \\ &\quad = \Pr\left[\text{OUT}_{\mathsf{ideal}} = b \mid \text{VIEW}_{\mathsf{ideal}} = \vec{a}_i \bigwedge i^* < i\right] \cdot \Pr\left[\text{VIEW}_{\mathsf{ideal}} = \vec{a}_i \bigwedge i^* < i\right]. \end{aligned}$$

It follows from the description of $\mathcal{S}$ that $\Pr[\text{VIEW}_{\mathsf{hyb}} = \vec{a}_i \bigwedge i^* < i] = \Pr[\text{VIEW}_{\mathsf{ideal}} = \vec{a}_i \bigwedge i^* < i]$. Furthermore, conditioned on $i^* < i$ the output of P_2 is the correct output $f(x,y)$ in both the hybrid and ideal worlds. We conclude that Eq. (5) holds. ■

To complete the proof of Eq. (4), we prove that for any $x \in \{x_1, x_2\}$ and $y \in Y$, any $\vec{a}_i \in \{0,1\}^i$, and all $b \in \{0,1\}$ it holds that

$$\begin{aligned} &\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (\vec{a}_i, b) \bigwedge i^* \geq i\right] \\ &\quad = \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (\vec{a}_i, b) \bigwedge i^* \geq i\right]. \end{aligned} \quad (6)$$

This is the crux of the proof. Write $\vec{a}_i = (\vec{a}_{i-1}, a)$, $\text{VIEW}_{\mathsf{hyb}} = (\overline{\text{VIEW}}^{i-1}_{\mathsf{hyb}}, \text{VIEW}^i_{\mathsf{hyb}})$, and $\text{VIEW}_{\mathsf{ideal}} = (\overline{\text{VIEW}}^{i-1}_{\mathsf{ideal}}, \text{VIEW}^i_{\mathsf{ideal}})$. (In what follows, we also often leave x and y implicit in the interests of readability.) Then

$$\begin{aligned}
&\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}\right) = (\vec{a}_i, b) \bigwedge i^* \geq i\right] \\
&\quad = \Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}\right) = (a, b) \mid \overline{\text{VIEW}}^{i-1}_{\mathsf{hyb}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right] \cdot \Pr\left[\overline{\text{VIEW}}^{i-1}_{\mathsf{hyb}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right]
\end{aligned}$$

and

$$\begin{aligned}
&\Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}\right) = (\vec{a}_i, b) \bigwedge i^* \geq i\right] \\
&\quad = \Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}\right) = (a, b) \mid \overline{\text{VIEW}}^{i-1}_{\mathsf{ideal}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right] \cdot \Pr\left[\overline{\text{VIEW}}^{i-1}_{\mathsf{ideal}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right].
\end{aligned}$$

Once again, it follows readily from the description of $\mathcal{S}$ that

$$\Pr\left[\overline{\text{VIEW}}^{i-1}_{\mathsf{hyb}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right] = \Pr\left[\overline{\text{VIEW}}^{i-1}_{\mathsf{ideal}} = \vec{a}_{i-1} \bigwedge i^* \geq i\right].$$

Moreover, conditioned on the event that $i^* \geq i$, the random variables of $\text{VIEW}^i_{\mathsf{hyb}}$ and $\text{OUT}_{\mathsf{hyb}}$ (resp., $\text{VIEW}^i_{\mathsf{ideal}}$ and $\text{OUT}_{\mathsf{ideal}}$) are independent of $\overline{\text{VIEW}}^{i-1}_{\mathsf{hyb}}$ (resp., $\overline{\text{VIEW}}^{i-1}_{\mathsf{ideal}}$) for fixed x and y. Thus, Eq. (6) is proved once we show that

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}, \text{OUT}_{\mathsf{hyb}}\right) = (a, b) \mid i^* \geq i\right] = \Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}, \text{OUT}_{\mathsf{ideal}}\right) = (a, b) \mid i^* \geq i\right] \tag{7}$$

for all x, y, a, b as above. We prove this via case-by-case analysis. For convenience, we recall the table for f:

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

Case 1: $x = x_1$ and $y = y_1$. We analyze the hybrid world first, followed by the ideal world.

Hybrid world. We first consider the hybrid world where the parties are running protocol Π. If $\mathcal{A}$ aborts after receiving its iteration-i message, P_2 will output $\text{OUT}_{\mathsf{hyb}} = b_{i-1}$. Since $i^* \geq i$, we have $b_{i-1} = f(\hat{x}, y_1)$ where $\hat{x}$ is chosen uniformly from X. So $\Pr[\text{OUT}_{\mathsf{hyb}} = 0] = \Pr_{\hat{x} \leftarrow X}[f(\hat{x}, y_1) = 0] = 1/3$ and $\Pr[\text{OUT}_{\mathsf{hyb}} = 1] = 2/3$.

Since $i^* \geq i$, the value of $\text{VIEW}^i_{\mathsf{hyb}} = a_i$ is independent of the value of b_{i-1}. Conditioned on the event that $i^* \geq i$, we have $\Pr[i^* = i] = \alpha = 1/5$ and $\Pr[i^* > i] = 4/5$. If $i^* = i$, then $a_i = f(x, y) = f(x_1, y_1) = 0$. If $i^* > i$, then $a_i = f(x_1, \hat{y})$ where $\hat{y}$ is chosen uniformly from Y. So $\Pr[a_i = 1] = \Pr_{\hat{y} \leftarrow Y}[f(x_1, \hat{y}) = 1] = 1/2$ and $\Pr[a_i = 0] = 1/2$. Overall, then, we have

$$\begin{aligned}
\Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1) = 0 \mid i^* \geq i] &= \alpha \cdot 1 + (1 - \alpha) \cdot \frac{1}{2} = \frac{3}{5} \\
\Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1) = 1 \mid i^* \geq i] &= \alpha \cdot 0 + (1 - \alpha) \cdot \frac{1}{2} = \frac{2}{5}.
\end{aligned}$$

Putting everything together gives

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x_1,y_1),\text{OUT}_{\mathsf{hyb}}(x_1,y_1)\right)=(a,b)\mid i^*\geq i\right]=\begin{cases}\frac{3}{5}\cdot\frac{1}{3}=\frac{1}{5} & (a,b)=(0,0)\\ \frac{3}{5}\cdot\frac{2}{3}=\frac{2}{5} & (a,b)=(0,1)\\ \frac{2}{5}\cdot\frac{1}{3}=\frac{2}{15} & (a,b)=(1,0)\\ \frac{2}{5}\cdot\frac{2}{3}=\frac{4}{15} & (a,b)=(1,1)\end{cases} \tag{8}$$

Ideal world. We now turn our attention to the ideal world. Since we are conditioning on $i^* \geq i$, here it is also the case that $\Pr[i^* = i] = \alpha = 1/5$ and $\Pr[i^* > i] = 4/5$. Furthermore, if $i^* = i$ then $\text{VIEW}^i_{\mathsf{ideal}} = a_i = f(x_1, y_1) = 0$. Now, however, if $i^* = i$ then $\mathcal{S}$ has already sent x_1 to the trusted party computing f (in order to learn the value $f(x_1, y_1)$) and so P_2 will *also* output $f(x_1, y_1) = 0$, rather than some independent value b_{i-1}.

When $i^* > i$, then (by construction of $\mathcal{S}$) we have $\Pr[a_i = 0] = \Pr_{\hat{y}\leftarrow Y}[f(x_1, \hat{y}) = 0] = 1/2$ and $\Pr[a_i = 1] = 1/2$. Now, however, the output of P_2 depends on the value sent to the trusted party following an abort by $\mathcal{A}$, which in turn depends on a_i (cf. step 8(b) of $\mathcal{S}$). In particular, we have:

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_1,y_1)=0\mid a_i=0\bigwedge i^*>i]\\ &\quad=\ \Pr[\mathcal{S}\text{ sends }x_1\text{ to the trusted party}\mid a_i=0\bigwedge i^*>i]\ =\ 0,\end{aligned}$$

and

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_1,y_1)=0\mid a_i=1\bigwedge i^*>i]\\ &\quad=\ \Pr[\mathcal{S}\text{ sends }x_1\text{ to the trusted party}\mid a_i=1\bigwedge i^*>i]\ =\ 1/3\end{aligned}$$

(in calculating the above, recall that $x = x_1$). Putting everything together, we obtain

$$\begin{aligned}&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(0,0)\mid i^*\geq i\right]\\ &=\ \alpha\cdot\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(0,0)\mid i^*=i\right]\\ &\quad+(1-\alpha)\cdot\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(0,0)\mid i^*>i\right]\\ &=\ \alpha+(1-\alpha)\cdot 0\ =\ \frac{1}{5}.\end{aligned} \tag{9}$$

Similarly,

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(0,1)\mid i^*\geq i\right]\ =\ (1-\alpha)\cdot\frac{1}{2}\cdot 1\ =\ \frac{2}{5} \tag{10}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(1,0)\mid i^*\geq i\right]\ =\ (1-\alpha)\cdot\frac{1}{2}\cdot\frac{1}{3}\ =\ \frac{2}{15} \tag{11}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\right)=(1,1)\mid i^*\geq i\right]\ =\ (1-\alpha)\cdot\frac{1}{2}\cdot\frac{2}{3}\ =\ \frac{4}{15}, \tag{12}$$

in exact agreement with Eq. (8).

Case 2: $x = x_2$ and $y = y_1$. In all the remaining cases, the arguments are the same as before; just the numbers differ. Therefore, we will allow ourselves to be more laconic.

In the hybrid world, conditioned on $i^* \geq i$, the values of $\text{OUT}_{\mathsf{hyb}} = b_{i-1}$ and $\text{VIEW}^i_{\mathsf{hyb}} = a_i$ are again independent. The distribution of b_{i-1} is given by: $\Pr[b_{i-1} = 0] = \Pr_{\hat{x} \leftarrow X}[f(\hat{x}, y_1) = 0] = 1/3$ and $\Pr[b_{i-1} = 1] = 2/3$. As for the distribution of a_i, we have

$$\begin{aligned}\Pr[a_i = 1 \mid i^* \geq i] &= \alpha \cdot \Pr[a_i = 1 \mid i^* = i] + (1-\alpha) \cdot \Pr[a_i = 1 \mid i^* > i] \\ &= \alpha \cdot 1 + (1-\alpha) \cdot \Pr_{\hat{y} \leftarrow Y}[f(x_1, \hat{y}) = 1] \\ &= \frac{1}{5} + \frac{4}{5} \cdot \frac{1}{2} \;=\; \frac{3}{5},\end{aligned}$$

and so $\Pr[a_i = 0 \mid i^* \geq i] = 2/5$. Putting everything together gives

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x_2, y_1), \text{OUT}_{\mathsf{hyb}}(x_2, y_1)\right) = (a, b) \mid i^* \geq i\right] = \begin{cases} \frac{2}{5} \cdot \frac{1}{3} = \frac{2}{15} & (a,b) = (0,0) \\ \frac{2}{5} \cdot \frac{2}{3} = \frac{4}{15} & (a,b) = (0,1) \\ \frac{3}{5} \cdot \frac{1}{3} = \frac{1}{5} & (a,b) = (1,0) \\ \frac{3}{5} \cdot \frac{2}{3} = \frac{2}{5} & (a,b) = (1,1) \end{cases} \tag{13}$$

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x_2, y_1) = 1$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 1] = \Pr_{\hat{y} \leftarrow Y}[f(x_1, \hat{y}) = 1] = 1/2$ and $\Pr[a_i = 0] = 1/2$. The value of $\text{OUT}_{\mathsf{ideal}}$ is now dependent on the value of a_i (cf. step 8(b) of $\mathcal{S}$); specifically:

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_2, y_1) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_1 \text{ to the trusted party} \mid a_i = 0 \bigwedge i^* > i] \;=\; 1/3,\end{aligned}$$

and

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_2, y_1) = 0 \mid a_i = 1 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_1 \text{ to the trusted party} \mid a_i = 1 \bigwedge i^* > i] \;=\; 1/2\end{aligned}$$

(using the fact that $x = x_2$). Putting everything together, we obtain

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_2, y_1), \text{OUT}_{\mathsf{ideal}}(x_2, y_1)\right) = (0,0) \mid i^* \geq i\right] = (1-\alpha) \cdot \frac{1}{2} \cdot \frac{1}{3} = \frac{2}{15} \tag{14}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_2, y_1), \text{OUT}_{\mathsf{ideal}}(x_2, y_1)\right) = (0,1) \mid i^* \geq i\right] = (1-\alpha) \cdot \frac{1}{2} \cdot \frac{2}{3} = \frac{4}{15} \tag{15}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_2, y_1), \text{OUT}_{\mathsf{ideal}}(x_2, y_1)\right) = (1,0) \mid i^* \geq i\right] = (1-\alpha) \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{5} \tag{16}$$

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x_2, y_1), \text{OUT}_{\mathsf{ideal}}(x_2, y_1)\right) = (1,1) \mid i^* \geq i\right] = \alpha + (1-\alpha) \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{2}{5}, \tag{17}$$

in exact agreement with Eq. (13).

Case 3: $x = x_1$ and $y = y_2$. In the hybrid world, this case is exactly symmetric to the case when $x = x_2$ and $y = y_1$. Thus we obtain the same distribution as in Eq. (13).

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x_1, y_2) = 1$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 1] = \Pr_{\hat{y} \leftarrow Y}[f(x_2, \hat{y}) = 1] = 1/2$ and $\Pr[a_i = 0] = 1/2$. The value of $\text{OUT}_{\mathsf{ideal}}$ is dependent on the value of a_i (cf. step 8(b) of $\mathcal{S}$); specifically:

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_1, y_2) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_2 \text{ to the trusted party} \mid a_i = 0 \bigwedge i^* > i] \;=\; 1/3,\end{aligned}$$

and

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_1, y_2) = 0 \mid a_i = 1 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_2 \text{ to the trusted party} \mid a_i = 1 \bigwedge i^* > i] \;=\; 1/2\end{aligned}$$

(using the fact that $x = x_1$). Putting everything together, we obtain the same distribution as in Eqs. (14)–(17). The distributions in the hybrid and ideal worlds are, once again, in exact agreement.

Case 4: $x = x_2$ and $y = y_2$. In the hybrid world, this case is exactly symmetric to the case when $x = x_1$ and $y = y_1$. Thus we obtain the same distribution as in Eq. (8).

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x_2, y_2) = 0$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 1] = \Pr_{\hat{y} \leftarrow Y}[f(x_2, \hat{y}) = 1] = 1/2$ and $\Pr[a_i = 0] = 1/2$. The value of $\text{OUT}_{\mathsf{ideal}}$ is dependent on the value of a_i (cf. step 8(b) of $\mathcal{S}$); specifically:

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_2, y_2) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_2 \text{ to the trusted party} \mid a_i = 0 \bigwedge i^* > i] \;=\; 0,\end{aligned}$$

and

$$\begin{aligned}&\Pr[\text{OUT}_{\mathsf{ideal}}(x_2, y_2) = 0 \mid a_i = 1 \bigwedge i^* > i] \\ &\quad = \Pr[\mathcal{S} \text{ sends } x_2 \text{ to the trusted party} \mid a_i = 1 \bigwedge i^* > i] \;=\; 1/3\end{aligned}$$

(using the fact that $x = x_2$). Putting everything together, we obtain the same distribution as in Eqs. (9)–(12). The distributions in the hybrid and ideal worlds are, once again, in exact agreement. This completes the proof of Claim 5. ■

The preceding claims along with Proposition 1 conclude the proof of Theorem 4.1. ■

5 A Lower Bound for Functions with an Embedded XOR

In the previous section we have shown a protocol that enables completely fair computation of certain functions that contain an embedded XOR. That protocol, however, has round complexity $\omega(\log n)$. (The round complexity may be worse, depending on α, but if α is constant then the round complexity is $m = \omega(\log n)$.) In this section we prove that this is inherent for any function that has an embedded XOR.

5.1 Preliminaries

Let f be a single-output, boolean function with an embedded XOR; that is, a function for which there exist inputs x_0, x_1, y_0, y_1 such that $f(x_i, y_j) = i \oplus j$. Let Π be an $r(n)$-round protocol that securely computes f with complete fairness. Here we denote the two parties executing the protocol by A and B. We present some basic conventions below, as well as the specification of a series of fail-stop adversaries that we will use in our proof.

Notation and conventions: We assume that A sends the first message in protocol Π, and B sends the last message. A *round* of Π consists of a message from A followed by a message from B. If A aborts before sending its ith-round message (but after sending the first $i - 1$ messages), then

we denote by b_{i-1} the value output by B (so B outputs b_0 if A sends nothing). If B aborts before sending its ith-round message (but after sending the first $i-1$ messages), then we denote by a_i the value output by A (so A outputs a_1 if B sends nothing). If neither party aborts, then B outputs b_r and A outputs a_{r+1}.

Proof overview. We consider executions of Π in which each party begins with input distributed uniformly in $\{x_0, x_1\}$ or $\{y_0, y_1\}$, respectively. We describe a series of $4r$ fail-stop adversaries $\{A_{i1}, A_{i0}, B_{i1}, B_{i0}\}_{i=1}^{r}$ where, intuitively, the aim of adversary A_{ib} is to guess B's input while *simultaneously* biasing B's output toward the bit b. (The aim of adversary B_{ib} is exactly analogous.) We show that if $r = O(\log n)$, then one of these adversaries succeeds with "high" probability even though, as explained next, this is not possible in the ideal world.

In the ideal world evaluation of f (when B chooses its input at random in $\{y_0, y_1\}$), it is certainly possible for an adversary corrupting A to learn B's input with certainty (this follows from the fact that f contains an embedded XOR), and it may be possible, depending on f, to bias B's output with certainty. It is *not* possible, however, to do both simultaneously with high probability. (We formally state and prove this below.) This gives us our desired contradiction whenever $r = O(\log n)$, and shows that no protocol with this many rounds can be completely fair.

Descriptions of the adversaries. Before giving the formal specification of the adversaries, we provide an intuitive description of adversary A_{i1}. (The other adversaries rely on the same intuition.) A_{i1} chooses a random input $x \in \{x_0, x_1\}$ and runs the protocol honestly for $i-1$ rounds. It then computes the value it would output if B aborted the protocol at the current point, i.e., it computes a_i. If $a_i = 1$, then A_{i1} continues the protocol for one more round (hoping that this will cause B to output 1 also) and halts. If $a_i = 0$, then A_{i1} halts immediately (hoping that B's output does not yet "match" A_{i1}'s, and that B will still output 1). In addition to this behavior during the protocol, A_{i1} also guesses B's input, in the natural way, based on its own input value x and the value of a_i it computed. In particular, if $x = x_\sigma$ then A_{i1} guesses that B's input is $y_{a_i \oplus \sigma}$ (since $f(x_\sigma, y_{a_i \oplus \sigma}) = a_i$).

Say B's input is y. Intuitively, because the protocol is completely fair, if the output that A_{i1} computes in round i is biased toward the correct value of $f(x, y)$, it must be that the last message sent by A_{i1} has relatively limited relevance (i.e., that B would output the same bit whether A_{i1} sends its ith round message or not). In particular, in the case of A_{r1}, the computed output must be equal to $f(x, y)$ (with all but negligible probability), and therefore the last message of the protocol is, in some sense, unnecessary. Using induction (for a logarithmic number of steps) we will demonstrate that the same holds for each of the prior rounds, and conclude that a protocol running in $\mathcal{O}(\log n)$ rounds can be transformed into an empty protocol in which neither party sends anything. This is, of course, impossible; therefore, no such protocol exists.

We now formally describe the adversaries.

Adversary A_{i1}:

1. Choose $x \in_R \{x_0, x_1\}$.
2. Run the honest A for the first $i-1$ rounds (using input x) and compute a_i:
   (a) If $a_i = 1$ and $x = x_0$, then output $\mathsf{guess}(y = y_1)$, send the ith round message, and halt.
   (b) If $a_i = 1$ and $x = x_1$, then output $\mathsf{guess}(y = y_0)$, send the ith round message, and halt.
   (c) If $a_i = 0$ and $x = x_0$, then output $\mathsf{guess}(y = y_0)$ and halt immediately.
   (d) If $a_i = 0$ and $x = x_1$, then output $\mathsf{guess}(y = y_1)$ and halt immediately.

Adversary A_{i0}:

1. Choose $x \in_R \{x_0, x_1\}$.
2. Run the honest A for the first $i-1$ rounds (using input x) and compute a_i:
   (a) If $a_i = 0$ and $x = x_0$, then output $\mathsf{guess}(y = y_0)$, send the ith round message and halt.
   (b) If $a_i = 0$ and $x = x_1$, then output $\mathsf{guess}(y = y_1)$, send the ith round message and halt.
   (c) If $a_i = 1$ and $x = x_0$, then output $\mathsf{guess}(y = y_1)$ and halt immediately.
   (d) If $a_i = 1$ and $x = x_1$, then output $\mathsf{guess}(y = y_0)$ and halt immediately.

Adversary B_{i1}:

1. Choose $y \in_R \{y_0, y_1\}$.
2. Run the honest B for the first $i-1$ rounds (using input y), receive A's ith round message, and compute b_i:
   (a) If $b_i = 1$ and $y = y_0$, then output $\mathsf{guess}(x = x_1)$, send the ith round message, and halt.
   (b) If $b_i = 1$ and $y = y_1$, then output $\mathsf{guess}(x = x_0)$, send the ith round message, and halt.
   (c) If $b_i = 0$ and $y = y_0$, then output $\mathsf{guess}(x = x_0)$ and halt immediately.
   (d) If $b_i = 0$ and $y = y_1$, then output $\mathsf{guess}(x = x_1)$ and halt immediately.

Adversary B_{i0}:

1. Choose $y \in_R \{y_0, y_1\}$.
2. Run the honest B for the first $i-1$ rounds (using input y), receive A's ith round message, and compute b_i:
   (a) If $b_i = 0$ and $y = y_0$, then output $\mathsf{guess}(x = x_0)$, send the ith round message, and halt.
   (b) If $b_i = 0$ and $y = y_1$, then output $\mathsf{guess}(x = x_1)$, send the ith round message, and halt.
   (c) If $b_i = 1$ and $y = y_0$, then output $\mathsf{guess}(x = x_1)$ and halt immediately.
   (d) If $b_i = 1$ and $y = y_1$, then output $\mathsf{guess}(X = x_0)$ and halt immediately.

Success probability for A_{i1}: As preparation for the proof that follows, we calculate the probability that A_{i1} succeeds in *simultaneously* guessing B's input y correctly, and having B output 1. By construction, if (say) A_{i1} uses $x = x_0$ as input and obtains $a_i = 0$, then it guesses correctly iff $y = y_0$. Furthermore, since it received $a_i = 0$ it does not send its ith round message; thus, by our notation, B outputs 1 if $b_{i-1} = 1$. There are three other possible ways for this to occur as well:

$$\begin{aligned}
&\Pr[A_{i1} \text{ guesses } y \wedge B \text{ outputs } 1] \\
&\quad = \Pr[x = x_0 \wedge y = y_0 \wedge a_i = 0 \wedge b_{i-1} = 1] + \Pr[x = x_0 \wedge y = y_1 \wedge a_i = 1 \wedge b_i = 1] \\
&\qquad + \Pr[x = x_1 \wedge y = y_1 \wedge a_i = 0 \wedge b_{i-1} = 1] + \Pr[x = x_1 \wedge y = y_0 \wedge a_i = 1 \wedge b_i = 1].
\end{aligned}$$

The calculations are similar for A_{i0}, B_{i1}, and B_{i0} so we present them with no further explanation.

Success probability for A_{i0}:

$$\begin{aligned}\Pr[A_{i0} \text{ guesses } y \wedge B \text{ outputs } 0] \\ = \ & \Pr[x = x_0 \wedge y = y_0 \wedge a_i = 0 \wedge b_i = 0] + \Pr[x = x_0 \wedge y = y_1 \wedge a_i = 1 \wedge b_{i-1} = 0] \\ & + \Pr[x = x_1 \wedge y = y_1 \wedge a_i = 0 \wedge b_i = 0] + \Pr[x = x_1 \wedge y = y_0 \wedge a_i = 1 \wedge b_{i-1} = 0].\end{aligned}$$

Success probability for B_{i1}:

$$\begin{aligned}\Pr[B_{i1} \text{ guesses } x \wedge A \text{ outputs } 1] \\ = \ & \Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_i = 1] + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_{i+1} = 1] \\ & + \Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_i = 1] + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_{i+1} = 1].\end{aligned}$$

Success probability for B_{i0}:

$$\begin{aligned}\Pr[B_{i0} \text{ guesses } x \wedge A \text{ outputs } 0] \\ = \ & \Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_{i+1} = 0] + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] \\ & + \Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_{i+1} = 0] + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 0].\end{aligned}$$

5.2 The Proof

We begin by showing that, in the ideal model, it is impossible for an adversary to bias the output of the honest party while simultaneously guessing the honest party's input, with probability greater than 1/2. Note that an adversary can certainly do one or the other. For example, if the honest B uses input $y \in_R \{y_0, y_1\}$ and an adversarial A uses input x_0, then A learns the input of B (by observing if the output is 0 or 1). Furthermore, if there exists a value x' for which $f(x', y_0) = f(x', y_1) = 1$ then A can completely bias the output of B to be 1.[6] In the first case, however, B's output is a random bit; in the second case, A learns no information about B's input. The following claim proves that these two extremes represent, in some sense, the best possible strategies:

Claim 7 *Consider an ideal-world evaluation of f (with complete fairness), where the honest party B chooses its input y uniformly from $\{y_0, y_1\}$ and the corrupted A^* outputs a guess for y following its interaction with the trusted party. For any A^* and any $\sigma \in \{0, 1\}$, it holds that*

$$\Pr[A^* \textit{ guesses } y \wedge B \textit{ outputs } \sigma] \leq \frac{1}{2}.$$

An analogous claim holds for the case when A is honest.

Proof: We consider the case of an honest B. Let $X_0 \stackrel{\text{def}}{=} \{x \mid f(x, y_0) = f(x, y_1) = 0\}$, and likewise $X_1 \stackrel{\text{def}}{=} \{x \mid f(x, y_0) = f(x, y_1) = 1\}$. Let $X_\oplus = \{x \mid f(x, y_0) \neq f(x, y_1)\}$. Note that X_0, X_1, and $X_\oplus$ partition the set of all inputs for A^*. In the following, when we say "A^* sends x" we mean that it sends x to the trusted party in the ideal model.

[6] We stress that this is different from the case of boolean XOR, where it is impossible to bias the honest party's output at all in the ideal model (when the honest party uses a random input).

Fix any $\sigma \in \{0,1\}$. Clearly $\Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_{\bar{\sigma}}] = 0$ since B *always* outputs $\bar{\sigma}$ when A^* sends $x \in X_{\bar{\sigma}}$. Also,

$$\Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_\sigma] = \Pr[A^* \text{ guesses } y \mid A^* \text{ sends } x \in X_\sigma] = \frac{1}{2},$$

where the first equality is because when A^* sends $x \in X_\sigma$, then party B *always* outputs σ and the second equality is because, in that case, A^* learns no information about B's input (which was chosen uniformly from $\{y_0, y_1\}$). Finally,

$$\Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_\oplus] \leq \Pr[B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_\oplus] = \frac{1}{2},$$

because B's input is chosen uniformly from $\{y_0, y_1\}$.

We thus have

$$\begin{aligned}
&\Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma] \\
&\quad = \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \wedge A^* \text{ sends } x \in X_{\bar{\sigma}}] \\
&\qquad + \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \wedge A^* \text{ sends } x \in X_\sigma] \\
&\qquad + \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \wedge A^* \text{ sends } x \in X_\oplus] \\
&\quad = \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_{\bar{\sigma}}] \cdot \Pr[A^* \text{ sends } x \in X_{\bar{\sigma}}] \\
&\qquad + \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_\sigma] \cdot \Pr[A^* \text{ sends } x \in X_\sigma] \\
&\qquad + \Pr[A^* \text{ guesses } y \wedge B \text{ outputs } \sigma \mid A^* \text{ sends } x \in X_\oplus] \cdot \Pr[A^* \text{ sends } x \in X_\oplus] \\
&\quad \leq \frac{1}{2} \cdot \big(\Pr[A^* \text{ sends } x \in X_\sigma] + \Pr[A^* \text{ sends } x \in X_\oplus]\big) \leq \frac{1}{2},
\end{aligned}$$

proving the claim. ■

The above claim, along with the assumed security of Π (with complete fairness), implies that for every inverse polynomial $\mu = 1/\mathsf{poly}$ we have

$$\Pr[B_{i0} \text{ guesses } x \wedge A \text{ outputs } 0] \leq \frac{1}{2} + \mu(n) \tag{18}$$

$$\Pr[B_{i1} \text{ guesses } x \wedge A \text{ outputs } 1] \leq \frac{1}{2} + \mu(n) \tag{19}$$

$$\Pr[A_{i0} \text{ guesses } y \wedge B \text{ outputs } 0] \leq \frac{1}{2} + \mu(n) \tag{20}$$

$$\Pr[A_{i1} \text{ guesses } y \wedge B \text{ outputs } 1] \leq \frac{1}{2} + \mu(n) \tag{21}$$

for sufficiently large n and all $1 \leq i \leq r(n)$.

We now prove a claim that states, informally, that if both parties can compute the correct output with high probability after running i rounds of Π, then they can also compute the correct output with high probability even when B does not send its ith-round message.

Claim 8 *Fix a function μ and a value of n for which Equations (18)–(21) hold for $1 \leq i \leq r(n)$, and let $\mu = \mu(n)$. For any $1 \leq i \leq r(n)$, if the following inequalities hold:*

$$\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_{i+1} = 1] - \frac{1}{4}\right| \leq \mu \tag{22}$$

$$\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_{i+1} = 1] - \frac{1}{4}\right| \leq \mu \tag{23}$$

$$\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_{i+1} = 0] - \frac{1}{4}\right| \leq \mu \tag{24}$$

$$\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_{i+1} = 0] - \frac{1}{4}\right| \leq \mu \tag{25}$$

when x is chosen uniformly from $\{x_0, x_1\}$ and y is chosen uniformly from $\{y_0, y_1\}$, then:

$$\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] - \frac{1}{4}\right| \leq 4\mu \tag{26}$$

$$\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 1] - \frac{1}{4}\right| \leq 4\mu \tag{27}$$

$$\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_i = 0] - \frac{1}{4}\right| \leq 4\mu \tag{28}$$

$$\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_i = 0] - \frac{1}{4}\right| \leq 4\mu \tag{29}$$

when x and y are chosen in the same way.

The first four equations represent the probability with which both parties receive correct output after executing the first i rounds of Π (i.e., after B sends its message in round i), for all possible choices of their inputs. The last four equations consider the same event, but when B does not send its message in round i. The claim asserts that the fact that B does not send its message in round i has a limited effect on the probability with which the parties obtain correct outputs.

Proof: We first prove Equation (26). That $\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] \leq \frac{1}{4} + 4\mu$ is immediate, since $\Pr[y = y_0 \wedge x = x_1] = \frac{1}{4}$. We must therefore prove the corresponding lower bound. Combining Equations (18), (24), and (25), and using our earlier calculation for the success probability for B_{i0}, we obtain

$$\begin{aligned}
\frac{1}{2} + \mu &\geq \Pr[B_{i0} \text{ guesses } x \wedge A \text{ outputs } 0] \\
&= \Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_{i+1} = 0] + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] \\
&\quad + \Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_{i+1} = 0] + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 0] \\
&\geq \frac{1}{4} - \mu + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] \\
&\quad + \frac{1}{4} - \mu + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 0] \\
&= \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] + \Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 0] \\
&\quad + \frac{1}{2} - 2\mu\,,
\end{aligned}$$

implying

$$\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] \leq 3\mu\,. \tag{30}$$

We also have

$$\begin{aligned}
\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 0] + \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] \\
&= \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1] \\
&\geq \Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_{i+1} = 1] \geq \frac{1}{4} - \mu\,,
\end{aligned}$$

using Equation (22) for the final inequality. Combined with Eq. (30), we conclude that

$$\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] \geq \frac{1}{4} - 4\mu,$$

proving Equation (26).

Using a symmetric argument, we can similarly prove Equation (27). Using an exactly analogous argument, but with adversary B_{i1} in place of B_{i0}, we can prove Equations (28) and (29). ■

The proof of the following claim exactly parallels the proof of the preceding claim, but using adversaries A_{i0} and A_{i1} instead of adversaries B_{i0} and B_{i1}.

Claim 9 *Fix a function μ and a value of n for which Equations (18)–(21) hold for $1 \leq i \leq r(n)$, and let $\mu = \mu(n)$. For any $1 \leq i \leq r(n)$, if the following inequalities hold:*

$$\begin{aligned}
\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_i = 1 \wedge a_i = 1] - \frac{1}{4}\right| &\leq \mu \\
\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_i = 1 \wedge a_i = 1] - \frac{1}{4}\right| &\leq \mu \\
\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_i = 0 \wedge a_i = 0] - \frac{1}{4}\right| &\leq \mu \\
\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_i = 0 \wedge a_i = 0] - \frac{1}{4}\right| &\leq \mu
\end{aligned}$$

when x is chosen uniformly from $\{x_0, x_1\}$ and y is chosen uniformly from $\{y_0, y_1\}$, then:

$$\begin{aligned}
\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_{i-1} = 1 \wedge a_i = 1] - \frac{1}{4}\right| &\leq 4\mu \\
\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_{i-1} = 1 \wedge a_i = 1] - \frac{1}{4}\right| &\leq 4\mu \\
\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_{i-1} = 0 \wedge a_i = 0] - \frac{1}{4}\right| &\leq 4\mu \\
\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_{i-1} = 0 \wedge a_i = 0] - \frac{1}{4}\right| &\leq 4\mu
\end{aligned}$$

when x and y are chosen in the same way.

We now prove the following theorem.

Theorem 5.1 *Let f be a two-party function containing an embedded XOR. Then any protocol securely computing f with complete fairness (assuming one exists) requires $\omega(\log n)$ rounds.*

Proof: Let Π be a protocol computing f with complete fairness using $r = r(n)$ rounds. Set $\mu = 1/\mathsf{poly}(n)$ for some polynomial to be fixed later. By correctness of Π, we have that for n sufficiently large

$$\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_r = 1 \wedge a_{r+1} = 1] - \frac{1}{4}\right| \leq \mu(n)$$

$$\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_r = 1 \wedge a_{r+1} = 1] - \frac{1}{4}\right| \leq \mu(n)$$
$$\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_r = 0 \wedge a_{r+1} = 0] - \frac{1}{4}\right| \leq \mu(n)$$
$$\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_r = 0 \wedge a_{r+1} = 0] - \frac{1}{4}\right| \leq \mu(n)$$

when x and y are chosen uniformly from $\{x_0, x_1\}$ and $\{y_0, y_1\}$, respectively. Taking n large enough so that Equations (18)–(21) also hold for $1 \leq i \leq r(n)$, we see that Claim 8 may be applied with $i = r$. Since the conclusion of Claim 8 is the assumption of Claim 9 and vice versa, the claims can be repeatedly applied r times, yielding:

$$\begin{aligned}
\left|\Pr[y = y_0 \wedge x = x_1 \wedge b_0 = 1 \wedge a_1 = 1] - \frac{1}{4}\right| &\leq 4^{2r(n)} \cdot \mu(n) \\
\left|\Pr[y = y_1 \wedge x = x_0 \wedge b_0 = 1 \wedge a_1 = 1] - \frac{1}{4}\right| &\leq 4^{2r(n)} \cdot \mu(n) \\
\left|\Pr[y = y_0 \wedge x = x_0 \wedge b_0 = 0 \wedge a_1 = 0] - \frac{1}{4}\right| &\leq 4^{2r(n)} \cdot \mu(n) \\
\left|\Pr[y = y_1 \wedge x = x_1 \wedge b_0 = 0 \wedge a_1 = 0] - \frac{1}{4}\right| &\leq 4^{2r(n)} \cdot \mu(n)\,.
\end{aligned}$$

If $r = \mathcal{O}(\log n)$, then $p(n) \stackrel{\text{def}}{=} 4^{2r(n)}$ is polynomial. Taking $\mu(n) = 1/16p(n)$ implies that, for n sufficiently large, A and B can both correctly compute (with probability at least 3/4) the value $f(x, y)$, for all $x \in \{x_0, x_1\}$ and $y \in \{y_0, y_1\}$, *without any interaction at all.* This is impossible, and so we conclude that $r = \omega(\log n)$. ■

References

[1] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[2] D. Beaver. Foundations of secure interactive computing. In *Advances in Cryptology — Crypto '91*, volume 576 of *LNCS*, pages 377–391. Springer, 1992.

[3] D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 468–473. IEEE, 1989.

[4] M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts. *IEEE Trans. Information Theory*, 36(1):40–46, 1990.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10. ACM Press, 1988.

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

[7] D. Boneh and M. Naor. Timed commitments. In *Advances in Cryptology — Crypto 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, 2000.

[8] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[9] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19. ACM Press, 1988.

[10] B. Chor and E. Kushilevitz. A zero-one law for Boolean privacy. *SIAM Journal on Discrete Math.*, 4:36–47, 1991.

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

[12] R. Cleve. Controlled gradual disclosure schemes for random bits and their applications. In *Advances in Cryptology — Crypto '89*, volume 435 of *LNCS*, pages 573–588. Springer, 1990.

[13] I. Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8(4):201–222, 1995.

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

[15] M. Franklin. *Complexity and Security of Distributed Protocols*. PhD thesis, Columbia University, 1993.

[16] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In *Advances in Cryptology — Crypto '87*, volume 293 of *LNCS*, pages 135–155. Springer, 1988.

[17] J. A. Garay, P. D. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. In *3rd Theory of Cryptography Conference — TCC 2006*, volume 3876 of *LNCS*, pages 404–428. Springer, 2006.

[18] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, Cambridge, UK, 2004.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[20] S. Goldwasser and L. A. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology — Crypto '90*, volume 537 of *LNCS*, pages 77–93. Springer, 1991.

[21] S. Gordon and J. Katz. Complete fairness in multi-party computation without an honest majority. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 19–35. Springer, 2009.

[22] S. D. Gordon and J. Katz. Partial fairness in secure two-party computation. In *Advances in Cryptology — Eurocrypt 2010*, volume 6110 of *LNCS*, pages 157–176. Springer, 2010.

[23] J. Kilian. Founding cryptography on oblivious transfer. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 20–31. ACM Press, 1988.

[24] J. Kilian. A general completeness theorem for two-party games. In *23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 553–560. ACM Press, 1991.

[25] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, 2003.

[26] M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 23–30. IEEE, 1983.

[27] S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology — Crypto '91*, volume 576 of *LNCS*, pages 392–404. Springer, 1992.

[28] T. Moran, M. Naor, and G. Segev. An optimally fair coin toss. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 1–18. Springer, 2009.

[29] B. Pinkas. Fair secure two-party computation. In *Advances in Cryptology — Eurocrypt 2003*, volume 2656 of *LNCS*, pages 87–105. Springer, 2003.

[30] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 73–85. ACM Press, 1989.

[31] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164. IEEE, 1982.

[32] A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.

A Complete Fairness for Other Functions using Protocol 2

A.1 Preliminary Discussion

Before specifying the more general functions for which Protocol 2 (cf. Figure 4) can be applied, we briefly discuss how we chose the value $\alpha = 1/5$ for the specific f of Section 4.2. This will provide some intuition that will be helpful in the section that follows. It should be clear that our entire discussion in this appendix assumes the specific simulation strategy described in the proof of Theorem 4.1. It may be the case that a different simulation strategy would allow for other values of α, or there may exist a different protocol altogether for computing f.

Consider the case of a malicious P_1 who aborts after receiving its iteration-i message, and let the parties' inputs be $x = x_1$, $y = y_1$ (note $f(x_1, y_1) = 0$). We use the notation as in the proof of

Claim 5, so that $\text{VIEW}^i_{\mathsf{hyb}}$ denotes the value a_i that P_1 reconstructs in iteration i and $\text{OUT}_{\mathsf{hyb}}$ denote the output of the honest P_2. The protocol itself ensures that in the hybrid world we have

$$\begin{aligned}&\Pr[\big(\text{VIEW}^i_{\mathsf{hyb}}(x_1,y_1),\text{OUT}_{\mathsf{hyb}}(x_1,y_1)\big)=(0,0)\mid i^*\geq i]\\ &\quad=\ \Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1,y_1)=0\mid i^*\geq i]\cdot\Pr[\text{OUT}_{\mathsf{hyb}}(x_1,y_1)=0\mid i^*\geq i],\end{aligned}$$

since $\text{OUT}_{\mathsf{hyb}}=b_{i-1}$ is independent of $\text{VIEW}^i_{\mathsf{hyb}}=a_i$ when $i^*\geq i$. We have

$$\Pr[\text{OUT}_{\mathsf{hyb}}(x_1,y_1)=0\mid i^*\geq i]=\Pr_{\hat{x}\leftarrow X}[f(\hat{x},y_1)=0]=1/3$$

and

$$\begin{aligned}&\Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1,y_1)=0\mid i^*\geq i]\\ &\quad=\ \alpha\cdot\Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1,y_1)=0\mid i^*=i]+(1-\alpha)\cdot\Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1,y_1)=0\mid i^*>i]\\ &\quad=\ \alpha+(1-\alpha)\cdot\Pr_{\hat{y}\leftarrow Y}[f(x_1,\hat{y})=0]\\ &\quad=\ \alpha+(1-\alpha)\cdot\frac{1}{2}\,,\end{aligned}$$

where the first equality holds since $\Pr[i^*=i\mid i^*\geq i]=\alpha$. Putting everything together we see that

$$\Pr[\big(\text{VIEW}^i_{\mathsf{hyb}}(x_1,y_1),\text{OUT}_{\mathsf{hyb}}(x_1,y_1)\big)=(0,0)\mid i^*\geq i]=\frac{1}{3}\cdot\left(\alpha+(1-\alpha)\cdot\frac{1}{2}\right).$$

In the ideal world, our simulation strategy ensures that, conditioned on $i^*\geq i$, the simulator $\mathcal{S}$ sends $x=x_1$ to the trusted party with probability α; when this occurs, the simulator will then set $\text{VIEW}^i_{\mathsf{ideal}}=a_i=f(x_1,y_1)=0$, and the honest party P_2 will output $f(x_1,y_1)=0$. Therefore, regardless of anything else the simulator might do,

$$\Pr[\big(\text{VIEW}^i_{\mathsf{ideal}}(x_1,y_1),\text{OUT}_{\mathsf{ideal}}(x_1,y_1)\big)=(0,0)\mid i^*\geq i]\ \geq\ \alpha.$$

If we want the ideal-world and hybrid-world distributions to be equal, then this requires

$$\alpha\leq\left(\alpha+(1-\alpha)\cdot\frac{1}{2}\right)\cdot\frac{1}{3}\,,$$

which is equivalent to requiring $\alpha\leq 1/5$. A similar argument applied to the other possible values for x,y shows that $\alpha\leq 1/5$ suffices for all of them. Setting $\alpha=1/5$ minimizes the number of rounds of the protocol.

Having fixed the value of α, we now explain how we determined the simulator's actions (for a malicious P_1) in step 8(b). We begin by introducing some notation that we will also use in the following section.

Define $p_{x_i}\stackrel{\text{def}}{=}\Pr_{\hat{y}\leftarrow Y}[f(x_i,\hat{y})=1]$ and, similarly, define $p_{y_i}\stackrel{\text{def}}{=}\Pr_{\hat{x}\leftarrow X}[f(\hat{x},y_i)=1]$. Let x' be as in the description of $\mathcal{S}$ in the proof of Claim 5. If $\mathcal{A}$ aborts in round $i<i^*$ after receiving the bit a_i, then we denote the event that $\mathcal{S}$ sends x_i to the ideal functionality computing f by $X^{(a_i)}_{x'\to x_i}$. Using this notation, we have from step 8(b) of $\mathcal{S}$ that:

$$\Pr[X^{(1)}_{x_1\to x_1}]=\frac{1}{3}\qquad\Pr[X^{(1)}_{x_1\to x_2}]=\frac{1}{2}\qquad\Pr[X^{(1)}_{x_1\to x_3}]=\frac{1}{6}\,.$$

Consider once again the case $x = x_1$ and $y = y_1$. In the hybrid world, by construction of Protocol 2, we have

$$\begin{aligned}
&\Pr[\big(\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1), \text{OUT}_{\mathsf{hyb}}(x_1, y_1)\big) = (1, 1) \mid i^* \geq i] \\
&\quad = \Pr[\text{VIEW}^i_{\mathsf{hyb}}(x_1, y_1) = 1 \mid i^* \geq i] \cdot \Pr[\text{OUT}_{\mathsf{hyb}}(x_1, y_1) = 1 \mid i^* \geq i] \\
&\quad = (1 - \alpha) \cdot p_{x_1} \cdot p_{y_1}.
\end{aligned}$$

(Note that if $i^* = i$, which occurs with probability α, then $a_i = f(x_1, y_1) = 0$.) Because of the way $\mathcal{S}$ is defined, in the ideal world we have

$$\begin{aligned}
&\Pr[\big(\text{VIEW}^i_{\mathsf{ideal}}(x_1, y_1), \text{OUT}_{\mathsf{ideal}}(x_1, y_1)\big) = (1, 1) \mid i^* \geq i] \\
&\quad = \Pr[\text{VIEW}^i_{\mathsf{ideal}}(x_1, y_1) = 1 \mid i^* \geq i] \cdot \Pr[\text{OUT}_{\mathsf{ideal}}(x_1, y_1) = 1 \mid \text{VIEW}^i_{\mathsf{ideal}}(x_1, y_1) = 1 \bigwedge i^* \geq i] \\
&\quad = (1 - \alpha) \cdot p_{x_1} \cdot \left(\Pr[X^{(1)}_{x_1 \to x_2}] + \Pr[X^{(1)}_{x_1 \to x_3}]\right).
\end{aligned}$$

If we want these to be equal, this requires $\Pr[X^{(1)}_{x_1 \to x_2}] + \Pr[X^{(1)}_{x_1 \to x_3}] = p_{y_1} = \frac{2}{3}$.

Proceeding similarly for the case when $x = x_1$ and $y = y_2$ and looking at the probability that $a_i = 0$ and the output of P_2 is 1, we derive

$$\Pr[X^{(1)}_{x_1 \to x_1}] + \Pr[X^{(1)}_{x_1 \to x_3}] = \frac{\alpha \cdot (p_{y_2} - 1)}{(1 - \alpha)(1 - p_{x_1})} + p_{y_2} = \frac{1}{2}.$$

Combining the above two with the constraint that $\Pr[X^{(1)}_{x_1 \to x_1}] + \Pr[X^{(1)}_{x_1 \to x_2}] + \Pr[X^{(1)}_{x_1 \to x_3}] = 1$ we obtain the unique feasible values used in step 8(b) of $\mathcal{S}$ (for the case $x = x_1$). The case of $x = x_2$ follows via a similar analysis.

Looking at the problem more generally, we observe that for certain functions f (e.g., the boolean XOR function), the problem is over-constrained and *no* feasible solution exists (regardless of the choice of α). In the following section we will argue that our protocol can be applied to any function f for which the above constraints can be satisfied for all possible inputs x, y.

A.2 Characterization of Functions for which Protocol 2 Applies

In this section we characterize a class of functions that can be securely computed with complete fairness using Protocol 2. The proof is a generalization of the proof from Section 4.2.

Notation. We assume a single-output, boolean function $f : X \times Y \to \{0, 1\}$ defined over a finite domain, where $X = \{x_1, \ldots x_\ell\}$ and $Y = \{y_1, \ldots, y_m\}$. We let M_f denote the $\ell \times m$ matrix whose entry at position (i, j) is $f(x_i, y_j)$, and let $\mathbf{v}_y$ denote the column of M_f corresponding to the input y of P_2. For every input $x \in X$ of player P_1 we define

$$p_x \stackrel{\text{def}}{=} \Pr_{\hat{y} \leftarrow Y}[f(x, \hat{y}) = 1],$$

where $\hat{y}$ is chosen uniformly from the domain Y of player P_2. Equivalently, $p_x = \frac{\sum_{y \in Y} f(x,y)}{m}$. We define p_y, for $y \in Y$, symmetrically. In addition, let $\bar{p}_x \stackrel{\text{def}}{=} 1 - p_x$ and $\bar{p}_y \stackrel{\text{def}}{=} 1 - p_y$.

We set α as follows:

$$\alpha \stackrel{\text{def}}{=} \min_{(i,j)} \left\{ \frac{\big|1 - f(x_i, y_j) - p_{x_i}\big| \cdot \big|1 - f(x_i, y_j) - p_{y_j}\big|}{\big|1 - f(x_i, y_j) - p_{x_i}\big| \cdot \big|1 - f(x_i, y_j) - p_{y_j}\big| + \big|f(x_i, y_j) - p_{y_j}\big|} \right\}, \tag{31}$$

where the minimum is taken over $1 \leq i \leq \ell$ and $1 \leq j \leq m$. By simple calculation, one can show that $0 < \alpha \leq 1$ and, in fact, $\alpha < 1$ unless f is a constant function (in which case completely fair computation of f is trivial). Using this value of α we define, for $x \in X$, the m-dimensional row vector $\vec{C}_x^{(0)}$, indexed by $y \in Y$, as follows:

$$\vec{C}_x^{(0)}(y) \stackrel{\text{def}}{=} \begin{cases} p_y & \text{if } f(x,y) = 1 \\ \frac{\alpha \cdot p_y}{(1-\alpha)\cdot \bar{p}_x} + p_y & \text{if } f(x,y) = 0 \end{cases}.$$

Similarly, we define $\vec{C}_x^{(1)}$ via:

$$\vec{C}_x^{(1)}(y) \stackrel{\text{def}}{=} \begin{cases} \frac{\alpha \cdot (p_y - 1)}{(1-\alpha)\cdot p_x} + p_y & \text{if } f(x,y) = 1 \\ p_y & \text{if } f(x,y) = 0 \end{cases}$$

(The denominators, above, are never 0.)

A row vector $(p_1, \ldots, p_\ell)$ of real numbers is a *probability vector* if $0 \leq p_i \leq 1$ for all i, and $\sum_i p_i = 1$. We are now ready to prove the following:

Theorem A.1 *Let f be a single-output, boolean function, and let M_f and $\vec{C}_{x_i}^{(b)}$ be as defined above. If for all $b \in \{0,1\}$ and $x \in X$ there exists a probability vector $\vec{X}_x^{(b)} = (p_1, \ldots, p_\ell)$ such that*

$$\vec{X}_x^{(b)} \cdot M_f = \vec{C}_x^{(b)},$$

then there exists a protocol that securely computes f with completes fairness.

Proof: We take Protocol 2 with α computed as in Eq. (31). Simulation for a corrupted P_2 follows exactly along the lines of the proof of Claim 4; recall that the simulator in that case did not rely on any specific properties of the function f or the value of α. We therefore focus our attention on the case when the adversary $\mathcal{A}$ corrupts P_1. In this case, our simulator $\mathcal{S}$ is almost identical to the simulator described in the proof of Claim 5 (except, of course, that it uses the appropriate value of α); the only significant change is how we deal with an abort in iteration $i < i^*$ (this corresponds to step 8(b) in the simulator from the proof of Claim 5). For completeness, we describe the modified simulator in its entirety, although we once again ignore the presence of the MAC-tags and keys for simplicity.

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the input x', the auxiliary input, and the security parameter n. The simulator also chooses $\hat{x} \in X$ uniformly at random.
2. $\mathcal{S}$ receives the input x of $\mathcal{A}$ to the computation of the functionality $\mathsf{ShareGen}'$.
   (a) If $x \notin X$, then $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from the computation of $\mathsf{ShareGen}'$, sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.
   (b) Otherwise, if the input is some $x \in X$, then $\mathcal{S}$ chooses uniformly distributed shares $a_1^{(1)}, \ldots, a_m^{(1)}$ and $b_1^{(1)}, \ldots, b_m^{(1)}$. Then, $\mathcal{S}$ gives these shares to $\mathcal{A}$ as its output from the computation of $\mathsf{ShareGen}'$.
3. If $\mathcal{A}$ sends abort to the trusted party computing $\mathsf{ShareGen}'$, then $\mathcal{S}$ sends $\hat{x}$ to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise (i.e., if $\mathcal{A}$ sends $\mathsf{continue}$), $\mathcal{S}$ proceeds as below.

4. Choose i^* according to a geometric distribution with parameter α.

5. For $i = 1$ to $i^* - 1$:

   (a) $\mathcal{S}$ chooses $\hat{y} \in Y$ uniformly at random, computes $a_i = f(x, \hat{y})$, and sets $a_i^{(2)} = a_i^{(1)} \oplus a_i$. It gives $a_i^{(2)}$ to $\mathcal{A}$.

   (b) If $\mathcal{A}$ aborts, then $\mathcal{S}$ chooses x' according to the distribution defined by[7] $\vec{X}_x^{(a_i)}$, and sends x' to the trusted party computing f. It then outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.

6. For $i = i^*$ to m:

   (a) If $i = i^*$ then $\mathcal{S}$ sends x to the trusted party computing f and receives $z = f(x, y)$.

   (b) $\mathcal{S}$ sets $a_i^{(2)} = a_i^{(1)} \oplus z$ and gives $a_i^{(2)}$ to $\mathcal{A}$.

   (c) If $\mathcal{A}$ aborts, then $\mathcal{S}$ then outputs whatever $\mathcal{A}$ outputs, and halts. If $\mathcal{A}$ does not abort, then $\mathcal{S}$ proceeds.

7. If $\mathcal{S}$ has not yet halted, and has not yet sent anything to the trusted party computing f (this can only happen if $i^* > m$ and $\mathcal{A}$ has not aborted), then it sends $\hat{x}$ to the trusted party. Then $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

(The simulator constructed in Claim 5 branched depending on the value of x, but this was only a simplification due to the fact that the input x_3, there, completely determined the output. In general there need not be any such input.)

We borrow the same notation as in our proof of Claim 5. Examining that proof, we see that the proof here will proceed identically up to the point where we need to show that, for all inputs x, y and all $a, b \in \{0, 1\}$:

$$\Pr\left[\left(\text{VIEW}_{\mathsf{hyb}}^i, \text{OUT}_{\mathsf{hyb}}\right) = (a, b) \mid i^* \geq i\right] = \Pr\left[\left(\text{VIEW}_{\mathsf{ideal}}^i, \text{OUT}_{\mathsf{ideal}}\right) = (a, b) \mid i^* \geq i\right] \tag{32}$$

(This is Eq. (7) there. As was done there, we suppress explicit mention of the inputs when the notation becomes cumbersome.) We now fix arbitrary x, y and show that the above holds. We consider two sub-cases depending on the value of $f(x, y)$.

Case 1: x and y are such that $f(x, y) = 0$. In the hybrid world, when $\mathcal{A}$ aborts after receiving its iteration-i message, then P_2 outputs $\text{OUT}_{\mathsf{hyb}} = b_{i-1}$ and the value of $\text{VIEW}_{\mathsf{hyb}}^i = a_i$ is independent of the value of b_{i-1}. By definition of the protocol, we have

$$\Pr[b_{i-1} = 0 \mid i^* \geq i] = \bar{p}_y \quad \text{and} \quad \Pr[b_{i-1} = 1 \mid i^* \geq i] = p_y,$$

since $b_{i-1} = f(\hat{x}, y)$ for $\hat{x}$ chosen uniformly from X. As for a_i, we have

$$\Pr\left[a_i = 0 \mid i^* \geq i\right] = \alpha + (1 - \alpha) \cdot \bar{p}_x \quad \text{and} \quad \Pr\left[a_i = 1 \mid i^* \geq i\right] = (1 - \alpha) \cdot p_x\,.$$

[7] This is understood in the natural way; i.e., x_j is chosen with probability $\vec{X}_x^{(a_i)}(j)$.

Since b_{i-1} and a_i are independent, we conclude that

$$\Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (a,b) \mid i^* \geq i\right] = \begin{cases} (\alpha + (1-\alpha)\cdot\bar{p}_x)\cdot\bar{p}_y & (a,b) = (0,0) \\ (\alpha + (1-\alpha)\cdot\bar{p}_x)\cdot p_y & (a,b) = (0,1) \\ (1-\alpha)\cdot p_x \cdot \bar{p}_y & (a,b) = (1,0) \\ (1-\alpha)\cdot p_x \cdot p_y & (a,b) = (1,1) \end{cases}$$

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x,y) = 0$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 0] = \bar{p}_x$. The value of $\text{OUT}_{\mathsf{ideal}}$ is now dependent on the value of a_i (cf. step 5(b) of the simulator described in this section); specifically, we have:

$$\begin{aligned} &\Pr[\text{OUT}_{\mathsf{ideal}}(x,y) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &= \Pr[\mathcal{S} \text{ sends } x' \text{ to the trusted party s.t. } f(x',y) = 0 \mid a_i = 0 \bigwedge i^* > i] \\ &= \sum_{\bar{x}:\, f(\bar{x},y)=0} \Pr_{x' \leftarrow \vec{X}^{(0)}_x}[x' = \bar{x}] \end{aligned}$$

and, in the general case,

$$\Pr[\text{OUT}_{\mathsf{ideal}}(x,y) = b \mid a_i = a \bigwedge i^* > i] = \sum_{\bar{x}:\, f(\bar{x},y)=b} \Pr_{x' \leftarrow \vec{X}^{(a)}_x}[x' = \bar{x}].$$

We therefore have, for example,

$$\begin{aligned} \Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (0,0) \mid i^* \geq i\right] &= \alpha + (1-\alpha)\cdot\bar{p}_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=0} \Pr_{x' \leftarrow \vec{X}^{(0)}_x}[x' = \bar{x}] \\ &= \alpha + (1-\alpha)\cdot\bar{p}_x \cdot \left(1 - \vec{X}^{(0)}_x \cdot \mathbf{v}_y\right) \\ &= \alpha + (1-\alpha)\cdot\bar{p}_x \cdot \left(1 - \vec{C}^{(0)}_x(y)\right) \\ &= \alpha + (1-\alpha)\cdot\bar{p}_x \cdot \left(1 - \frac{\alpha\cdot p_y}{(1-\alpha)\cdot\bar{p}_x} - p_y\right) \\ &= (\alpha + (1-\alpha)\cdot\bar{p}_x)\cdot\bar{p}_y, \end{aligned}$$

(The second equality uses the definitions of $\vec{X}^{(0)}_x$ and $\mathbf{v}_y$; the third equality uses the assumption, from the theorem, that $\vec{X}^{(0)}_x \cdot \mathbf{v}_y = \vec{C}^{(0)}_x(y)$. We then use the definition of $\vec{C}^{(0)}_x(y)$ and re-arrange using algebra.) This is equal to the associated probability in the hybrid world, as computed above.

For completeness, we include the calculations for the remaining cases:

$$\begin{aligned} &\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (0,1) \mid i^* \geq i\right] \\ &= (1-\alpha)\cdot\bar{p}_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=1} \Pr_{x' \leftarrow \vec{X}^{(0)}_x}[x' = \bar{x}] \\ &= (1-\alpha)\cdot\bar{p}_x \cdot \left(\vec{X}^{(0)}_x \cdot \mathbf{v}_y\right) \\ &= (1-\alpha)\cdot\bar{p}_x \cdot \vec{C}^{(0)}_x(y) \\ &= (1-\alpha)\cdot\bar{p}_x \cdot \left(\frac{\alpha\cdot p_y}{(1-\alpha)\cdot\bar{p}_x} + p_y\right) \\ &= (\alpha + (1-\alpha)\cdot\bar{p}_x)\cdot p_y \quad = \quad \Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (0,1) \mid i^* \geq i\right]. \end{aligned}$$

$$
\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (1,0) \mid i^* \geq i\right] \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=0} \Pr_{x' \leftarrow \vec{X}^{(1)}_x}[x' = \bar{x}] \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \left(1 - \vec{X}^{(1)}_x \cdot \mathbf{v}_y\right) \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \left(1 - \vec{C}^{(1)}_x(y)\right) \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot (1-p_y) \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \bar{p}_y \quad = \quad \Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (1,0) \mid i^* \geq i\right].
\end{aligned}
$$

$$
\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (1,1) \mid i^* \geq i\right] \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=1} \Pr_{x' \leftarrow \vec{X}^{(1)}_x}[x' = \bar{x}] \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \left(\vec{X}^{(1)}_x \cdot \mathbf{v}_y\right) \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot \vec{C}^{(1)}_x(y) \\
&\quad = \quad (1-\alpha)\cdot p_x \cdot p_y \quad = \quad \Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (1,1) \mid i^* \geq i\right].
\end{aligned}
$$

Equality holds, in all cases, between the corresponding probabilities in the ideal and hybrid worlds. We thus conclude that Eq. (32) holds for all x, y with $f(x,y) = 0$.

Case 2: x and y are such that $f(x,y) = 1$. We provide the calculations with limited discussion. In the hybrid world, we have

$$
\Pr\left[\left(\text{VIEW}^i_{\mathsf{hyb}}(x,y), \text{OUT}_{\mathsf{hyb}}(x,y)\right) = (a,b) \mid i^* \geq i\right] = \begin{cases} ((1-\alpha)\cdot \bar{p}_x)\cdot \bar{p}_y & (a,b) = (0,0) \\ ((1-\alpha)\cdot \bar{p}_x)\cdot p_y & (a,b) = (0,1) \\ (\alpha + (1-\alpha)\cdot p_x)\cdot \bar{p}_y & (a,b) = (1,0) \\ (\alpha + (1-\alpha)\cdot p_x)\cdot p_y & (a,b) = (1,1) \end{cases}
$$

In the ideal world, if $i^* = i$ then $\text{OUT}_{\mathsf{ideal}} = \text{VIEW}^i_{\mathsf{ideal}} = f(x,y) = 1$. If $i^* > i$, then the distribution of $\text{VIEW}^i_{\mathsf{ideal}} = a_i$ is given by $\Pr[a_i = 0] = \bar{p}_x$, and the value of $\text{OUT}_{\mathsf{ideal}}$ is now dependent on the value of a_i. Working out the details, we have:

$$
\begin{aligned}
&\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (0,0) \mid i^* \geq i\right] \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \sum_{\bar{x}:\, f(\bar{x},y)=0} \Pr_{x' \leftarrow \vec{X}^{(0)}_x}[x' = \bar{x}] \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \left(1 - \vec{X}^{(0)}_x \cdot \mathbf{v}_y\right) \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \left(1 - \vec{C}^{(0)}_x(y)\right) \\
&\quad = \quad (1-\alpha)\cdot \bar{p}_x \cdot \bar{p}_y.
\end{aligned}
$$

$$
\Pr\left[\left(\text{VIEW}^i_{\mathsf{ideal}}(x,y), \text{OUT}_{\mathsf{ideal}}(x,y)\right) = (0,1) \mid i^* \geq i\right]
$$

$$
\begin{aligned}
&= (1-\alpha)\cdot\bar{p}_x\cdot\sum_{\bar{x}:\ f(\bar{x},y)=1}\Pr_{x'\leftarrow\vec{X}_x^{(0)}}[x'=\bar{x}]\\
&= (1-\alpha)\cdot\bar{p}_x\cdot\left(\vec{X}_x^{(0)}\cdot\mathbf{v}_y\right)\\
&= (1-\alpha)\cdot\bar{p}_x\cdot\left(\vec{C}_x^{(0)}(y)\right)\\
&= (1-\alpha)\cdot\bar{p}_x\cdot p_y.
\end{aligned}
$$

$$
\begin{aligned}
&\Pr\left[\left(\textsc{view}^i_{\mathsf{ideal}}(x,y),\textsc{out}_{\mathsf{ideal}}(x,y)\right)=(1,0)\mid i^*\geq i\right]\\
&\quad= (1-\alpha)\cdot p_x\cdot\sum_{\bar{x}:\ f(\bar{x},y)=0}\Pr_{x'\leftarrow\vec{X}_x^{(1)}}[x'=\bar{x}]\\
&\quad= (1-\alpha)\cdot p_x\cdot\left(1-\vec{X}_x^{(1)}\cdot\mathbf{v}_y\right)\\
&\quad= (1-\alpha)\cdot p_x\cdot\left(1-\vec{C}_x^{(1)}(y)\right)\\
&\quad= (1-\alpha)\cdot p_x\cdot\left(1-\frac{\alpha\cdot(p_y-1)}{(1-\alpha)\cdot p_x}-p_y\right)\\
&\quad= (\alpha+(1-\alpha)\cdot p_x)\cdot\bar{p}_y.
\end{aligned}
$$

$$
\begin{aligned}
&\Pr\left[\left(\textsc{view}^i_{\mathsf{ideal}}(x,y),\textsc{out}_{\mathsf{ideal}}(x,y)\right)=(1,1)\mid i^*\geq i\right]\\
&\quad= \alpha+(1-\alpha)\cdot p_x\cdot\sum_{\bar{x}:\ f(\bar{x},y)=1}\Pr_{x'\leftarrow\vec{X}_x^{(1)}}[x'=\bar{x}]\\
&\quad= \alpha+(1-\alpha)\cdot p_x\cdot\left(\vec{X}_x^{(1)}\cdot\mathbf{v}_y\right)\\
&\quad= \alpha+(1-\alpha)\cdot p_x\cdot\left(\vec{C}_x^{(1)}(y)\right)\\
&\quad= \alpha+(1-\alpha)\cdot p_x\cdot\left(\frac{\alpha\cdot(p_y-1)}{(1-\alpha)\cdot p_x}+p_y\right)\\
&\quad= (\alpha+(1-\alpha)\cdot p_x)\cdot p_y.
\end{aligned}
$$

Once again, equality holds between the corresponding probabilities in the ideal and hybrid worlds in all cases. This concludes the proof of the theorem. ■