Protocols for Multiparty Coin Toss with Dishonest Majority

Amos Beimel[1,⋆], Eran Omri[2,⋆⋆], and Ilan Orlov[1,⋆⋆⋆]

[1] Dept. of Computer Science, Ben Gurion University, Be'er Sheva, Israel
[2] Dept. of Computer Science, Bar Ilan University, Ramat Gan, Israel

Abstract. Coin-tossing protocols are protocols that generate a random bit with uniform distribution. These protocols are used as a building block in many cryptographic protocols. Cleve [STOC 1986] has shown that if at least half of the parties can be malicious, then, in any r-round coin-tossing protocol, the malicious parties can cause a bias of $\Omega(1/r)$ to the bit that the honest parties output. However, for more than two decades the best known protocols had bias $\frac{t}{\sqrt{r}}$, where t is the number of corrupted parties. Recently, in a surprising result, Moran, Naor, and Segev [TCC 2009] have shown that there is an r-round two-party coin-tossing protocol with the optimal bias of $O(1/r)$. We extend Moran et al. results to the *multiparty model* when less than 2/3 of the parties are malicious. The bias of our protocol is proportional to $1/r$ and depends on the gap between the number of malicious parties and the number of honest parties in the protocol. Specifically, for a constant number of parties or when the number of malicious parties is somewhat larger than half, we present an r-round m-party coin-tossing protocol with optimal bias of $O(1/r)$.

1 Introduction

Secure multiparty computation in the malicious model allows distrustful parties to compute a function securely. Designing such secure protocols is a delicate task with a lot of subtleties. An interesting and basic functionality for secure computation is coin tossing – generating a random bit with uniform distribution. This is a simple task where the parties have no inputs. However, already this task raises questions of fairness and how malicious parties can bias the output. Understanding what can be achieved for coin tossing in various settings can be considered as a step towards understanding general secure and fair multiparty computation. Indeed, some of the early works on secure computation were on coin tossing, e.g., [3, 4, 6]. Furthermore, coin tossing is used as a basic tool in constructing many protocols that are secure in the malicious model. Secure

⋆ Supported by ISF grant 938/09.
⋆⋆ This research was generously supported by the European Research Council as part of the ERC project "LAST". Part of the research was done while the author was a post-doctoral fellow at BGU supported by the ISF grant 860/06.
⋆⋆⋆ Supported by ISF grant 938/09 and by the Frankel Center for Computer Science.

T. Rabin (Ed.): CRYPTO 2010, LNCS 6223, pp. 538–557, 2010.

protocols for coin tossing are a digital analogue of physical coin tossing, which have been used throughout history to resolve disputes.

The main problem in designing coin-tossing protocols is the prevention of a bias of the output. The bias of a coin-tossing protocol measures the maximum influence of the adversary controlling a subset of malicious parties on the output of the honest parties, where the bias is 0 if the output is always uniformly distributed and the bias is 1/2 if the adversary can force the output to be always (say) 1. To demonstrate the problems of designing a coin-tossing protocol, we describe Blum's two-party coin-tossing protocol [3].

Example 1 (Blum's coin-tossing protocol [3]). To toss a coin, Alice and Bob execute the following protocol.

- Alice chooses a random bit a and sends a commitment $c = \text{commit}(a)$ to Bob.
- Bob chooses a random bit b and sends it to Alice.
- Alice sends the bit a to Bob together with de-commit(c).
- If Bob does not abort during the protocol, Alice outputs $a \oplus b$, otherwise she outputs a random bit.
- If Alice does not abort during the protocol and c is a commitment to a, then Bob outputs $a \oplus b$, otherwise he outputs a random bit.

If Alice is malicious, then she can bias the output toward (say) 1. If $a \oplus b = 1$, she opens the commitment and Bob outputs 1. However, if $a \oplus b = 0$, Alice aborts, and Bob outputs 1 with probability 1/2. All together, the probability that the honest Bob outputs 1 is 3/4. It can be proved that this is the best that Alice can do in this protocol, and hence, the bias of the protocol is 1/4. This protocol demonstrates the problems caused by parties aborting the protocol and the need to define how the output of the other parties is computed after such aborts.

While the above protocol is a significant improvement over naive protocols whose bias is 1/2, the protocol still has a constant bias. If *more than half* of the parties are honest, then, using general secure multiparty protocols there are constant-round protocols with negligible bias (assuming a broadcast channel), e.g., the protocol of [14]. Cleve [6] proved that when at least half of the parties can be malicious, the bias of every protocol with r rounds is $\Omega(1/r)$. In particular, this proves that without honest majority no protocol with polynomial number of rounds (in the security parameter) can have negligible bias. On the positive side, it was shown in [2, 6] that there is a two-party protocol with bias $O(1/\sqrt{r})$. This can be generalized to an m-party protocol that tolerates any number of malicious parties and has bias $O(t/\sqrt{r})$. Cleve and Impagliazzo [7] have shown that, in a model where commitments are available only as black-box (and no other assumptions are made), the bias of every coin-tossing protocol is $\Omega(1/\sqrt{r})$.[1] The protocols of [3, 2, 6] are in this model.

[1] The lowerbound of [7] holds in a stronger model which we will not discuss in this paper.

The question if there is a coin-tossing protocol without an honest majority that has bias $O(1/r)$ was open for many years. Recently, in a surprising and elegant result, Moran, Naor, and Segev [12] have shown that there is an r-round two-party coin-tossing protocol with bias $O(1/r)$. Moran et al. ask the following question:

> "An interesting problem is to identify the optimal trade-off between the number of parties, the round complexity, and the bias. Unfortunately, it seems that several natural variations of our approach fail to extend to the case of more than two parties. Informally, the main reason is that a coalition of malicious parties can guess the threshold round with a pretty good probability by simulating the protocol among themselves for any possible subset."

1.1 Our Results

Our main contribution is a multi-party coin-tossing protocol that has small bias when less than 2/3 of the parties are malicious.

Theorem 1 (Informal). *Let m, t, and r be integers such that $m/2 \leq t < 2m/3$. There exists an r-round m-party coin-tossing protocol tolerating t malicious parties that has bias $O(2^{2^{k+1}}/r')$, where $k = 2t - m$ and $r' = r - O(k+1)$.*

The above protocol nearly has the desired dependency on r, i.e., the dependency implied by the lower bound of Cleve [6]. However, its dependency on k has, in general, a prohibitive cost. Nevertheless, there are interesting cases where the bias is $O(1/r)$.

Corollary 1 (Informal). *Let m, t be* constants *such that $m/2 \leq t < 2m/3$ and r be an integer. There exists an r-round m-party coin-tossing protocol tolerating t malicious parties that has bias $O(1/r)$.*

For example, we construct an r-round 5-party coin-tossing protocol tolerating 3 malicious parties that has bias $8/(r - O(1))$ (this follows from our general construction in Sections 4–6).

Notice that the protocol of Theorem 2 depends on k and not on the number of malicious parties t. Thus, it is efficient when k is small.

Corollary 2 (Informal). *Let m, r be integers and $t = m/2 + O(1)$. There exists an r-round m-party coin-tossing protocol tolerating t malicious parties that has bias $O(1/r)$.*

For example, for any even m we construct an r-round m-party coin-tossing protocol tolerating $m/2$ malicious parties that has bias $1/(2r - O(1))$. Furthermore, even when $t = 0.5m + 0.5 \log\log m - 1$, the bias of our protocol is small, namely, $O(m/(r - O(\log\log m)))$.

We also reduce the bias compared to previous protocols when more than 2/3 of the parties are malicious. The bias of the m-party protocol of [2, 6] is $O(t/\sqrt{r})$. We present a protocol whose bias is $O(1/\sqrt{r})$ when t/m is constant, that is, when the fraction of malicious parties is constant we get rid of the factor t in the bias.

Communication Model. We consider a communication model where all parties can only communicate through an authenticated broadcast channel. On one hand, if a party broadcasts a message, then all other parties see *the same* message. This ensures some consistency between the information the parties have. On the other hand, there are no private channels and all parties see all messages. We assume a synchronous model, however, the adversary is rushing.[2]

We note that our results can be translated to a model with authenticated point-to-point channels with a PKI infrastructure (in an m-party protocol, the translation will increase the number of rounds by a multiplicative factor of $O(m)$). Thus, our results hold in the two most common models for secure multiparty computation.

The Idea of Our Protocol. We generalize the two-party protocol of Moran et al. [12] to the multi-party setting. In the protocol of [12] in each round Alice and Bob get bits that are their output if the other party aborts: If a party aborts in round i, then the other party outputs the bit it got in round $i-1$. Furthermore, there is a special round i^*; prior to round i^* the bits given to Alice and Bob are random independent bits and from round i^* onward the bits given to Alice and Bob are the same fixed bit. The adversary can bias the output only if it guesses i^*. In our protocol, in each round there are many bits. We define a collection of subsets of the parties and each subset gets a bit. The bits are chosen similarly to [12]: prior to i^* they are independent and from i^* onward they are fixed. In our case we cannot give the bits themselves to the parties. We rather use a few layers of secret-sharing schemes to store these bits. For every subset in the collection, we use the first secret-sharing scheme to share the bit of the subset among the parties of the subset. We use an additional secret-sharing scheme to share the shares of the first secret-sharing scheme. The threshold in the latter secret sharing scheme is chosen such that the protocol can proceed until enough parties aborted. In the round when the number of aborted parties ensures that there is an honest majority, an appropriate subset in the collection is chosen, its bit is reconstructed, and this bit is the output of the honest parties. The description of how to implement these ideas appears in Sections 4–6.

The construction of Moran et al. [12] is presented in two phases. In the first phase they present a protocol with a trusted dealer, for which an adversary can only inflict bias $O(1/r)$. Then, they show how to implement this protocol in the real-world, using a constant round secure-with-abort multiparty protocol, as well as secret sharing and authentication schemes. This can be seen as a general transformation from any two-party coin-tossing protocol with a trusted dealer, into a real world two-party coin-tossing protocol. We observe that the transformation of Moran et al. to a real-world protocol requires some further care trying to generalize it for the multiparty case. We show how this can be achieved by adopting the definition of secure multiparty computation of [1], which requires the protocol to detect a cheating party, that is, at the end of

[2] If there is synchronous broadcast without a rushing adversary then coin tossing is trivial.

the protocol either the honest parties hold a correct output or all honest parties agree on a party that cheated during the protocol.

2 Preliminaries

A multi-party coin-tossing protocol with m parties is defined using m probabilistic polynomial-time Turing machines $p_1, \ldots, p_m$ having the security parameter 1^n as their only input. The coin-tossing computation proceeds in rounds, in each round, the parties broadcast and receive messages on a broadcast channel. The number of rounds in the protocol is typically expressed as some polynomially-bounded function r in the security parameter. At the end of protocol, the (honest) parties should hold a common bit w. We denote by CoinToss() the ideal functionality that gives the honest parties the same uniformly distributed bit w, that is, $\Pr[w = 0] = \Pr[w = 1] = 1/2$. In our protocol, the output bit w will have some bias.

In this work we consider a malicious static computationally-bounded (i.e., non-uniform probabilistic polynomial-time) adversary that is allowed to corrupt some subset of parties. That is, before the beginning of the protocol, the adversary corrupts a subset of the players that may deviate arbitrarily from the protocol, and thereafter the adversary controls the messages sent by the corrupted parties. The honest parties follow the instructions of the protocol.

The parties communicate in a synchronous network, using only a broadcast channel. The adversary is rushing, that is, in each round the adversary hears the messages sent by the honest parties before broadcasting the messages of the corrupted parties for this round (thus, the messages broadcast by corrupted parties can depend on the messages of the honest parties broadcast in this round).

The security of multiparty computation protocols is defined using the real vs. ideal paradigm. In this paradigm, we consider the real-world model, in which protocols are executed. We then formulate an ideal model for executing the task at hand. This ideal model involves a trusted party whose functionality captures the security requirements from the task. Finally, we show that the real-world protocol "emulates" the ideal-world protocol: For any real-life adversary $\mathcal{A}$ there should exist an ideal-model adversary $\mathcal{S}$ (also called simulator) such that the global output of an execution of the protocol with $\mathcal{A}$ in the real-world model is distributed similarly to the global output of running $\mathcal{S}$ in the ideal model.

1/p-Secure Computation. As explained in the introduction, the ideal functionality CoinToss() cannot be implemented when there is no honest majority. We use $1/p$-secure computation, defined by [9, 10], to capture the divergence from the ideal worlds. This notion applies to general secure computation. We start with some notation.

Definition 1 (1/p-indistinguishability). *A function $\mu(\cdot)$ is* negligible *if for every positive polynomial $q(\cdot)$ and all sufficiently large n it holds that $\mu(n) < 1/q(n)$. A* distribution ensemble $X = \{X_n\}_{n\in\mathbb{N}}$ *is an infinite sequence of random variables indexed by $n \in N$. For a fixed function p, two distribution ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are* computationally $1/p$-indistinguishable,

denoted $X \overset{1/p}{\approx} Y$*, if for every non-uniform polynomial-time algorithm* D *there exists a negligible function* $\mu(\cdot)$ *such that for every* n*,*

$$|\Pr[D(X_n) = 1] - \Pr[D(Y_n)) = 1]| \leq \frac{1}{p(n)} + \mu(\cdot).$$

We next define the notion of $1/p$-secure computation [9, 10]. The definition uses the standard real/ideal paradigm [8, 5], except that we consider a completely fair ideal model (as typically considered in the setting of honest majority), and require only $1/p$-indistinguishability rather than indistinguishability. In the coin-tossing protocol, the parties do not have inputs. Thus, to simplify the definitions, we define secure computation without inputs (except for the security parameters).

Definition 2 ($1/p$-secure computation [9, 10]). *Let* $p = p(n)$ *be a function. An* m*-party protocol* Π *is said to* $1/p$*-securely compute a functionality* $\mathcal{F}$ *if for every non-uniform probabilistic polynomial-time adversary* $\mathcal{A}$ *in the real model, there exists a non-uniform probabilistic polynomial-time adversary* $\mathcal{S}$ *in the ideal model such that the following two distribution ensembles are computationally* $1/p$*-indistinguishable* $\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(\text{aux})}(1^n)\right\}_{n\in\mathbb{N}} \overset{1/p}{\approx} \left\{\text{REAL}_{\Pi,\mathcal{A}(\text{aux})}(1^n)\right\}_{n\in\mathbb{N}}$*, where* $\text{REAL}_{\Pi,\mathcal{A}(\text{aux})}(1^n)$ *is a random variable consisting of the view of the adversary (i.e., its random input and the messages it got) and the output of the honest parties following an execution of* Π*, and* $\text{IDEAL}_{\mathcal{F},\mathcal{S}(\text{aux})}(1^n)$ *is a random variable consisting of the output of the adversary* $\mathcal{S}$ *in the ideal world execution and the output of the honest parties in that execution.*

Definition 3. *We say that a protocol is a coin-tossing protocol with bias* $1/p$ *if it is a* $1/p$*-secure protocol for the functionality* CoinToss()*.*

2.1 The Two-Party Protocol of Moran et al.

Moran, Naor, and Segev [12] present a two-party coin-tossing protocol with optimal bias with respect to round complexity (i.e., meeting the lowerbound of Cleve [6]). We next briefly review their protocol, which later serves as the basis for our construction. Following the presentation of [12], we first describe a construction that uses an on-line trusted party acting as a dealer. Later, we describe how the trusted party can be eliminated.

The main underlying idea is that the dealer chooses a special round during which the parties actually unknowingly learn the output of the protocol. If the adversary guesses this round, it can bias the output by aborting. If the adversary aborts (or behaves maliciously) in any other time, then there is no bias. However, this special round is uniformly selected (out of r possible rounds) and then concealed such that the adversary is unable to guess it with probability exceeding $1/r$. Therefore, the overall bias achievable by any adversary is $O(1/r)$.

More specifically, for two parties Alice and Bob to jointly toss a random coin, the protocol proceeds as follows. In a preprocessing phase, the trusted

party selects a special round number $i^* \in \{1, \ldots, r\}$, uniformly at random, and selects bits $a_1, \ldots, a_{i^*-1}, b_1, \ldots, b_{i^*-1}$, independently, uniformly at random. It then uniformly selects a bit $w \in \{0,1\}$ and sets $a_i = b_i = w$ for all $i^* \leq i \leq r$. Thereafter, the protocol proceeds in rounds: In round i, the dealer gives Alice the bit a_i and Bob the bit b_i. If none of the parties abort, then at the end of the protocol both output $a_r = b_r = w$. If a party prematurely aborts in some round i, the honest party outputs the bit it received in the previous round (i.e., a_{i-1} or b_{i-1} respectively). If one party aborts before the other party received its first bit (i.e., a_1 or b_1), then the other party outputs a random bit.

The security of the protocol follows from the fact that, unless the adversary aborts in round i^*, it cannot bias the output of the protocol. The view of any of the parties up to round $i \leq i^*$ is independent of the value of i^*, hence, any adversary corrupting a single party can guess i^* with probability at most $1/r$.

To eliminate the trusted party, Moran et al. first turn the trusted party from an on-line dealer into an off-line dealer, i.e., one that computes some values in a preprocessing phase, deals them to the parties, and halts. To achieve this, they use a 2-out-of-2 secret-sharing scheme and an authentication scheme. The trusted party selects i^*, bits $a_1, \ldots, a_{i^*-1}, b_1, \ldots, b_{i^*-1}$, and a bit $w \in \{0,1\}$ as before. It then selects random shares for a_i and b_i for each $i \in \{1, \ldots, r\}$. That is, it computes shares $a_i^{(A)} \oplus a_i^{(B)} = a_i$ and $b_i^{(A)} \oplus b_i^{(B)} = b_i$. At the beginning of the protocol, the trusted party sends to Alice her shares of the a_i's, that is $a_i^{(A)}$, and the shares $b_i^{(A)}$ together with an authentication of the $b_i^{(A)}$ (i.e., authenticated shares of the b_i's), and sends to Bob his shares of the b_i's and authenticated shares of the a_i's. The protocol now proceeds in rounds. In each round i Bob sends to Alice his authenticated share of a_i, so Alice can reconstruct the bit a_i. Alice then sends to Bob her authenticated share of b_i. An adversary cannot forge an authentication, and is, thus, essentially limited to aborting in deviating from the prescribed protocol.

The off-line dealer is then replaced by a (constant round) secure-with-abort two-party protocol [11] for computing the preprocessing functionality. That is, at the end of the initialization protocol, the parties get the authenticated shares of the a_i's and the b_i's, while the underlying i^* and authentication keys stay secret. The security of the 2-party preprocessing protocol guarantees that a bounded adversary is essentially as powerful as in a computation with an off-line dealer.

3 Coin Tossing with Dishonest Majority – A Warm-Up

In this section we present two warm-up constructions for multiparty coin-tossing with bias $O(1/r)$ where r is the number of rounds in the protocol. The first construction considers the case that at most half of the parties are malicious (however, there is no majority of honest parties). The second construction solves the problem of coin tossing with 5 parties, where at most 3 are malicious. These two protocols demonstrate the main difficulties in constructing multiparty coin-tossing protocols with dishonest majority, alongside the techniques we use to overcome these difficulties. In Sections 4–6, we present a construction for the

general case that combines ideas from the two constructions presented in this section.

The main issue of any coin-tossing protocol is how to deal with premature aborts. The protocol must instruct any large enough subset of parties (i.e., at least as large as the set of honest parties) how to jointly reconstruct a bit if all other parties abort the protocol. Since there is no honest majority, an adversary controlling some set of parties can compute the output of this set assuming that the other parties abort. The problem in designing a protocol is how to ensure that this information does not enable the adversary to bias the output.

3.1 Multiparty Coin Tossing When Half of the Parties Can Be Malicious

In this section we present a protocol with optimal (up to a small constant) bias with respect to round complexity, when up to half the parties may be corrupt. We next give an informal description of the protocol with an on-line trusted party who acts as a dealer.

To construct a protocol for multiparty coin tossing for the case that up to half the parties may be malicious, the parties simulate the 2-party protocol of [12]. That is, we partition the parties into two sets A and B, one will simulate Alice and the other will simulate Bob. The main difficulty is that the adversary is not restricted to corrupting parties only in one of these sets. To overcome this problem, in our partition A contains a single party p_1, and the set B consists of the parties $p_2, \ldots, p_m$. If the adversary corrupts p_1, it gains full access to the view of Alice in the 2-party protocol; however, in this case a strict majority of the parties simulating Bob is honest, and the adversary will gain no information about the bits of Bob, i.e., the b_i's.

We next describe the protocol. In a preprocessing phase, the dealer uniformly selects $i^* \in \{1, \ldots, r\}$ and then uniformly and independently selects $a_1, \ldots, a_{i^*-1}, b_1, \ldots, b_{i^*-1}$. Finally, it uniformly selects $w \in \{0, 1\}$ and sets $a_i = b_i = w$ for all $i^* \leq i \leq r$. In each round i, the dealer sends a_i to A, selects random shares of b_i in Shamir's $m/2$-out-of-$(m-1)$ secret-sharing scheme, and sends each share to the appropriate party in B. We stress that formally (to model a rushing adversary), the dealer first sends the malicious parties their messages, allows them to abort, and proceeds as described below.

During the execution some parties might abort; we say that a party is active if it has not aborted. If a party p_j prematurely aborts, then the trusted party notifies all currently active parties that p_j has aborted. We next describe the actions when a party aborts:

- If p_1 aborts in round i, then the parties in B reconstruct b_{i-1}, output it, and halt. In this case, since p_1 is corrupt, at least $m/2$ honest parties exist in B and, thus, they will be able to reconstruct the output.
- If in round i parties from B abort such that less than $m/2$ active parties remain in B, then p_1 broadcasts a_{i-1} to the remaining $m/2-1$ parties in B and all (honest) parties output a_{i-1} and halt. In this case p_1 must be honest and hence can be trusted to broadcast a_{i-1}.

– While there are still at least $m/2$ active parties in B (i.e., at most $m/2-1$ of them abort) and p_1 is active, the protocol proceeds without a change.

To prevent cheating, the dealer needs to sign the messages given to the parties. We omit these details in this section.

Recall that at most $m/2$ out of the m parties are malicious. Thus, if p_1 is corrupted, then at most $(m/2)-1$ parties in B are corrupted, and they cannot reconstruct b_i. To see that the above protocol is $O(1/r)$-secure is now straightforward. An adversary wishing to bias the protocol must cause premature termination. To do so, it must either corrupt p_1 (and gain no information on the b_i's) or otherwise corrupt $m/2$ parties in B (hence, leaving p_1 uncorrupted). Thus, for any adversary in the multi-party protocol there is an adversary corrupting Alice or Bob in the on-line setting of the two party protocol of [12] that is essentially as powerful. An important feature that we exploit in our protocol is the fact that in the two-party protocol Bob does not need its bit b_{i-1} if Alice does not abort. Thus, in our protocol the parties in B do not reconstruct b_{i-1} unless p_1 aborts in round i.

More work is required in order to eliminate the trusted dealer, however, the arguments justifying such a move are a special case of those described in Section 6.

3.2 A 5-Party Protocol That Tolerates up to 3 Malicious Parties

In this section we consider the case where $m=5$ and $t=3$, i.e., a 5-party protocol where up to 3 of the parties may be malicious. As in previous protocols, we first sketch our construction assuming there is a special on-line trusted dealer. This dealer interacts with the parties in rounds, sharing bits to subsets of parties, and proceeds with the normal execution as long as at least 4 of the parties are still active.

Denote the trusted dealer by T and the parties by $p_1,\ldots,p_5$. Let $S_1,\ldots,S_{10}$ be all possible triplets of parties, i.e., $S_j \subset \{p_1,\ldots,p_5\}$ such that $|S_j|=3$. Denote by $\sigma^i_{S_j}$ a bit to be recovered by S_j if the protocol terminates in round $i+1$. In a preprocessing phase, the dealer T selects uniformly at random $i^* \in \{1,\ldots,r\}$, indicating the special round in this five-party protocol. Then, for every $0 \le i < i^*$ it selects $\sigma^i_{S_j}$ independently and uniformly at random for each $j \in \{1,\ldots,10\}$. Finally, it independently and uniformly selects a random bit w and sets $\sigma^i_{S_j}=w$, for every $i \in \{i^*,\ldots,r\}$ and for every $j \in \{1,\ldots,10\}$.

The dealer T interacts with $p_1,\ldots,p_5$ in rounds, where round i, for $1 \le i \le r$ consists of three phases:

First phase. The dealer sends to the adversary all the bits $\sigma^i_{S_j}$ such that there is a majority of corrupted parties in S_j, i.e., at least 2 parties in S_j are controlled by the adversary.

Second phase. The adversary sends to T a list of parties that abort in the current round. If there are less than 4 active parties (i.e., there are either 2

or 3 active parties),[3] T sends $\sigma_{S_j}^{i-1}$ to the active parties, where S_j is the lexicographically first triplet that contains all of the active parties. The honest parties output this bit and halt.

Third phase. If at least 4 parties are active, T notifies the active parties that the protocol proceeds normally.

If after r rounds, there are at least 4 active parties, T simply sends w to all active parties and the honest parties output this bit.

As an example of a possible execution of the protocol, assume that p_1 aborts in round 4 and p_3 and p_4 abort in round 26. In this case, T sends $\sigma_{\{p_1,p_2,p_5\}}^{25}$ to p_2 and p_5, which output this bit.

Recall that the adversary obtains the bit S_j if at least two parties in S_j are malicious. If the adversary causes the dealer to halt, then, either there are two active parties, both of them must be honest, or there are three active parties and at most one of them is malicious. In either case, the adversary does not know $\sigma_{S_j}^{i-1}$ in advance. Furthermore, the dealer reveals the appropriate bit $\sigma_{S_j}^{i-1}$ to the active parties. Jumping ahead, these properties are later preserved in a real world protocol by using a 2-out-of-3 secret-sharing scheme.

We next argue that any adversary can bias the output of the above protocol by at most $O(1/r)$. As in the protocol of Moran et al., the adversary can only bias the output by causing the protocol to terminate in round i^*. In contrast to the protocol of [12], in our protocol if in some round there are two bits σ_S^i and $\sigma_{S'}^i$ that the adversary can obtain such that $\sigma_S^i \neq \sigma_{S'}^i$, then the adversary can deduce that $i \neq i^*$. However, there are at most 7 bits that the adversary can obtain in each round (i.e., the bits of sets S containing at least two malicious parties). For a round i such that $i < i^*$, the probability that all these bits are equal to (say) 0 is $(1/2)^7$. Such rounds are indistinguishable to the adversary from round i^*. Intuitively, the best an adversary can do is guess one of these rounds, and therefore cannot succeed guessing with probability better than $1/2^7$. Thus, the bias the adversary can cause is $2^7/r$.

Eliminating the Dealer of the 5-Party Protocol. We eliminate the trusted on-line dealer in a few steps using a few layers of secret-sharing schemes. First, we change the on-line dealer, so that in each round i it shares the bit σ_S^i of each subset S among the parties of S using a 2-out-of-3 secret-sharing scheme – called *inner* secret-sharing scheme. The same requirement on σ_S^i as in the above protocol are preserved using this inner secret-sharing scheme. That is, the adversary is able to obtain information on σ_S^i only if it controls at least two of the parties in S. On the other hand, if the adversary does not control at least two parties in S (i.e., there is an honest majority in S), then, in round i, the honest parties can reconstruct σ_S^{i-1} (if so instructed by the protocol).

Next we turn the on-line dealer into an off-line dealer. That is, we show that it is possible for the dealer to only interact with the parties once, sending each

[3] The reason for requiring that the dealer does not continue when at least two parties abort will become clear when we transform the protocol to a protocol with an off-line dealer.

party some input, so that thereafter, the parties interact in rounds (without the dealer) and in each round i each party learns its shares in the ith inner secret-sharing scheme. That is, in each round i, each party p learns a share of σ_S^i in a 2-out-of-3 secret-sharing scheme, for every triplet S such that $p \in S$. For this purpose, the dealer computes, in a preprocessing phase, the appropriate shares for the inner secret-sharing scheme. The shares of each round for each party p are then shared in a 2-out-of-2 secret-sharing scheme, where p gets one of the two shares (this serves as a mask, allowing only p to later reconstruct its shares of the appropriate σ_S^i's). All parties get shares in a 4-out-of-5 Shamir secret-sharing scheme of the other share of the 2-out-of-2 secret sharing. We call the resulting secret-sharing scheme the *outer* scheme.

The use of the 4-out-of-5 secret-sharing scheme plays a crucial role in eliminating the on-line dealer. On the one hand, it guarantees that an adversary, corrupting at most three parties, cannot reconstruct the shares of round i before round i. On the other hand, at least two parties must not reveal their shares in order to prevent a reconstruction of the outer scheme (this is why we cannot proceed after 2 parties aborted). Hence, the protocol proceed normally as long as at least 4 parties are active. If, indeed, at least two parties abort (in round i), then the remaining parties use their shares of the inner scheme to reconstruct the bit σ_S^{i-1} for the appropriate triplet S.

To prevent malicious parties from cheating, by say, sending false shares and causing reconstruction of wrong secrets, every message that a party should send during the execution of the protocol is signed in the preprocessing phase (together with the appropriate round number and with the party's index). In addition, the dealer sends a verification key to each of the parties. To conclude, the off-line dealer gives each party the signed shares for the outer secret sharing scheme together with the verification key.

The protocol with the off-line dealer proceeds in rounds. In round i of the protocol all parties broadcast their (signed) shares in the outer (4-out-of-5) secret-sharing scheme. Thereafter, each party can unmask the message it receives (with its share in the appropriate 2-out-of-2 secret-sharing scheme) to obtain its shares in the 2-out-of-3 sharing of the bits σ_S^i (for the appropriate sets S's to which the party belongs). If a party stops broadcasting messages or broadcasts improperly signs messages, then all other parties consider it as aborted. If two or more parties abort, the remaining parties reconstruct the bit of the lexicographically first triplet that contains all of them, as described above. In the special case of premature termination already in the first round, the remaining parties engage in a fully secure protocol (with honest majority) to toss a completely random coin.

Finally, we replace the off-line dealer by using a secure with abort protocol with cheat detection computing the functionality computed by the dealer. The details of this final step are given in Section 6.

The above construction can be generalized in a straightforward manner to any number m of parties and any number t of malicious parties such that $t < 2m/3$. However, in the protocol described in Section 4 the bias on the output is substantially smaller; this is done using a better way for distributing bits to subsets.

4 Coin-Tossing with Dishonest Majority – Our Main Construction

In Sections 4–6 we present our main result – a coin-tossing protocol that has nearly optimal bias and can tolerate up to 2/3 fraction of malicious parties. More specifically, we prove the following theorem:

Theorem 2. *If enhanced trap-door permutations exist, then for any m, t, and r such that $t < 2m/3$, there is an r-round m-party coin-tossing protocol tolerating up to t malicious parties and has bias $O\left(2^{2^{k+1}}/r'\right)$, where $k = 2t - m$ and $r' = r - O(k+1)$.*

In the above theorem k is the difference between the number of malicious parties and the number of honest parties, i.e., $k = t - (m - t) = 2t - m$.

Following [12], we describe our protocol in two steps. In Section 5, we describe Protocol CoinTossWithDealer$_r$ that uses an online trusted party. In Section 6, we get rid of the on-line dealer. This simplifies the description and understanding of our protocols. More importantly, we can prove the security of our main protocol in a modular way. We first prove

Theorem 3. *Protocol* CoinTossWithDealer$_r$ *is an r-round m-party coin-tossing protocol with an on-line dealer tolerating up to t malicious parties that has bias $O\left(2^{2^{k+1}}/r\right)$.*

We then consider the on-line dealer of Protocol CoinTossWithDealer$_r$ as an ideal functionality. In this protocol, the honest parties do not send any messages and in each round the dealer sends messages to the parties; we consider an interactive functionality sending the messages that the dealer sends. We prove

Theorem 4. *Let $t < 2m/3$. If enhanced trap-door permutations exist, the protocol presented in Section 6 is a computationally-secure implementation with $r'+O(k+1)$ rounds of the dealer functionality in Protocol* CoinTossWithDealer$_{r'}$.

The above theorem is proved using the hybrid model techniques of Canetti [5]. Theorem 2 follows from Theorem 3 and Theorem 4 by a composition argument.

We stress that constructing fair coin-tossing protocols assuming a trusted dealer is an easy task, e.g., the trusted party can choose a random bit and send it to each party. However, when considering a rushing adversary one cannot eliminate the trusted party in this protocol. The coin-tossing protocol we describe, Protocol CoinTossWithDealer$_r$, is designed such that it is possible to transform it to a protocol with no trusted party.

5 Coin-Tossing with Dishonest Majority and an On-Line Dealer

In this section we describe a protocol with a special trusted party T who acts as an on-line dealer interacting with the parties in rounds. In the protocol we

describe, in every round the trusted party T chooses bits for some subsets of parties (the collection of subsets that receive a bit is part of the design of the protocol). Since, in the real world, the adversary can be rushing, the interaction between the parties and T in each round has three phases. In the first phase, for each set S that contains enough malicious parties, the trusted party sends the bit of the set to the malicious parties in the subset S. In the second phase, malicious parties may abort the computation (and by that prevent later reconstruction of some of the information). To do so, these parties send to T an "abort" message. Finally, in the third phase, the actual (ideal) secret sharing takes place.

Protocol CoinTossWithDealer$_r$

Inputs: The input of each party p_i is the security parameter 1^n, a polynomial $r = r(n)$ specifying the number of rounds in the protocol, and an upper bound t on the number of corrupted parties.

Underlying Subsets: Let $P_j = \{p_j\}$ for $1 \le j \le k+1$ and $P_{k+2} = \{p_{k+2}, \ldots, p_m\}$. Define $Q_J = \cup_{j \in J} P_j$ for each $J \subset \{1, \ldots, k+2\}$.

For each subset P_j define a reconstruction threshold value o_j: For $1 \le j \le k+1$ define $o_j = 1$ and define $o_{k+2} = m - t$. Finally, $o_J = \sum_{j \in J} o_j$ for each $J \subset \{1, \ldots, k+2\}$.

Instructions for the (trusted) dealer:

The preprocessing phase: Select a bit σ_J^i for every subset Q_J and every round i as follows:

1. Select $i^* \in \{1, \ldots, r\}$ and $w \in \{0, 1\}$ independently with uniform distribution.
2. For each $J \subset \{1, \ldots, k+2\}$, select $\sigma_J^0, \ldots, \sigma_J^{i^*-1}$ independently with uniform distribution.
3. For each $J \subset \{1, \ldots, k+2\}$, set $\sigma_J^{i^*} = \ldots = \sigma_J^r = w$ for $i^* \le i \le r$.

Interaction rounds: In each round $1 \le i \le r$ of the protocol, interact with the parties in three phases:

- **The peeking phase:** For each $J \subset \{1, \ldots, k+2\}$, if Q_J contains at least o_J malicious parties, send the bit σ_J^i to all malicious parties in Q_J.
- **The abort phase:** Upon receiving an $abort_j$ message from party p_j, remove party p_j from the list of active parties and notify all parties that party p_j is inactive. (Ignore all other types of messages.)
  If at least $m - t$ parties have aborted so far, move to the premature termination process.
- **The main phase:** Send "proceed" to all parties.

Premature termination process: This round consists of two phases, after which the protocol terminates and all honest parties hold the same output.

- **The abort phase:** Upon receiving an $abort_j$ message from party p_j, remove party p_j from the list of active parties.
- **The default output phase:** Let D be the set of indices of parties that aborted the protocol thus far, i.e., $D = \{j \mid p_j \text{ has aborted}\}$.

- If $|D \cap \{k+2, \ldots, m\}| \geq m - t$ then $J = \{1, \ldots, k+1\} \setminus D$.
- If $|D \cap \{k+2, \ldots, m\}| < m - t$ then $J = (\{1, \ldots, k+1\} \setminus D) \cup \{k+2\}$.
- Send $w' = \sigma_J^{i-1}$ to all parties.

Normal termination: This phase is executed if the last round of the protocol is completed.
Send w to all parties.

Instructions for honest parties: Upon receiving output y from the dealer, output y. (Honest parties do not send any message throughout the protocol.)

We next informally explain why the protocol has small bias, that is, we give a sketch of the proof of Theorem 3. First, we claim that the adversary can bias the output only if the premature termination occurs in round i^*:

1. If the premature termination round occurs after round i^* (or does not occur at all), then the output is already fixed.
2. If the premature termination round occurs before round i^*, then the adversary does not know the random bit σ_J^{i-1} that the honest parties output:
   (a) If $|D \cap \{k+2, \ldots, m\}| \geq m - t$, then $J = \{1, \ldots, k+1\} \setminus D$ and $o_J = |J|$. There are at most t corrupt parties and at least $m - t$ of them are in $Q_{\{k+2\}}$, thus, at most $t - (m - t) = k$ in $\{p_1, \ldots, p_{k+1}\}$. In other words, there is at least one honest (and therefore active) party in Q_J, and the trusted party does not send the bit σ_J^{i-1} to the parties in Q_J.
   (b) If $|D \cap \{k+2, \ldots, m\}| < m - t$, then $J = (\{1, \ldots, k+1\} \setminus D) \cup \{k+2\}$. Let $\alpha = |D \cap \{1, \ldots, k+1\}|$. In this case, $o_J = k+1-\alpha+m-t = t+1-\alpha$. The set Q_J contains at most $t-\alpha < o_J$ corrupt parties, thus, these parties do not get the bit σ_J^{i-1} from the trusted party.

Thus, the adversary can bias the output only if it guesses i^*. If $\sigma_J^i \neq \sigma_{J'}^i$ for two bits that the adversary gets from the trusted party, then it can learn that $i < i^*$. It can be shown that the adversary can get at most 2^{k+1} such bits (out of the 2^{k+2} bits). With probability $1/2^{2^{k+1}}$ all these bits are all equal in a round prior to i^* and the adversary cannot distinguish such round from i^*. By Lemma 2 in [9], this implies that the adversary can guess i^* with probability at most $2^{2^{k+1}}/r$. Therefore, the bias is $O(2^{2^{k+1}}/r)$.

Roughly speaking, transforming the above informal arguments into a formal proof, which uses the real vs. ideal paradigm, works as follows. We define a simulator $\mathcal{S}$ that for an adversary $\mathcal{A}$, first uses the ideal CoinToss() functionality to toss a completely fair coin w_S (this coin is the output of the honest parties in the simulated execution). Then, in order to simulate the view of $\mathcal{A}$, the simulator $\mathcal{S}$ runs $\mathcal{A}$ internally and interacts with $\mathcal{A}$ playing the role of T (with $w = w_S$); the only difference is that in the a premature termination, it always sends the parties w_S. The arguments of the above proof sketch show that view of $\mathcal{A}$ together with the output of the honest parties are identically distributed whenever premature termination does not occur in the special round i^*. The above bound on the ability of any adversary to correctly guess i^* finalizes the proof. The formal proof will appear in the full version of this paper.

6 Omitting the On-Line Dealer

In this section we show how Protocol CoinTossWithDealer$_r$, presented in Section 5, can be transformed into a real-world protocol. That is, we present a fully secure m-party protocol implementing the ideal functionality described in Protocol CoinTossWithDealer$_r$. The resulting protocol has r' rounds, where $r' = r + c(k+1)$, for some constant c, and is executed in a network where the parties communicate via an authenticated broadcast channel. Before formally describing our construction, we outline its main components.

The inner secret-sharing scheme. To implement the ideal secret sharing functionality of the trusted party T in the CoinTossWithDealer$_r$ protocol to share the bits σ^i_J, we use an o_J-out-of-$|Q_J|$ Shamir secret-sharing scheme. That is, in each round i, each party $p_j \in Q_J$ obtains a share $S^{i,J}_j$ in a o_J-out-of-$|Q_J|$ secret-sharing of σ^i_J. The same requirement on σ^i_J as in the ideal protocol are preserved using this inner secret-sharing scheme. That is, the adversary is able to obtain information on σ^i_J only if it controls at least o_J of the parties in Q_J. On the other hand, if, in a premature termination in round i, at least o_J parties in Q_J work together, then they can reconstruct σ^{i-1}_J.

The outer secret-sharing scheme. In the ideal protocol, the adversary never learns anything about the bits σ^i_J before round i begins. To achieve this property in the real-world protocol, the shares of the inner secret-sharing schemes of all rounds are shared, in a preprocessing step, using a $(t+1)$-out-of-m secret-sharing scheme. The $t+1$ threshold guarantees that the adversary cannot see the shares of the inner secret-sharing scheme for a given round i without the help of honest parties, which will not be given before round i.

In each round i the parties send messages so that each party can reconstruct its shares in the inner secret-sharing schemes of round i. Since all messages are broadcast and all parties can see them, the shares that party p_j receives in round i are masked by using yet another layer of secret-sharing. Specifically, a share $S^{i,J}_j$ to be reconstructed by p_j in round i is signed and shared (already in the preprocessing phase) in a 2-out-of-2 secret sharing scheme, such that one share is given to p_j and the other is shared among all parties in a $(t+1)$-out-of-m secret-sharing scheme. We refer to the combination of these two layers of secret-sharing as the outer secret-sharing scheme.

Premature Termination. The $t+1$ threshold of the outer secret sharing scheme allows a successful reconstruction (of the shares of the inner scheme) as long as at least $t+1$ parties participate in the reconstruction. This allows the real-world protocol to proceed with normal interaction rounds as long as less than $m-t$ parties have aborted (as does the ideal-world protocol). This property is crucial to the success of the real world protocol, since in the complementary event that during round i the number of parties that have aborted is at least $m-t$, then an honest majority is guaranteed (since $t < 2m/3$). Thus, in a premature termination in round i, the active parties can engage in a fully secure multiparty computation of the appropriate functionality, i.e., the CoinToss functionality in the special case that $i = 1$ and a reconstruction functionality otherwise.

Signatures. In order to confine adversarial strategies to premature aborts, the messages that the parties send are signed (together with the appropriate round number and the index of the sending party), and a verification key is given to all parties. Furthermore, all shares in the inner secret-sharing scheme are signed (as they are used as messages if reconstruction is required). Any message failing to comply with the prescribed protocol is considered an abort message. Since all messages are publicly broadcast, all parties can keep record of all aborts.

The preliminary phase. The goal of the preliminary phase is to compute the MultiShareGen$_r$ functionality, which computes the bits for the underlying sets and the signed shares for the inner and outer secret-sharing schemes. As an honest majority is not guaranteed, it is not possible to implement this functionality by a secure protocol with fairness. That is, we cannot implement an ideal functionality where a trusted party computes the MultiShareGen$_r$ functionality and sends the appropriate output to each party. However, since the outputs of the MultiShareGen$_r$ functionality do not reveal any information regarding the output of the protocol to any subset of size at most t, fairness is not essential for this part of the computation. We use a protocol with cheat detection, that is, if the protocols fails at least one corrupt party is identified by all honest parties. The computation is then repeated without the detected malicious parties.

More formally, we compute the MultiShareGen$_r$ functionality using a multiparty computation protocol that is secure-with-abort with cheat-detection. Informally, this means that we use a protocol that implements the following ideal model: the trusted party computes the MultiShareGen$_r$ functionality and gives the outputs of the corrupted parties to the adversary; the adversary either sends "proceed", in which case, the trusted party sends the appropriate output to each honest party; otherwise, the adversary sends "abort$_j$" (where, p_j is in the set of corrupted parties) to the trusted party, which in turn notifies the honest parties that p_j is malicious. Using methods from Pass [13], one can obtain a constant-round multiparty protocol secure-with-abort with cheat-detection. Since this protocol is repeated at most $k+1$ times before an honest majority is guaranteed, the round complexity of the preliminary phase is $O(k)$.

We first present the initialization functionality of the protocol,

Functionality MultiShareGen$_r$

Computing default bits

1. Choose $w \in \{0,1\}$ and $i^* \in \{1,\ldots,r\}$ uniformly at random.
2. For each $i \in \{1,\ldots,r\}$ and for each $J \subset \{1,\ldots,k+2\}$,
   (a) if $i \in \{1,\ldots,i^*-1\}$, then choose independently and uniformly at random $\sigma_J^i \in \{0,1\}$.
   (b) if $i \in \{i^*,\ldots,r\}$, then set $\sigma_J^i = w$.

Computing signed shares of the inner secret sharing scheme

3. Compute $(K_{\text{sign}}, K_{\text{ver}}) \leftarrow \text{Gen}(1^n)$.
4. For each $i \in \{1,\ldots,r\}$ and for each $J \subset \{1,\ldots,k+2\}$
   (a) Choose random secret shares of σ_J^i in an o_J-out-of-$|Q_J|$ Shamir's secret sharing scheme for the parties in Q_J.
   For each party $p_j \in Q_J$, let $S_j^{i,J}$ be its share of σ_J^i.

(b) For each share $S_j^{i,J}$, add the corresponding set index and the round number and sign:
$R_j^{i,J} \leftarrow (S_j^{i,J}, J, i, \mathrm{Sign}((S_j^{i,J}, J, i), K_{\mathrm{sign}})$.

Computing shares of the outer secret sharing scheme

5. For each $i \in \{1, \ldots, r\}$, for each $J \subset \{1, \ldots, k+2\}$, and for each $p_j \in Q_J$, share p_j's signed share $R_j^{i,J}$ using a 2-out-of-2 secret sharing scheme; one share is given to p_j (a private mask only p_j obtains) and the other share is shared among all parties in a $(t+1)$-out-of-m secret-sharing scheme.

Signing the messages of all parties

6. Compute the message $m_{(q,i)}$ that $p_q \in P$ broadcasts in round i by concatenating (1) p_q's identity, (2) the round number i, and (3) the shares of $R_j^{i,J}$ (for all J and for all j such that $p_j \in Q_J$) produced in Step 5 for p_q (excluding p_q's private masks).
7. Compute $M_{(q,i)} \leftarrow (m_{(q,i)}, \mathrm{Sign}(m_{(q,i)}, K_{\mathrm{sign}}))$.

Outputs: Each party p_j receives

- The verification key K_{ver}.
- The messages $M_{(j,1)}, \ldots, M_{(j,r)}$ that p_j broadcasts during the protocol.
- p_j's private masks which were produced in Step 6 for each $J \subset \{1, \ldots, k+2\}$ such that $p_j \in Q_J$.

Next, we formally define the m-party coin-tossing protocol tolerating $t < 2m/3$ malicious parties without any dealer.

Protocol MultiPartyCoinToss$_r$

Joint input: Security parameter 1^n.

Preliminary phase:

- The parties execute a secure with abort protocol with cheat detection computing Functionality MultiShareGen$_r$.
- If a party aborts, then this phase is repeated without the parties that were identified as cheaters so far.
- If the first phase was repeated $k+1$ times (thus, an honest majority is guaranteed), the parties use a multiparty secure protocol (with fairness) to toss a fair coin, output this resulting bit, and halt.
- Denote the set of indices of inactive parties (i.e., parties that cheated or aborted so far) by D.

In each round $i = 1, \ldots, r$ **do:**

- Each party $p_j \in P$ broadcasts $M_{(j,i)}$ (containing its shares in the outer secret-sharing scheme).
- If $\mathrm{Ver}(M_{(j,i)}, K_{\mathrm{ver}}) = 0$ or if p_j broadcasts an invalid or no message, then all parties mark p_j as inactive, i.e., set $D \leftarrow D \cup \{j\}$. If $|D| \geq m - t$, then the premature termination step is executed.

Premature termination step

- If $i = 1$, then the active parties use a multiparty secure protocol (with fairness) to toss a fair coin, output this resulting bit, and halt.
- Otherwise,

1. Each party p_j reconstructs $R_j^{i-1,J}$, the signed share of the "inner secret sharing scheme" produced in Step (4) of Functionality MultiShareGen$_r$, for every $J \subset \{1,\ldots,k+2\}$ such that $p_j \in Q_J$.
2. The active parties execute a secure multiparty protocol with an honest majority to compute Functionality Reconstruction, where the input of each party p_j is $R_j^{i-1,J}$ for every $J \subset \{1,\ldots,k+2\}$ such that $p_j \in Q_J$.
3. The active parties output the output of this protocol, and halt.

At the end of round r:

- Each active party p_j broadcasts the signed shares $R_j^{r,J}$ for each J such that $p_j \in Q_J$.
- Each active party reconstructs the bit σ_J^r for the lexicographically first set J such that at least o_J parties broadcast properly signed shares $R_j^{r,J}$, outputs σ_J^r, and halts.

Functionality Reconstruction

Joint Input: The indices of inactive parties, D, and the verification key, K_{ver}.

Private Input of p_j: A set of signed shares $R_j^{i-1,J}$ for each $J \subset \{1,\ldots,k+2\}$ such that $p_j \in Q_J$.

Computation:

1. For each p_j, if p_j sends a message that is not appropriately signed or malformed, then $D \leftarrow D \cup \{j\}$.
2. Define the set J:
   - If $|D \cap \{k+2,\ldots,m\}| \geq m-t$ then $J = \{1,\ldots,k+1\} \setminus D$.
   - If $|D \cap \{k+2,\ldots,m\}| < m-t$ then $J = (\{1,\ldots,k+1\} \setminus D) \cup \{k+2\}$.
3. Reconstruct σ_J^{i-1} from the shares of the active parties in Q_J.

Outputs: Each honest party p_j output the value σ_J^{i-1}.

We next claim that Functionality Reconstruction is well-defined, that is, if the functionality is computed (after premature termination in round $i > 1$), then, indeed, σ_J^{i-1} can be reconstructed. To see this, observe that the number of parties in the appropriate set Q_J that participate in the computation (i.e., not in D) is at least the reconstruction threshold o_J: If $|D \cap \{k+2,\ldots,m\}| \geq m-t$, then Q_J contains only active parties and $|Q_J| = o_J$. Notice that we already proved that in this case $|Q_J| \geq 1$. If $|D \cap \{k+2,\ldots,m\}| \leq m-t-1$, then $o_J = |J|-1+m-t$ and $|Q_J \setminus \{p_j : j \in D\}| = |J|-1+|\{k+2,\ldots,m\} \setminus D|$. To prove that $|Q_J \setminus \{p_j : j \in D\}| \geq o_J$, it suffices to show that $|\{k+2,\ldots,m\} \setminus D| \geq (m-k-1)-(m-t-1) = t-k = t-(2t-m) = m-t$.

7 Coin-Tossing Protocol for Any Constant Fraction of Corrupted Parties

In this section we describe an r-round m-party coin-tossing protocol that tolerates up to t dishonest parties, where t is some constant fraction of m, that is, $t = (1-\varepsilon)m$, for some (constant) $0 < \varepsilon$. The bias of our protocol is $O\left(\varepsilon/\sqrt{r-t}\right)$.

Before our work, the best known protocol for this scenario is an extension of Blum's two-party coin-tossing protocol [3] to an r-round m-party protocol that has bias $O\left(t/\sqrt{r-t}\right)$ [2, 6]. In this protocol, in each round i of the protocol, the parties jointly select a random bit σ_i in two phases. In the first phase, each party commits to a "private" random bit, and in the second phase the private bits are all revealed and the output bit σ_i is taken to be the XOR of all private bits. The output of the whole protocol is taken to be the value of the majority of the σ_i's. When there is a premature abort in round i, the remaining parties repeat the computation of round i and continue with the prescribed computation.

Intuitively, the best strategy for a rushing adversary to bias the output of the protocol, say toward 0, is in each round i to instruct a corrupted party to abort before the completion of the revealing phase if $\sigma_i = 1$. This is possible, since the rushing adversary learns σ_i before the completion of round i, specifically, a corrupted party can delay its message until all honest parties reveal their bit. This can go on at most t times, adding a total bias of $O\left(t/\sqrt{r}\right)$, whenever $r = \Omega(t^2)$.

We use the notion of cheat detection to limit the adversary to abort in a constant number of rounds. Roughly speaking, we follow the general structure of the above protocol in computing the σ_i's and taking the majority over them. However, we compute each σ_i using a secure with abort with cheat-detection protocol, such that either the computation is completed or at least a constant fraction of the malicious parties abort (specifically, $m - t$ malicious parties abort). Next, we briefly describe the computation of each σ_i in our protocol. That is, we show how to obtain a constant round secure-with-abort with cheat detection protocol to compute a random bit that identifies at least $m - t$ cheating parties. Let m_i be the number of active parties at the beginning of round i and t_i be a bound on the number of active corrupted parties at the beginning of round i (that is, if t' parties have aborted in rounds $1, \ldots, i-1$, then $t_i = t - t'$). We assume the existence of a constant round secure-with-abort with cheat detection protocol.

In the first phase, a preprocessing phase, active parties execute a constant round secure-with-abort with cheat detection protocol to compute a $(t_i + 1)$-out-of-m_i secret sharing of a random bit σ_i. That is, at the end of this phase, each party holds a share in a $(t_i + 1)$-out-of-m_i Shamir secret sharing scheme of σ_i. To confine adversarial strategies to aborts, the share that each party receives is signed and a verification key is given to all parties. In a second phase, a revealing phase, all parties reveal their shares and reconstruct σ_i. Broadcasting anything other than a signed share is treated as abort. To see that the above protocol achieves the required properties, observe that after the first phase the adversary cannot reconstruct σ_i. Thus, by aborting the preprocessing round, malicious parties cannot bias the output. We stress that they are able to cause the preprocessing phase to fail, at the cost of at least one malicious party being detected by all honest parties. in such a case, the preprocessing stage is repeated without the detected party. This, however, can only happen at most t times in total, throughout the whole protocol. In the revealing phase, a rushing adversary is already able to learn σ_i before the corrupted parties broadcast their messages

and thus can bias the output by not broadcasting these messages. However, by the properties of the secret sharing scheme, at least $m - t$ parties will have to not broadcast their message, and hence, effectively abort the computation. Hence, the adversary can do this at most $\frac{1-2\varepsilon}{\varepsilon}$ times throughout the protocol, before an honest majority among active parties is guaranteed. Thus, the majority function is applied to $\Omega(r-t)$ random bits, of which the adversary can bias $\frac{1-2\varepsilon}{\varepsilon}$. Thus, the total bias of the protocol is $O\left(\frac{1}{\varepsilon\sqrt{r-t}}\right)$.

Acknowledgments. We are grateful to Yehuda Lindell for many helpful discussions and great advice. We thank Oded Goldreich and Gil Segev for suggesting this problem and for useful conversations.

References

[1] Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 137–156. Springer, Heidelberg (2007)
[2] Averbuch, B., Blum, M., Chor, B., Goldwasser, S., Micali, S.: How to implement Bracha's $O(\log n)$ Byzantine agreement algorithm (1985) (unpublished manuscript)
[3] Blum, M.: Coin flipping by telephone a protocol for solving impossible problems. SIGACT News 15(1), 23–27 (1983)
[4] Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudo-random bits. SIAM J. on Computing 13, 850–864 (1984)
[5] Canetti, R.: Security and composition of multiparty cryptographic protocols. J. of Cryptology 13(1), 143–202 (2000)
[6] Cleve, R.: Limits on the security of coin flips when half the processors are faulty. In: Proc. of the 18th STOC, pp. 364–369 (1986)
[7] Cleve, R., Impagliazzo, R.: Martingales, collective coin flipping and discrete control processes (1993) (manuscript)
[8] Goldreich, O.: Foundations of Cryptography, Voume II Basic Applications. Cambridge University Press, Cambridge (2004)
[9] Gordon, D., Katz, J.: Partial fairness in secure two-party computation. Cryptology ePrint Archive, Report 2008/206 (2008), `http://eprint.iacr.org/`
[10] Katz, J.: On achieving the "best of both worlds" in secure multiparty computation. In: Proc. of the 39th STOC, pp. 11–20. ACM Press, New York (2007)
[11] Lindell, Y.: Parallel coin-tossing and constant-round secure two-party computation. J. of Cryptology 16(3), 143–184 (2003)
[12] Moran, T., Naor, M., Segev, G.: An optimally fair coin toss. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 1–18. Springer, Heidelberg (2009)
[13] Pass, R.: Bounded-concurrent secure multi-party computation with a dishonest majority. In: Proc. of the 36th STOC, pp. 232–241 (2004)
[14] Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: Proc. of the 21st STOC, pp. 73–85 (1989)