Open access • Proceedings Article • DOI:10.1145/1374376.1374436

Complete fairness in secure two-party computation — Source link

Dov S. Gordon, Hazay Carmit, Jonathan Katz, Yehuda Lindell

Institutions: University of Maryland, College Park, Bar-Ilan University

Published on: 17 May 2008 - Symposium on the Theory of Computing

Topics: Secure two-party computation and Secure multi-party computation

Complete Fairness in Secure Two-Party Computation[*]

S. Dov Gordon
Dept. of Computer Science
University of Maryland
gordon@cs.umd.edu

Carmit Hazay
Dept. of Computer Science
Bar-Ilan University
harelc@cs.biu.ac.il

Jonathan Katz
Dept. of Computer Science
University of Maryland
jkatz@cs.umd.edu

Yehuda Lindell
Dept. of Computer Science
Bar-Ilan University
lindell@cs.biu.ac.il

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 that if either party receives its output, then the other party does too. Cleve (STOC 1986) showed that complete fairness cannot be achieved *in general* in the two-party setting; specifically, he showed (essentially) that it is impossible to compute Boolean XOR with complete fairness. Since his work, the accepted folklore has been that *nothing* non-trivial can be computed with complete fairness, and the question of complete fairness in secure two-party computation has been treated as closed since the late '80s.

In this paper, we demonstrate that this widely held folklore belief is *false* by showing completely-fair secure protocols for various non-trivial two-party functions including Boolean AND/OR as well as Yao's "millionaires' problem". Surprisingly, we show that it is even possible to construct completely-fair protocols for certain functions containing an "embedded XOR", although in this case we also prove a lower bound showing that a super-logarithmic number of rounds are necessary. Our results demonstrate that the question of completely-fair secure computation without an honest majority is far from closed.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General; F.m [**Theory of Computation**]: Miscellaneous

General Terms

Theory, Security

Keywords

Fairness, cryptography, secure computation

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 [10] 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 ruled 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 [15]) 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

[*] This work was supported by US-Israel Binational Science Foundation grant #2004240. The third author was also supported by NSF CAREER award #0447075.

STOC'08, May 17–20, 2008, Victoria, British Columbia, Canada.
Copyright 2008 ACM 978-1-60558-047-0/08/05 ...$5.00.

[1] Throughout this discussion, we refer to the notion of *complete* fairness where honest parties receive their entire output if the adversary receives its own output. Notions of *partial* fairness have also been considered; see below.

with abort", since the adversary in this case may abort the protocol when it receives its output).

Protocols achieving security with complete fairness when a majority of parties are honest are known for arbitrary functionalities, [5, 8, 2, 26] (assuming a broadcast channel), as are protocols achieving security with abort for any number of corrupted parties [16, 15] (under suitable cryptographic assumptions, and continuing to assume a broadcast channel). Since the work of Cleve, however, there has been no progress towards a better understanding of complete fairness *without* an honest majority. That is, no impossibility results have been shown for other functionalities, nor have any completely-fair protocols been constructed. (Feasibility results for authenticated broadcast [24, 11] could be viewed as an exception, but broadcast is anyway trivial in the two-party setting.) In short, the question has been treated as closed for over two decades.

Our results. Cleve's work shows that *certain functions* cannot be computed with complete fairness without an honest majority. As mentioned above, however, the prevailing "folklore" interpretation of this result is 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 functions *can* be computed with complete fairness in the two-party case. We demonstrate two protocols for two classes of Boolean functions. Our first protocol implies, among other things:

Theorem *(Under suitable assumptions) there is a constant-round protocol for securely computing the two-party Boolean OR and AND functions with complete fairness.*

Note that OR and AND are non-trivial: they cannot be computed with information-theoretic privacy [9] and are complete for secure computation with abort [19, 20, 4].

Our first protocol also yields completely-fair computation of Yao's *millionaires' problem* (i.e., the "greater than" function) for polynomial-size domains. More generally, our first protocol can be applied to obtain completely-fair secure computation of any function (over polynomial-size domains) that does not contain an *embedded XOR*, i.e., inputs x_0, x_1, y_0, y_1 such that $f(x_i, y_j) = i \oplus j$.

Given the class of functions for which our first protocol applies, and the fact that Cleve's result rules out completely-fair computation of XOR, a natural conjecture might be that any function containing an embedded XOR cannot be computed with complete fairness. Our second protocol shows that this conjecture is false:

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

Our second protocol is more general than our first (in the sense that any function that can be computed with complete fairness using our first protocol can also be computed with complete fairness using our second protocol), but we consider both protocols to be of independent interest. Furthermore, the round complexity of our protocols is incomparable: the number of rounds in the first protocol is linear in the size of the domain of the function being computed, and in the second protocol it is super-logarithmic in the security parameter. We show that the latter is inherent for functions containing an embedded XOR:

Theorem *Let f be a function with 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 [10], but proceeds differently: Cleve only considered *bias* whereas we must jointly consider both *bias* and *privacy* (since, for certain functions containing an embedded XOR, it may be possible for an adversary to bias the output even in the ideal world). This makes the proof considerably more complex.

Related work. Questions of fairness, and possible relaxations thereof, have been studied since the early days of secure computation [27, 13, 3, 17]. One of the main techniques to result from this line of work is "gradual release" [22, 12, 3, 17, 6, 25, 14] which, informally, guarantees that at any point in the protocol both the adversary and the honest parties can obtain their output by investing a "similar" amount of work. Gradual release is applicable to general functionalities, but achieves only *partial* fairness. In contrast, we are interested in obtaining *complete* fairness, exactly as in the case of an honest majority. To the best of our knowledge, ours are the first positive results for completely-fair secure computation *without* an honest majority.

2. DEFINITIONS

We denote the security parameter by n. We use the standard definitions of secure multi-party computation (see [15]) for both "security with complete fairness" and "security with abort". We stress, however, that we will be considering security *with complete fairness* even though we are in the two-party setting (this is not standard). Our protocols will rely on information-theoretic MACs, defined and constructed in the standard way.

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 outputs of each party. 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. If X_n and Y_n are of size 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 (and also the collection $\mathcal{F}$) as a *function.*

3. FAIRNESS FOR THE MILLIONAIRES' PROBLEM (AND MORE)

In this section, we describe a protocol for securely computing the classic millionaires' problem (and related functionalities) with complete fairness. 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 $\mathcal{F} = \{f_n\}_{n \in \mathbb{N}}$ denote a function of the above form, where $m = m(n)$ is the size of the domain of each input. (In

ShareGen

Inputs: Let the inputs to $\mathsf{ShareGen}$ be x_i and y_j with $1 \leq i, j \leq m$. (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 $a_i = b_j = f(x_i, y_j)$, where x_i and y_j are the parties' inputs.
   - 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 $1 \leq i \leq m$, 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 $1 \leq i \leq m$, 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.

general, the domains need not be the same size; we make this assumption for simplicity.) For some fixed value of n, let $X = \{x_1, \ldots, x_m\}$ denote the valid inputs for the first party and let $Y = \{y_1, \ldots, y_m\}$ denote the valid inputs for the second party. (We leave the security parameter implicit when this will not cause confusion.) By suitably ordering these elements, we may write f_n as follows:

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

Viewed in this way, f_n is exactly the millionaires' problem or, equivalently, the "greater than" function. In the full version of this work, we show that *any* function with a polynomial-size domain that does not have an embedded XOR is essentially equivalent to the function f_n defined above (in the sense that such a function can be transformed into one of this type without any loss). The remainder of this section is thus devoted to a proof of the following theorem (where we assume that the function is already transformed to the above form):

THEOREM 1. *Let $m = \mathsf{poly}(n)$ and let $\mathcal{F} = \{f_n\}$ be any function with domain size m that does not have an embedded XOR. Then, assuming the existence of enhanced trapdoor permutations, there exists a protocol that securely computes $\mathcal{F} = \{f_n\}$ with complete fairness.*

Our protocol requires $\Theta(m)$ rounds, which explains why we require $m = \mathsf{poly}(n)$. When $m = 2$, we obtain a protocol for computing Boolean AND (and, by symmetry, OR) with complete fairness. For the remainder of this section, we write f in place of f_n.

3.1 The Protocol

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 value $f(x_i, y_j)$ in iteration i, and P_2 learns the output in iteration j. (I.e., 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 $f(x_k, y_j)$.)

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. There are two possibilities: P_1 either aborts before iteration i, or after iteration 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 clearly achieved. In the second case, P_1 obtains the output $f(x_i, y)$ in iteration i and then aborts in some iteration $k \geq i$. 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, P_2 will (locally) compute and output $f(x_k, y) = f(x_k, y_j)$. We claim that $f(x_k, y_j) = f(x_i, y_j)$, and so the output of P_2 is equal to the output obtained by P_1 (and fairness is achieved). Here we rely on the specific properties of f: since $j \geq k \geq i$ we have $f(x_i, y_j) = 0 = f(x_k, y_j)$. This is the key observation that enables us to obtain fairness for this function.

We formalize the above 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. Let $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code (MAC). We assume it is an m-time MAC with information-theoretic security, though a computationally-secure MAC would also suffice.

We will rely on a sub-protocol for securely computing a randomized functionality $\mathsf{ShareGen}$ defined in Figure 1. In our protocol, the parties will compute $\mathsf{ShareGen}$ as a result of which P_1 obtains shares $a_1^{(1)}, b_1^{(1)}, a_2^{(1)}, b_2^{(1)}, \ldots$ and

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 $\bot$ 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 $\bot$, it outputs $f(x_1, y)$ and halts. Otherwise, the parties proceed.
   - (c) Denote the output of P_1 from π by $a_1^{(1)}, \dots, a_m^{(1)}$, $(b_1^{(1)}, t_1^b), \dots, (b_m^{(1)}, t_m^b)$, and k_a.
   - (d) Denote the output of P_2 from π by $(a_1^{(2)}, t_1^a), \dots, (a_m^{(2)}, t_m^a)$, $b_1^{(2)}, \dots, b_m^{(2)}$, and k_b.
2. **For $\boldsymbol{i = 1, \dots, m}$ do:**

   $\boldsymbol{P_2}$ sends the next share to $\boldsymbol{P_1}$:
   - (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).

   $\boldsymbol{P_1}$ sends the next share to $\boldsymbol{P_2}$:
   - (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.

P_2 obtains shares $a_1^{(2)}, b_1^{(2)}, a_2^{(2)}, b_2^{(2)}, \dots$. (The functionality $\mathsf{ShareGen}$ 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 it. If 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 sends $a_i^{(2)}$ to P_1, and then P_1 sends $b_i^{(1)}$ to P_2.

Let π be a constant-round protocol that securely computes $\mathsf{ShareGen}$ *with abort*. Such a protocol can be constructed using standard tools for secure two-party computation, assuming the existence of enhanced trapdoor permutations [21]. Our protocol for computing f is given in Figure 2.

Theorem 2. *If* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is an i.t.-secure, m-time MAC, and π securely computes* $\mathsf{ShareGen}$ *with abort, then Protocol 1 securely computes $\{f_n\}$ with complete fairness.*

Proof. Let Π denote Protocol 1. We analyze Π in a hybrid model where the parties have access to a trusted party computing $\mathsf{ShareGen}$. (Note: since π securely computes $\mathsf{ShareGen}$ with abort, the adversary may abort the trusted party computing $\mathsf{ShareGen}$ before it sends output 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. This suffices to prove the theorem [7]. We first analyze the case when P_1 is corrupted:

Claim 1. *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*

$$\left\{\text{IDEAL}_{f,\mathcal{S}}(x,y,n)\right\}_{x\in X, y\in Y, n\in\mathbb{N}} \overset{\mathrm{s}}{\equiv} \left\{\text{HYBRID}^{\mathsf{ShareGen}}_{\Pi,\mathcal{A}}(x,y,n)\right\}_{x\in X, y\in Y, n\in\mathbb{N}}.$$

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, and the security parameter n.
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 $\bot$ to $\mathcal{A}$ as its output from the computation of $\mathsf{ShareGen}$, sends x_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts.
   - (b) If $x' \in X$, then $\mathcal{S}$ chooses uniformly-distributed shares $a_1^{(1)}, \dots, a_m^{(1)}$ and $b_1^{(1)}, \dots, b_m^{(1)}$. 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 $a_1^{(1)}, \dots, a_m^{(1)}$, $(b_1^{(1)}, t_1^b), \dots, (b_m^{(1)}, t_m^b)$, and k_a as its output from $\mathsf{ShareGen}$.

3. If $\mathcal{A}$ sends abort to the trusted third party computing ShareGen, then $\mathcal{S}$ sends x_1 to the trusted party computing f, outputs whatever $\mathcal{A}$ outputs, and halts. Otherwise, $\mathcal{S}$ 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 $j = 1, \ldots, i-1$, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ chooses $a_j^{(2)}$ such that $a_j^{(1)} \oplus a_j^{(2)} = \textsc{null}$, 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. Then:

      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 $(\hat{b}_i^{(1)}, \hat{t}_i^b)$ in this iteration. 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 $j = i+1, \ldots, m$, simulator $\mathcal{S}$ works as follows:
   (a) $\mathcal{S}$ chooses $a_j^{(2)}$ such that $a_j^{(1)} \oplus a_j^{(2)} = \textsc{null}$, 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 current iteration. 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 outputs whatever $\mathcal{A}$ outputs and halts.

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). It is straightforward to prove that this is the case with all but negligible probability by relying on the security of the MAC. 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.

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 its view 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 1. 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 the above 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 iteration j (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 Protocol 1, 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 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 $\mathcal{A}$ did not send 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, with $i \leq \zeta < \ell$. By the specification of Protocol 1, party P_2 would output $f(x_\zeta, y) = f(x_\zeta, y_\ell)$. However, since $\zeta < \ell$ 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. ■

A proof for the case when P_2 is corrupted is similar to the above, and is given in the full version of this work. ■

ShareGen′

Inputs: Let the inputs to 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 $1 \leq i \leq m$, 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 $1 \leq i \leq m$, 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 ShareGen′, parameterized by a value α.

4. FAIRNESS FOR SOME FUNCTIONS WITH AN EMBEDDED XOR

In this section we present a protocol for completely-fair secure computation of a more general class of functions, including certain functions with an embedded XOR. For simplicity, we consider functions $\mathcal{F}$ defined over a finite domain, i.e., $\mathcal{F} = \{f_n\}$ where $f_n = f$ for all n (however, our result generalizes to the case of functions with a polynomial-size domain). We require $f : X \times Y \to \{0, 1\}$ to be a single-output, Boolean function, where X is the set of inputs for the first party and Y is the set of inputs for the second party.

The protocol, described in full in the following section, is parameterized by a constant $\alpha > 0$ that depends on the particular function f being computed. After describing the protocol in generic terms, we prove security of the protocol (instantiated with a particular value of α) for a *specific* function f. In the full version of this work, we describe a general class of functions for which the protocol can be applied to obtain complete fairness.

4.1 The Protocol

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.) In contrast to our earlier protocol, the values $a_1, b_1, \ldots, a_m, b_m$ are now generated *probabilistically* as follows: first, a value $i^* \in \{1, \ldots, m\}$ is chosen according to a geometric distribution (see below). 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 $f(x, y)$. As in the previous protocol, following the preliminary phase the parties interact for m iterations where they exchange their shares one-by-one, 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.)

If $m = \omega(\log n)$, we have $a_m = b_m = f(x, y)$ with all but negligible probability and so correctness holds. 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 it has then learned the correct value of $f(x, y)$ while P_2 has not. However, P_1 cannot identify iteration i^* with certainty (this holds even if it knows the other party's input y) and, even though it may guess i^* correctly with non-negligible probability, the fact that it can never be sure whether its guess is correct will suffice to ensure fairness. (This intuition merely provides a way of understanding the protocol; the formal proof validates this intuition.)

Formal description of the protocol. Let $m = \omega(\log n)$. As in Section 3, we use an m-time MAC with information-theoretic security. We also, again, rely on a sub-protocol π computing a functionality ShareGen′ that generates shares (and associated MAC tags) for the parties; see Figure 3. (As before, π securely compute 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 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

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, 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 $\boldsymbol{i = 1, \ldots, m}$ do:**
   $\boldsymbol{P_2}$ sends the next share to $\boldsymbol{P_1}$:
   (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).
   $\boldsymbol{P_1}$ sends the next share to $\boldsymbol{P_2}$:
   (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.

is heads with probability α) until the first head appears, and letting i^* be the number of tosses performed. 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. Furthermore, when α is constant and $m = \omega(\log n)$, then $i^* \leq m$ with all but negligible probability. Our second protocol is given in Figure 4.

4.2 Proving Fairness for a Particular Function

Protocol 2 does *not* 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 the full version of this work we describe a general class of functions for which this is the case. Here, we prove security of the protocol for the following particular function f:

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

We deliberately chose a function that is simple, yet nontrivial. In particular, we chose a function that has an embedded XOR in order to demonstrate that the presence of an embedded XOR is not a barrier to achieving complete fairness. For this f, we set $\alpha = 1/5$ in Protocol 2. (In fact, any $\alpha \leq 1/5$ would work.) We explain how we arrived at this value of α in the full version.

Theorem 3. *If* $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ *is an i.t.-secure, m-time MAC, and π securely computes* $\mathsf{ShareGen}'$ *with abort, then Protocol 2, with $\alpha = 1/5$, securely computes f with complete fairness.*

Proof Idea: Due to space limitations, we are not able to include the complete proof. We therefore provide an outline of the proof that highlights some of the main ideas. We refer the reader to the full version of this work for further details.

As in the previous section, we analyze Protocol 2 (with $\alpha = 1/5$) in a hybrid model where there is a trusted party computing $\mathsf{ShareGen}'$. (Again, we stress that since π securely computes $\mathsf{ShareGen}'$ with abort, the adversary may abort the trusted party computing $\mathsf{ShareGen}'$ before it sends output to the honest party.) We 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.

The case where P_2 is corrupted is fairly straightforward since, in every iteration, P_2 sends its share first. We therefore focus on the case of a corrupted P_1. We construct a simulator $\mathcal{S}$ given black-box access to $\mathcal{A}$. For readability, we ignore the presence of the MAC tags and keys, and assume that a MAC forgery does not occur. When we say $\mathcal{A}$ "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[2] x', the auxiliary input, and the security parameter n. The simulator also chooses

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

$\hat{x} \in X$ 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 $x \notin X$, 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) If $x \in X$, 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 $\mathcal{A}$ aborts 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, $\mathcal{S}$ proceeds as below.

4. Choose i^* according to a geometric distribution with parameter α. Then 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 $\mathcal{A}$.
   (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 $\mathcal{A}$. (A fresh $\hat{y}$ is chosen in every iteration.)
   (b) If $\mathcal{A}$ aborts, then:
      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.

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

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

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 $\mathcal{S}$ 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 not 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. (Recall we are ignoring here the possibility that $\mathcal{A}$ outputs a MAC forgery; this introduces only a negligible statistical difference.)

Observe that the case of $x = x_3$ is straightforward: here, $\mathcal{S}$ does not need to send anything to the trusted party until *after* $\mathcal{A}$ aborts, since $a_i = 1$ for all i. (This is because $f(x_3, y) = 1$ 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 $x \in \{x_1, x_2\}$. Even for this case we will not be able to give the complete proof; instead, to provide some intuition, we show that $\mathcal{S}$ as above provides a good simulation for one (natural) *particular* adversary.

Specifically, let us consider a hybrid-world adversary $\mathcal{A}$ that sends its input $x \in \{x_1, x_2\}$ to the trusted party computing $\mathsf{ShareGen}'$, does not abort the trusted party, and then computes a_1 after receiving $a_1^{(2)}$ from P_2. If $a_1 = 0$, it aborts immediately; otherwise, it runs the protocol honestly to the end. We now analyze the joint distribution of $\mathcal{A}$'s view (that we will denote by $\textsc{view}_{\mathcal{A}}$) and P_2's output (that we will denote by $\textsc{out}_2$) in both the hybrid world and the ideal world, given our simulator $\mathcal{S}$ as described above. We perform this analysis for all possible values of the parties' inputs.

Case 1: $x = x_1$ and $y = y_1$. Note that $f(x, y) = 0$. We look first at the hybrid world. The probability that $a_1 = 0$ is $\alpha + \frac{1}{2} \cdot (1 - \alpha) = \frac{3}{5}$, and in this case P_1 aborts before P_2 has received any shares. If P_1 aborts then, according to the protocol, P_2 outputs $b_0 = f(\hat{x}, y_1)$ where $\hat{x}$ is chosen uniformly from X. So $\Pr[\textsc{out}_2 = 0] = \frac{1}{3}$ and we conclude that

$$\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0, 0)] = \frac{3}{5} \cdot \frac{1}{3} = \frac{1}{5} \tag{1}$$

and

$$\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0, 1)] = \frac{3}{5} \cdot \frac{2}{3} = \frac{2}{5} \,. \tag{2}$$

(We do not explicitly include in $\textsc{view}_{\mathcal{A}}$ the shares $\mathcal{A}$ received from $\mathsf{ShareGen}'$, since these are simply random values.)

The probability that $a_1 = 1$ is $\frac{1}{2} \cdot (1 - \alpha) = \frac{2}{5}$. When $a_1 = 1$ then $\mathcal{A}$ runs the protocol honestly to completion, and $\textsc{out}_2 = f(x, y) = 0$.

Let us now look at the ideal-world execution of $\mathcal{S}$ with a trusted party computing f. We first examine the case when $a_1 = 0$. This can happen in two ways:

- With probability $\alpha = 1/5$, we have $i^* = 1$. In this case the simulator sends x_1 to the trusted party, learns the correct value of $f(x_1, y_1) = 0$, and sets $a_1 = 0$. Note that here we also have $\textsc{out}_2 = f(x_1, y_1) = 0$.

- With probability $1 - \alpha = 4/5$, we have $i^* > i$. In this case the simulator generates a_1 by choosing random $\hat{y}$ and setting $a_1 = f(x_1, \hat{y})$; thus, a_1 is a random bit. If it turns out that $a_1 = 0$ (which occurs with probability $\frac{1}{2}$), then $\mathcal{A}$ aborts right away; in response, $\mathcal{S}$ sends x_2 to the trusted party with probability $\frac{1}{3}$ and sends x_3 to the trusted party with the remaining probability (see step 8(b)(i)). In either eventuality, we have $\textsc{out}_2 = 1$ (since $f(x_2, y_1) = f(x_3, y_1) = 1$).

Combining the above, we see that

$$\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,0)] = \Pr[i^* = 1] = \frac{1}{5}$$

and

$$\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,1)] = \frac{1}{2} \cdot \Pr[i^* > 1] = \frac{1}{2} \cdot \frac{4}{5} = \frac{2}{5},$$

in agreement with Eqs. (1) and (2).

We have $a_1 = 1$ with probability $\frac{1}{2} \cdot \Pr[i^* > 1] = \frac{2}{5}$. When this occurs, $\mathcal{A}$ will never abort and so we have $\textsc{out}_2 = f(x_1, y_1) = 0$. Furthermore, it can be verified that the view of $\mathcal{A}$ in this case is distributed identically to the view of $\mathcal{A}$ in the hybrid world, conditioned on $a_1 = 1$.

Case 2: $\boldsymbol{x = x_1}$ and $\boldsymbol{y = y_2}$. The analysis is similar to the above. Here, $f(x, y) = 1$. Now, in the hybrid world, the probability that $a_1 = 0$ is $\frac{1}{2} \cdot (1 - \alpha) = \frac{2}{5}$, and $\mathcal{A}$ aborts immediately when this happens. When $\mathcal{A}$ aborts, P_2 outputs $b_0 = f(\hat{x}, y_2)$ where $\hat{x}$ is chosen uniformly from X. So $\Pr[\textsc{out}_2 = 0] = \frac{1}{3}$ and we conclude that

$$\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,0)] = \frac{2}{5} \cdot \frac{1}{3} = \frac{2}{15} \tag{3}$$

and

$$\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,1)] = \frac{2}{5} \cdot \frac{2}{3} = \frac{4}{15}. \tag{4}$$

We have $a_1 = 1$ with probability $\alpha + \frac{1}{2} \cdot (1 - \alpha) = \frac{3}{5}$. When $a_1 = 1$ then $\mathcal{A}$ runs the protocol honestly to completion, and $\textsc{out}_2 = f(x, y) = 1$.

In the ideal world, we first examine the case when $a_1 = 0$. This can only happen when $i^* > 1$, and so occurs with probability $\frac{2}{5}$. Once $\mathcal{A}$ aborts, $\mathcal{S}$ sends x_2 to the trusted party with probability $\frac{1}{3}$ and sends x_3 to the trusted party with the remaining probability (see step 8(b)(i)); thus, we have $\Pr[\textsc{out}_2 = 0] = \frac{1}{3}$. We therefore conclude that

$$\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,0)] = \frac{2}{5} \cdot \frac{1}{3} = \frac{2}{15}$$

and

$$\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,1)] = \frac{2}{5} \cdot \frac{2}{3} = \frac{4}{15},$$

in agreement with Eqs. (3) and (4).

In the ideal world, it also holds that $\Pr[a_1 = 1] = \frac{3}{5}$. Conditioned on this event, $\mathcal{A}$ never aborts and so $\textsc{out}_2 = f(x_1, y_2) = 1$. Furthermore, it can be verified that the view of $\mathcal{A}$ in this case is distributed identically to the view of $\mathcal{A}$ in the hybrid world, conditioned on $a_1 = 1$.

Case 3: $\boldsymbol{x = x_2}$ and $\boldsymbol{y = y_1}$. Here, $f(x, y) = 1$. In the hybrid world, the probability that $a_1 = 0$ is $\frac{1}{2} \cdot (1 - \alpha) = \frac{2}{5}$, and $\mathcal{A}$ aborts immediately when this happens. When $\mathcal{A}$ aborts, P_2 outputs $b_0 = f(\hat{x}, y_1)$ where $\hat{x}$ is chosen uniformly from X. So $\Pr[\textsc{out}_2 = 0] = \frac{1}{3}$, and Eqs. (3) and (4) hold here as well.

We have $a_1 = 1$ with probability $\alpha + \frac{1}{2} \cdot (1 - \alpha) = \frac{3}{5}$. When $a_1 = 1$ then $\mathcal{A}$ runs the protocol honestly to completion, and $\textsc{out}_2 = f(x, y) = 1$.

In the ideal world, $a_1 = 0$ occurs with probability $\frac{2}{5}$. When $\mathcal{A}$ aborts, $\mathcal{S}$ sends x_1 to the trusted party with probability $\frac{1}{3}$ and sends x_3 to the trusted party with the remaining probability; thus, we have $\Pr[\textsc{out}_2 = 0] = \frac{1}{3}$ and so $\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,0)] = \frac{2}{15}$ and $\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,1)] = \frac{4}{15}$, just as in the hybrid world.

In the ideal world, it also holds that $\Pr[a_1 = 1] = \frac{3}{5}$. Conditioned on this event, $\mathcal{A}$ never aborts and so $\textsc{out}_2 = f(x_1, y_2) = 1$. Once again, it can be verified that the view of $\mathcal{A}$ in this case is distributed identically to the view of $\mathcal{A}$ in the hybrid world, conditioned on $a_1 = 1$.

Case 4: $\boldsymbol{x = x_2}$ and $\boldsymbol{y = y_2}$. Now $f(x, y) = 0$. In the hybrid world, the probability that $a_1 = 0$ is $\alpha + \frac{1}{2} \cdot (1 - \alpha) = \frac{3}{5}$. When this occurs, P_2 outputs $b_0 = f(\hat{x}, y_1)$ where $\hat{x}$ is chosen uniformly from X. So $\Pr[\textsc{out}_2 = 0] = \frac{1}{3}$ and Eqs. (1) and (2) hold here as well.

The probability that $a_1 = 1$ is $\frac{1}{2} \cdot (1 - \alpha) = \frac{2}{5}$. When $a_1 = 1$ then $\mathcal{A}$ runs the protocol honestly to completion, and $\textsc{out}_2 = f(x, y) = 0$.

In the ideal world, $a_1 = 0$ can happen in two ways:

- With probability $\alpha = 1/5$, we have $i^* = 1$. In this case the simulator sends x_2 to the trusted party, learns the correct value of $f(x_1, y_1) = 0$, and sets $a_1 = 0$. Here, $\textsc{out}_2 = f(x_2, y_2) = 0$.

- With probability $1 - \alpha = 4/5$, we have $i^* > i$. In this case $\mathcal{S}$ generates a_1 by choosing random $\hat{y}$ and setting $a_1 = f(x_1, \hat{y})$; thus, a_1 is a random bit. If $a_1 = 0$, then $\mathcal{A}$ aborts; in response, $\mathcal{S}$ sends x_1 to the trusted party with probability $\frac{1}{3}$ and sends x_3 to the trusted party with the remaining probability (cf. step 8(b)(i)). In either eventuality, we have $\textsc{out}_2 = 1$ (since $f(x_1, y_2) = f(x_3, y_2) = 1$).

We thus see that $\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,0)] = \frac{1}{5}$, and $\Pr[(\textsc{view}_{\mathcal{A}}, \textsc{out}_2) = (0,1)] = \frac{2}{5}$, as in the hybrid world.

The probability that $a_1 = 1$ is $\frac{1}{2} \cdot \Pr[i^* > 1] = \frac{2}{5}$. When this occurs, $\mathcal{A}$ will never abort and so we have $\textsc{out}_2 = f(x_1, y_1) = 0$. Furthermore, it can be verified that the view of $\mathcal{A}$ in this case is distributed identically to the view of $\mathcal{A}$ in the hybrid world, conditioned on $a_1 = 1$. $\blacksquare$

5. BOUNDING THE ROUND COMPLEXITY

Since Cleve's impossibility proof [10] rules out complete fairness for Boolean XOR, one might conjecture that any function containing an *embedded XOR* (i.e., for which there exist inputs x_0, x_1, y_0, y_1 such that $f(x_i, y_j) = i \oplus j$) cannot be securely computed with complete fairness. In the previous section we have shown that this is not the case. The protocol shown there, however, requires $\omega(\log n)$ rounds and one might wonder whether this can be improved. (In particular, Protocol 1 required $O(1)$ rounds when the input domains were of fixed size.) The following theorem shows that the round complexity of our protocol is optimal:

THEOREM 4. *Let f be a function with an embedded XOR. Then any protocol securely computing f with complete fairness (assuming one exists) requires $\omega(\log n)$ rounds.*

A proof appears in the full version of this work.

6. CONCLUSIONS AND OPEN QUESTIONS

We have shown the first positive results for secure two-party computation of non-trivial functionalities with complete fairness. Our results re-open a line of research that was previously thought to be closed. A host of interesting questions remain:

- Can the round complexity of our first protocol be improved, or can a lower bound be shown?
- We currently only know how to show impossibility of complete fairness for functionalities that imply coin tossing. Can further impossibility results be shown? Ultimately, of course, we would like a complete characterization of when complete fairness is possible.
- We currently only have feasibility results for single-output, 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.
- What can be said with regard to complete fairness in the *multi-party* setting, without an honest majority? (This question is interesting both with and without the assumption of a broadcast channel.) Though initial feasibility results have recently been shown [18], the previous questions apply here as well.

7. REFERENCES

[1] D. Beaver. Foundations of Secure Interactive Computing. In *Crypto '91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

[2] D. Beaver. Secure Multi-Party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *Journal of Cryptology* 4(2):75–122, 1991.

[3] D. Beaver and S. Goldwasser. Multiparty Computation with Faulty Majority. In 30*th FOCS*, pages 468–473, 1989.

[4] A Beimel, T. Malkin, and S. Micali. The All-or-Nothing Nature of Two-Party Secure Computation. In *Crypto '99*, Springer-Verlag (LNCS 1666), pages 80–97, 1999.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In 20*th STOC*, pages 1–10, 1988.

[6] D. Boneh and M. Naor. Timed Commitments. In *Crypto 2000*, Springer-Verlag (LNCS 1880), pages 236–254, 2000.

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

[8] D. Chaum, C. Crépeau, and I. Damgård. Multi-party Unconditionally Secure Protocols. In 20*th STOC*, pages 11–19, 1988.

[9] B. Chor and E. Kushilevitz. A Zero-One Law for Boolean Privacy. *SIAM Journal of Discrete Math* 4(1):36–47, 1991.

[10] R. Cleve. Limits on the Security of Coin Flips when Half the Processors are Faulty. In 18*th STOC*, pages 364–369, 1986.

[11] D. Dolev and H. Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[12] S. Even, O. Goldreich, and A. Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM* 28(6):637–647, 1985.

[13] Z. Galil, S. Haber, and M. Yung. Cryptographic Computation: Secure Fault Tolerant Protocols and the Public Key Model. In *Crypto '87*, Springer-Verlag (LNCS 293), pages 135–155, 1988.

[14] J. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. Resource Fairness and Composability of Cryptographic Protocols. In 3*rd TCC*, Springer-Verlag (LNCS 3876), pages 404–428, 2006.

[15] O. Goldreich. *Foundations of Cryptography: Volume 2*. Cambridge University Press, 2004.

[16] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game — A Completeness Theorem for Protocols with Honest Majority. In 19*th STOC*, pages 218–229, 1987.

[17] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *Crypto '90*, Springer-Verlag (LNCS 537), pages 77–93, 1990.

[18] S.D. Gordon and J. Katz. Complete Fairness in Multi-Party Computation Without an Honest Majority. Manuscript in submission.

[19] J. Kilian. A General Completeness Theorem for Two-Party Games. In 23*rd STOC*, pages 553–560, 1991.

[20] J. Kilian. More General Completeness Theorems for Secure Two-Party Computation. In 32*nd STOC*, pages 316–324, 2000.

[21] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. *Journal of Cryptology* 16(3): 143–184, 2003.

[22] M. Luby, S. Micali and C. Rackoff. How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin. In 24*th FOCS*, pages 11–21, 1983.

[23] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *Crypto '91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.

[24] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM* 27(2):228–234, 1980.

[25] B. Pinkas. Fair Secure Two-Party Computation. In *Eurocrypt 2003*, Springer-Verlag (LNCS 2656), pages 87–105, 2003.

[26] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In 21*st STOC*, pages 73–85, 1989.

[27] A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986.