An Almost-Optimally Fair Three-Party Coin-Flipping Protocol

[Extended Abstract] [*]

Iftach Haitner[†]
School of Computer Science
Tel Aviv University
Israel
iftachh@cs.tau.ac.il

Eliad Tsfadia[‡]
School of Computer Science
Tel Aviv University
Israel
eliadtsf@post.tau.ac.il

ABSTRACT

In a multiparty *fair* coin-flipping protocol, the parties output a common (close to) unbiased bit, even when some corrupted parties try to bias the output. Cleve [STOC 1986] has shown that in the case of dishonest majority (i.e., at least half of the parties can be corrupted), in *any* m-round coin-flipping protocol, the corrupted parties can bias the honest parties' common output bit by $\Omega(\frac{1}{m})$. For more than two decades, the best known coin-flipping protocols against dishonest majority had bias $\Theta(\frac{\ell}{\sqrt{m}})$, where ℓ is the number of corrupted parties. This was changed by a recent breakthrough result of Moran et al. [TCC 2009], who constructed an m-round, *two*-party coin-flipping protocol with optimal bias $\Theta(\frac{1}{m})$. In a subsequent work, Beimel et al. [Crypto 2010] extended this result to the multiparty case in which *less than* $\frac{2}{3}$ of the parties can be corrupted. Still for the case of $\frac{2}{3}$ (or more) corrupted parties, the best known protocol had bias $\Theta(\frac{\ell}{\sqrt{m}})$. In particular, this was the state of affairs for the natural three-party case.

We make a step towards eliminating the above gap, presenting an m-round, three-party coin-flipping protocol, with bias $\frac{O(\log^2 m)}{m}$. Our approach (which we also apply for the two-party case) does not follow the "threshold round" paradigm used in the work of Moran et al. and Beimel et al., but rather is a variation of the majority protocol of Cleve, used to obtain the aforementioned $\Theta(\frac{\ell}{\sqrt{m}})$-bias protocol.

[*]Full version can be found at [26].

[†]Research supported by ISF grant 1076/11, the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11) and US-Israel BSF grant 2010196.

[‡]Research supported by ISF grant 1076/11, the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), US-Israel BSF grant 2010196 and Check Point Institute for Information Security.

Categories and Subject Descriptors

F.0 [**Theory of Computation**]: General

General Terms

Theory

Keywords

coin-flipping protocols; fairness; fair computation

1. INTRODUCTION

In a multiparty *fair* coin-flipping (-tossing) protocol, the parties output a common (close to) unbiased bit, even though some corrupted parties try to bias the output. More formally, such protocols should satisfy the following two properties: first, when all parties are honest (i.e., follow the prescribed protocol), they all output the *same* bit, and this bit is unbiased (i.e., uniform over $\{0,1\}$). Second, even when some parties are corrupted (i.e., collude and arbitrarily deviate from the protocol), the remaining parties should still output the *same* bit, and this bit should not be too biased (i.e., its distribution should be close to uniform over $\{0,1\}$). We emphasize that, unlike weaker variants of coin-flipping protocol known in the literature, the honest parties should output a common bit, regardless of what the corrupted parties do. In particular, they are not allowed to abort if a cheat was noticed.

When a majority of the parties are honest, efficient and *completely* fair coin-flipping protocols are known as a special case of secure multiparty computation with an honest majority [9].[1] When an honest majority is not guaranteed, however, the situation is more complex.

Negative results.

Cleve [13] showed that for *any* efficient two-party m-round coin-flipping protocol, there exists an efficient adversary to bias the output of the honest party by $\Theta(1/m)$. This lower bound extends to the multiparty case via a simple reduction.

Positive results.

Assuming one-way functions exist, Cleve [13] showed that a simple m-round majority protocol can be used to derive a

[1]Throughout, we assume a broadcast channel is available to the parties.

t-party coin-flipping protocol with bias $\Theta(\frac{\ell}{\sqrt{m}})$ (against dishonest majority), where ℓ is the number of corrupted parties. For more than two decades, Cleve's protocol was the best known fair coin-flipping protocol (without honest majority), under *any* hardness assumption, and for *any* number of parties. In a recent breakthrough result, Moran et al. [33] constructed an m-round, *two*-party coin-flipping protocol with optimal bias of $\Theta(\frac{1}{m})$. The result holds for any efficiently computable m, and under the assumption that oblivious transfer protocols exist. In a subsequent work, Beimel et al. [6] extended the result of [33] for the multiparty case in which *less than* $\frac{2}{3}$ of the parties can be corrupted. More specifically, for any $\ell < \frac{2}{3} \cdot t$, they presented an m-round, t-party protocol, with bias $\frac{2^{2\ell - t}}{m}$ against (up to) ℓ corrupted parties.

Still for the case of $\frac{2}{3}$ (or more) corrupted parties, the best known protocol was the $\Theta(\frac{\ell}{\sqrt{m}})$-bias majority protocol of [13]. In particular, this was the state of affairs for the natural three-party case (where two parties are corrupt).

1.1 Our Result

We present an almost-optimally fair, three-party coin-flipping protocol.

Theorem 1.1 (main theorem, informal). *Assuming the existence of oblivious transfer protocols, then for any $m \in \text{poly}$ there exists an m-round, three-party coin-flipping protocol, with bias $\frac{O(\log^2 m)}{m}$ (against one, or two, corrupted parties).*

As a building block towards constructing our three-party protocol, we present an alternative construction for two-party, almost-optimally fair coin-flipping protocols. Our approach does not follows the "threshold round" paradigm used in [33, 6], but rather is a variation of the aforementioned $\Theta(\frac{\ell}{\sqrt{m}})$-bias, coin-flipping protocol of [13].

1.2 Additional Related Work

Cleve and Impagliazzo [14] showed that in the *fail-stop model*, any two-party m-round coin-flipping protocol has bias $\Omega(\frac{1}{\sqrt{m}})$; adversaries in this model are computationally unbounded, but they must follow the instructions of the protocol, except for being allowed to abort prematurely. Dachman-Soled et al. [15] showed that the same holds for $o(m/\log m)$-round protocols in the random-oracle model — the parties have oracle access to a uniformly chosen function over m bit strings.

There is a vast literature concerning coin-flipping protocols with weaker security guarantees. Most notable among these are protocols that are *secure with abort*. According to this security definition, if a cheat is detected or if one of the parties aborts, the remaining parties are not required to output anything. This form of security is meaningful in many settings, and it is typically much easier to achieve; assuming one-way functions exist, secure-with-abort protocols of negligible bias are known to exist against any number of corrupted parties [11, 27, 34]. To a large extent, one-way functions are also necessary for such coin-flipping protocols [10, 25, 28, 31].

Coin-flipping protocols were also studied in a variety of other models. Among these are collective coin-flipping in the *perfect information model*: parties are computationally unbounded and all communication is public [3, 8, 17, 36, 37], and protocols based on physical assumptions, such as quantum computation [1, 4, 5] and tamper-evident seals [32].

Perfectly fair coin-flipping protocols (i.e., zero bias) are a special case of protocols for *fair* secure function evaluation (SFE). Intuitively, the security of such protocols guarantees that when the protocol terminates, either everyone receives the (correct) output of the functionality, or no one does. While Cleve [13]'s result yields that some functions do not have fair SFE, it was recently shown by Gordon et al. [23] that many interesting function families do have (perfectly) fair SFE.

1.3 Our Techniques

The following is a high-level description of the ideas underlying our three-party fair coin flipping protocol. We start by describing the two-party protocol of Moran et al. [33], and explain why natural extensions of their approach (such as the one used in [6]) fall short when it comes to constructing three-party fair protocols. We next explain our new approach for two-party protocols, and then extend this approach to three parties.

Throughout, we assume without loss of generality that if a corrupted party aborts in a given round, it sends an abort message to all other parties at the *end* of this round (after seeing the messages sent by the non-aborting parties). To keep the discussion simple, we focus on security against fail-stop adversaries — the parties follow the prescribed protocol, but might abort prematurely. Achieving this level of security is the heart of the matter, since (assuming one-way functions exist) there exists a round-preserving reduction from protocols secure against fail-stop adversaries into protocols of full-fledged security [20].

1.3.1 *The Two-Party Protocol of Moran et al.*

For $m \in \mathbb{N}$, the $(2m)$-round, two-party protocol $(\mathsf{P}_0, \mathsf{P}_1)$ of Moran et al. [33] is defined as follows.[2] Following a common paradigm for fair multiparty computations [6, 22, 30], the protocol starts by the two parties using oblivious transfer (OT) to securely compute the following "share generating" random function.

Algorithm 1.2 (function $\mathsf{SharesGen}$).

1. *Uniformly sample* $c \leftarrow \{0,1\}$ *and* $i^* \leftarrow [m]$ $(= \{1,\ldots,m\})$.
2. *For* $i = 1$ *to* m, *let*
   (a) $(d_i^0, d_i^1) = \begin{cases} \text{uniform } \textit{sample from } \{0,1\}^2, & i < i^* - 1 \\ (c,c), & \textit{otherwise.} \end{cases}$
   (b) $c_i = \begin{cases} \perp, & i < i^* \\ c, & \textit{otherwise.} \end{cases}$
3. *Split each of the* $3m$ *values* $d_1^0, d_1^1, \ldots, d_m^0, d_m^1, c_1, \ldots, c_m$ *into two "shares," using a 2-out-of-2 secret sharing scheme, and output the two sets of shares.*

Protocol 1.3 $((\mathsf{P}_0, \mathsf{P}_1))$.

Initial step: The parties securely compute the function $\mathsf{SharesGen}$, *where each party gets one set of shares.*

[2] The protocol described below is a close variant of the original protocol of Moran et al. [33], which serves our presentation better.

Main loop: For $i = 1$ to m, do

(a) P_0 sends to P_1 its share of d_i^1, and P_1 sends to P_0 its share of d_i^0.

- *P_0 reconstructs the value of d_i^0, and P_1 reconstructs the value of d_i^1.*

(b) Each party sends to the other party its share of c_i.

- *Both parties reconstruct the value of c_i.*

Output: The parties output c_i, for the first i for which $c_i \neq \perp$.

Abort: If P_0 aborts, party P_1 outputs the value of d_i^1 for the maximal *$i \in [m]$ for which it has reconstructed this value. If there is no such i, P_1 outputs a uniform bit. (The case that P_1 aborts is analogously defined).*

We start with few observations regarding the secure computation of $\mathsf{SharesGen}$ done in the above protocol.

- The computation of $\mathsf{SharesGen}$ is *not* fair: the parties get their parts of the output (i.e., their shares) in an *arbitrary* manner. Specifically, the corrupted party might prematurely abort after learning its part of the output, preventing the other party from getting its part.
- Since $\mathsf{SharesGen}$ is efficient (in m), assuming OT protocols exist, an (unfair) secure computation of $\mathsf{SharesGen}$ exists for any efficiently computable m.
- Ignoring negligible terms (due to the imperfection of secure computation using OT), the output of each party (when seen on its own) is a set of uniform strings. In particular, it contains *no information* about the other party's shares, or about the values of c and i^*.

  By construction, a party outputs a uniform bit if the other party aborts before the end of the secure computation phase. Hence, it makes no sense for a party to abort during this phase.
- Given the above observation, it is instructive to pretend that at the first step of the protocol, the output of a random execution of $\mathsf{SharesGen}$ was given to the parties by an *honest dealer*.

Note that in each round of the above protocol, both honest parties send their messages without waiting for the other party's message. Hence, the above protocol is symmetric with respect to the parties' role. However, since we assume no simultaneous channel (which would have trivialized the whole question), the corrupted party can postpone sending its message until it gets the message of the honest party, and then decide whether to send its message for this round or abort.

Security of the protocol.

At least on the intuitive level, the security proof of the above protocol is rather simple. Since the protocol is symmetric, we assume for concreteness that P_0 is corrupted and tries to bias the expected output of P_0 away from $\frac{1}{2}$. The following random variables are defined with respect to a random execution of $(\mathsf{P}_0, \mathsf{P}_1)$: let V be the view of the corrupted P_0, right after sending the abort message, and let V^- be the view V without this abort message (V^- and V are set to the full view, if no abort occurred). Finally, for a view v, let $\mathsf{val}(v)$ be the expected outcome of the non-aborting parties, conditioned on v, and assuming the non-aborting parties act *honestly* in the rest of the execution. It is not hard to verify that the bias obtained by P_0 is *exactly* $\left|\mathsf{val}(V) - \mathsf{val}(V^-)\right|$.

It is also easy to see that by aborting in round (i, b), for some $i \in [m]$, party P_0 gains nothing (i.e., $\mathsf{val}(V) = \mathsf{val}(V^-)$), where the (i,j)'th round of the execution stands for the j'th step of the i'th loop in the execution. A slightly more complicated math yields that by aborting in round (i, a), party P_0 only gains $\Theta(\frac{1}{m})$ bias. It follows that the maximal bias obtained by a fail-stop strategy for P_0 is $\Theta(\frac{1}{m})$.

Fairness via defense.

Let us present a different view of the protocol of [33]. Consider a variant of this protocol without the d_i's. Namely, the parties reconstruct $c_1, \ldots, c_m$ one at a time, until they reach $c_i \neq \perp$. When an abort occurs, the remaining party outputs an unbiased coin if it has not yet reconstructed c, and outputs c otherwise. It is easy to see that an aborting attacker can bias the output of the other party in this degenerate variant by $\frac{1}{4}$; that is, it simply waits until it reconstructs c and then aborts for biasing the other party's output towards $1 - c$.

The role of "defense" values $(d_i^0, d_i^1), \ldots, (d_m^0, d_m^1)$ is to prevent such an attack; if a party aborts after reconstructing c, the other party is guaranteed to output c as well. The problem is, however, that the defense values themselves might cause a problem: a corrupted party might abort after reconstructing its defense value for the i'th round (and not only after reconstructing c_i). Indeed, by aborting in these rounds, a corrupted party does gain a bias, but only $\Theta(\frac{1}{m})$.

On extending Moran et al.'s protocol for the three-party case.

We next explain why the approach of Moran et al. does not seem to be useful for constructing fair, three-party coin-flipping protocols.

In a three-party fair coin-flipping protocol, one should deal with two *non-simultaneous* aborts: after one party aborts, the remaining two parties should interact in a two-party protocol to agree on their common coin. Since one of the remaining parties might be corrupted as well, this two-party protocol needs to be a fair coin-flipping protocol as well. Moreover, the expected outcome of the latter two-party protocol, whose shares are given *before* each round, should be equal (up to an additive difference of $\Theta(\frac{1}{m})$) to the value of the three-party protocol *after* this round — the expected outcome of the protocol given the reconstructed shares. Otherwise, an aborting party can significantly bias the output of the two other parties.

Consider the following natural extension of Moran et al.'s protocol to a three-party protocol. The value of $c_1, \ldots, c_m$ are as in the two-party protocol (now shared between the three parties). The defense values are not bits, but rather two vectors of shares for the two remaining parties (different shares for each possible pair), to enable them to interact in some fair two-party coin-flipping protocol if the third party aborts.

Assume that in the i'th round of the "outer" three-party protocol, the value of c_i is one (i.e., $c_i = c = 1$), and consider the two-party protocol executed by the remaining parties, if a party aborts in this round. The outcome of the remaining party in the case of a premature abort in this underlying two-party protocol should be also one. Otherwise, two corrupted parties can mount the following two-phase attack: first aborting in the outer three-party protocol after seeing $c_i = 1$, and then prematurely aborting in the inner two-party protocol, knowing that the other party will output something that is far from one. Now, assume that in the i'th round of the "outer" three-party protocol, the value of c_i is $\perp$ (i.e., $i < i^*$), and consider again the two-party protocol executed by the remaining parties if party aborts in this round. It is easy to see that expected outcome of this two-party protocol should be close to $\frac{1}{2}$ (i.e., unbiased). Thus, the defense values, to be constructed by each party during the execution of this two-party protocol, cannot all be of the same value.

These restrictions on the two-party protocol defense values ruin the security of the outer three-party protocol; in each round i, two corrupted (and thus colluding) parties can reconstruct the *whole* two-party execution that they should engage in if the other (in this case, the honest) party aborts in this round. By checking whether the defense values of this two-party execution are all ones (indicating that $c = 1$), all zeros (indicating that $c = 0$), or mixed (indicating that $c_i =\perp$), they get enough information for biasing the output of the protocol by a constant value.

What causes the above three-party protocol to fail is that its value in a given round might be changed by $\frac{1}{2}$ (say from $\frac{1}{2}$ to 1). As we argued above, the (long) defense values reconstructed *before* each round in the three-party protocol have to contain many (i.e., m) samples drawn according to the value of the protocol at the *end* of the round. It follows that two corrupted parties might extrapolate, at the *beginning* of such a round, the value of protocol when this round *ends*, thus rendering the protocol insecure.

1.3.2 Our Smooth Two-Party Protocol

Given the above understanding, our first step is to construct a two-party coin-flipping protocol, whose value only changes *slightly* (i.e., smoothly) between consecutive rounds. In the next section we use a *derandomized* version of such a smooth coin-flipping protocol as a building block for constructing an (almost) optimally fair three-party protocol.

Consider the $\Theta(\frac{1}{\sqrt{m}})$-bias coin-flipping protocol of Cleve [13]: in each of the m rounds, the parties reconstruct the value of a coin $c_i \in \{-1, 1\}$, and the final outcome is set to $\mathsf{sign}(\sum_{i\in[m]} c_i)$. Since the value of $\sum c_i$ is close to being uniform over $[-\sqrt{m}, \sqrt{m}]$, the value of the first coin c_1 changes the protocol's value by $\Theta(\frac{1}{\sqrt{m}})$. This sounds like a good start toward achieving a smooth coin-flipping protocol. The problem is, however, that with probability $\Theta(\frac{1}{\sqrt{m}})$, the sum of $c_1, \ldots, c_{m-1}$ is exactly zero. Hence, with this probability, the final coin changes the protocol's value by $\frac{1}{2}$.

We overcome the above problem by using a *weighted* majority protocol. In the first round the parties reconstruct m-coins (in a single shot), reconstruct $(m - 1)$ coins in the second round, and so on, until in the very last round only a single coin is reconstructed. Now the value of $\sum c_i$ (now each c_i is an integer) is close to being uniform over $[-m, m]$, and the last round determines the outcome only with probability $\Theta(\frac{1}{m})$ (versus $\Theta(\frac{1}{\sqrt{m}})$ in the unweighted version). Other rounds also enjoy a similar smoothness property. We emphasize that the resulting protocol *does not* guarantee small bias (but only a $\Theta(\frac{1}{\sqrt{m}})$-bias). Rather, we take advantage of its smoothness and use it as a building block of a fair two-party protocol (and then of a three-party protocol).

The protocol.

As in Moran et al. [33], the parties start by securely computing a share generating function, and then use its outputs to slowly reconstruct the output of the protocol.

Let $\mathcal{B}er(\delta)$ be the Bernoulli distribution over $\{0, 1\}$, taking the value one with probability δ and zero otherwise, let $\mathcal{C}_n$ be distribution induced by the sum of n uniform coins over $\{-1, 1\}$, let $\mathsf{B}_n(k) := \Pr_{x\leftarrow\mathcal{C}_n}[x \geq k]$, and finally, for $i \in [n]$ let $\mathsf{sum}_n(i) := \sum_{j=i}^{n}(n + 1 - j) = \frac{(n-i+1)(n-i+2)}{2}$.

We start by describing the share generating function and then use it to describe the protocol.

Algorithm 1.4 (function $\mathsf{TwoPartySharesGen}$).

```
1. For $z \in \{0,1\}$, sample $d_0^z \leftarrow \{0,1\}$.
2. For $i = 1$ to $m$,
    (a) Sample $c_i \leftarrow \mathcal{C}_{m+1-i}$.
    (b) For $z \in \{0,1\}$, sample $d_i^z \leftarrow \mathcal{B}er(\delta_i)$, for $\delta_i = \mathsf{B}_{\mathsf{sum}_m(i+1)}(-\sum_{j=1}^{i} c_i)$.[3]
3. Split each of the $3m$ values $d_1^0, d_1^1, \ldots, d_m^0, d_m^1, c_1, \ldots, c_m$ into two "shares", using a 2-out-of-2 secret sharing scheme, to create two sets of shares: $\mathbf{s}^{\#0}$ and $\mathbf{s}^{\#1}$.
4. Output $(d_0^0, \mathbf{s}^{\#0}), (d_0^1, \mathbf{s}^{\#1})$.
```

Protocol 1.5 ($\pi_2 = (\mathsf{P}_0^2, \mathsf{P}_1^2)$).

Initial step: The parties securely compute the function $\mathsf{TwoPartySharesGen}$. *Let* $(d_0^i, \mathbf{s}^{\#i})$ *be the local output of* P_i^2.

Main loop: For $i = 1$ *to* m, *do*

- *(a)* P_0^2 *sends to* P_1^2 *its share of* d_i^1, *and* P_1^2 *sends to* P_0^2 *its share of* d_i^0.
  - P_0^2 *reconstructs the value of* d_i^0, *and* P_1^2 *reconstructs the value of* d_i^1.
- *(b) Each party sends to the other party its share of* c_i.
  - *Both parties reconstruct the value of* c_i.

Output: Both parties output one if $\sum_{j=1}^{m} c_i \geq 0$, *and zero otherwise.*

Abort: If P_0^2 *aborts, party* P_1^2 *outputs the value of* d_i^1, *for the* maximal $i \in [m]$ *for which it has reconstructed this value (note that by construction such an* i *always exists).*

The case that P_1^2 *aborts is analogously defined.*

[3] δ_i is the probability that the protocol's output is one, given the value of the "coins" $c_1 \ldots, c_i$ (and assuming no abort).

Namely, the parties interact in a weighted majority protocol, where in the i'th round, they reconstruct, in an unfair manner, $(m+1-i)$ coins (i.e., c_i). If a party aborts, the remaining party outputs a defense value given to it by the honest dealer (implemented via the secure computation of $\mathsf{TwoPartySharesGen}$).

A few remarks are in place. First, we will only define the protocol for $m \equiv 1 \bmod 4$.[4] Hence, $\sum_{j=1}^{m} c_i \neq 0$, and the protocol's output is a uniform bit when played by the honest parties. Second, if P_0^2 aborts in the first round, the party P_1^2 could simply output a uniform bit. We make P_0^2 output d_0^1, since this be useful when the two-party protocol will be later used as part of the three-party protocol. Finally, one can define the above protocol without exposing the coins c_i's to the parties (in this case, the honest parties output (d_m^0, d_m^1) as the final outcome). We expose the coins to the parties to make the analysis of the protocol easier to follow.

Security of the protocol.

Note that the defense value given in round (i,a) (i.e., step a of the i'th loop) is distributed according to the expected outcome of the protocol, conditioned on the value of the coin to *be given* in round (i,b). These defense values make aborting in round (i,b), for any value of i, harmless. So it is left to argue that aborting in round (i,a), for any value of i, is not too harmful either. We show that, using the fact that the value reconstructed in round (i,a) gives only a very noisy signal about the value of c_i.

Since the protocol is symmetric, we assume for concreteness that the corrupted party is P_0^2. Similar to the analysis of Moran et al.'s protocol sketched above, it suffices to bound the value of $\left|\mathsf{val}(V) - \mathsf{val}(V^-)\right|$.

Assume that P_0^2 aborts in round (i,b). By construction, $\mathsf{val}(V^-) = \delta_i$. Since, the defense of P_1^2 in round (i,b) is sampled according to $\mathcal{B}er(\delta_i)$, it is also the case that $\mathsf{val}(V) = \delta_i$.

Assume now that P_0^2 aborts in round (i,a). By construction, $\mathsf{val}(V) = \delta_{i-1}$. Note that V^- does contains some information about δ_i, i.e., a sample from $\mathcal{B}er(\delta_i)$, and thus $\mathsf{val}(V^-)$ is typically different from $\mathsf{val}(V)$. Yet, since V^- contains only a sample from $\mathcal{B}er(\delta_i)$, and this is a very noisy signal for the actual value of δ_i, we manage to prove the following.

$$\left|\mathsf{val}(V) - \mathsf{val}(V^-)\right| = \frac{(\delta_i - \delta_{i-1})^2}{\delta_{i-1}}. \qquad (1)$$

Equation (1) yields that $\left|\mathsf{val}(V) - \mathsf{val}(V^-)\right| = O(\frac{1}{m})$, since by the "smoothness" of the protocol (i.e., the value of the game does not change drastically between consecutive rounds) it follows that $\left|\frac{\delta_i - \delta_{i-1}}{\delta_{i-1}}\right| \in O(\frac{1}{\sqrt{m}})$ with high probability.[5]

1.3.3 Our Three-Party Protocol

We start by applying a generic approach, introduced by Beimel et al. [6], to try and extend our fair two-party protocol into a three-party one. We then explain why this generic approach does not work in our case, and show how to modify it to get a fair three-party protocol.

In the first attempted three-party protocol, the parties interact in the following variant of the two-party protocol π_2 described in Protocol 1.5. As the first step, the three parties (securely) compute the function $\mathsf{ThreePartySharesGen}$, defined next, rather than $\mathsf{TwoPartySharesGen}$ as in protocol π_2. Let $\mathcal{C}_\varepsilon$ be a ε-biased coin over $\{-1,1\}$. Namely, $\mathcal{C}_\varepsilon$ is the Bernoulli probability distribution over $\{-1,1\}$, taking the value 1 with probability $\frac{1}{2}+\varepsilon$ and -1 otherwise. Let $\mathcal{C}_{n,\varepsilon}$ be the sum of n independent ε-biased coins over $\{-1,1\}$ and let $\mathsf{B}_{n,\varepsilon}(k) := \Pr_{x \leftarrow \mathcal{C}_{n,\varepsilon}}[x \geq k]$. For $\varepsilon \in [-\frac{1}{2}, \frac{1}{2}]$, we define the function $\mathsf{TwoPartySharesGen}^\varepsilon$ as the following variant of the function $\mathsf{TwoPartySharesGen}$ defined above: (1) the "coin" c_i is sampled according to $\mathcal{C}_{m+1-i,\varepsilon}$ (and not according $\mathcal{C}_{m+1-i}$ as in $\mathsf{TwoPartySharesGen}$); (2) the initial defense values d_0^0 and d_0^1 are sampled according to $\mathcal{B}er(\mathsf{B}_{\mathsf{sum}_m(1),\varepsilon}(0))$ (and not $\mathcal{B}er(\frac{1}{2}) = \mathcal{B}er(\mathsf{B}_{\mathsf{sum}_m(1)}(0))$ as in $\mathsf{TwoPartySharesGen}$).

Algorithm 1.6 (function $\mathsf{ThreePartySharesGen}$).

```
1. For $i = 1$ to $m$,
    (a) Sample $c_i \leftarrow \mathcal{C}_{m+1-i}$.
    (b) Let $\varepsilon_i \in [-\frac{1}{2}, \frac{1}{2}]$ be the value such that $\mathsf{B}_{\mathsf{sum}_m(1),\varepsilon_i}(0) = \delta_i = \mathsf{B}_{\mathsf{sum}_m(i+1),0}(-\sum_{j=1}^{i} c_j)$.[6]
    (c) For each pair of the three parties, generate shares for an execution of $\pi_2$, by calling $\mathsf{TwoPartySharesGen}^{\varepsilon_i}$.
2. Split the values of $c_1,\ldots,c_m$ and the defense values into three set of shares using a 3-out-of-3 secret sharing scheme, and output the three sets.
```

Protocol 1.7 ($\pi_3 = (\mathsf{P}_0^3, \mathsf{P}_1^3, \mathsf{P}_2^3)$).

Initial step: The parties securely compute the function $\mathsf{ThreePartySharesGen}$, *where each party gets one set of shares.*

Main loop: For $i = 1$ to m, do

(a) Each party sends to the other parties its share of their defense values.

- *Each pair $(\mathsf{P}_z^3, \mathsf{P}_{z'}^3)$ of the parties reconstructs a pair of two sets of shares $d_i^{z,z'} = ((d_i^{z,z'})_z, (d_i^{z,z'})_{z'})$, to serve as input for an execution of the two-party protocol if the third party aborts (i.e., P_z reconstructs $(d_i^{z,z'})_z$, and $\mathsf{P}_{z'}$ reconstructs $(d_i^{z,z'})_{z'}$).*

(b) Each party sends the other parties its share of c_i.

- *All parties reconstruct the value of c_i.*

Output: The parties output one if $\sum_{j=1}^{m} c_i \geq 0$, and zero otherwise.

Abort:
- *If P_0^3 aborts, the parties P_1^3 and P_2^3 use the shares of $d_i^{1,2}$, for the* maximal *$i \in [m]$ that has been reconstructed, to interact in π^2 (starting right after the share reconstruction phase). If no such i exists, the parties interact in the (full, unbiased) two-party protocol π^2.*

  The case that P_1^3 or P_2^3 aborts is analogously defined.

[4]Defining the protocol for $m \equiv 2 \bmod 4$ works as well.

[5]The $O(\log^2 m)$ factor in our actual result does not appear in the above informal argument.

[6]Namely, ε_i is set to the number such that a fresh new game with ε_i-biased coins, has the same value (i.e., expected outcome) as that of the current game at this point (i.e., conditioning on $c_1,\ldots,c_i$).

- *If two parties abort in the same round, the remaining party acts as if one party has only aborted in the very beginning of the two-party protocol.*

Similar to the analysis for the two-party protocol sketched above, it suffices to show that the defense values reconstructed by a pair of corrupted parties in round (i, a) — the inputs for the two-party protocols — do not give too much information about the value of δ_i — the expected outcome of the three-party protocol conditioned on the coins reconstructed at round (i, b). Note that once two corrupted parties are given these defense values, which happens in round (i, a), they can *immediately* reconstruct the whole two-party execution induced by them. This two-party execution effectively contains $\Theta(m)$ *independent* samples from $\mathcal{B}er(\delta_i)$: one sample is given explicitly as the final output of the execution, and the value of $2m$ additional samples can be extrapolated from the $2m$ defense values given to the two parties. Many such independent samples can be used to approximate the value of δ_i with accuracy $\Theta(\frac{1}{m})$. It follows that in round (i, a), two corrupted parties can rush and estimate the value of δ_i with high accuracy, and then use it to bias the outcome of the three-party protocol by $\left|\delta_i - \delta_{i-1} - \Theta(\frac{1}{m})\right| \in \Omega(\frac{1}{\sqrt{m}})$.

Derandomizing the above protocol.

We handle the above problem by modifying the way the defense values given to the parties in the three-party protocol are sampled. We modify the share generating function $\mathsf{TwoPartySharesGen}^{\varepsilon}$ to first draw $\Theta(m^2)$ independent samples from $\mathcal{C}_{\varepsilon}$, and then use these samples via a simple derandomization technique for drawing the $\Theta(m)$ defense values given in the three-party protocol. This constitute a saving in comparing to the original $\mathsf{TwoPartySharesGen}^{\varepsilon}$ which, at least for its straightforward implementation, requires $\Theta(m^3)$ (independent) samples from $\mathcal{C}_{\varepsilon}$. When called with $\varepsilon = \varepsilon_i$, the definition of ε_i yields that the information such $\Theta(m^2)$ samples contain about the value of δ_i is roughly the same as in a *constant* number of independent samples from $\mathcal{B}er(\delta_i)$. It follows that the defense values given to the parties in round (i, a) of the resulting three-party protocol, leak only a little information about the value of δ_i. This yields that a pair of colluding parties cannot bias the output of the three-party protocol by more than $\Theta(\frac{1}{m})$.

1.4 Open Problems

The existence of an optimally fair three-party coin-flipping protocol (without the $O(\log^2 m)$ factor) is still an interesting open question. A more fundamental question is whether there exists a fair coin-flipping protocol for any number of parties (against any number of corrupted parties). While constructing (at least, almost) optimally fair, m-round coin-flipping protocols for a constant (or even $\log(m)$) number of parties seems within the reach of our current technique, handling a super-logarithmic number of parties, not to mention $\Omega(m)$, seems to require a completely new approach, an may not be possible at all.

Paper Organization

Formal definition of the security model is given in Section 2. We also state there (Section 2.3.1) a new game-based definition of fair coin-flipping protocols, which is equivalent to the standard real/ideal definition. Our results are formally stated in Section 3. Missing proofs can be found in the full version of the paper [26].

2. PRELIMINARIES

2.1 Multi-Party Protocols

The following discussion is restricted to no private input protocols (such restricted protocols suffice for our needs).

A t-party protocol is defined using t Touring Machines (TMs) $\mathsf{P}_1, \ldots, \mathsf{P}_t$, having the security parameter 1^{κ} as their common input. In each round, the parties broadcast and receive messages on a broadcast channel. At the end of protocol, each party outputs some binary string.

The parties communicate in a synchronous network, using only a broadcast channel: when a party broadcasts a message, all other parties see *the same* message. This ensures some consistency between the information the parties have. There are no private channels and all the parties see all the messages, and can identify their sender. We do not assume simultaneous broadcast. It follows that in each round, some parties might hear the messages sent by the other parties before broadcasting their messages. We assume that if a party aborts, it first broadcasts the message Abort to the other parties, and without loss of generality only does so at the end of a round in which it is supposed to send a message. A protocol is *efficient*, if its parties are PPTM, and the protocol's number of rounds is a computable function of the security parameter.

This work focuses on efficient protocols, and on malicious, static PPT adversaries for such protocols. An adversary is allowed to corrupt some subset of the parties; before the beginning of the protocol, the adversary corrupts a subset of the parties that from now on may arbitrarily deviate from the protocol. Thereafter, the adversary sees the messages sent to the corrupted parties and controls their messages. We also consider the so called *fail-stop* adversaries. Such adversaries follow the prescribed protocol, but might abort prematurely. Finally, the honest parties follow the instructions of the protocol to its completion.

2.2 The Real vs. Ideal Paradigm

The security of multiparty computation protocols is defined using the *real* vs. *ideal* paradigm [12, 19]. In this paradigm, the *real-world model*, in which protocols is executed is compared to an *ideal model* for executing the task at hand. The latter model involves a trusted party whose functionality captures the security requirements of the task. The security of the real-world protocol is argued by showing that it "emulates" the ideal-world protocol, in the following sense: for any real-life adversary A, there exists an ideal-model adversary (also known as simulator) $\mathbb{A}$ such that the global output of an execution of the protocol with A in the real-world model is distributed similarly to the global output of running $\mathbb{A}$ in the ideal model. The following discussion is restricted to random, no-input functionalities. In addition, to keep the presentation simple, we limit our attention to uniform adversaries.[7]

The Real Model.

[7]All results stated in this paper, straightforwardly extend to the non-uniform settings.

Let π be an t-party protocol and let A be an adversary controlling a subset $\mathcal{C} \subseteq [t]$ of the parties. Let $\mathrm{REAL}_{\pi,\mathsf{A},\mathcal{C}}(\kappa)$ denote the output of A (i.e., without loss of generality its view: its random input and the messages it received) and the outputs of the honest parties, in a random execution of π on common input 1^κ.

Recall that an adversary is *fail stop*, if until they abort, the parties in its control follow the prescribed protocol (in particular, they property toss their private random coins). We call an execution of π with such a fail-stop adversary, a fail-stop execution.

The Ideal Model.

Let f be a t-output functionality. If f gets a security parameter (given in unary), as its first input, let $f_\kappa(\cdot) = f(1^\kappa, \cdot)$. Otherwise, let $f_\kappa = f$.

An ideal execution of f with respect to an adversary $\mathbb{A}$ controlling a subset $\mathcal{C} \subseteq [t]$ of the "parties" and a security parameter 1^κ, denoted $\mathrm{IDEAL}_{f,\mathbb{A},\mathcal{C}}(\kappa)$, is the output of the adversary $\mathbb{A}$ and that of the trusted party, in the following experiment.

Experiment 2.1.

1. *The trusted party sets $(y_1, \ldots, y_t) = f_\kappa(X)$, where X is a uniform element in the domain of f_κ, and sends $\{y_i\}_{i \in \mathcal{C}}$ to $\mathbb{A}(1^\kappa)$.*
2. *$\mathbb{A}(1^\kappa)$ sends the description of a subset $\mathcal{J} \subseteq \mathcal{C}$ to the trusted party, and locally outputs some value.*
3. *The trusted party outputs $\{o_i\}_{i \in [t] \setminus \mathcal{C}}$, where o_i is equal to y_i in case $\mathcal{J} = \emptyset$, and the description of $\mathcal{J}$ otherwise.*

An adversary $\mathbb{A}$ is non-aborting, if it always sets $\mathcal{J} = \emptyset$.

2.2.1 δ-Secure Computation

The following definitions adopts the notion of δ-secure computation [7, 21, 30] for our restricted settings.

Definition 2.2 (δ-secure computation). *An efficient t-party protocol π computes a t-output functionality f in a* δ-secure *manner [resp., against fail-stop adversaries], if for every $\mathcal{C} \subsetneq [t]$ and every [resp., fail-stop]* PPT *adversary A controlling the parties indexed by $\mathcal{C}$,*[8] *there exists a* PPT *$\mathbb{A}$ controlling the same parties, such that*

$$\mathrm{SD}\left(\mathrm{REAL}_{\pi,\mathsf{A},\mathcal{C}}(\kappa), \mathrm{IDEAL}_{f,\mathbb{A},\mathcal{C}}(\kappa)\right) \leq \delta(\kappa),$$

for large enough κ.

A protocol securely compute a functionality f, if it computes f in a $\mathrm{neg}(\kappa)$-secure manner.

The protocol π computes f in a simultaneous δ-secure *manner, if the above is achieved by a* non-aborting *$\mathbb{A}$.*

Note that being simultaneous δ-secure is a very strong requirement, as it dictates that the cheating real adversary has no way to prevent the honest parties from getting their part of the output, and this should be achieved with no simultaneous broadcast mechanism.

[8]The requirement that $\mathcal{C}$ is a *strict* subset of $[t]$, is merely for notational convinced.

2.3 Fair Coin-Flipping Protocols

Definition 2.3 (δ-fair coin-flipping). *For $t \in \mathbb{N}$ let* $\mathsf{CoinFlip}_t$ *be the t-output functionality from $\{0,1\}$ to $\{0,1\}^t$, defined by $\mathsf{CoinFlip}(b) = b^t$. A t-party protocol π is* δ-fair coin-flipping protocol, *if it computes* $\mathsf{CoinFlip}_t$ *in a* simultaneous δ-secure *manner.*

2.3.1 Proving Fairness

The following lemma reduces the task of proving fairness of a coin-flipping protocol, against fail-stop adversaries, to proving the protocol is correct: the honest parties always output the same bit, and this bit is uniform in an all honest execution, and to proving the protocol is unbiased: a fail-stop adversary cannot bias the output of the honest parties by too much.

Definition 2.4 (correct coin-flipping protocols). *A protocol is a* correct coin flipping, *if*

- *When interacting with an fails-stop adversary controlling a subset of the parties, the honest parties* always *output the same bit, and*
- *The common output in a random* honest *execution of π, is uniform over $\{0,1\}$.*

Given a partial view of a fail-stop adversary, we are interesting in the expected outcome of the parties, conditioned on this and the adversary making no further aborts.

Definition 2.5 (view value). *Let π be a protocol in which the honest parties always output the same bit value. For a partial view v of the parties in a fail-stop execution of π, let $\mathsf{C}_\pi(v)$ denote the parties' full view in an* honest *execution of π conditioned on v (i.e., all parties that do not abort in v act honestly in $\mathsf{C}_\pi(v)$). Let $\mathsf{val}_\pi(v) = \mathrm{E}_{v' \leftarrow \mathsf{C}_\pi(v)}[\mathrm{out}(v')]$, where $\mathrm{out}(v')$ is the common output of the non-aborting parties in v'.*

Finally, a protocol is unbiased, if no fail-stop adversary can bias the common output of the honest parties by too much.

Definition 2.6 (unbiased coin-flipping protocols). *A t-party, m-round protocol π is α-*unbiased, *if the following holds for every fail-stop adversary A controlling the parties indexed by a subset $\mathcal{C} \subset [t]$. Let V be A's view in a random execution of π in which A controls the parties indexed by $\mathcal{C}$, and let I_j be the index of the j'th round in which A sent an abort message (set to m, if no such round). Let V_i be the prefix of V at the end of the i'th round, letting V_0 being the empty view, and let V_i^- be the prefix of V_i with the i'th round abort messages (if any) removed. Then*

$$\mathop{\mathrm{E}}_{V}\left[\left|\sum_{j \in |\mathcal{C}|} \mathsf{val}(V_{I_j}) - \mathsf{val}(V_{I_j}^-)\right|\right] \leq \alpha,$$

where $\mathsf{val} = \mathsf{val}_\pi$ is according to Definition 2.5.

The following lemma is an alternative characterization of fair coin-flipping protocols (against fail-stop adversaries).

Lemma 2.7. *Let π be a correct, α-unbiased coin-flipping protocol with $\alpha(\kappa) \leq \frac{1}{2} - \frac{1}{p(\kappa)}$, for some $p \in \mathrm{poly}$. Then π is a $(\alpha(\kappa) + \mathrm{neg}(\kappa))$-secure coin-flipping protocol against fail-stop adversaries.*

Proof. Omitted. $\square$

2.4 Oblivious Transfer

DEFINITION 2.8. *The* $\binom{1}{2}$ *oblivious transfer* (OT *for short) functionality, is the two-output functionality* f *over* $\{0,1\}^3$*, defined by* $f(\sigma_0, \sigma_1, i) = ((\sigma_0, \sigma_1), (\sigma_i, i))$.

Protocols the securely compute OT, are known under several hardness assumptions (cf., [2, 16, 18, 24, 29, 35]).

2.5 f-Hybrid Model

Let f be a t-output functionality. The f-hybrid model is identical to the real model of computation discussed above, but in addition, each t-size subset of the parties involved, has access to a trusted party realizing f. It is important to emphasize that the trusted party realizes f in a *non-simultaneous* manner: it sends a random output of f to the parties in an arbitrary order. When a party gets its part of the output, it instructs the trusted party to either continue sending the output to the other parties, or to send them the abort symbol (i.e., the trusted party "implements" f in a perfect non-simultaneous manner).

All notions given in Sections 2.2 and 2.3 naturally extend to the f-hybrid model, for any functionality f. In addition, the proof of Lemma 2.7 straightforwardly extends to this model.

3. OUR RESULTS

In this section we formally state our main results. The proofs that follow the intuition given in Section 1.3, can be found in the full version of this paper [26].

3.1 Two-Party Protocols

THEOREM 3.1. *There exist a polynomial-time oracle-aided protocol* Π^2 *and a polynomial-time computable function* TwoPartySharesGen*, such that* $\Pi^2(1^m)$ *is an* m*-round, two-party,* $O(\frac{\log^2 m}{m})$*-fair coin-flipping protocol against unbounded fail-stop adversaries in the* TwoPartySharesGen*-hybrid model, for any* $m \equiv 1 \bmod 4$.

PROOF. Omitted. □

Using standard means, the above theorem yields the existence of an almost optimal two-party fair coin-flipping protocol, in the real (non-hybrid) model.

THEOREM 3.2. *Assuming protocols for securely computing* OT *exist, then for any polynomially bounded, polynomial-time computable, integer function* m*, there exists an* m*-round,* $\frac{O(\log^2 m)}{m}$*-fair, two-party coin-flipping protocol.*

PROOF. Omitted. □

3.2 Three-Party Protocols

THEOREM 3.3. *There exist a polynomial-time oracle-aided protocol* Π^3 *and a polynomial-time computable function* ThreePartySharesGen*, such that* $\Pi^3(1^m)$ *is an* m*-round, three-party,* $O(\frac{\log^2 m}{m})$*-fair coin-flipping protocol against unbounded fail-stop adversaries in the* ThreePartySharesGen*-hybrid model, for any* $m \equiv 1 \bmod 4$.

PROOF. Omitted. □

As in the two-party case, the above theorem yields the existence of an almost optimal three-party fair coin-flipping protocol, in the real (non-hybrid) model.

THEOREM 3.4. *Assuming protocols for securely computing* OT *exist, then for any polynomially bounded, polynomial-time computable, integer function* m*, there exists an* m*-round,* $\frac{O(\log^2 m)}{m}$*-fair, three-party coin-flipping protocol.*

PROOF. Omitted. □

References

[1] D. Aharonov, A. Ta-Shma, U. Vazirani, and A. C. Yao. Quantum bit escrow. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2000.

[2] W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology – EUROCRYPT 2001*, 2001.

[3] N. Alon and M. Naor. Coin-flipping games immune against linear-sized coalitions. *SIAM Journal on Computing*, pages 46–54, 1993.

[4] A. Ambainis. A new protocol and lower bounds for quantum coin flipping. *J. Comput. Syst. Sci.*, 68(2): 398–416, 2004.

[5] A. Ambainis, H. Buhrman, Y. Dodis, and H. Röhrig. Multiparty quantum coin flipping. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity*, pages 250–259, 2004.

[6] A. Beimel, E. Omri, and I. Orlov. Protocols for multiparty coin toss with dishonest majority. In *Advances in Cryptology – CRYPTO 2010*, pages 538–557, 2010.

[7] A. Beimel, Y. Lindell, E. Omri, and I. Orlov. $1/p$-secure multiparty computation without honest majority and the best of both worlds. pages 277–296, 2011.

[8] M. Ben-Or and N. Linial. Collective coin flipping. *ADVCR: Advances in Computing Research*, 5, 1989.

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

[10] I. Berman, I. Haitner, and A. Tentes. Coin flipping of any constant bias implies one-way functions. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*.

[11] M. Blum. How to exchange (secret) keys. *ACM Transactions on Computer Systems*, 1983.

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

[13] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–369, 1986.

[14] R. Cleve and R. Impagliazzo. Martingales, collective coin flipping and discrete control processes. Manuscript, 1993.

[15] D. Dachman-Soled, Y. Lindell, M. Mahmoody, and T. Malkin. On the black-box complexity of optimally-fair coin tossing. In *tcc11*, pages 450–467, 2011.

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

[17] U. Feige. Noncryptographic selection protocols. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, 1999.

[18] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 197–206, 2008.

[19] O. Goldreich. *Foundations of Cryptography – VOLUME 2: Basic Applications.* Cambridge University Press, 2004.

[20] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, pages 691–729, 1991. Preliminary version in *FOCS'86*.

[21] S. D. Gordon and J. Katz. Partial fairness in secure two-party computation. pages 157–176, 2010.

[22] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. pages 413–422, 2008.

[23] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. *Journal of the ACM*, 58(6):24, 2011.

[24] I. Haitner. Implementing oblivious transfer using collection of dense trapdoor permutations. In *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004*, pages 394–409, 2004.

[25] I. Haitner and E. Omri. Coin Flipping with Constant Bias Implies One-Way Functions. pages 110–119, 2011.

[26] I. Haitner and E. Tsfadia. An almost-optimally fair three-party coin-flipping protocol. www.cs.tau.ac.il/~iftachh/papers/3PartyCF/QuasiOptimalCF_Full.pdf, 2014. Manuscript.

[27] I. Haitner, M. Nguyen, S. J. Ong, O. Reingold, and S. Vadhan. Statistically hiding commitments and statistical zero-knowledge arguments from any one-way function. *SIAM Journal on Computing*, pages 1153–1218, 2009.

[28] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 230–235, 1989.

[29] Y. Kalai. Smooth projective hashing and two-message oblivious transfer. In *Advances in Cryptology – EUROCRYPT 2005*, 2005.

[30] J. Katz. On achieving the "best of both worlds" in secure multiparty computation. pages 11–20, 2007.

[31] H. K. Maji, M. Prabhakaran, and A. Sahai. On the Computational Complexity of Coin Flipping. In *Proceedings of the 51th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 613–622, 2010.

[32] T. Moran and M. Naor. Basing cryptographic protocols on tamper-evident seals. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2005.

[33] T. Moran, M. Naor, and G. Segev. An optimally fair coin toss. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2009*, pages 1–18, 2009.

[34] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, pages 151–158, 1991.

[35] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.

[36] A. Russell and D. Zuckerman. Perfect information leader election in log* n + 0 (1) rounds. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, 1999.

[37] M. Saks. A robust noncryptographic protocol for collective coin flipping. *SIJDM: SIAM Journal on Discrete Mathematics*, 2, 1989.