A Full Characterization of Functions that Imply Fair Coin Tossing and Ramifications to Fairness*

Gilad Asharov[1], Yehuda Lindell[1], and Tal Rabin[2]

[1] Department of Computer Science, Bar-Ilan University, Israel
[2] IBM T.J. Watson Research Center, New York
asharog@cs.biu.ac.il, lindell@biu.ac.il, talr@us.ibm.com

Abstract. It is well known that it is impossible for two parties to toss a coin fairly (Cleve, STOC 1986). This result implies that it is impossible to securely compute with fairness any function that can be used to toss a fair coin. In this paper, we focus on the class of deterministic Boolean functions with finite domain, and we ask for which functions in this class is it possible to information-theoretically toss an unbiased coin, given a protocol for securely computing the function with fairness. We provide a *complete characterization* of the functions in this class that imply and do not imply fair coin tossing. This characterization extends our knowledge of which functions cannot be securely computed with fairness. In addition, it provides a focus as to which functions may potentially be securely computed with fairness, since a function that cannot be used to fairly toss a coin is not ruled out by the impossibility result of Cleve (which is the *only* known impossibility result for fairness). In addition to the above, we draw corollaries to the feasibility of achieving fairness in two possible fail-stop models.

1 Introduction

1.1 Background

In the setting of secure multiparty computation, some mutually distrusting parties wish to compute some joint function of their inputs in the presence of adversarial behaviour. Loosely speaking, the security requirements from such a computation are that nothing is learned from the protocol other than the output (privacy), that the output is distributed according to the prescribed functionality (correctness), and that parties cannot choose their inputs as a function of the others' inputs (independence of inputs). Another important property is that of *fairness* which, intuitively, means that either *everyone* receives the output or *no one* does.

It is well known that when a majority of the parties are honest, it is possible to securely compute any functionality while guaranteeing all of the security properties mentioned above, including fairness [8,2,4,11]. Furthermore, when there is no

* The first two authors were funded by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 239868, and by the ISRAEL SCIENCE FOUNDATION (grant No. 189/11). Much of this work was carried out while they were at the IBM T.J. Watson Research Center, New York.

A. Sahai (Ed.): TCC 2013, LNCS 7785, pp. 243–262, 2013.

honest majority, including the important case of two parties where one may be corrupted, it is possible to securely compute any functionality while guaranteeing all of the security properties mentioned above *except for fairness* [13,8,6]. The fact that fairness is not achieved in this latter case is inherent, as was shown in the seminal work of Cleve [5] who proved that there exist functions that cannot be computed by two parties with complete fairness. Specifically, Cleve showed that the very basic and natural functionality of coin-tossing, where two parties toss an unbiased coin, cannot be computed fairly. The impossibility result of Cleve implies that fairness cannot be achieved *in general*. That is, Cleve's result proves that it is impossible to securely compute with complete fairness any function that can be used to toss a fair coin (like the boolean XOR function).

Until recently, the accepted folklore from the time of Cleve's result was that *only trivial functions* can be securely computed with complete fairness without an honest majority. This changed recently with the surprising work of Gordon et al. [9] who showed that this folklore is incorrect and that there exist some non-trivial boolean functions that *can* be computed fairly, in the two party setting. They showed that *any* function that does not contain an embedded XOR (i.e., inputs x_1, x_2, y_1, y_2 such that $f(x_1, y_1) = f(x_2, y_2) \neq f(x_1, y_2) = f(x_2, y_1)$) can be computed fairly in the malicious settings. Examples of functions without an embedded XOR include the boolean OR and AND functions and Yao's millionaires problem [13] (i.e., the greater-than function). This possibility result changes our understanding regarding fairness, and re-opens the question of which functions can be computed with complete fairness. Given the possibility result mentioned above, and given the fact that Cleve's impossibility result rules out completely fair computation of boolean XOR, a natural conjecture is that the presence of an embedded XOR serves as a barrier to a fair computation of a given function. However, [9] showed that this is also incorrect: they give an example of a function that *does* contain an embedded XOR and construct a protocol that securely computes this function with fairness.

Since [9], there have been no other works that further our understanding regarding which (boolean) functions can be computed fairly without an honest majority in the two party setting. Specifically, Cleve's impossibility result is the only known function that cannot be computed fairly, and the class of functions for which [9] shows possibility are the only known possible functions. There is therefore a large class of functions for which we have no idea as to whether or not they can be securely computed with complete fairness.

1.2 Our Work

Motivated by the fundamental question of characterizing which functions can be computed with complete fairness, we analyze which functions *cannot* be computed fairly since they are already ruled out by Cleve's original result. That is, we show which boolean functions "imply" the coin-tossing functionality. We provide a simple property (criterion) on the truth table of a given boolean function. We then show that for every function that satisfies this property, it holds that the existence of a protocol that fairly computes the given function implies the

existence of a protocol for fair coin-tossing in the presence of a fail-stop or malicious adversary, in contradiction to Cleve's impossibility result. This implies that the functions that satisfy the property cannot be computed fairly. The property is very simple, clean and general.

The more challenging and technically interesting part of our work is a proof that the property is *tight*. Namely, we show that a function f that does not satisfy the property *cannot* be used to construct a fair coin-tossing protocol (in the information theoretic setting). More precisely, we show that it is impossible to construct a fair two-party coin-tossing protocol, even if the parties are given access to a trusted party that computes f *fairly* for them. We prove this impossibility by showing the existence of an (inefficient) adversary that can bias the outcome with non-negligible probability. Thus, we prove that it is not possible to toss a coin with information-theoretic security, when given access to fair computations of f. We stress that this "impossibility" result is actually a source of optimism, since it *may* be possible to securely compute such functions with complete fairness. Indeed, the fair protocols presented in [9] are for functions for which the property does not hold.[1]

It is important to note that our proof that functions that do not satisfy the property do not imply coin tossing is very different from the proof of impossibility by Cleve. Specifically, the intuition behind the proof by Cleve is that since the parties exchange messages in turn, there must be a point where one party has more information than the other about the outcome of the coin-tossing protocol. If that party aborts at this point, then this results in bias. This argument holds since the parties cannot exchange information simultaneously. In contrast, in our setting, the parties *can* exchange information simultaneously via the computation of f. Thus, our proof is conceptually very different to that of Cleve, and in particular, is not a reduction to the proof by Cleve.

The Criterion. Intuitively, the property that we define over the function's truth table relates to the question of whether or not it is possible for one party to singlehandedly change the probability that the output of the function will be 1 (or 0) based on how it chooses its input. In order to explain the criterion, we give two examples of functions that imply coin-tossing, meaning that a fair secure protocol for computing the function implies a fair secure protocol for coin tossing. We discuss how each of the examples can be used to toss a coin fairly, and this in turn will help us to explain the criterion. The functions are given below:

(a)

	y_1	y_2	y_3
x_1	0	1	1
x_2	1	0	0
x_3	0	0	1

(b)

	y_1	y_2	y_3
x_1	1	0	0
x_2	0	1	0
x_3	0	0	1

[1] We remark that since our impossibility result is information theoretic, there is the possibility that some of the functions for which the property does not hold do imply coin tossing computationally. In such a case, the impossibility result of Cleve still applies to them. See more discussion in "open questions" below.

Consider function (a), and assume that there exists a fair protocol for this function. We show how to toss a fair coin using a single invocation of the protocol for f. Before doing so, we observe that the output of a single invocation of the function can be expressed by multiplying the truth-table matrix of the function by probability vectors.[2] Specifically, assume that party P_1 chooses input x_i with probability p_i, for $i = 1, 2, 3$ (thus $p_1 + p_2 + p_3 = 1$ since it must choose some input); likewise, assume that P_2 chooses input y_i with probability q_i. Now, let M_f be the "truth table" of the function, meaning that $M_f[i, j] = f(x_i, y_j)$. Then, the output of the invocation of f upon the inputs chosen by the parties equals 1 with probability exactly $(p_1, p_2, p_3) \cdot M_f \cdot (q_1, q_2, q_3)^T$.

We are now ready to show how to toss a coin using f. First, note that there are two complementary rows; these are the rows specified by inputs x_1 and x_2. This means that if P_1 chooses one of the inputs in $\{x_1, x_2\}$ uniformly at random, then no matter what distribution over the inputs (corrupted) P_2 uses, the result is a uniformly chosen coin. In order to see this, observe that when we multiply the vector $(\frac{1}{2}, \frac{1}{2}, 0)$ (the distribution over the input of P_1) with the matrix M_f, the result is the vector $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. This means that no matter what input P_2 will choose, or what distribution over the inputs it may use, the output is 1 with probability 1/2 (formally, the output is 1 with probability $\frac{1}{2} \cdot q_1 + \frac{1}{2} \cdot q_2 + \frac{1}{2} \cdot q_3 = \frac{1}{2}$ because $q_1 + q_2 + q_3 = 1$). This means that if P_1 is honest, then a corrupted P_2 cannot bias the output. Likewise, there are also two complementary columns (y_1 and y_3), and thus, if P_2 chooses one of the inputs in $\{y_1, y_3\}$ uniformly at random, then no matter what distribution over the inputs (a possibly corrupted) P_1 uses, the result is a uniform coin.

In contrast, there are no two complementary rows or columns in the function (b). However, if P_1 chooses one of the inputs $\{x_1, x_2, x_3\}$ uniformly at random (i.e., each input with probability one third), then no matter what distribution P_2 will use, the output is 1 with probability 1/3. Similarly, if P_2 chooses a uniformly random input, then no matter what P_1 does, the output is 1 with the same probability. Therefore, a single invocation of the function f in which the honest party chooses the uniform distribution over its inputs results in a coin that equals 1 with probability exactly $\frac{1}{3}$, irrespective of what the other party inputs. In order to obtain an unbiased coin that equals 1 with probability $\frac{1}{2}$ the method of von-Neumann [12] can be used. This method works by having the parties use the function f to toss two coins. If the resulting coins are different (i.e, 01 or 10), then they output the result of the first invocation. Otherwise, they run the protocol again. This yields a coin that equals 1 with probability $\frac{1}{2}$ since the probability of obtaining 01 equals the probability of obtaining 10. Thus, conditioned on the results being different, the probability of outputting 0 equals the probability of outputting 1.

The criterion is a direct generalization of the examples shown above. Let $f : \{x_1, \ldots, x_\ell\} \times \{y_1, \ldots, y_\ell\} \to \{0, 1\}$ be a function, and let M_f be the truth table representation as described above. We say that the function has the criterion if there exist two probability vectors, $\mathbf{p} = (p_1, \ldots, p_m)$, $\mathbf{q} = (q_1 \ldots, q_\ell)$

[2] $\mathbf{p} = (p_1, \ldots, p_m)$ is a probability vector if $p_i \geq 0$ for every $1 \leq i \leq m$, and $\sum_{i=1}^{m} p_i = 1$.

such that $\mathbf{p} \cdot M_f$ and $M_f \cdot \mathbf{q}^T$ are both vectors that equal δ everywhere, for some $0 < \delta < 1$. Observe that if such probability vectors exist, then the function implies the coin-tossing functionality as we described above. Specifically, P_1 chooses its input according to distribution $\mathbf{p}$, and P_2 chooses its inputs according to the distribution $\mathbf{q}$. The result is then a coin that equals 1 with probability δ. Using the method of von-Neumann, this can be used to obtain a uniformly distributed coin. We conclude:

Theorem 1.1 (informal). *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0,1\}$ be a function that satisfies the aforementioned criterion. Then, the existence of a protocol for securely computing f with complete fairness implies the existence of a fair coin tossing protocol.*

An immediate corollary of this theorem is that any such function cannot be securely computed with complete fairness, as this contradicts the impossibility result of Cleve [5].

As we have mentioned above, the more interesting and technically challenging part of our work is a proof that the criterion is tight. That is, we prove the following theorem:

Theorem 1.2 (informal). *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0,1\}$ be a function that does* not *satisfy the aforementioned criterion. Then, there exists an exponential-time adversary that can bias the outcome of every coin-tossing protocol that uses ideal and fair invocations of f.*

This result has a number of ramifications. Most notably, it helps focus our research on the question of fairness in two-party secure computation. Specifically, the only functions that can potentially be computed securely with fairness are those for which the property does not hold. In these functions one of the parties can partially influence the outcome of the result singlehandedly, a fact that is used inherently in the protocol of [9] for the function with an embedded XOR. This does not mean that all functions of this type can be fairly computed. However, it provides a good starting point. In addition, our results define the set of functions for which Cleve's impossibility result suffices for proving that they cannot be securely computed with fairness. Given that no function other than those implying coin tossing has been ruled out since Cleve's initial result, understanding exactly what is included in this impossibility is of importance.

On Fail-Stop Adversaries. Our main results above consider the case of malicious adversaries. In addition, we explore the fail-stop adversary model where the adversary follows the protocol like an honest party, but can halt early. This model is of interest since the impossibility result of Cleve [5] for achieving fair coin tossing holds also for fail-stop adversaries. In order to prove theorems regarding the fail-stop model, we first provide a definition of security with complete fairness for fail-stop adversaries that follows the real/ideal simulation paradigm. Surprisingly, this turns out not to be straightforward and we provide two natural formulations that are very different regarding feasibility. The formulations differ regarding the ideal-world adversary/simulator. The question that arises is

whether or not the simulator is allowed to use a different input to the prescribed one. In the semi-honest model (which differs only in the fact that the adversary cannot halt early) the standard formulation is to not allow the simulator to change the prescribed input, whereas in the malicious model the simulator is always allowed to change the prescribed input. We therefore define two fail-stop models. In this first, called "fail-stop1", the simulator is allowed to either send the trusted party computing the function the prescribed input of the party or an abort symbol $\perp$, but nothing else. In the second, called "fail-stop2", the simulator may send any input that it wishes to the trusted party computing the function. Note, however, that if there was no early abort then the prescribed input must be used because such an execution is identical to an execution between two honest parties.

Observe that in the first model, the honest party is guaranteed to receive the output on the prescribed inputs, unless it receives abort. In addition, observe that any protocol that is secure in the presence of malicious adversaries is secure also in the fail-stop2 model. However, this is not true of the fail-stop1 model (this is due to the fact that the simulator in the ideal model for the case of malicious adversaries is more powerful than in the fail-stop1 ideal model since the former can send any input whereas the latter can only send the prescribed input or $\perp$).

We remark that Cleve's impossibility result holds in both models, since the parties do not have inputs in the coin-tossing functionality, and therefore there is no difference in the ideal-worlds of the models in this case. In addition, the protocols of [9] that are secure for malicious adversaries are secure for fail-stop2 (as mentioned above, this is immediate), but are *not* secure for fail-stop1.

We show that in the fail-stop1 model, it is impossible to securely compute with complete fairness any function containing an embedded XOR. We show this by constructing a coin-tossing protocol from any such function, that is secure in the fail-stop model. Thus, the only functions that can potentially be securely computed with fairness are those with no embedded XOR but with an embedded OR (if a function has neither, then it is trivial and can be computed unconditionally and fairly); we remark that there are very few functions with this property. We conclude that in the fail-stop1 model, fairness cannot be achieved for almost all non-trivial functions. We remark that [9] presents secure protocols that achieve complete fairness for functions that have no embedded XOR; however, they are not secure in the fail-stop1 model, as mentioned.

Regarding the fail-stop2 model, we prove an analogous result to Theorem 1.2. In the proof of Theorem 1.2, the adversary that we construct changes its input in one of the invocations of f and then continues honestly. Thus, it is malicious and not fail-stop2. Nevertheless, we show how the proof can be modified in order to hold for the fail-stop2 model as well.

These extensions for fail-stop adversaries deepen our understanding regarding the feasibility of obtaining fairness. Specifically, any protocol that achieves fairness for any non-trivial function (or at least any function that has an embedded XOR), must have the property that the simulator can send any input in the

ideal model. Stated differently, the input that is effectively used by a corrupted party cannot be somehow committed, thereby preventing this behaviour. This also explains why the protocols of [9] have this property.

1.3 Open Questions

In this work we provide an almost complete characterization regarding what functions imply and do not imply coin tossing. Our characterisation is not completely tight since the impossibility result of Theorem 1.2 only holds in the information-theoretic setting; this is due to the fact that the adversary needs to carry out inefficient computations. Thus, it is conceivable that coin tossing can be achieved computationally from some such functions. It is important to note, however, that any function that does not fulfil our criterion implies oblivious transfer (OT). Thus, any protocol that uses such a function has access to OT and all that is implied by OT (e.g., commitments, zero knowledge, and so on). Thus, any such computational construction would have to be inherently nonblack-box in some sense. Our work also only considers finite functions (where the size of the domain is not dependent on the security parameter); extensions to other function classes, including non-Boolean functions, is also of interest.

The main open question left by our work is to characterize which functions for which the criterion does not hold can be securely computed with complete fairness. Our work is an important step to answering this question by providing a clearer focus than was previously known. Observe that in order to show that a function that does not fulfil the criterion cannot be securely computed with complete fairness, a new impossibility result must be proven. In particular, it will not be possible to reduce the impossibility to Cleve [5] since such a function does not imply coin tossing.

2 Definitions

The Coin-Tossing Functionality. We define the coin-tossing functionality simply by $f^{\mathsf{ct}}(\lambda, \lambda) = (U_1, U_1)$, where λ denotes the empty input and U_1 denotes the uniform distribution over $\{0,1\}$. That is, the functionality receives no input, chooses a uniformly chosen bit and gives both parties the same bit. This yields the following definition:

Definition 2.1 (Coin-Tossing by Simulation). *A protocol π is a* secure coin-tossing protocol via simulation *if it securely computes f^{ct} with complete fairness in the presence of malicious adversaries.*

The above definition provides very strong simulation-based guarantees, which is excellent for our positive results. However, when proving impossibility, it is preferable to rule out even weaker, non-simulation based definitions. We now present a weaker definition where the guarantee is that the honest party outputs an unbiased coin, irrespective of the cheating party's behaviour.

However, we stress that since our impossibility result only holds with respect to an all-powerful adversary (as discussed in the introduction), our definition is stronger than above since it requires security in the presence of any adversary, and not just polynomial-time adversaries.

Notations. Denote by $\langle P_1, P_2 \rangle$ a two party protocol where both parties act honestly. For $\ell \in \{1,2\}$, let $\mathsf{out}_\ell \langle P_1^*, P_2^* \rangle$ denote the output of party P_ℓ^* in an execution of P_1^* with P_2^*. In some cases, we also specify the random coins that the parties use in the execution; $\langle P_1(r_1), P_2(r_2) \rangle$ denotes an execution where P_1 acts honestly and uses random tape r_1 and P_2 acts honestly and uses random tape r_2. Let $r(n)$ be a polynomial that bounds the number of rounds of the protocol π, and let $c(n)$ be an upper bound on the length of the random tape of the parties. Let Uni denote the uniform distribution over $\{0,1\}^{c(n)} \times \{0,1\}^{c(n)}$. We are now ready to define a coin-tossing protocol:

Definition 2.2 (Information-Theoretic Coin-Tossing). *A polynomial-time protocol $\pi = \langle P_1, P_2 \rangle$ is an* unbiased coin-tossing protocol*, if the following hold:*

1. **(agreement)** *There exists a negligible function $\mu(\cdot)$ such that for every n it holds that:*
$$\Pr_{r_1, r_2 \leftarrow \mathsf{Uni}} \left[\mathsf{out}_1 \langle P_1(r_1), P_2(r_2) \rangle \neq \mathsf{out}_2 \langle P_1(r_1), P_2(r_2) \rangle \right] \leq \mu(n) \ .$$
2. **(no bias)** *For every adversary $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that for every $b \in \{0,1\}$ and every $n \in \mathbb{N}$:*
$$\Pr \left[\mathsf{out}_1 \langle P_1, \mathcal{A} \rangle = b \right] \leq \frac{1}{2} + \mu(n) \quad \text{and} \quad \Pr \left[\mathsf{out}_2 \langle \mathcal{A}, P_2 \rangle = b \right] \leq \frac{1}{2} + \mu(n) \ .$$

Observe that both requirements together guarantee that two honest parties will output the same uniformly distributed bit, except with negligible probability.

Function Implication. In the paper, we study whether or not a function f "implies" the coin-tossing functionality. We now formally define what we mean by "function implication". Our formulation uses the notion of a hybrid model, which is a combination of the ideal and real models (see [3,6]). Specifically, let f be a function. Then, an execution in the f-hybrid model consists of real interaction between the parties (like in the real model) and ideal invocations of f (like in the ideal model). The ideal invocations of f take place via a trusted party that receives inputs and sends the output of f on those inputs to both parties, exactly like in the ideal model. We stress that in our ideal model both parties receive the output of f simultaneously since we are considering fair secure computation. We are now ready for the definition.

Definition 2.3. *Let $f : X \times Y \to Z$ and $g : X' \times Y' \to Z'$ be functions. We say that* function f implies function g in the presence of malicious adversaries *if there exists a protocol that securely computes g in the f-hybrid model with complete fairness, in the presence of static malicious adversaries. We say that f* information-theoretically *implies g if the above holds with statistical security.*

Note that if g can be securely computed with fairness (under some assumption), then every function f computationally implies g. Thus, this is only of interest for functions g that either cannot be securely computed with fairness, or for which this fact is not known.

3 The Criterion

In this section we define the criterion, and explore its properties. We start with the definition of δ-balanced functions.

3.1 δ-Balanced Functions

A vector $\mathbf{p} = (p_1, \ldots, p_k)$ is a probability vector if $\sum_{i=1}^{k} p_i = 1$, and for every $1 \leq i \leq k$ it holds that $p_i \geq 0$. Let $\mathbf{1}_k$ be the all one vector of size k. In addition, for a given function $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0, 1\}$, let M_f denote the matrix defined by the truth table of f. That is, for every $1 \leq i \leq m$, $1 \leq j \leq \ell$, it holds that $M_f[i, j] = f(x_i, y_j)$.

Informally, a function is balanced if there exist probabilities over the inputs for each party that determine the probability that the output equals 1, irrespective of what input the other party uses. Assume that P_1 chooses its input according to the probability vector $(p_1, \ldots, p_m)$, meaning that it uses input x_i with probability p_i, for every $i = 1, \ldots, m$, and assume that party P_2 uses the j^{th} input y_j. Then, the probability that the output equals 1 is obtained by multiplying $(p_1, \ldots, p_m)$ with the j^{th} column of M_f. Thus, a function is balanced on the left, or with respect to P_1, if when multiplying $(p_1, \ldots, p_m)$ with M_f the result is a vector with values that are all equal. Formally:

Definition 3.1. *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0, 1\}$ be a function, and let $0 \leq \delta_1, \delta_2 \leq 1$ be constants. We say that f is δ_1-*left-balanced *if there exists a probability vector $\mathbf{p} = (p_1, \ldots, p_m)$ such that:*

$$(p_1, \ldots, p_m) \cdot M_f = \delta_1 \cdot \mathbf{1}_\ell = (\delta_1, \ldots, \delta_1) \ .$$

*Likewise, we say that the function f is δ_2-*right-balanced *if there exists a probability vector $\mathbf{q} = (q_1, \ldots, q_\ell)$ such that:*

$$M_f \cdot (q_1, \ldots, q_\ell)^T = \delta_2 \cdot \mathbf{1}_m^T \ .$$

*If f is δ_1-left-balanced and δ_2-right-balanced, we say that f is (δ_1, δ_2)-balanced. If $\delta_1 = \delta_2$, then we say that f is δ-*balanced*, where $\delta = \delta_1 = \delta_2$. We say that f is* strictly δ-balanced *if $\delta_1 = \delta_2$ and $0 < \delta < 1$.*

Note that a function may be δ_2-right-balanced for some $0 \leq \delta_2 \leq 1$ but not left balanced. For example, consider the function defined by the truth table $M_f \stackrel{\text{def}}{=} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$. This function is right balanced for $\delta_2 = \frac{1}{2}$ by taking $\mathbf{q} = (\frac{1}{2}, \frac{1}{2}, 0)$. However, it is not left-balanced for any δ_1 because for every probability vector

$(p_1, p_2) = (p, 1-p)$ it holds that $(p_1, p_2) \cdot M_{\tilde{f}} = (p, 1-p, 1)$, which is not balanced for any p. Likewise, a function may be δ_2-right-balanced, but not left balanced.

We now prove a simple but somewhat surprising proposition, stating that if a function is (δ_1, δ_2)-balanced, then δ_1 and δ_2 must actually equal each other. Thus, any (δ_1, δ_2)-balanced function is actually δ-balanced.

Proposition 3.2. *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0,1\}$ be a (δ_1, δ_2)-balanced function for some constants $0 \leq \delta_1, \delta_2 \leq 1$. Then, $\delta_1 = \delta_2$, and so f is δ-balanced.*

Proof: Under the assumption that f is (δ_1, δ_2)-balanced, we have that there exist probability vectors $\mathbf{p} = (p_1, \ldots, p_m)$ and $\mathbf{q} = (q_1, \ldots, q_\ell)$ such that $\mathbf{p} \cdot M_f = \delta_1 \cdot \mathbf{1}_\ell$ and $M_f \cdot \mathbf{q}^T = \delta_2 \cdot \mathbf{1}_m^T$. Observe that since $\mathbf{p}$ and $\mathbf{q}$ are probability vectors, it follows that for every constant c we have $\mathbf{p} \cdot (c \cdot \mathbf{1}_m^T) = c \cdot (\mathbf{p} \cdot \mathbf{1}_m^T) = c$; likewise $(c \cdot \mathbf{1}_\ell) \cdot \mathbf{q}^T = c$. Thus,

$$\mathbf{p} \cdot M_f \cdot \mathbf{q}^T = \mathbf{p} \cdot \left(M_f \cdot \mathbf{q}^T \right) = \mathbf{p} \cdot \left(\delta_2 \cdot \mathbf{1}_m^T \right) = \delta_2$$

and

$$\mathbf{p} \cdot M_f \cdot \mathbf{q}^T = \left(\mathbf{p} \cdot M_f \right) \cdot \mathbf{q}^T = \left(\delta_1 \cdot \mathbf{1}_\ell \right) \cdot \mathbf{q}^T = \delta_1,$$

implying that $\delta_1 = \delta_2$. ∎

Note that a function can be both δ_2-right-balanced and δ_2'-right-balanced for some $\delta_2 \neq \delta_2'$. For example, consider the function M_f, which was defined above. It is easy to see that the function is δ_2-right-balanced for every $1/2 \leq \delta_2 \leq 1$ (by multiplying with the probability vector $(1-\delta_2, 1-\delta_2, 2\delta_2 - 1)^T$ from the right). Nevertheless, in cases where a function is δ_2-right-balanced for multiple values, Proposition 3.2 implies that the function cannot be left-balanced for *any* δ_1. Likewise, if a function is δ_1-left balanced for more than one value of δ_1, it cannot be right-balanced.

3.2 The Criterion

The criterion for determining whether or not a function implies coin-tossing is simply the question of whether the function is *strictly* δ-balanced for some δ. Formally:

Property 3.3. *A function $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0,1\}$ is* strictly balanced *if it is δ-balanced for some $0 < \delta < 1$.*

Observe that if M_f has a monochromatic row (i.e., there exists an input x such that for all y_i, y_j it holds that $f(x, y_i) = f(x, y_j)$), then there exists a probability vector $\mathbf{p}$ such that $\mathbf{p} \cdot M_f = 0 \cdot \mathbf{1}_\ell$ or $\mathbf{p} \cdot M_f = 1 \cdot \mathbf{1}_\ell$; likewise for a monochromatic column. Nevertheless, we stress that the existence of such a row and column does not imply f is strictly balanced since it is required that δ be strictly between 0 and 1, and not equal to either.

3.3 Exploring the δ-Balanced Property

In this section we prove some technical lemmas regarding the property that we will need later in the proof. First, we show that if a function f is not left-balanced for any $0 \leq \delta \leq 1$ (resp. not right balanced), then it is *not close* to being balanced. More precisely, it seems possible that a function f can be not δ-balanced, but is only negligibly far from being balanced (i.e., there may exist some probability vector $\mathbf{p} = \mathbf{p}(n)$ (that depends on the security parameter n) such that all the values in the vector $\mathbf{p} \cdot M_f$ are at most negligibly far from δ, for some $0 \leq \delta \leq 1$). In the following claim, we show that this situation is impossible. Specifically, we show that if a function is not δ balanced, then there exists some constant $c > 0$, such that for any probability vector $\mathbf{p}$, there is a distance of at least c between two values in the vector $\mathbf{p} \cdot M_f$. This holds also for probability vectors that are functions of the security parameter n (as can be the case in our setting of secure protocols).

Lemma 3.4. *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0,1\}$ be a function that is not left balanced for any $0 \leq \delta_1 \leq 1$ (including $\delta_1 = 0, 1$). Then, there exists a constant $c > 0$, such that for any probability vector $\mathbf{p} = \mathbf{p}(n)$, it holds that:*

$$\max_i(\delta_1, \ldots, \delta_\ell) - \min_i(\delta_1, \ldots, \delta_\ell) \geq c$$

where $(\delta_1, \ldots, \delta_\ell) = \mathbf{p} \cdot M_f$, and M_f is the matrix representation of f.

Proof: Let P^m be the set of all probability vectors of size m. That is, $\mathrm{P}^m \subseteq [0,1]^m$ (which itself is a subset of $\mathbb{R}^m$), and each vector sums up to one. P^m is a closed and bounded space. Therefore using the Heine-Borel theorem, P^m is a compact space.

We start by defining a function $\phi : \mathrm{P}^m \to [0,1]$ as follows:

$$\phi(\mathbf{p}) = \max_i(\mathbf{p} \cdot M_f) - \min_i \cdot(\mathbf{p} \cdot M_f)$$

Clearly, the function $\mathbf{p} \cdot M_f$ (where M_f is fixed and $\mathbf{p}$ is the variable) is a continuous function. Moreover, the maximum (resp. minimum) of a continuous function is itself a continuous function. Therefore, from composition of continuous functions we have that the function ϕ is continuous. Using the extreme value theorem (a continuous function from a compact space to a subset of the real numbers attains its maximum and minimum), there exists some probability vector $\mathbf{p}_{\min}$ for which for all $\mathbf{p} \in \mathrm{P}^m$, $\phi(\mathbf{p}_{\min}) \leq \phi(\mathbf{p})$. Since f is not δ-balanced, $\mathbf{p}_{\min} \cdot M_f \neq \delta \cdot \mathbf{1}_\ell$ for any $0 \leq \delta \leq 1$, and so $\phi(\mathbf{p}_{\min}) > 0$. Let $c \stackrel{\text{def}}{=} \phi(\mathbf{p}_{\min})$. This implies that for any probability vector $\mathbf{p}$, we have that $\phi(\mathbf{p}) \geq \phi(\mathbf{p}_{\min}) = c$. That is:

$$\max_i(\delta_1, \ldots, \delta_\ell) - \min_i(\delta_1, \ldots, \delta_\ell) \geq c \tag{1}$$

where $(\delta_1, \ldots, \delta_\ell) = \mathbf{p} \cdot M_f$. We have proven this for *all* probability vectors of size m. Thus, it holds also for every probability vector $\mathbf{p}(n)$ that is a function of n, and for all n's (this is true since for every n, $\mathbf{p}(n)$ defines a concrete probability vector for which Eq. (1) holds). ∎

A similar claim holds for the case where f is not right balanced.

4 Strictly-Balanced Functions Imply Coin Tossing

In this section, we show that any function f that is strictly balanced can be used to fairly toss a coin. Intuitively, this follows from the well known method of Von Neumann [12] for obtaining an unbiased coin toss from a biased one. Specifically, given a coin that is heads with probability ϵ and tails with probability $1-\epsilon$, Von Neumann showed that you can toss a coin that is heads with probability exactly $1/2$ by tossing the coin twice in each phase, and stopping the first time that the pair is either heads-tails or tails-heads. Then, the parties output heads if the pair is heads-tails, and otherwise they output tails. This gives an unbiased coin because the probability of heads-tails equals the probability of tails-heads (namely, both probabilities equal $\epsilon \cdot (1-\epsilon)$). Now, since the function f is strictly δ-balanced it holds that if party P_1 chooses its input via the probability vector $(p_1, \ldots, p_m)$ then the output will equal 1 with probability δ, irrespective of what input is used by P_2; likewise if P_2 chooses its input via $(q_1, \ldots, q_\ell)$ then the output will be 1 with probability δ irrespective of what P_1 does. This yields a coin that equals 1 with probability δ and thus Von Neumann's method can be applied to achieve unbiased coin tossing. We stress that if one of the parties aborts early and refuses to participate, then the other party proceeds by itself (essentially, tossing a coin with probability δ until it concludes). We have the following theorem:

Theorem 4.1. *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0,1\}$ be a strictly-balanced function for some constant $0 < \delta < 1$, as in Property 3.3. Then, f information-theoretically and computationally implies the coin-tossing functionality f^{ct} with malicious adversaries.*

Application to Fairness. Cleve [5] showed that there does not exist a protocol that securely computes the fair coin-tossing functionality in the plain model. Since any strictly-balanced function f implies the coin-tossing functionality, a protocol for f in the plain model implies the existence of a protocol for coin-tossing in the plain model. We therefore conclude:

Corollary 4.2. *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0,1\}$ be a strictly-balanced function. Then, f cannot be securely computed with fairness (with computational or information-theoretic security).*

5 Unbalanced Functions Do Not Information-Theoretically Imply Coin Tossing

We now show that any function f that is *not* strictly-balanced (for all δ) does *not* information-theoretically imply the coin-tossing functionality. Stated differently, there does not exist a protocol for fairly tossing an unbiased coin in the f-hybrid model, with statistical security. Observe that in the f-hybrid model, it is possible for the parties to simultaneously exchange information, in some sense, since both parties receive output from calls to f at the same time. Thus, Cleve-type arguments [5] that are based on the fact that one party must know more than

the other party at some point do not hold. We prove our result by showing that for every protocol there exists an unbounded malicious adversary that can bias the result. Our unbounded adversary needs to compute probabilities, which can actually be approximated given an $\mathcal{NP}$-oracle. Thus, it is possible to interpret our technical result also as a black-box separation, if desired.

As we have mentioned in the introduction, although we prove an "impossibility result" here, the implication is the opposite. Specifically, our proof that an unbalanced[3] f cannot be used to toss an unbiased coin implies that it may be possible to securely compute such functions with fairness. Indeed, the functions that were shown to be securely computable with fairness in [9] are unbalanced.

Recall that a function is not strictly balanced if is not δ-balanced for any $0 < \delta < 1$. We treat the case that the function is not δ-balanced at all separately from the case that it *is* δ-balanced but for $\delta = 0$ or $\delta = 1$. In the proof we show that in both of these cases, such a function cannot be used to construct a fair coin tossing protocol.

Theorem 5.1. *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0, 1\}$ be a function that is* not *left-balanced, for any $0 \leq \delta_1 \leq 1$. Then, f does not information-theoretically imply the coin-tossing functionality with malicious adversaries.*

Proof Idea: We begin by observing that if f does not contain an embedded OR (i.e., inputs x_0, x_1, y_0, y_1 such that $f(x_0, y_0) = f(x_1, y_0) = f(x_0, y_1) \neq f(x_1, y_1)$) or an embedded XOR (i.e., inputs x_0, x_1, y_0, y_1 such that $f(x_0, y_0) = f(x_1, y_1) \neq f(x_0, y_1) = f(x_1, y_0)$), then it is trivial and can be computed by simply having one party send the output to the other. This is because such a function depends only on the input of one party. Thus, by [5], it is impossible to fairly toss an unbiased coin in the f-hybrid model, since this is the same as fairly tossing an unbiased coin in the plain model. Thus, we consider only functions f that have an embedded OR or an embedded XOR.

In addition, we consider coin-tossing protocols that consist of calls to f only, and no other messages. This is due to the fact that we can assume that any protocol consists of rounds, where each round is either an invocation of f or a message consisting of a single bit being sent from one party to the other. Since f has an embedded OR or an embedded XOR, messages of a single bit can be sent by invoking f. This is due to the fact that in both cases there exist inputs x_0, x_1, y_0, y_1 such that $f(x_1, y_0) \neq f(x_1, y_1)$ and $f(x_0, y_1) \neq f(x_1, y_1)$. Thus, in order for P_2 to send P_1 a bit, the protocol can instruct the parties to invoke f where P_1 always inputs x_1, and P_2 inputs y_0 or y_1 depending on the bit that it wishes to send; similarly for P_1. Thus, any non-trivial function f enables "bit transmission" in the above sense. Observe that if one of the parties is malicious and uses an incorrect input, then this simply corresponds to sending an incorrect bit in the original protocol.

Intuition. The fact that f is not balanced implies that in any single invocation of f, one party is able to have some effect on the output by choosing its input

[3] Note, that the name unbalanced is a bit misleading as the complement of not being strictly balanced also includes being 1 or 0-balanced.

appropriately. That is, if the function is non-balanced on the left then the party on the right can use an input not according to the prescribed distribution in the protocol, and this will change the probability of the output of the invocation being 1 (for example). However, it may be possible that the ability to somewhat influence the output in individual invocations is not sufficient to bias the overall computation, due to the way that the function calls are composed. Thus, in the proof we need to show that an adversary is in fact capable of biasing the overall protocol. We demonstrate this by showing that there exist crucial invocations where the ability to bias the outcome in these invocation suffice for biasing the overall outcome. Then, we show that such invocations are always reached in any execution of the protocol, and that the adversary can (inefficiently) detect when such an invocation has been reached and can (inefficiently) compute which input it needs to use in that invocation in order to bias the output.

We prove the above by considering the execution tree of the protocol, which is comprised of calls to f and the flow of the computation based on the output of f in each invocation (i.e., the parties proceed left in the tree if the output of f in this invocation is 0; and right otherwise). Observe that a path from the root of the tree to a leaf-node represents a protocol execution. We show that in *every* path from the root to a leaf, there exists at least one node with the property that influencing the output of the single invocation of that node yields a bias in the final outcome. In addition, we describe the strategy of the adversary to detect such a node and choose its input for that node in order to obtain a bias.

In more detail, for every node v in the execution tree of the protocol, the adversary calculates (in an inefficient manner) the probability that the output of the computation equals 1, assuming that v is reached in the execution. Observe that the probability of obtaining 1 at the root node is at most negligibly far from 1/2 (since it is a secure coin-tossing protocol), and that the probability of obtaining 1 at a leaf node is either 1 or 0, depending on whether the output at the given leaf is 1 or 0 (the way that we define the tree is such that the output is fully determined by the leaf). Using a pigeon-hole like argument, we show that on every path from the root to a leaf there must be at least one node where the probability of outputting 1 given that this node is reached is significantly different than the probability of outputting 1 given that the node's child on the path is reached. We further show that this difference implies that the two children of the given node yield significantly different probabilities of outputting 1 (since the probability of outputting 1 at a node v is the weighted-average of outputting 1 at the children, based on the probability of reaching each child according to the protocol). This implies that in every protocol execution, there exists an invocation of f where the probability of outputting 1 in the entire protocol is significantly different if the output of this invocation of f is 0 or 1. Since f is not balanced, it follows that for any distribution used by the honest party to choose its input for this invocation, there exist two inputs that the corrupted party can use that result in significantly different probabilities of obtaining 1. In particular, at least one of these probabilities is *significantly different from the probability of obtaining 1 in this call when both parties are honest and follow the*

protocol.[4] Thus, the adversary can cause the output of the entire execution to equal 1 with probability significantly different than 1/2, which is the probability when both parties play honestly.

The above description does not deal with question of whether the output will be biased towards 0 or 1. In fact we design two adversaries, one that tries to bias the output towards 0 and the other towards 1. Then we show that at least one of these adversaries will be successful (see Footnote 4 for an explanation as to why only one of the adversaries may be successful). The two adversaries are similar and very simple. They search for the node on the path of the execution where the bias can be created and there make their move. In all nodes until and after that node they behave honestly (i.e., choose inputs for the invocations of f according to the input distribution specified by the protocol). We analyze the success of the adversaries and show that at least one of them biases the output with noticeable probability. The full proof appears in [1]. ■

The above theorem proves impossibility for the case that the function is not balanced. As we have mentioned, we must separately deal with the case that the function *is* balanced, but not *strictly* balanced; i.e., the function is either 0-balanced or 1-balanced. The main difference in this case is that not all nodes which have significantly different probabilities in their two children can be used by the adversary to bias the outcome. This is due to the fact that the protocol may specify an input distribution for the honest party at such a node that forces the output to be either 0 or 1 (except with negligible probability), and so the "different child" is only reached with negligible probability. This can happen since the function *is* balanced with $\delta = 0$ or $\delta = 1$. The proof therefore shows that this cannot happen too often, and the adversary can succeed enough to bias the output. The following is proven in [1]:

Theorem 5.2. *Let* $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0, 1\}$ *be a* 1*-balanced or a* 0*-balanced function. Then,* f *does not information-theoretically imply the coin-tossing protocol.*

Conclusion: Combining Theorems 4.1, 5.1 and 5.2, we obtain the following:

Corollary 5.3. *Let* $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0, 1\}$ *be a function.*

1. *If* f *is strictly-balanced, then* f *implies the coin-tossing functionality (computationally and information theoretically).*
2. *If* f *is not strictly-balanced, then* f *does not information-theoretically imply the coin-tossing functionality with malicious adversaries.*

Impossibility in the OT-Hybrid Model. Our proof of impossibility holds only in the information-theoretic setting since the adversary must carry out computations that do not seem to be computable in polynomial-time. It is natural

[4] Observe that one of these probabilities may be the same as the probability of obtaining 1 in an honest execution, in which case choosing that input will not result in any bias. Thus, the adversary may be able to bias the output of the entire protocol towards 1 or may be able to bias the output of the entire protocol towards 0, but not necessarily both.

to ask whether or not the impossibility result still holds in the computational setting. We do not have an answer to this question. However, as a step in this direction, we show that the impossibility still holds if the parties are given access to an ideal oblivious transfer (OT) primitive as well as to the function f. That is, we prove the following:

Theorem 5.4. *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0, 1\}$ be a function. If f is not strictly-balanced, then the pair of functions (f, OT) do not information-theoretically imply the coin tossing functionality with malicious adversaries.*

Proof: In order to see that this is the case, first observe that if f has an embedded-OR then it implies oblivious transfer [10]. Thus, f can be used to obtain OT, and so the question of whether f implies coin tossing or (f, OT) imply coin tossing is the same. It thus remains to consider the case that f does not have an embedded OR but does have an embedded XOR (if it has neither then it is trivial and so clearly cannot imply coin tossing, as we have mentioned). We now show that in such a case f must be strictly balanced, and so this case is not relevant. Let x_1, x_2, y_1, y_2 be an embedded XOR in f; i.e., $f(x_1, y_1) = f(x_2, y_2) \neq f(x_1, y_2) = f(x_2, y_1)$. Now, if there exists a y_3 such that $f(x_1, y_3) = f(x_2, y_3)$ then f has an embedded OR. Thus, x_1 and x_2 must be complementary rows (as in example function (a) in the Introduction). Likewise, if there exists an x_3 such that $f(x_3, y_1) = f(x_3, y_2)$ then f has an embedded OR. Thus, y_1 and y_2 must be complementary columns. We conclude that f has two complementary rows and columns, and as we have shown in the Introduction, this implies that f is strictly balanced with $\delta = \frac{1}{2}$. ■

6 Fairness in the Presence of Fail-Stop Adversaries

In order to study the feasibility of achieving fair secure computation in the fail-stop model, we must first present a definition of security for this model. To the best of our knowledge, there is no simulation-based security definition for the fail-stop model in the literature. As we have mentioned in the introduction, there are two natural ways of defining security in this model, and it is not clear which is the "correct one". We therefore define two models and study feasibility for both. In the first model, the ideal-model adversary/simulator must either send the party's prescribed input to the trusted party computing the function, or a special abort symbol $\perp$, but nothing else. This is similar to the semi-honest model, except that $\perp$ can be sent as well. We note that if $\perp$ is sent, then both parties obtain $\perp$ as output and thus fairness is preserved.[5] This is actually a very strong requirement from the protocol since both parties either learn the prescribed output, or they both output $\perp$. In the second model, the ideal adversary can send any input

[5] It is necessary to allow an explicit abort in this model since if the corrupted party does not participate at all then the output cannot be computed. The typical solution to this problem, which is to take some default input, is not appropriate here because this means that the simulator can change the input of the corrupted party. Thus, such an early abort must result in output $\perp$.

that it wishes to the trusted party, just like a malicious adversary. We remark that if the real adversary does not abort a real protocol execution, then the result is the same as an execution of two honest parties and thus the output is computed from the prescribed inputs. This implies that the ideal adversary can really only send a different input in the case that the real adversary halts before the protocol is completed. As we have mentioned in the Introduction, the impossibility result of Cleve [5] for coin-tossing holds in both models, since the parties have no input, and so for this functionality the models are identical.

6.1 Fail-Stop 1

In this section we define and explore the first fail-stop model.

Execution in the Ideal World. An ideal execution involves parties P_1 and P_2, an adversary $\mathcal{S}$ who has corrupted one of the parties, and the trusted party. An ideal execution for the computation of f proceeds as follows:

Inputs: P_1 and P_2 hold inputs $x \in X$, and $y \in Y$, respectively; the adversary $\mathcal{S}$ receives the security parameter 1^n and an auxiliary input z.

Send Inputs to Trusted Party: The honest party sends its input to the trusted party. The corrupted party controlled by $\mathcal{S}$ may send its prescribed input or $\perp$.

Trusted Party Sends Outputs: If an input $\perp$ was received, then the trusted party sends $\perp$ to both parties. Otherwise, it computes $f(x, y)$ and sends the result to both parties.

Outputs: The honest party outputs whatever it was sent by the trusted party, the corrupted party outputs nothing and $\mathcal{S}$ outputs an arbitrary function of its view.

We denote by $\mathrm{IDEAL}^{\text{f-stop-1}}_{f,\mathcal{S}(z)}(x, y, n)$ 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.

Security. The real model is the same as is defined in the standard definition of secure two-party computation [6], except that we consider adversaries that are fail-stop only. This means that the adversary must behave exactly like an honest party, except that it can halt whenever it wishes during the protocol. We stress that its decision to halt or not halt, and when, may depend on its view. We are now ready to present the security definition.

Definition 6.1 (Security – Fail-Stop1). *Protocol π* securely computes *f* with complete fairness in the fail-stop1 model *if for every non-uniform probabilistic polynomial-time fail-stop adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that:*

$$\left\{\mathrm{IDEAL}^{\text{f-stop-1}}_{f,\mathcal{S}(z)}(x, y, n)\right\} \stackrel{\mathrm{c}}{\equiv} \left\{\mathrm{REAL}_{\pi,\mathcal{A}(z)}(x, y, n)\right\}$$

where $x \in X$, $y \in Y$, $z \in \{0,1\}^*$ and $n \in \mathbb{N}$.

Exploring Fairness in the Fail-stop-1 Model. We first observe that if a function contains an embedded XOR, then it cannot be computed fairly in this model.

Theorem 6.2. *Let $f : \{x_1, \ldots, x_m\} \times \{y_1, \ldots, y_\ell\} \to \{0,1\}$ be a function that contains an embedded XOR. Then, f implies the coin-tossing functionality and thus cannot be computed fairly.*

Proof: Assume that f contains an embedded XOR; i.e., there exist inputs x_1, x_2, y_1, y_2 such that $f(x_1, y_1) = f(x_2, y_2) \neq f(x_1, y_2) = f(x_2, y_1)$. We can easily construct a protocol for coin-tossing using f that is secure in the fail-stop model. Party P_1 chooses input $x \in \{x_1, x_2\}$ uniformly at random, P_2 chooses $y \in \{y_1, y_2\}$ uniformly at random, and the parties invoke the function f where P_1 inputs x and P_2 inputs y. In case the result of the invocation is $\perp$, the other party chooses its output uniformly at random.

Since the adversary is fail-stop1, it must follow the protocol specification (including choosing its input in the invocation of f correctly until it aborts when it can input $\perp$). In both cases, it is easy to see that the honest party outputs an unbiased coin. Formally, for any given fail-stop adversary $\mathcal{A}$ we can construct a simulator $\mathcal{S}$: $\mathcal{S}$ receives from the coin tossing functionality f^{ct} the bit b, and invokes the adversary $\mathcal{A}$. If $\mathcal{A}$ sends the trusted party computing f the symbol $\perp$, then $\mathcal{S}$ responds with $\perp$. Otherwise, (if $\mathcal{A}$ sends some real value - either x_1, x_2 if it controls P_1, or y_1, y_2 if it controls P_2), then $\mathcal{S}$ responds with the bit b that it received from f^{ct} as if it is the output of the ideal call to f. It is easy to see that the ideal and real distributions are identical. ■

As we have mentioned, if a function does not contain an embedded XOR or OR then it is trivial and can be computed fairly (because the output depends on only one of the parties' inputs). It therefore remains to consider the feasibility of fairly computing functions that have an embedded OR but no embedded XOR. Gordon et. al [9] present a protocol for securely computing any function of this type with complete fairness, in the presence of a malicious adversary. However, the security of their protocol relies inherently on the ability of the simulator to send the trusted party an input that is not the corrupted party's prescribed input. Thus, their protocol seems not to be secure in this model.

The problem of securely computing functions that have an embedded OR but no embedded XOR therefore remains open. We remark that there are very few functions of this type, and these functions have a very specific structure, as discussed in [9].

6.2 Fail-Stop 2

In this section we define and explore the second fail-stop model. In this case, the ideal adversary can send any value it wishes to the trusted party (and the output of the honest party is determined accordingly). It is easy to see that in executions where the real adversary does not abort the output is the same as between two honest parties. Thus, the ideal adversary is forced to send the

prescribed input of the party in this case. Observe that the ideal model here is identical to the ideal model for the case of malicious adversaries. Thus, the only difference between this definition and the definition of security for malicious adversaries is the quantification over the real adversary; here we quantify only over fail-stop real adversaries. Otherwise, all is the same.

Definition 6.3 (Security – Fail-Stop2). *Protocol* π securely computes f with complete fairness in the fail-stop2 model *if for every non-uniform probabilistic polynomial-time fail-stop adversary* $\mathcal{A}$ *in the real world, there exists a non-uniform probabilistic polynomial-time adversary* $\mathcal{S}$ *in the ideal model such that:*

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z)}(x,y,n)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z)}(x,y,n)\right\}$$

where $x \in X$*,* $y \in Y$*,* $z \in \{0,1\}^*$ *and* $n \in \mathbb{N}$*, and* IDEAL *denotes the standard ideal model for malicious adversaries.*

In the g-hybrid-model for fail-stop2 adversaries, where the parties have access to a trusted party computing function g for them, a corrupted party may provide an incorrect input to an invocation of g as long as it halts at that point. This may seem arbitrary. However, it follows naturally from the definition since a secure fail-stop2 protocol is used to replace the invocations of g in the real model. Thus, if a fail-stop adversary can change its input as long as it aborts in the real model, then this capability is necessary also for invocations of g in the g-hybrid model.

Exploring Fairness in the Fail-Stop-2 Model. In the following we show that the malicious adversaries that we constructed in the proofs of Theorem 5.1 and Theorem 5.2 can be modified to be *fail-stop2*. We remark that the adversaries that we constructed did not abort during the protocol execution, but rather continued after providing a "different" input in one of the f invocations. Thus, they are not fail-stop2 adversaries. In order to prove the impossibility for this case, we need to modify the adversaries so that they *halt* at the node v for which they can bias the outcome of the invocation (i.e., a node v for which v's children in the execution tree have significantly different probabilities for the output of the entire execution equalling 1). Recall that in this fail-stop model, the adversary is allowed to send a different input than prescribed in the invocation at which it halts; thus, this is a valid attack strategy. In the full paper we prove:

Theorem 6.4. *Let* $f:\{x_1,\ldots,x_m\}\times\{y_1,\ldots,y_\ell\} \to \{0,1\}$ *be a function that is* not δ*-balanced, for any* $0 < \delta < 1$*. Then,* f *does not information-theoretically imply the coin-tossing protocol in the fail-stop2 model.*

We prove the above by considering two possible cases, relating to the potential difference between the honest party outputting 1 at a node when the other party aborts at that node but until then was fully honest, or when the other party continues honestly from that node (to be more exact, we consider the average of these differences over all nodes). First, assume that there is a noticeable difference between an abort after fully honest behavior and a fully honest execution.

In this case, we construct a fail-stop adversary who plays honestly until an appropriate node where such a difference occurs and then halts. (In fact, such an adversary is even of the fail-stop1 type). Next, assume that there is *no noticeable difference* between an abort after fully honest behavior and a fully honest execution. Intuitively, this means that continuing honestly or halting makes no difference. Thus, if we take the malicious adversaries from Section 5 and modify them so that they halt immediately after providing malicious input (as allowed in the fail-stop2 model), then we obtain that there is no noticeable difference between the original malicious adversary and the fail-stop2 modified adversary. We remark that this is not immediate since the difference in this case is between aborting and not aborting without giving any malicious input. However, as we show, if there is no difference when honest inputs are used throughout, then this is also no difference when a malicious input is used.

We conclude that one of the two types of fail-stop2 adversaries described above can bias any protocol.

Acknowledgements. We thank Gene S. Kopp and John D. Wiltshire-Gordon for helpful discussions.

References

1. Full version of this work. Cryptology ePrint Archive (2013)
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: 20th STOC, pp. 1–10 (1988)
3. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13(1), 143–202 (2000)
4. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: The 20th STOC, pp. 1–10 (1988)
5. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: The 18th STOC, pp. 364–369 (1986)
6. Goldreich, O.: The Foundations of Cryptography. Basic Applications, vol. 2. Cambridge University Press (2004)
7. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. JACM 38(1), 691–729 (1991)
8. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: The 19th STOC, pp. 218–229 (1987)
9. Gordon, S.D., Hazay, C., Katz, J., Lindell, Y.: Complete fairness in secure two-party computation. JACM 58(6), 24 (2011)
10. Kilian, J.: A general completeness theorem for two-party games. In: 23rd STOC, pp. 553–560 (1991)
11. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: The 21st STOC, pp. 73–85 (1989)
12. von Neumann, J.: Various Techniques Used in Connection with Random Digits. Journal of Research of the National Bureau of Standards 12, 36–38 (1951)
13. Yao, A.: How to generate and exchange secrets (extended abstract). In: The 27th FOCS, pp. 162–167 (1986)