On Complete Primitives for Fairness

Dov Gordon[1,⋆], Yuval Ishai[2,3,⋆⋆], Tal Moran[4], Rafail Ostrovsky[3,⋆⋆⋆], and Amit Sahai[3,†]

[1] University of Maryland, gordon@cs.umd.edu
[2] Technion, Israel, yuvali@cs.technion.il
[3] University of California, Los Angeles, {rafail,sahai}@cs.ucla.edu
[4] Harvard SEAS, talm@seas.harvard.edu

Abstract. For secure two-party and multi-party computation with abort, classification of which primitives are *complete* has been extensively studied in the literature. However, for *fair* secure computation, where (roughly speaking) either all parties learn the output or none do, the question of complete primitives has remained largely unstudied. In this work, we initiate a rigorous study of completeness for primitives that allow fair computation. We show the following results:

- **No "short" primitive is complete for fairness.** In surprising contrast to other notions of security for secure two-party computation, we show that for fair secure computation, no primitive of size $O(\log k)$ is complete, where k is a security parameter. This is the case even if we can enforce parallelism in calls to the primitives (i.e., the adversary does not get output from any primitive in a parallel call until it sends input to all of them). This negative result holds regardless of any computational assumptions.
- **A fairness hierarchy.** We clarify the fairness landscape further by exhibiting the existence of a "fairness hierarchy". We show that for every "short" $\ell = O(\log k)$, no protocol making (serial) access to any ℓ-bit primitive can be used to construct even a $(\ell + 1)$-bit simultaneous broadcast.
- **Positive results.** To complement the negative results, we exhibit a k-bit primitive that *is* complete for two-party fair secure computation. We show how to generalize this result to the multi-party setting.
- **Fairness combiners.** We also introduce the question of constructing a protocol for fair secure computation from primitives that may be faulty. We show that this is possible when a majority of the instances are honest. On the flip side, we show that this result is tight: no functionality is complete for fairness if half (or more) of the instances can be malicious.

⋆ Work supported by US-Israel Binational Science Foundation grant 2004240 and NSF grant 0830464.

⋆⋆ Supported in part by ISF grant 1310/06, BSF grants 2008411, and NSF grants 0627781, 0716835, 0716389, 0830803, 0916574.

⋆⋆⋆ Supported in part by IBM Faculty Award, Xerox Innovation Group Award, the Okawa Foundation Award, Intel, Teradata, BSF grant 2008411, NSF grants 0716835, 0716389, 0830803, 0916574 and U.C. MICRO grant.

† Research supported in part from BSF grant 2008411, NSF grants 0916574, 0830803, 0627781 and 0716389, an equipment grant from Intel and an Okawa Foundation Research Grant.

1 Introduction

In the setting of secure multi-party computation, n participants wish to jointly compute a function while maintaining several security properties, such as the privacy of their inputs, correctness of the outputs, and others. The security of their computation is defined by comparing their view in the protocol to an ideal world — one that embodies complete security — and proving that they are indistinguishable to an outside observer. In this ideal world there is an additional trusted party that privately receives the inputs from all participants, performs the computation on their behalf, and returns the outputs.

In the real world, of course, relying on an outside party is undesirable. In some of the most fundamental results in modern cryptography, many beautiful techniques have been developed in order to remove the trusted party while retaining most of the security properties that he affords. In fact, in the two-party setting there is only one security property for which the trusted third party has remained essential. Informally, we say a computation is *fair* if either all players receive their output or none of them do. It is easy to see that in the ideal world, where there is an additional trusted party, computations are always fair. This is a difficult property to achieve in the real world, as a malicious player may abort the protocol prematurely.

In 1986, Richard Cleve [20] proved that, for general functionalities, fairness is impossible to achieve unless the majority of parties is honest. Specifically, he showed that even the very basic functionality of coin-tossing cannot be fairly computed by a two-party protocol. Recalling that fairness is immediate with the help of a third party, in this paper we address a very natural question.

> What is the *minimum* amount of help required to be able to compute all functions fairly?

We think of this helper as a naive black box, or a *primitive*, with no knowledge of the function being computed. It is charged with a fixed task: it takes inputs from each player, and then simultaneously outputs some fixed function of the inputs to all players. Clearly we can compute any function fairly if this primitive is sufficiently complex: we can simply define its input to be a description of the function being computed, along with the inputs to that function. (Indeed, this was demonstrated by Fitzi et al. [26], as discussed below.) However, our interest is in reducing the complexity of these primitives. In particular, we study the minimum input size to such primitives that will enable the fair computation of any function.

Interestingly, there has been extensive research on very similar questions in the context of *unfair* secure computation. When there is no honest majority among the players, it is known that oblivious transfer is both necessary and sufficient for computational security (without fairness) [51, 31, 42]. Similarly, in addition to the impossibility of attaining fairness without an honest majority, it is also known that we cannot achieve information-theoretic security in this setting [19], and there is a long line of research identifying the minimum primitives that enable information theoretic security (without fairness). Surprisingly, very little work has addressed the parallel questions with respect to fairness.

1.1 Our Results

The main theme that recurs throughout our results is that when looking at primitives for fair computation, input size matters. We classify primitives according to the length of the inputs provided by the two parties: a k-bit primitive accepts inputs of size k from both parties.

No "Short" Primitive is Complete for Fairness For many other notions of security which are unrealizable in the plain model (such as unconditional security and UC security) there exist *finite* functionalities which are complete for that setting (e.g., [32, 39, 41, 42]). In other words, to enable (unfair) secure computation in these models, it suffices to give access to a trusted implementation of some simple function with short inputs. Surprisingly, we show that no primitive of size $O(\log \kappa)$ is complete for fair computation (where κ is the security parameter). Our impossibility result holds even if we allow parallel calls to the primitives (where the adversary does not get output from any primitive in a parallel call until it sends input to all of them), and even if the adversary may only deviate from the protocol by aborting early.

Coin-Flipping and Simultaneous Broadcast are not Complete for Fairness Coin-flipping is perhaps the simplest fair functionality that is known to be unrealizable in the plain model [20]. Simultaneous broadcast is important because it is one of the most natural candidates for a complete primitive for fairness. It is often the first thing that comes to mind when thinking about how to construct a fair protocol. Lending weight to this intuition, Katz proved that simultaneous broadcast is complete for *partial fairness* [40]. Finally, although we know that is also unrealizable in the plain model (even a 1-bit simultaneous broadcast implies fair coin-flipping), it can be constructed from conceivable non-standard assumptions such as timed commitments [12] or physical limits on signal propagation. Surprisingly, our results imply that simultaneous broadcast of any size (as well as coin-flipping) is not complete for fair computation (this is a direct corollary of our "No Short Primitive" result).

A Fairness Hierarchy We clarify the fairness landscape further by exhibiting the existence of a "fairness hierarchy." We show that for every "short" k (when the adversary can run in time poly(2^k)), no k-bit primitive can be used to construct even a $k + 1$-bit simultaneous broadcast. This result is almost tight: given a "long" k'-bit simultaneous broadcast and a semantically-secure encryption scheme with keys of length k', we can construct a simultaneous broadcast of any length by first exchanging encrypted inputs and then exchanging keys. The fairness hierarchy complements the first proof of impossibility for achieving fair computation through short primitives (described above). The latter demonstrates that there exists *some* function that cannot be fairly computed, even with (parallel) access to a short primitive. In the hierarchy result, we only consider serial access to a short, k-bit primitive, but we demonstrates that it does not enable fair computation for even the most simple $k + 1$ bit primitives.

Positive Results Previous work by Fitzi et al. [26] proposed the *Universal Black Box* (UBB) primitive and showed that is is complete for fair secure computation in both the two-party and multi-party settings. However, the UBB primitive requires as input a description of functionality to be computed (a more detailed description appears in Sec. 3.2). Thus, its input size and running time depend on the complexity of the target functionality.

In this paper, we show that there exists a much simpler primitive that is complete for two-party fair computation. This primitive implements a "Fair Reconstruction" procedure for a secret sharing scheme. Before calling the primitive, the parties first run an unfair secure computation that outputs *non-malleable secret shares* of the desired function's output. They then call the primitive using these secret shares as their input. The input to the primitive depends on a security parameter and on the output size of the functionality being computed (but not on its description). With the addition of computational assumptions (the existence of one-way functions), the input size can be made to depend only on the security parameter.

We also show how this result can be generalized to the multi-party setting. This generalization is straightforward if we are satisfied by fairness without robustness (i.e., if a single malicious party can cause the entire computation to abort). In the full-security case, however, our results exhibit a trade-off between the input size and the number of primitive invocations that may be required to complete the protocol: for n parties, we describe a primitive that has input size $O(n^2)$ but requires $O(n)$ invocations in the worst case, and a more complex primitive that has input size $O(2^n)$, but only requires a single invocation. (In contrast, the input size of the UBB primitive grows with the number of parties, the functionality's input length and description size — however, it requires only a single invocation.)

Fairness Combiners We next consider the orthogonal question of constructing a protocol for fair computation from primitives which may be faulty. Questions of this nature have been studied in the context of many other primitives (e.g., [37, 38]). We show a functionality that is complete for two-party fair computation when the majority of its instances are honest. On the flip side, we show that this result is tight: no functionality is complete for fairness if half (or more) of the instances can be malicious.

1.2 Related Work

The issue of fairness has not been neglected; there has been a lot of diverging work on achieving fairness through relaxations in the security definition [10, 21, 23, 29, 40, 34], relaxations in the communication model [43, 40] and by enabling dispute resolution through a trusted third party [2, 3, 13, 47]. The recent counter-intuitive results of Gordon, Hazay, Katz and Lindell [35, 36], showing that some non-trivial functions *can* be fairly computed in the plain model, have caused a surge of renewed interest in the subject.

Fair Exchange and Contract Signing One of the earliest related problems in the cryptographic literature is that of fairly exchanging items or information. In a fair exchange, either both parties receive the item, or neither party does. Although we are interested in fair computation of arbitrary functions, the problem of fair exchange turns out to be the crux of the matter. Exemplifying this, both the security definitions and the solutions for the more general setting were usually used in this setting first.

In the plain model, Cleve showed that perfectly fair exchange is impossible [20]. Boneh and Naor gave a similar lower bound for fair contract signing [12]. However, relaxed definitions of fairness are still possible to achieve.

One very common relaxation is that each party would have to perform a similar amount of computation to compute its output. This definition is weak enough to be satisfied by protocols in the plain model (using standard cryptographic assumptions), and is usually accomplished by releasing the information "gradually" ([10, 25, 28, 21, 24, 11, 48, 29]). In a similar approach, the output of the parties is masked with noise that decreases over the time, allowing their confidence in the output to increase as the protocol proceeds ([45, 6, 33]).

More recently, the fair exchange problem has been studied in the *optimistic* model: this model, introduced by Asokan [2], uses a trusted third party (TTP) but requires that the TTP be involved in the protocol only if one of the parties is malicious.

A third model was proposed by Chen, Kudla and Paterson [16] and extended by Lindell [44]. In this model fairness is *legally* rather than technically enforceable: the guarantee is the honest party will either receive her output, or a "check" from the other party (for a pre-agreed amount). In order to invalidate the check in court, the paying party will have to reveal information which will allow the honest party to compute her output.

Fairness in General Secure Computation Positive results in all three models have been extended to the general two-party computation setting [51, 13, 44]. Although a complete primitive for fairness is implied in these works, the construction in each is specific to the model. In contrast, our positive result (Thm. 1) is generic: in any of the fairness models above, it is sufficient to implement our simple complete primitive for fairness in order to get generic fair computation in that model.

A fourth relaxation of the fairness definition, *partial fairness*, was proposed by Katz [40]. This definition is phrased in the language of secure computation: informally, a protocol realizes a functionality with ε-partial fairness if there exists an ideal-world simulator whose output cannot be distinguished from the real-world adversary with advantage greater than ε. Katz showed that simultaneous broadcast (SB) is a complete primitive for ε-partial fairness (for any fixed ε). Our positive result uses techniques similar to his.

Gordon and Katz study partial fairness in the plain model [34], and show that even partial fairness is impossible to achieve in general in the plain model. Their proof gives a specific functionality that is complete for (perfect) fairness, and our proof of Thm. 2 has a similar flavor.

In the multi-party computation setting with an honest majority and a broadcast channel, completely fair computation is possible for any functionality, even without computational assumptions [9, 15, 49]. When there is no honest majority, Cleve's lower-bound applies and general fair computation cannot be achieved at all in the plain model. Lepinski, Micali, Peikert, and shelat devised a protocol for completely fair multi-party computation with any number of malicious parties by relying on "envelope" primitives;

communication primitives with special physical properties [43]. Our complete primitives for the multi-party case are possibly less amenable to implementation via simple physical means (such as envelopes), but allow us to separate the unfair computation from the calls to the complete primitive (whereas the two are intertwined in [43]).

We note that we have mentioned only a small fraction of the related work on fair-exchange and fair computation. See [13, 47, 48, 44] for more works in this area.

Classifying Primitives in Secure Computation Our result in Sec. 5 defines a fairness hierarchy, based on the input size of the primitives. While classification based on input size seems less useful in other contexts of secure computation, other hierarchies have been studied. For example, classification according to the number of calls to a primitive [5, 7], classification by privacy level [17] and by reductions to other primitives [46]. While these works may share some of our goals, namely to further understand the theoretical underpinnings of secure computation, their methods are quite different, and they do not address fairness.

2 Definitions

Definition 1 (Non-Malleable Secret Sharing). *A* 2-out-of-2 non-malleable secret sharing scheme *(NMSS scheme) is defined by a pair of algorithms* $(\mathsf{Share}, \mathsf{Rec})$ *with the following properties:*

- $\mathsf{Share}(s, r)$ *returns* 2 *shares,* (s_0, s_1) *(where s_i is the share of the i-th party) such that if r is picked at random, then a single share reveals no information about s.*
- $\mathsf{Rec}(\mathsf{Share}(s, r)) = (s, 0)$ *for every s, r. The second output of* Rec *serves as a flag which is set to 0 if the secret has been successfully reconstructed.*
- *Any attempt by a player to modify their share (independently of the remaining share) is detected with overwhelming probability. Formally, we say that* $(\mathsf{Share}, \mathsf{Rec})$ *is ε-non-malleable if for every secret s, every (computationally unbounded) adversary $\mathcal{A}$ can win the following game with at most ε probability:*
  - *$\mathcal{A}$ corrupts one of the parties.*
  - *Random shares* (s_0, s_1) *from* $\mathsf{Share}(s, r)$ *are given to the* 2 *parties.*
  - *Based on the share $s_{\mathcal{A}}$ it observed, $\mathcal{A}$ computes a new share $s^*_{\mathcal{A}}$.*
  - *$\mathcal{A}$ wins if $s^*_{\mathcal{A}} \neq s_{\mathcal{A}}$ and* $\mathsf{Rec}(s^*_{\mathcal{A}}, s_H) = (s', 0)$ *for some secret s', where s_H is the share received by the uncorrupted party.*

We note that similar notions of robust secret sharing from the literature (cf. [22] and references therein) are weaker in that they define $\mathcal{A}$ to win the above game only if $s' \neq s$. While this weaker notion does not suffice for our purposes, previous constructions (including the ones from [22]) in fact also satisfy our stronger requirement. We include a construction of an NMSS scheme in the full version.

The following functionality will play a role in several of our results:

Definition 2 (Fair Reconstruction). $\mathsf{FairRec}_\ell(x, y)$ *is defined:*

$$\mathsf{FairRec}_\ell(x, y) = \begin{cases} (s, s) & \textit{if } \mathsf{Rec}(x, y) = (s, 0) \\ (\bot, \bot) & \textit{otherwise} \end{cases}$$

where $x, y \in \{0, 1\}^\ell$.

Intuitively, the $\mathsf{FairRec}$ functionality is just a fair implementation of Rec: it takes a non-malleable secret share from each player, and outputs the result of Rec to both players if and only if the secret was successfully reconstructed. We will prove that it is complete for fairness in Section 3.1. Interestingly, it will also play a key role in our proofs of impossibility in Section 4.

Secure Function Evaluation (SFE) We use the standard definitions for secure function evaluation in the standalone model. Due to limited space, we do not repeat them here, but refer the reader to [14] for a complete definition.

The definition of SFE given in [14] guarantees output delivery, as do all of the protocols that we present. Our impossibility results hold even if we only consider a slightly weaker definition of SFE where the adversary is allowed to abort before giving input to the trusted party. In the two-party setting these notions are equivalent. We note that any (polynomial-time computable) functionality can be computed according to a relaxed notion of security in which the adversary receives his output first and may choose to abort immediately afterwards. Since our interest in this work is in *fair* secure-computation, we will always refer to the stronger notion of security described above, except when explicitly stating a computation is "secure-with-abort". We sometimes (informally) refer to a protocol as *fair* when we actually mean that it is *secure* according to this stricter notion (which includes fairness).

Definition 3 (k-bit Primitives). *We say a protocol Π implementing some functionality $\mathcal{F}$ has access to a k-bit primitive g, if in every round of the protocol, the players may submit k-bit inputs to a trusted computation that securely implements g. We write Π^g to make explicit the fact that Π has access to g.*

We often consider k-bit primitives where $k = k(\kappa)$ is a function of the security parameter. In this case, when we say g is a k-bit primitive we mean that g is an infinite sequence of primitives, such that for every $\kappa \in \mathbb{N}$ there is defined a $k(\kappa)$-bit primitive g_κ in that sequence.

We sometimes informally refer to primitives as "short" or "long". A k-bit primitive is considered "short" if $k = O(\log \kappa)$, where κ is the security parameter. A k bit primitive is considered "long" if $k = \Omega(\kappa)$.

Definition 4 (Complete Primitive for Fairness). *For a functionality $\mathcal{F}$ and a primitive g, we say the* fairness of $\mathcal{F}$ reduces to g *if there exists a protocol Π^g that securely computes $\mathcal{F}$. Let C be a class of functionalities. We say that g* is C-complete for fairness *if, for all $\mathcal{F} \in C$, the fairness of $\mathcal{F}$ reduces to g.*

When g is C-complete for fairness and C is the class of all functions, we may omit it and say that g is *complete for fairness*.

Definition 5 (Parallel Primitives). *For a primitive g, we denote $\mathsf{par}_k(g)$ the primitive that consists of k independent copies of g with* enforced parallelism. *The parallelism is enforced in that none of the copies of g in $\mathsf{par}_k(g)$ send output to any party until all k copies have received input from all parties. We use Π_p^g to denote that protocol Π has access to $\mathsf{par}_k(g)$.*

Definition 6 (Simultaneous Broadcast). *This primitive was originally defined by Chor, Goldwasser, Micali and Awerbuch [18], in the context of multi-party computation. In the two-party setting, we define the primitive* Simultaneous Broadcast *(*SB*) that takes one input value from each player and (fairly) swaps them. Formally,* $\mathsf{SB}(x, y) = (y, x)$*. We refer to a k-bit* SB *when the input sizes are at most k-bits long.*

3 Fairness-Complete Primitives

3.1 Fairness-Complete Primitives for Two-Party Computation

In this section we demonstrate an ideal function that is complete for two-party fairness. In order to compute some function $\mathcal{F}(x, y) = \{F_0(x, y), F_1(x, y)\}$ fairly, the parties will first compute a related function $\mathcal{F}'(x, y)$ that provides player i with an *encryption* of $F_i(x, y)$, along with a share of the corresponding decryption key (generated using a 2-out-of-2 NMSS scheme). This reduces the problem to a simple exchange of the secret shares. Of course, if the players exchanged these on their own, one player might abort just at the point of exchange, recovering the decryption key (and thus his output) all alone. Instead, the ideal functionality $\mathsf{FairRec}$ takes the shares from each player and performs the reconstruction fairly; the non-malleability property of the secret sharing scheme enables the functionality to verify that both players have provided correct shares. The details follow:

Theorem 1. *Any two-party functionality $\mathcal{F}$ with output length m can be fairly computed in the OT-hybrid model by using a single call to* $\mathsf{FairRec}_{O(m)}$*. If one-way functions exist, then $\mathcal{F}$ can be fairly computed in the OT-hybrid model with a single call to* $\mathsf{FairRec}_{O(\kappa)}$*.*

We begin by defining a function $\mathcal{F}'$ related to $\mathcal{F}$ in the way described above. Specifically, let $(\mathsf{Enc}, \mathsf{Dec})$ be the encryption and decryption functions for a semantically secure symmetric encryption scheme. When using $\mathsf{FairRec}_{O(m)}$, as in the first part of the theorem, the encryption scheme is a one-time pad (with key length m). When using $\mathsf{FairRec}_{O(\kappa)}$, any semantically-secure symmetric encryption scheme may be used (with key length $O(\kappa)$). Then we define:

$$\mathcal{F}'(x, y) = \left\{F'_0(x, y), F'_1(x, y)\right\} = \{(\mathsf{Enc}_{k_{\mathsf{Enc}}}(F_0(x, y)), s_0), (\mathsf{Enc}_{k_{\mathsf{Enc}}}(F_1(x, y)), s_1)\}$$

where $(s_0, s_1) = \mathsf{Share}(k_{\mathsf{Enc}}, r)$, and r is chosen uniformly at random.

The size of the input to $\mathsf{FairRec}$ is the size of the share of one decryption key. The fair computation of $\mathcal{F}(x, y)$ follows easily:

1. Execute a secure-with-abort protocol to compute $\mathcal{F}'(x, y)$ in the OT-hybrid model (e.g., [41]).
2. Player $i \in \{0, 1\}$ parses the output $F'_i(x, y)$ as
$$z_i = (z_{\mathsf{Enc}}, z_{\mathsf{FairRec}}) = (\mathsf{Enc}_{k_{\mathsf{Enc}}}(F_i(x, y)), \mathsf{Share}(k_{\mathsf{Enc}}, r)_i)$$
and submits z_{FairRec} to the ideal function $\mathsf{FairRec}$
3. Let K_i denote the output that player i receives from $\mathsf{FairRec}$. If $K_i = \bot$, output $\bot$. Otherwise, output $\mathsf{Dec}_{K_i}(z_{\mathsf{Enc}})$.

Due to space limitations, we defer the proof of security for this protocol to the full version of the paper.

Standalone vs. Composable Security Note that the SFE security definitions we use in this paper are in the *standalone* setting, and in particular allow the ideal-world simulator to rewind the adversary. In the first part of Thm. 3.1, using $\mathsf{FairRec}_{O(m)}$ and a one-time pad, this is not a problem: we can use a *straight-line* simulator (that does not rewind the adversary) and prove security even under parallel composition. In the second part of the theorem, however (with a small $\mathsf{FairRec}$ functionality), we use rewinding in an essential way. Thus, $\mathsf{FairRec}_{O(\kappa)}$ may not be complete for fairness under parallel composition. The crux of the problem is that the adversary's actions may depend on the encryption used to reduce the input size to $\mathsf{FairRec}$ (this causes a *selective decommitment* problem). We note that given access to $\mathsf{par}_{O(m)}(\mathsf{FairRec}_{O(\kappa)})$ ($O(m)$ parallel calls to a small fair reconstruction functionality) we can construct a protocol that is provably secure under parallel composition: we can replace the encryption with an all-or-nothing transform of the output of $\mathcal{F}$, each share of which is then split between the parties using a 2-out-of-2 NMSS scheme. These shares are fairly reconstructed in parallel using $\mathsf{par}_{O(m)}(\mathsf{FairRec}_{O(\kappa)})$.

3.2 Fairness-Complete Primitives for Multi-party Computation

If we only cared about achieving fairness, and not about guaranteeing output delivery, then it is possible to extend the protocol of the previous section to multi-party setting in a straightforward way by using n-out-of-n NMSS. However, guaranteeing robustness as well is a bit more subtle (note that in the two-party case, robustness and fairness are equivalent). Below, we describe three different primitives that are complete for *robust* secure computation. Recall that in a robust, ideal-world computation, the trusted third party will never abort; instead it replaces the inputs of aborting players with a default value, guaranteeing that the honest parties always receive output. Each of the three primitives has a different trade-off between the call complexity (how many times the functionality must be invoked) and the input size to the primitive (depending on the number of participating parties and the description of the function to be evaluated).

Universal Black Box (UBB) This primitive was originally proposed by Fitzi et al. [26]: The UBB receives as input from each party both a circuit (specifying the function to be computed) and an argument to that function. It then partitions the parties according to the circuit that they provide. For each set of parties that gave the same circuit as input, the UBB primitive outputs to that set of parties the evaluation of that circuit on the arguments given by those parties, using default arguments for the remaining parties. It is easy to see that a single call to this primitive can be used to securely compute any function.

Fair Consistent Reconstruction (FCR) Before using the FCR primitive to compute a functionality F, the n parties first perform a secure computation with abort for the related function F'. The function $F'(x_1, \ldots, x_n)$ chooses random keys $k_1, \ldots, k_n$ and computes $k = \sum_{i=1}^{n} k_i$ and $o = \mathsf{Enc}_k(F(x_1, \ldots, x_n))$. It also generates a key pair (sk, vk) for some public-key signature scheme, and computes $\sigma_i = \mathsf{Sign}(sk, k_i)$ for every $i \in [n]$. The output to party i is (o, vk, k_i, σ_i). We note that if any player does abort during this computation[5], the remaining players simply substitute some default input for him and begin again.

The parties then invoke FCR primitive on the values (vk, k_i, σ_i) as received from the execution of F'. The output of FCR is as follows:

Case 1: If not all parties submit the same value for vk, then FCR partitions the players according to the values of vk they submitted and returns these partitions.

Case 2: If any player submits (vk, k_i, σ_i) such that $\mathsf{Vrfy}(vk, k_i, \sigma_i) = 0$ (i.e. they submitted an invalid signature on key k_i), FCR returns a message identifying player i as a cheater.

Case 3: If all parties submit valid signatures under the same key vk, then FCR returns $k = \sum_{i=1}^{n} k_i$.

If players receive output indicating that case 1 occurred, then they begin the entire computation again using default values for anyone outside their own partition. (Note that all honest players are guaranteed to be in the same partition.) If they receive output indicating that case 2 occurred, then they begin the computation again excluding the cheating party and using a default value for his input. Finally, if they receive output indicating that case 3 occurred, they simply use k to decrypt o and recover the output of F. This process may continue for a total of $n-1$ iterations until the protocol terminates. We defer the proof of security to the full version of the paper.

All-Subsets Reconstruction (ASR) The ASR primitive is, essentially, a version of the FCR primitive that can be used to fairly compute any functionality with a single invocation (rather than n). The price is that its input size is exponential in the number of parties (although still independent of the complexity of the target functionality). The input from party i is a set of inputs to the FCR primitive: one input to FCR for every subset $S \subseteq [n]$ of the parties such that $\{i\} \subset S$. The ASR primitive internally runs the entire reconstruction protocol using FCR:

1. ASR begins by running FCR using the inputs corresponding to the set of all parties.
2. If reconstruction of k fails because FCR ends in either case 1 or case 2, ASR recurses, either after partitioning the players, or after dropping a cheater, and calls FCR with the appropriate corresponding inputs.
3. If the reconstruction succeeds, ASR simply outputs k to all players still "active" in the successful call to FCR.

In order to use this primitive to compute a functionality F, the n parties first perform a secure computation with abort for the related function F'', described below. Let F' be the function computed in the protocol that uses FCR, and let out_i be the output of player i from F'. Then for every subset of parties S, $F''(x_1, \ldots, x_n)$ computes the outputs $(\mathsf{out}_1^{(S)}, \ldots, \mathsf{out}_n^{(S)}) = F'(x_1^{(S)}, \ldots, x_n^{(S)})$ where

$$x_i^{(S)} = \begin{cases} x_i & \text{if } i \in S \\ \perp & \text{if } i \notin S \end{cases}$$

[5] There exist protocols in which all players are correctly informed of the identity of the aborting player [31, 30].

(i.e., $\mathsf{out}_i^{(S)}$ is the output of the function F' to party i assuming only the parties in S are honest and the remaining parties abort). Note that by the definition of F', $\mathsf{out}_i^{(S)} = (o^{(S)}, vk^{(S)}, k_i^{(S)}, \sigma_i^{(S)})$. F'' outputs $\mathsf{out}_i^{(S)}$ to player i, for each set S such that $\{i\} \subset S$.

The parties invoke ASR with these outputs. If an honest party i receives the output $\perp$, it assumes it is the only honest party and computes the functionality F on its own. Otherwise, it uses the output k to decrypt $o^{(S)}$ (for any S such that $i \in S$). We again defer the proof of security to the full version.

4 Limits on Fairness-Completeness

In this section we show that there does not exist a finite (i.e. "short") primitive that is complete for fairness. More specifically, we prove that the $\mathsf{FairRec}_\kappa$ function cannot be fairly computed even if the players are given parallel access to a primitive of size $O(\log \kappa)$. There are two main ideas behind the proof. For simplicity, imagine for now that the entire protocol consisted of a single call to this short primitive. Our first observation is that because the primitive is short, the adversary can locally simulate it, computing its output for each possible input of the other party. This will play a crucial role in our proof, but it does not itself suffice: so far the adversary has no way of knowing *which* of these outputs are correct. However, because the primitive is supposed to be complete for fairness, it allows us to compute the $\mathsf{FairRec}$ functionality, which has a very useful property: its output is *verifiable*. That is, when two parties are given inputs generated by Share, then the correct output of $\mathsf{FairRec}$ is $(s, 0)$, where the flag 0 indicates that s is the correct output. Furthermore, for incorrect inputs, with overwhelming probability the output of $\mathsf{FairRec}$ is $(\perp, \perp)$. The adversary simply computes the primitive for every possible input of the other player, and outputs s when he recovers it.

When we consider a protocol with many calls to the primitive (including parallel calls), we combine the above ideas using a standard hybrid argument. If the adversary aborts before any invocations of the primitive, he cannot learn anything about the output s. On the other hand, if he behaves honestly in all invocations, he should always recover s. We prove below that there is some specific invocation for which the adversary can gain a non-negligible advantage over the honest party by aborting and simulating the input to that invocation as described above. Finally, he can guess which invocation will allow this advantage with significant probability. Formally, we prove:

Theorem 2. *Let g be an $O(\log \kappa)$-bit primitive. Then for any polynomial p, $\mathsf{par}_{p(\kappa)}(g)$ is not complete for fairness.*

Note that for any $k \geq 1$, $\mathsf{par}_k(g)$ is a more powerful primitive than g (i.e., if the fairness of $\mathcal{F}$ reduces to g then the fairness of $\mathcal{F}$ also reduces to $\mathsf{par}_k(g)$). We are proving an impossibility, so starting with a more powerful primitive strengthens our results. Our proof will hold even if we restrict the adversarial behavior to aborting early.

Proof. Suppose there exists such a primitive g and polynomial p. Consider the $r = r(\kappa)$ round protocol Π_p^g that fairly computes $\mathsf{FairRec}_\kappa(x, y)$ while making a call to $\mathsf{par}_{p(\kappa)}(g)$ in each round. We can think of this call as $p(\kappa)$ parallel calls to g. Without loss of generality, we assume that these calls to g constitute the only communication between the players[6]. Let $q = p \cdot r$ be the total number of calls to g. For each round $i \in r$, we define some arbitrary ordering σ_i on the parallel calls to g that occur in that round. This induces a natural ordering over all q calls to g, where for $i < j$, calls in round i are ordered before calls in round j. We let g_k denote the k^{th} call to g according to this ordering.

Consider an execution of Π_p^g in which $\mathsf{Share}(s, r) = (s_0, s_1)$ is used to generate shares, for a random s and r. Player j gets the share s_j. We let the value a_i denote the output of player 0 when player 1 acts honestly for the first i calls to g (according to the ordering previously described) and then aborts. We define b_i in the symmetric way. Note that by correctness of Π_p^g, and the definition of $\mathsf{FairRec}_\kappa$, for all i

$$\Pr[a_i \neq s \wedge a_i \neq \perp] = \mathsf{negl}(\kappa) = \Pr[b_i \neq s \wedge b_i \neq \perp]$$

and

$$\Pr[a_q = s] = \Pr[b_q = s] = 1 - \mathsf{negl}(\kappa).$$

where the probability is over the random tapes of the players. Furthermore, by the definition of $\mathsf{FairRec}$ and the properties of a NMSS scheme,

$$\Pr[a_0 \neq \perp] = \mathsf{negl}(\kappa) = \Pr[b_0 \neq \perp].$$

It follows that for every large enough κ, there exists a polynomial $p'(\kappa)$ and a round i such that either

$$\Pr[a_i = s] - \Pr[b_{i-1} = s] \geq \frac{1}{p'(\kappa)}.$$

or

$$\Pr[b_i = s] - \Pr[a_{i-1} = s] \geq \frac{1}{p'(\kappa)}.$$

Without loss of generality, we will assume the former, and we demonstrate an adversary $\mathcal{A}$ that breaks the security of Π_p^g with probability at least $1/(q \cdot p'(\kappa))$.

$\mathcal{A}$ begins by choosing a random value $i^* \in [1, \ldots, q]$, and plays honestly for the first $i^* - 1$ calls to g (i.e., submits correct values to g) and then aborts. Note that the resulting output of player 1 is b_{i^*-1}. The adversary now attempts to compute the value of a_{i^*} by simulating the side of player 1. Note, however, that by definition, the value of a_{i^*} depends on honest input to g_{i^*} from both players, and $\mathcal{A}$ may not know (anything) about player 1's input to g_{i^*}. Here we use the fact that g has short inputs, and that $\mathsf{FairRec}$ is verifiable. $\mathcal{A}$ goes through all possible inputs $\beta \in \{0, 1\}^{O(\log \kappa)}$ that player 1 might have sent to g_{i^*}, and for each such value he simulates g internally, using as input his own (honest) value that he *would* have sent if he had not aborted, and β. He computes a_{i^*} from his view in the (real) interaction with player 1, and the simulated output of g_{i^*}. Since one of these values of β is the value used by player 1 in the actual execution, it follows that the correct value of a_{i^*} is among this set of outputs. Furthermore, if some simulated $a_{i^*} = s' \neq \perp$ then $s' = s$ with overwhelming probability. $\mathcal{A}$ outputs

[6] This is without loss of generality because we can always modify g to do message transmission, in addition to its original functionality. Note also that if less than $p(\kappa)$ calls are needed in a particular round, the players can make extra calls with random inputs, ignoring the outputs, to make the total number of calls $p(\kappa)$.

$s' \neq \perp$ if this occurs, and $\perp$ otherwise. By our assumption, there exists an i such that $\Pr[a_i = s \wedge b_{i-1} = \perp] \geq \frac{1}{p'(\kappa)}$. Hence, $\mathcal{A}$ recovers s without the honest party receiving output with probability $1/(q \cdot p'(\kappa))$, contradicting the fairness of protocol.

Theorem 3. *Simultaneous broadcast is not complete for fairness.*

The fact that short simultaneous broadcast (cf. Section 2) is not complete for fairness follows from Theorem 2. We give two different proofs that a large simultaneous broadcast is not complete for fairness.

Proof (First proof of Thm. 3.). The first proof follows as a corollary of Theorem 2, and Lemma 1 (below). This is true because if long-SB were complete, then by Lemma 1, short-SB would also be complete, contradicting Theorem 2.

Lemma 1. *Let g denote the k-bit* SB *primitive. For any $p \in \mathbb{N}$, there exists a protocol Π_p^g that implements kp-bit* SB *with perfect security.*

Proof. The protocol is the (trivial) one round protocol in which both parties split their inputs into p blocks of size k, submit block i to instance g_i, and output the concatenation of the p outputs (maintaining the order). The proof of the lemma is straightforward, so we omit a formal exposition. We simply note that the decision of an adversary to change its input to any instance(s) of g (including the decision to abort in some instance(s)) is entirely independent of the actions or input of the honest party. The simulator simply recovers the p values that the adversary intended for g (recall that an abort is treated as input 0^k), concatenates them, and forwards them to the trusted party. After receiving output, the simulator rewinds the adversary, parses the output into blocks, and uses block i as the honest player's input to g_i.

In our full version we provide a second proof that is more direct and is of independent interest. Due to space constraints we omit the proof here.

5 A Fairness Hierarchy

In this section we show that for small k, fairness cannot be "amplified" at all (with regards to input size). Specifically, for small values of k we show that no k-bit functionality can be used to build $(k+1)$-SB, even if standard cryptographic assumptions are allowed. Unlike in Section 4, here we assume that the players do not have parallel access to the primitive. More formally:

Theorem 4. *For $k = O(\log \kappa)$, the fairness of $(k+1)$-*SB *does not reduce to any k-bit primitive.*

To gain some intuition for how we prove the theorem, consider that in an ideal world execution of simultaneous broadcast, if the players inputs are chosen independently, then (by definition) each player's output is independent of their own input. However, we show below that in any real world protocol constructing $(k+1)$-SB from k-bit functionalities, this property cannot be guaranteed. We demonstrate that there always exists some round in which an adversary can gain information about the other party's input, as well as some later round in which it can still affect the other party's output by choosing whether or not to abort. By choosing whether to abort in the later round based on what is learned in the earlier round, the adversary can correlate the output of the honest player with his input.

There are two main ideas behind the proof. The first idea is the one used in the proof of Theorem 2: because the $k = O(\log \kappa)$-bit inputs to the primitive are small, the adversary can gain an "information lead" of one round by testing all $2^k = \text{poly}(\kappa)$ inputs that the honest party might send to the k-bit primitive in the next round. For each of these possible outputs from the primitive, the adversary computes the $k + 1$-bit value that he *would* have output in the protocol if this were the last thing he received in the protocol (i.e. if the honest player aborted immediately afterwards). In this way the adversary computes the set of all "potential outputs" that he could possibly output if the honest party sends a single additional message and then aborts.

Unfortunately, unlike in the proof of Theorem 2, we cannot argue here that the adversary recognizes the correct output of the protocol among this set: in Theorem 2 the output was verifiable, while the output of SB is not. Instead, we rely on a different observation: there are twice as many possible input values to the $k + 1$-SB protocol as there are potential outputs from the k-bit primitive used in any particular round. Thus, for every round, at least half of the possible $k + 1$ bit inputs to SB will not appear in the set of potential outputs. We use this fact to show that there exist two inputs, y, y', and a round i such that if the honest player has input y, the potential output set in round i contains y but not y', while if his input is y', it contains y' but not y. Furthermore, we will prove that at least one of the parties can affect the other's output by sending a random message in some later round, $i' > i$, and then aborting. The adversarial strategy is as follows: he runs the protocol honestly until round i, and then determines which of the two inputs the honest party has. Depending on what he learns, the then chooses whether to complete the protocol honestly, or to later abort after round i'. By making this decision, he creates a correlation between the honest party's output and his input, violating the security of the simultaneous broadcast.

Due to space considerations, we defer the formal proof to the full version of the paper.

6 Fairness Combiners

We have demonstrated that one possible approach for achieving fair secure computation is to rely on a trusted third party to implement the FairRec functionality. A natural question that arises is, how much can we distribute that trust? Instead of trusting a single party, can we use multiple parties, guaranteeing both fairness and security so long as some number of them act honestly?

This can be thought of as a fairness analogue of *combiners* [37, 38]; the two computing parties (clients) have access to n "fairness providers" (servers), of which at least $n - t$ are guaranteed to be honest. Two questions arise: what are the values of t and n for which this is possible, and what are the minimal requirements of the honest fairness providers? Below, we show that combining is possible if and only if $n > 2t$.

6.1 Combining with an Honest Majority

If the clients and servers had a broadcast channel and a complete point to point network, the problem would simply be a restricted case of secure multiparty computation where only two parties have input and output. In this case we could use the protocol of Rabin and Ben-Or to fairly compute any functionality [49], since a strict majority of the servers are honest, and at least one client is honest. However, we are interested in using *independent* servers, where each provider communicates only with the two clients and is not expected to know anything about the other primitives.

Instead we model the problem as a secure multiparty computation over an *incomplete* point-to-point network, in which there only exist communication channels between the two computing players, and from each player to each server. The adversary is allowed to corrupt at most one of the computing players, and at most t of the fairness providers. Finally, we assume the existence of oblivious transfer, although we discuss how to make our results unconditional in the full version. Viewing the problem this way, one solution is to emulate the missing channels (including a broadcast channel), enabling the use of the general result of Rabin and Ben-Or. We outline this solution below.

The two clients begin by executing an *unfair* secure computation (using oblivious transfer) to establish correlated randomness for each pair of servers. This randomness will include shared secret keys that enable any two servers to authenticate and encrypt messages to one another, as well as additional correlated randomness that will enable broadcast for any $n > 2t$ [4, 27]. In order to prevent the clients from learning the correlated randomness, the computation will actually output NMSS shares of the output, one share to each client, which they then relay to the appropriate servers. If the protocol to establish these keys ends unfairly, then the honest client simply aborts; no information is leaked and no harm is done, since this execution is independent of their inputs. Otherwise, the servers inform both clients that they have successfully reconstructed their keys and randomness. If anyone indicates otherwise, all players immediately abort the protocol. With the communication channels in place, the players can now execute the protocol of Rabin and Ben-Or [49] to compute the desired functionality.[7]

In the full version of the paper, we show how to construct a more efficient protocol for this task and discuss in greater detail the security properties we can guarantee.

Impossibility of Combining with a Faulty Majority The previous result is tight; if the majority of the fairness providers are corrupt, they do not help us to achieve fairness. Specifically, we consider the case where the players have access to two fairness providers, one of which is corrupt, and show that any function that can be computed in this model can be computed in the plain model. Since there exist functions that cannot be computed in the plain model [20], it follows that the same functions cannot be computed in our setting. Below, when a protocol Π permits calls to two instances of some primitive g, we denote this by Π^{g_1,g_2}.

[7] We note that the protocol of Rabin and Ben-Or is *robust*, which means that even if some servers abort, the clients can continue the computation with the remaining servers. This enables even two honest clients to complete their computation correctly, so long as less than half of the servers are corrupt.

Theorem 5. *Let g_1 and g_2 be two instances of some arbitrary functionality g. If Π^{g_1,g_2} securely computes function $\mathcal{F}$, even when an adversary corrupts one of the instances of g, then there exists a protocol Π' that securely computes $\mathcal{F}$ without access to any primitives.*

Proof (Sketch). The proof is a standard partitioning argument. Π' simply follows the description of Π^{g_1,g_2}, delegating the responsibilities for computing g_1 and g_2 to players 0 and 1 respectively. By our assumption, so long as one of the two instances is executed fairly and securely, Π' fairly and securely computes $\mathcal{F}$. Since one of the players is honest, the primitive that they control will always be executed honestly.

References

1. *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, 2-4 May 1988, Chicago, Illinois, USA*. ACM, 1988.
2. N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *CCS*, pages 7–17. ACM, April 1997.
3. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, Apr. 2000.
4. B. Baum-Waidner, B. Pfitzmann, and M. Waidner. Unconditional byzantine agreement with good majority. In C. Choffrut and M. Jantzen, editors, *STACS*, volume 480 of *Lecture Notes in Computer Science*, pages 285–295. Springer, 1991.
5. D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC*, pages 479–488. ACM, 1996.
6. D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. pages 468–473, 1989.
7. A. Beimel and T. Malkin. A quantitative approach to reductions in secure computation. In M. Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 238–257. Springer, 2004.
8. M. Bellare, editor. *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*. Springer, 2000.
9. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC* [1], pages 1–10.
10. M. Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1(2):175–193, May 1983. Previously published in ACM STOC 1983 proceedings, pages 440–447.
11. D. Boneh and M. Naor. Timed commitments. pages 236–254.
12. D. Boneh and M. Naor. Timed commitments. In Bellare [8], pages 236–254.
13. C. Cachin and J. Camenisch. Optimistic fair secure computation. In Bellare [8], pages 93–111.
14. R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
15. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC* [1], pages 11–19.
16. L. Chen, C. Kudla, and K. G. Paterson. Concurrent signatures. In C. Cachin and J. Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 287–305. Springer, 2004.

17. B. Chor, M. Geréb-Graus, and E. Kushilevitz. On the structure of the privacy hierarchy. *J. Cryptology*, 7(1):53–60, 1994.
18. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS*, pages 383–395. IEEE, 1985.
19. B. Chor and E. Kushilevitz. A zero-one law for boolean privacy. *SIAM J. Discrete Math.*, 4(1):36–47, 1991.
20. R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369. ACM, 1986.
21. R. Cleve. Controlled gradual disclosure schemes for random bits and their applications. In G. Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 573–588. Springer, 1990.
22. R. Cramer, Y. Dodis, S. Fehr, C. Padró, and D. Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 471–488. Springer, 2008.
23. I. Damgård. Practical and provably secure release of a secret and exchange of signatures. *J. Cryptology*, 8(4):201–222, 1995.
24. I. Damgård. Practical and provably secure release of a secret and exchange of signatures. *J. Cryptology*, 8(4):201–222, 1995.
25. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
26. M. Fitzi, J. A. Garay, U. M. Maurer, and R. Ostrovsky. Minimal complete primitives for secure multi-party computation. *J. Cryptology*, 18(1):37–61, 2005.
27. M. Fitzi, N. Gisin, U. M. Maurer, and O. von Rotz. Unconditional byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In L. R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 482–501. Springer, 2002.
28. Z. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. pages 135–155.
29. J. A. Garay, P. D. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. In S. Halevi and T. Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 404–428. Springer, 2006.
30. O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.
31. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.
32. O. Goldreich and R. Vainish. How to solve any protocol problem - an efficiency improvement. In C. Pomerance, editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 1988.
33. S. Goldwasser and L. Levin. Fair computation and general functions in the presence of immoral majority.
34. D. Gordon and J. Katz. Partial fairness in secure two-party computation. Cryptology ePrint Archive, Report 2008/206, 2008. http://eprint.iacr.org/2008/206.
35. S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. In R. E. Ladner and C. Dwork, editors, *STOC*, pages 413–422. ACM, 2008.
36. S. D. Gordon and J. Katz. Complete fairness in multi-party computation without an honest majority. In Reingold [50], pages 19–35.
37. D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen. On robust combiners for oblivious transfer and other primitives. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

38. A. Herzberg. Folklore, practice and theory of robust combiners. *Journal of Computer Security*, 17(2):159–189, 2009.
39. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.
40. J. Katz. On achieving the "best of both worlds" in secure multiparty computation. In D. S. Johnson and U. Feige, editors, *STOC*, pages 11–20. ACM, 2007.
41. J. Kilian. Founding cryptography on oblivious transfer. In *STOC* [1], pages 20–31.
42. J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky. Reducibility and completeness in private computations. *SIAM Journal on Computing*, 29(4):1189–1208, 2000.
43. M. Lepinski, S. Micali, C. Peikert, and A. shelat. Completely fair sfe and coalition-safe cheap talk. In S. Chaudhuri and S. Kutten, editors, *PODC*, pages 1–10. ACM, 2004.
44. A. Y. Lindell. Legally-enforceable fairness in secure two-party computation. In T. Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 121–137. Springer, 2008.
45. M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin. In *FOCS*, pages 11–21. IEEE, 1983.
46. H. K. Maji, M. Prabhakaran, and M. Rosulek. Complexity of multi-party computation problems: The case of 2-party symmetric secure function evaluation. In Reingold [50], pages 256–273.
47. S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*, pages 12–19. ACM, 2003.
48. B. Pinkas. Fair secure two-party computation. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 87–105. Springer, 2003.
49. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85. ACM, 1989.
50. O. Reingold, editor. *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*. Springer, 2009.
51. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE, 1986.