An Optimally Fair Coin Toss

Tal Moran* Moni Naor*† Gil Segev*

Abstract

We address one of the foundational problems in cryptography: the bias of coin-flipping protocols. Coin-flipping protocols allow mutually distrustful parties to generate a common unbiased random bit, guaranteeing that even if one of the parties is malicious, it cannot significantly bias the output of the honest party. A classical result by Cleve [STOC '86] showed that for any two-party r-round coin-flipping protocol there exists an efficient adversary that can bias the output of the honest party by $\Omega(1/r)$. However, the best previously known protocol only guarantees $O(1/\sqrt{r})$ bias, and the question of whether Cleve's bound is tight has remained open for more than twenty years.

In this paper we establish the optimal trade-off between the round complexity and the bias of two-party coin-flipping protocols. Under standard assumptions, we show that Cleve's lower bound is tight: we construct an r-round protocol with bias $O(1/r)$.

*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. Email: {tal.moran,moni.naor,gil.segev}@weizmann.ac.il. Research supported in part by a grant from the Israel Science Foundation.

†Incumbent of the Judith Kleeman Professorial Chair.

1 Introduction

A coin-flipping protocol allows mutually distrustful parties to generate a common unbiased random bit. Such a protocol should satisfy two properties. First, when all parties are honest and follow the instructions of the protocol, their common output is a uniformly distributed bit. Second, even if some of the parties collude and deviate from the protocol's instructions, they should not be able to significantly bias the common output of the honest parties.

When a majority of the parties are honest, efficient and completely fair coin-flipping protocols are known as a special case of general multiparty computation with an honest majority [6] (assuming a broadcast channel). When an honest majority is not available, and in particular when there are only two parties, the situation is more complex. Blum's two-party coin-flipping protocol [8] guarantees that the output of the honest party is unbiased only if the malicious party does not abort prematurely. This satisfies a rather weak notion of fairness in which once the malicious party is labeled as a "cheater" the honest party is allowed halt without outputting any value. Blum's protocol can rely on the existence of any one-way function [18, 24], and Impagliazzo and Luby [19] showed that one-way functions are in fact essential even for such a seemingly weak notion. While this notion suffices for some applications, in many cases fairness is required to hold even if one of the parties aborts prematurely (consider, for example, an adversary that controls the communication channel and can prevent communication between the parties). In this paper we consider a stronger notion: even when the malicious party is labeled as a cheater, we require that the honest party outputs a bit.

Cleve's impossibility result. The latter notion of fairness turns out impossible to achieve in general. Specifically, Cleve [10] showed that for any two-party r-round coin-flipping protocol there exists an efficient adversary that can bias the output of the honest party by $\Omega(1/r)$. Cleve's lower bound holds even under arbitrary computational assumptions: the adversary only needs to simulate an honest party, and decide whether or not to abort early depending on the output of the simulation. However, the best previously known protocol (with respect to bias) only guaranteed $O(1/\sqrt{r})$ bias [5, 10], and the question of whether Cleve's bound was tight has remained open for over twenty years.

Fairness in secure computation. The bias of coin-flipping protocols can be viewed as a particular case of the more general framework of fairness in secure computation. Typically, the security of protocols is formalized by comparing their execution in the *real model* to an execution in an *ideal model* where a trusted party receives the inputs of the parties, performs the computation of their behalf, and then sends *all* parties their respective outputs. Executions in the ideal model guarantee *complete fairness*: either all parties learn the output, or neither party does. Cleve's result, however, shows that without an honest majority complete fairness is generally impossible to achieve, and therefore the formulation of secure computation (see [13]) weakens the ideal model to one in which fairness is not guaranteed. Informally, a protocol is "secure-with-abort" if its execution in the real model is indistinguishable from an execution in the ideal model allowing the ideal-model adversary to chose whether the honest parties receive their outputs.

Recently, Katz [20] suggested an alternate relaxation: keep the ideal model unchanged (i.e., all parties always receive their outputs), but relax the notion of indistinguishability by asking that the real model and ideal model are distinguishable with probability at most $1/p(n) + \nu(n)$, for a polynomial $p(n)$ and a negligible function $\nu(n)$ (we refer the reader to Section 2 for a formal definition). Protocols satisfying this requirement are said to be $1/p$-secure, and intuitively, such protocols guarantee complete fairness in the real model except with probability $1/p$. In the context of coin-flipping protocols, any $1/p$-secure protocol has bias at most $1/p$. However, the definition of $1/p$-security is more general and applies to a larger class of functionalities.

1.1 Our Contributions

In this paper we establish the optimal trade-off between the round complexity and the bias of two-party coin-flipping protocols. We prove the following theorem:

Theorem 1.1. *Assuming the existence of oblivious transfer, for any polynomial $r = r(n)$ there exists an r-round two-party coin-flipping protocol that is $1/(4r - c)$-secure, for some constant $c > 0$.*

We prove the security of our protocol under the simulation-based definition of $1/p$-security[1], which for coin-flipping protocols implies, in particular, that the bias is at most $1/p$. We note that our result not only identifies the optimal trade-off asymptotically, but almost pins down the exact leading constant: Cleve showed that any r-round two-party coin-flipping protocol has bias at least $1/(8r + 2)$, and we manage to achieve bias of at most $1/(4r - c)$ for some constant $c > 0$.

Our approach holds in fact for a larger class of functionalities. We consider the more general task of sampling from a distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$: party P_1 receives a sample from $\mathcal{D}_1$ and party P_2 receives a correlated sample from $\mathcal{D}_2$ (in coin-flipping, for example, the joint distribution $\mathcal{D}$ produces the values $(0, 0)$ and $(1, 1)$ each with probability $1/2$). Before stating our result in this setting we introduce a standard notation: we denote by $\mathrm{SD}(\mathcal{D}, \mathcal{D}_1 \otimes \mathcal{D}_2)$ the statistical distance between the joint distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ and the direct-product of the two marginal distributions $\mathcal{D}_1$ and $\mathcal{D}_2$. We prove the following theorem which generalizes Theorem 1.1:

Theorem 1.2. *Assuming the existence of oblivious transfer, for any polynomially-sampleable distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ and polynomial $r = r(n)$ there exists an r-round two-party protocol for sampling from $\mathcal{D}$ that is $\frac{\mathrm{SD}(\mathcal{D}, \mathcal{D}_1 \otimes \mathcal{D}_2)}{2r-c}$-secure, for some constant $c > 0$.*

1.2 Related Work

Coin-flipping protocols. Simple variations of Blum's protocol are the most commonly used coin-flipping protocols whenever security with abort is sufficient. Specifically, an r-round protocol with bias $O(1/\sqrt{r})$ can be constructed by sequentially executing Blum's protocol $O(r)$ times, and outputting the majority of the intermediate output values [5, 10]. We note that in this protocol an adversary can indeed bias the output by $\Omega(1/\sqrt{r})$ by simply aborting prematurely. One of the most significant results on the bias of coin-flipping protocols gave reason to believe that the optimal trade-off between the round complexity and the bias is in fact $\Theta(1/\sqrt{r})$ (as provided by the latter variant of Blum's protocol): Cleve and Impagliazzo [11] showed that in the *fail-stop model*, any two-party r-round coin-flipping protocol has bias $\Omega(1/\sqrt{r})$. In the fail-stop model adversaries are computationally unbounded, but they must follow the instructions of the protocol except for being allowed to abort prematurely. In this model commitment schemes exist in a trivial fashion[2], and therefore the Cleve–Impagliazzo bound also applies to any protocol whose security relies commitment schemes in a black-box manner, such as Blum's protocol and its variants.

Coin-flipping protocols were also studied in a variety of other models. Among those are collective coin-flipping in the "perfect information model" in which parties are computationally unbounded and all communication is public [2, 7, 12, 25, 26], and protocols based on physical assumptions, such as quantum computation [1, 3, 4] and tamper-evident seals [23].

[1]In a very preliminary version of this work we proved our results with respect to the definition of bias (see Section 2), and motivated by [14, 20] we switch to the more general framework of $1/p$-secure computation.

[2]The protocol for commitment in the fail-stop model is simply to privately decide on the committed value and send the message "I am committed" to the other party.

Fair computation. Some of the techniques underlying our protocols found their origins in a recent line of research devoted for achieving various forms of fairness in secure computation. The technique of choosing a secret "threshold round", before which no information is learned, and after which aborting the protocol is essentially useless was suggested by Moran and Naor [23] as part of a coin-flipping protocol based on tamper-evident seals. It was later also used by Katz [20] for partially-fair protocols using a simultaneous broadcast channel, and by Gordon et al. [15] for completely-fair protocols for a restricted (but yet rather surprising) class of functionalities. Various techniques for hiding a meaningful round in game-theoretic settings were suggested by Halpern and Teague [17], Gordon and Katz [16], and Kol and Naor [21]. Katz [20] also introduced the technique of distributing shares to the parties in an initial setup phase (which is only secure-with-abort), and these shares are then exchanged by the parties in each round of the protocol.

Subsequent work. Our results in the two-party setting were very recently generalized by Gordon and Katz [14] to deal with the more general case of randomized functions, and not only distributions. Gordon and Katz showed that any efficiently-computable randomized function $f : X \times Y \to Z$ where at least one of X and Y is of polynomial size has an r-round protocol that is $O\left(\frac{\min\{|X|,|Y|\}}{r}\right)$-secure. In addition, they showed that even if both domains are of super-polynomial size but the range Z is of polynomial size, the f has an r-round protocol that is $O\left(\frac{|Z|}{\sqrt{r}}\right)$-secure. Gordon and Katz also showed a specific function $f : X \times Y \to Z$ where X, Y, and Z are of size super-polynomial which cannot be $1/p$-securely computed for any $p > 2$ assuming the existence of exponentially-hard one-way functions.

1.3 Paper Organization

The remainder of this paper is organized as follows. In Section 2 we review several notions and definitions that are used in the paper (most notably, the definition of $1/p$-secure computation). In Section 3 we describe a simplified variant of our protocol and prove its security. In Section 4 we describe a more refined and general variant of our protocol which settles Theorems 1.1 and 1.2. Finally, in Section 5 we pose several open problems.

2 Preliminaries

In this section we review the definitions of coin-flipping protocols, $1/p$-indistinguishability and $1/p$-secure computation (taken almost verbatim from [14, 20]), and one-time message authentication.

2.1 Coin-Flipping Protocols

A two-party coin-flipping protocol is defined via two probabilistic polynomial-time Turing machines (P_1, P_2), referred to as parties, that receive as input a security parameter 1^n. The parties exchange messages in a sequence of rounds, where in every round each party both sends and receives a message (i.e., a round consists of two moves). At the end of the protocol, P_1 and P_2 produce outputs bits c_1 and c_2, respectively. We denote by $(c_1|c_2) \leftarrow \langle P_1(1^n), P_2(1^n)\rangle$ the experiment in which P_1 and P_2 interact (using uniformly chosen random coins), and then P_1 outputs c_1 and P_2 outputs c_2. It is required that for all sufficiently large n, and every possible pair (c_1, c_2) that may be output by $\langle P_1(1^n), P_2(1^n)\rangle$, it holds that $c_1 = c_2$ (i.e., P_1 and P_2 agree on a common value). This requirement can be relaxed by asking that the parties agree on a common value with sufficiently high probability[3].

[3]Cleve's lower bound [10] holds under this relaxation as well. Specifically, if the parties agree on a common value with probability $1/2 + \epsilon$, then Cleve's proof shows that the protocol has bias at least $\epsilon/(4r + 1)$.

The security requirement of a coin-flipping protocol is that even if one of P_1 and P_2 is corrupted and arbitrarily deviates from the protocol's instructions, the bias of the honest party's output remains bounded. Specifically, we emphasize that a malicious party is allowed to abort prematurely, and in this case it is assumed that the honest party is notified on the early termination of the protocol. In addition, we emphasize that even when the malicious party is labeled as a cheater, the honest party must outputs a bit. For simplicity, the following definition considers only the case in which P_1 is corrupted, and an analogous definition holds for the case that P_2 is corrupted:

Definition 2.1. *A coin-flipping protocol* (P_1, P_2) *has bias at most* $\epsilon(n)$ *if for every probabilistic polynomial-time Turing machine* P_1^* *it holds that*

$$\left|\Pr\left[(c_1|c_2) \leftarrow \langle P_1^*(1^n), P_2(1^n)\rangle : c_2 = 1\right] - \frac{1}{2}\right| \leq \epsilon(n) + \nu(n) \ ,$$

for some negligible function $\nu(n)$ *and for all sufficiently large* n*.*

2.2 $1/p$-Indistinguishability and $1/p$-Secure Computation

$1/p$-Indistinguishability. A distribution ensemble $X = \{X(a,n)\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$, where $\mathcal{D}_n$ is a set that may depend on n. For a fixed polynomial $p(n)$, two distribution ensembles $X = \{X(a,n)\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ and $Y = \{Y(a,n)\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ 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 $\nu(n)$ such that for all sufficiently large $n \in \mathbb{N}$ and for all $a \in \mathcal{D}_n$ it holds that

$$\left|\Pr\left[D(X(a,n)) = 1\right] - \Pr\left[D(Y(a,n)) = 1\right]\right| \leq \frac{1}{p(n)} + \nu(n) \ .$$

$1/p$-Secure computation. A two-party protocol for computing a functionality $\mathcal{F} = \{(f^1, f^2)\}$ is a protocol running in polynomial time and satisfying the following functional requirement: if party P_1 holds input $(1^n, x)$, and party P_2 holds input $(1^n, y)$, then the joint distribution of the outputs of the parties is statistically close to $(f^1(x,y), f^2(x,y))$. In what follows we define the notion of $1/p$-secure computation [14, 20]. The definition uses the standard real/ideal paradigm [9, 13], 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 (we note that, in general, the notions of $1/p$-security and security-with-abort are incomparable). We consider active adversaries, who may deviate from the protocol in an arbitrary manner, and static corruptions.

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in a real protocol execution to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an ideal computation involving an incorruptible trusted party to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted party exists) can do no more harm than if it was involved in the above-described ideal computation.

Execution in the ideal model. The parties are P_1 and P_2, and there is an adversary $\mathcal{A}$ who has corrupted one of them. An ideal execution for the computation of $\mathcal{F} = \{f_n\}$ proceeds as follows:

Inputs: P_1 and P_2 hold the security parameter 1^n and inputs $x \in X_n$ and $y \in Y_n$, respectively. The adversary $\mathcal{A}$ receives an auxiliary input aux.

Send inputs to trusted party: The honest party sends its input to the trusted party. The corrupted party controlled by $\mathcal{A}$ may send any value of its choice. Denote the pair of inputs sent to the trusted party by (x', y').

Trusted party sends outputs: If $x' \notin X_n$ the trusted party sets x' to some default element $x_0 \in X_n$ (and likewise if $y' \notin Y_n$). Then, the trusted party chooses r uniformly at random and sends $f_n^1(x', y'; r)$ to P_1 and $f_n^2(x', y'; r)$ to P_2.

Outputs: The honest party outputs whatever it was sent by the trusted party, the corrupted party outputs nothing, and $\mathcal{A}$ outputs any arbitrary (probabilistic polynomial-time computable) function of its view.

We let $\mathsf{IDEAL}_{\mathcal{F},\mathcal{A}(\mathsf{aux})}(x, y, n)$ be the random variable consisting of the output of the adversary and the output of the honest party following an execution in the ideal model as described above.

Execution in the real model. We next consider the real model in which a two-party protocol π is executed by P_1 and P_2 (and there is no trusted party). In this case, the adversary $\mathcal{A}$ gets the inputs of the corrupted party and sends all messages on behalf of this party, using an arbitrary polynomial-time strategy. The honest party follows the instructions of π.

Let π be a two-party protocol computing $\mathcal{F}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input aux. We let $\mathsf{REAL}_{\pi,\mathcal{A}(\mathsf{aux})}(x, y, n)$ be the random variable consisting of the view of the adversary and the output of the honest party, following an execution of π where P_1 begins by holding input $(1^n, x)$, and P_2 begins by holding input $(1^n, y)$.

Security as emulation of an ideal execution in the real model . Having defined the ideal and real models, we can now define security of a protocol. Loosely speaking, the definition asserts that a secure protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated as follows:

Definition 2.2 (1/p-secure computation)**.** *Let $\mathcal{F}$ and π be as above, and fix a function $p = p(n)$. Protocol π is said to* $1/p$-securely compute *$\mathcal{F}$ 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*

$$\{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(\mathsf{aux})}(x, y, n)\}_{(x,y)\in X\times Y,\mathsf{aux}\in\{0,1\}^*} \stackrel{1/p}{\approx} \{\mathsf{REAL}_{\pi,\mathcal{A}(\mathsf{aux})}(x, y, n)\}_{(x,y)\in X\times Y,\mathsf{aux}\in\{0,1\}^*} .$$

2.3 One-Time Message Authentication

Message authentication codes provide assurance to the receiver of a message that it was sent by a specified legitimate sender, even in the presence of an active adversary who controls the communication channel. A message authentication code is defined via triplet $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ of probabilistic polynomial-time Turing machines such that:

1. The key generation algorithm Gen receives as input a security parameter 1^n, and outputs an authentication key k.

2. The authentication algorithm Mac receives as input an authentication key k and a message m, and outputs a tag t.

3. The verification algorithm Vrfy receives as input an authentication key k, a message m, and a tag t, and outputs a bit $b \in \{0, 1\}$.

The functionality guarantee of a message authentication code is that for any message m it holds that $\mathsf{Vrfy}(k, m, \mathsf{Mac}(k, m)) = 1$ with overwhelming probability over the internal coin tosses of Gen, Mac and Vrfy. In this paper we rely on message authentication codes that are one-time secure. That is, an authentication key is used to authenticate a single message. We consider an adversary that queries the authentication algorithm on a single message m of her choice, and then outputs a pair (m', t'). We say that the adversary forges an authentication tag if $m' \neq m$ and $\mathsf{Vrfy}(k, m', t') = 1$. Message authentication codes that are one-time secure exist in the information-theoretic setting, that is, even an unbounded adversary has only a negligible probability of forging an authentication tag. Constructions of such codes can be based, for example, on pair-wise independent hash functions [27].

3 A Simplified Protocol

In order to demonstrate the main ideas underlying our approach, in this section we present a simplified protocol. The simplification is two-fold: First, we consider the specific coin-flipping functionality (as in Theorem 1.1), and not the more general functionality of sampling from an arbitrary distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ (as in Theorem 1.2). Second, the coin-flipping protocol will only be $1/(2r)$-secure and not $1/(4r)$-secure.

We describe the protocol in a sequence of refinements. We first informally describe the protocol assuming the existence of a trusted third party. The trusted third party acts as a "dealer" in a pre-processing phase, sending each party an input that it uses in the protocol. In the protocol we make no assumptions about the computational power of the parties. We then eliminate the need for the trusted third party by having the parties execute a secure-with-abort protocol that implements its functionality (this can be done in a constant number of rounds).

The protocol. The joint input of the parties, P_1 and P_2, is the security parameter 1^n and a polynomial $r = r(n)$ indicating the number of rounds in the protocol. In the pre-processing phase the trusted third party chooses uniformly at random a value $i^* \in \{1, \dots, r\}$, that corresponds to the round in which the parties learn their outputs. In every round $i \in \{1, \dots, r\}$ each party learns one bit of information: P_1 learns a bit a_i, and P_2 learns a bit b_i. In every round $i \in \{1, \dots, i^* - 1\}$ (these are the "dummy" rounds) the values a_i and b_i are independently and uniformly chosen. In every round $i \in \{i^*, \dots, r\}$ the parties learn the same uniformly distributed bit $c = a_i = b_i$ which is their output in the protocol. If the parties complete all r rounds of the protocol, then P_1 and P_2 output a_r and b_r, respectively[4]. Otherwise, if a party aborts prematurely, the other party outputs the value of the previous round and halts. That is, if P_1 aborts in round $i \in \{1, \dots, r\}$ then P_2 outputs the value b_{i-1} and halts. Similarly, if P_2 aborts in round i then P_1 outputs the value a_{i-1} and halts.

[4]An alternative approach that reduces the *expected* number of rounds from r to $r/2 + 1$ is as follows. In round i^* the parties learn their output $c = a_{i^*} = b_{i^*}$, and in round $i^* + 1$ they learn a special value $a_{i^*+1} = b_{i^*+1} = \text{NULL}$ indicating that they should output the value from the previous round and halt. For simplicity (both in the presentation of the protocol and in the proof of security) we chose to present the protocol as always having r rounds, but this is not essential for our results.

More specifically, in the pre-processing phase the trusted third party chooses $i^* \in \{1,\dots,r\}$ uniformly at random and defines $a_1,\dots,a_r$ and $b_1,\dots,b_r$ as follows: First, it choose $a_1,\dots,a_{i^*-1} \in \{0,1\}$ and $b_1,\dots,b_{i^*-1} \in \{0,1\}$ independently and uniformly at random. Then, it chooses $c \in \{0,1\}$ uniformly at random and lets $a_{i^*} = \cdots = a_r = b_{i^*} = \cdots = b_r = c$. The trusted third party creates secret shares of the values $a_1,\dots,a_r$ and $b_1,\dots,b_r$ using an information-theoretically-secure 2-out-of-2 secret sharing scheme, and these shares are given to the parties. For concreteness, we use the specific secret-sharing scheme that splits a bit x into $(x^{(1)}, x^{(2)})$ by choosing $x^{(1)} \in \{0,1\}$ uniformly at random and letting $x^{(2)} = x \oplus x^{(1)}$. In every round $i \in \{1,\dots,r\}$ the parties exchange their shares for the current round, which enables P_1 to reconstruct a_i, and P_2 to reconstruct b_i. Clearly, when both parties are honest, the parties produce the same output bit which is uniformly distributed.

Eliminating the trusted third party. We eliminate the need for the trusted third party by relying on a *possibly unfair* sub-protocol that securely computes with abort the functionality $\mathsf{ShareGen}_r$, formally described in Figure 1. Such a protocol with a constant number of rounds can be constructed assuming the existence of oblivious transfer (see, for example, [22]). In addition, our protocol also relies on a one-time message authentication code $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ that is information-theoretically secure. The functionality $\mathsf{ShareGen}_r$ provides the parties with authentication keys and authentication tags so each party can verify that the shares received from the other party were the ones generated by $\mathsf{ShareGen}_r$ in the pre-processing phase. A formal description of the protocol is provided in Figure 2.

Functionality $\mathsf{ShareGen}_r$

Input: Security parameter 1^n.

Computation:

1. Choose $i^* \in \{1,\dots,r\}$ uniformly at random.
2. Define values $a_1,\dots,a_r$ and $b_1,\dots,b_r$ as follows:
   - For $1 \le i \le i^*-1$ choose $a_i, b_i \in \{0,1\}$ independently and uniformly at random.
   - Choose $c \in \{0,1\}$ uniformly at random, and for $i^* \le i \le r$ let $a_i = b_i = c$.
3. For $1 \le i \le r$, choose $\left(a_i^{(1)}, a_i^{(2)}\right)$ and $\left(b_i^{(1)}, b_i^{(2)}\right)$ as random secret shares of a_i and b_i, respectively.
4. Compute $k_1^a,\dots,k_r^a,k_1^b,\dots,k_r^b \leftarrow \mathsf{Gen}(1^n)$. For $1 \le i \le r$, let $t_i^a = \mathsf{Mac}_{k_i^a}\left(i||a_i^{(2)}\right)$ and $t_i^b = \mathsf{Mac}_{k_i^b}\left(i||b_i^{(1)}\right)$.

Output:

1. Party P_1 receives the values $a_1^{(1)},\dots,a_r^{(1)}$, $\left(b_1^{(1)}, t_1^b\right),\dots,\left(b_r^{(1)}, t_r^b\right)$, and $k^a = (k_1^a,\dots,k_r^a)$.
2. Party P_2 receives the values $\left(a_1^{(2)}, t_1^a\right),\dots,\left(a_r^{(2)}, t_r^a\right)$, $b_1^{(2)},\dots,b_r^{(2)}$, and $k^b = (k_1^b,\dots,k_r^b)$.

Figure 1: The ideal functionality $\mathsf{ShareGen}_r$.

Protocol $\mathsf{CoinFlip}_r$

Joint input: Security parameter 1^n.

Preliminary phase:

1. Parties P_1 and P_2 run protocol π for computing $\mathsf{ShareGen}_r(1^n)$ (see Figure 1).
2. If P_1 receives $\perp$ from the above computation, it outputs a uniformly chosen bit and halts. Likewise, if P_2 receives $\perp$ it outputs a uniformly chosen bit and halts. Otherwise, the parties proceed.
3. Denote the output of P_1 from π by $a_1^{(1)}, \ldots, a_r^{(1)}$, $\left(b_1^{(1)}, t_1^b\right), \ldots, \left(b_r^{(1)}, t_r^b\right)$, and $k^a = (k_1^a, \ldots, k_r^a)$.
4. Denote the output of P_2 from π by $\left(a_1^{(2)}, t_1^a\right), \ldots, \left(a_r^{(2)}, t_r^a\right)$, $b_1^{(2)}, \ldots, b_r^{(2)}$, and $k^b = (k_1^b, \ldots, k_r^b)$.

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

1. P_2 sends the next share to P_1:
   (a) P_2 sends $\left(a_i^{(2)}, t_i^a\right)$ to P_1.
   (b) P_1 receives $\left(\hat{a}_i^{(2)}, \hat{t}_i^a\right)$ from P_2. If $\mathsf{Vrfy}_{k_i^a}\left(i||\hat{a}_i^{(2)}, \hat{t}_i^a\right) = 0$ (or if P_1 received an invalid message or no message), then P_1 outputs a_{i-1} and halts (if $i = 1$ it outputs a uniformly chosen bit).
   (c) If $\mathsf{Vrfy}_{k_i^a}\left(i||\hat{a}_i^{(2)}, \hat{t}_i^a\right) = 1$ then P_1 reconstructs a_i using the shares $a_i^{(1)}$ and $\hat{a}_i^{(2)}$.
2. P_1 sends the next share to P_2:
   (a) P_1 sends $\left(b_i^{(1)}, t_i^b\right)$ to P_2.
   (b) P_2 receives $\left(\hat{b}_i^{(1)}, \hat{t}_i^b\right)$ from P_1. If $\mathsf{Vrfy}_{k_i^b}\left(i||\hat{b}_i^{(1)}, \hat{t}_i^b\right) = 0$ (or if P_2 received an invalid message or no message), then P_2 outputs b_{i-1} and halts (if $i = 1$ it outputs a uniformly chosen bit).
   (c) If $\mathsf{Vrfy}_{k_i^b}\left(i||\hat{b}_i^{(1)}, \hat{t}_i^b\right) = 1$ then P_2 reconstructs b_i using the shares $b_i^{(1)}$ and $b_i^{(2)}$

Output: P_1 and P_2 output a_r and b_r, respectively.

Figure 2: The coin-flipping protocol $\mathsf{CoinFlip}_r$.

Proof of security. The following theorem proves that the protocol is $1/(2r)$-secure. We then conclude the section by showing the our analysis is in fact tight: there exists an efficient adversary that can bias the output of the honest party by essentially $1/(2r)$.

Theorem 3.1. *For any polynomial $r = r(n)$, if protocol π securely computes* $\mathsf{ShareGen}_r$ *with abort, then protocol* $\mathsf{CoinFlip}_r$ *is $1/(2r)$-secure.*

Proof. We prove the (1/2r)-security of protocol $\mathsf{CoinFlip}_r$ in a hybrid model where a trusted party for computing $\mathsf{ShareGen}_r$ with abort is available. Using standard techniques (see [9]), it then follows that when replacing the trusted party computing $\mathsf{ShareGen}_r$ with a sub-protocol that security computes $\mathsf{ShareGen}_r$ with abort, the resulting protocol is $1/(2r)$-secure.

Specifically, for every polynomial-time hybrid-model adversary $\mathcal{A}$ corrupting P_1 and running $\mathsf{CoinFlip}_r$ in the hybrid model, we show that there exists a polynomial-time ideal-model adversary $\mathcal{S}$ corrupting P_1 in the ideal model with access to a trusted party computing the coin-flipping functionality such that the statistical distance between these two executions is at most $1/(2r)+\nu(n)$, for some negligible function $\nu(n)$. For simplicity, in the remainder of the proof we ignore the aspect of message authentication in the protocol, and assume that the only malicious behavior of the adversary $\mathcal{A}$ is early abort. This does not result in any loss of generality, since there is only a negligible probably of forging an authentication tag.

On input $(1^n, \mathsf{aux})$ the ideal-model adversary $\mathcal{S}$ invokes the hybrid-model adversary $\mathcal{A}$ on $(1^n, \mathsf{aux})$ and queries the trusted party computing the coin-flipping functionality to obtain a bit c. The ideal-model adversary $\mathcal{S}$ proceeds as follows:

1. $\mathcal{S}$ simulates the trusted party computing the $\mathsf{ShareGen}_r$ functionality by sending $\mathcal{A}$ independently and uniformly chosen shares $a_1^{(1)}, \ldots, a_r^{(1)}, b_1^{(1)}, \ldots, b_r^{(1)}$. If $\mathcal{A}$ aborts (i.e., if $\mathcal{A}$ sends abort to the simulated $\mathsf{ShareGen}_r$ after receiving the shares), then $\mathcal{S}$ outputs $\mathcal{A}$'s output and halts.

2. $\mathcal{S}$ chooses $i^* \in \{1, \ldots, r\}$ uniformly at random.

3. In every round $i \in \{1, \ldots, i^* - 1\}$, $\mathcal{S}$ chooses a random bit a_i, and sends $\mathcal{A}$ the share $a_i^{(2)} = a_i^{(1)} \oplus a_i$. If $\mathcal{A}$ aborts then $\mathcal{S}$ outputs $\mathcal{A}$'s output and halts.

4. In every round $i \in \{i^*, \ldots, r\}$, $\mathcal{S}$ sends $\mathcal{A}$ the share $a_{i^*+1}^{(2)} = a_{i^*+1}^{(1)} \oplus c$ (recall that c is the value received from the trusted party computing the coin-flipping functionality). If $\mathcal{A}$ aborts then $\mathcal{S}$ outputs $\mathcal{A}$'s output and halts.

5. At the end of the protocol $\mathcal{S}$ outputs $\mathcal{A}$'s output and halts.

We now consider the joint distribution of $\mathcal{A}$'s view and the output of the honest party P_2 in the ideal model and in the hybrid model. There are three cases to consider:

1. $\mathcal{A}$ aborts before round i^*. In this case the distributions are identical: in both models the view of the adversary is the sequence of shares, and the sequence of messages up to the round in which $\mathcal{A}$ aborted, and the output of P_2 is a uniformly distributed bit which is independent of $\mathcal{A}$'s view.

2. $\mathcal{A}$ aborts in round i^*. In this case $\mathcal{A}$'s view is identical in both models, but the distributions of P_2's output given $\mathcal{A}$'s view are not identical. In the ideal model, P_2 outputs the random bit c that was revealed to $\mathcal{A}$ by $\mathcal{S}$ in round i^* (recall that c is the bit received from the trusted party computing the coin-flipping functionality). In the hybrid model, however, the output of P_2 is the value b_{i^*-1} which is a random bit that is independent of $\mathcal{A}$'s view. Thus, in this case the statistical distance between the two distributions is $1/2$. However, this case occurs with probability at most $1/r$ since in both models i^* is independent of $\mathcal{A}$'s view until this round (that is, the probability that $\mathcal{A}$ aborts in round i^* is at most $1/r$).

3. $\mathcal{A}$ aborts after round i^* or does not abort. In this case the distributions are identical: the output of P_2 is the same random bit that was revealed to $\mathcal{A}$ in round i^*.

The above three cases imply that the statistical distance between the two distributions is at most $1/(2r)$, and this concludes the proof. ■

Claim 3.2. *In protocol* $\mathsf{CoinFlip}_r$ *there exists an efficient adversarial party* P_1^* *that can bias the output of* P_2 *by* $\frac{1-2^{-r}}{2r}$.

Proof. Consider the adversarial party P_1^* that completes the pre-processing phase, and then halts in the first round $i \in \{1, \ldots, r\}$ for which $a_i = 0$. We denote by Abort the random variable corresponding to the round in which P_1^* aborts, where $\mathsf{Abort} = \perp$ if P_1^* does not abort. In addition, we denote by c_2 the random variable corresponding to the output bit of P_2. Notice that if P_1^* aborts in round $j \le i^*$ then P_2 outputs a random bit, and if P_1^* does not abort then P_2 always outputs 1. Therefore, for every $i \in \{1, \dots, r\}$ it holds that

$$\begin{aligned}
\Pr[c_2 = 1 \mid i^* = i] &= \sum_{j=1}^{i^*} \Pr[\mathsf{Abort} = j \mid i^* = i] \Pr[c_2 = 1 \mid \mathsf{Abort} = j \wedge i^* = i] \\
&\quad + \Pr[\mathsf{Abort} = \perp \mid i^* = i] \Pr[c_2 = 1 \mid \mathsf{Abort} = \perp \wedge i^* = i] \\
&= \sum_{j=1}^{i^*} \Pr[a_1 = \cdots = a_{j-1} = 1, a_j = 0] \Pr[c_2 = 1 \mid \mathsf{Abort} = j \wedge i^* = i] \\
&\quad + \Pr[a_1 = \cdots = a_{i^*} = 1] \Pr[c_2 = 1 \mid \mathsf{Abort} = \perp \wedge i^* = i] \\
&= \sum_{j=1}^{i^*} \frac{1}{2^j} \cdot \frac{1}{2} + \frac{1}{2^{i^*}} \cdot 1 \\
&= \frac{1}{2} + \frac{1}{2^{i+1}} \quad .
\end{aligned}$$

This implies that

$$\begin{aligned}
\Pr[c_2 = 1] &= \sum_{i=1}^{r} \Pr[i^* = i] \Pr[c_2 = 1 \mid i^* = i] \\
&= \sum_{i=1}^{r} \frac{1}{r} \left(\frac{1}{2} + \frac{1}{2^{i+1}} \right) \\
&= \frac{1}{2} + \frac{1 - 2^{-r}}{2r} \quad .
\end{aligned}$$

■

4 The Generalized Protocol

In this section we prove Theorems 1.1 and 1.2 by presenting a more refined and generalized protocol. The improvements over the protocol presented in Section 3 are as follows:

- Improved security guarantee. In the simplified protocol party P_1 can bias the output of party P_2 (by aborting in round i^*), but party P_2 cannot not bias the output of party P_1. This is due to the fact that party P_1 always learns the output before party P_2 does. In the generalized protocol the party that learns the output before the other party is chosen uniformly at random (i.e., party P_1 learns the output before party P_2 with probability $1/2$). This is achieved by having the parties exchange a sequence of $2r$ values $(a_1, b_1), \dots, (a_{2r}, b_{2r})$ (using the same secret-sharing exchange technique as in the simplified protocol) with the following property: for odd values of i, party P_1 learns a_i before party P_2 learns b_i, and for even values of i party P_2 learns b_i before party P_1 learns a_i. Thus, party P_1 can bias the result only when i^* is odd, and party P_2 can bias the result only when i^* is even. The key point is that the parties can exchange the sequence of $2r$ shares in only $r+1$ rounds by combining some of their messages[5].

- A larger class of functionalities. We consider the more general task of sampling from a distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$: party P_1 receives a sample from $\mathcal{D}_1$ and party P_2 receives a correlated sample from $\mathcal{D}_2$ (in coin-flipping, for example, the joint distribution $\mathcal{D}$ produces the

[5]Recall that each round consists of two moves: a message from P_2 to P_1 followed by a message from P_1 to P_2.

values $(0,0)$ and $(1,1)$ each with probability $1/2$). Our generalized protocol can handle any polynomially-sampleable distribution $\mathcal{D}$.

In the following we describe the generalized protocol $\mathsf{Sampling}_r$ (Section 4.1), and then prove its security (Section 4.2).

4.1 Description of the Protocol

Joint input: Security parameter 1^n, and a polynomially-sampleable distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$.

Preliminary phase:

1. Parties P_1 and P_2 run protocol π for computing $\mathsf{ShareGen}_r(1^n, \mathcal{D})$ (see Figure 3).
2. If P_1 receives $\perp$ from the above computation, it outputs a random sample from $\mathcal{D}_1$ and halts. Likewise, if P_2 receives $\perp$ it outputs a random sample from $\mathcal{D}_2$ and halts. Otherwise, the parties proceed.
3. Denote the output of P_1 from π by $a_1^{(1)}, \dots, a_{2r}^{(1)}, \left(b_1^{(1)}, t_1^b\right), \dots, \left(b_{2r}^{(1)}, t_{2r}^b\right)$, and $k_1^a, \dots, k_{2r}^a$.
4. Denote the output of P_2 from π by $\left(a_1^{(2)}, t_1^a\right), \dots, \left(a_{2r}^{(2)}, t_{2r}^a\right)$, $b_1^{(2)}, \dots, b_{2r}^{(2)}$, and $k_1^b, \dots, k_{2r}^b$.

In round 1 do:

1. P_2 sends a share to P_1:
   (a) P_2 sends $\left(a_1^{(2)}, t_1^a\right)$ to P_1.
   (b) P_1 receives $\left(\hat{a}_1^{(2)}, \hat{t}_1^a\right)$ from P_2. If $\mathsf{Vrfy}_{k_1^a}\left(1 || \hat{a}_1^{(2)}, \hat{t}_1^a\right) = 0$, then P_1 outputs a random sample from $\mathcal{D}_1$ and halts. Otherwise, P_1 reconstructs a_1 using the shares $a_1^{(1)}$ and $\hat{a}_1^{(2)}$.
2. P_1 sends a pair of shares to P_2:
   (a) P_1 sends $\left(b_1^{(1)}, t_1^b\right)$ and $\left(b_2^{(1)}, t_2^b\right)$ to P_2.
   (b) P_2 receives $\left(\hat{b}_1^{(1)}, \hat{t}_1^b\right)$ and $\left(\hat{b}_2^{(1)}, \hat{t}_2^b\right)$ from P_1. If $\mathsf{Vrfy}_{k_1^b}\left(1 || \hat{b}_1^{(1)}, \hat{t}_1^b\right) = 0$, then P_2 outputs a random sample from $\mathcal{D}_2$ and halts. Otherwise, P_2 reconstructs b_1 using the shares $b_1^{(1)}$ and $b_1^{(2)}$.
   (c) If $\mathsf{Vrfy}_{k_2^b}\left(2 || \hat{b}_2^{(1)}, \hat{t}_2^b\right) = 0$, then P_2 outputs b_1 and halts. Otherwise, P_2 reconstructs b_2 using the shares $b_2^{(1)}$ and $b_2^{(2)}$.

In each round $j = 2, \dots, r$ do:

1. P_2 sends a pair of shares to P_1:
   (a) P_2 sends $\left(a_{2j-2}^{(2)}, t_{2j-2}^a\right)$ and $\left(a_{2j-1}^{(2)}, t_{2j-1}^a\right)$ to P_1.
   (b) P_1 receives $\left(\hat{a}_{2j-2}^{(2)}, \hat{t}_{2j-2}^a\right)$ and $\left(\hat{a}_{2j-1}^{(2)}, \hat{t}_{2j-1}^a\right)$ from P_2. If $\mathsf{Vrfy}_{k_{2j-2}^a}\left(2j-2 || \hat{a}_{2j-2}^{(2)}, \hat{t}_{2j-2}^a\right) = 0$, then P_1 outputs a_{2j-3} and halts. Otherwise, P_1 reconstructs a_{2j-2} using the shares $a_{2j-2}^{(1)}$ and $\hat{a}_{2j-2}^{(2)}$.
   (c) If $\mathsf{Vrfy}_{k_{2j-1}^a}\left(2j-1 || \hat{a}_{2j-1}^{(2)}, \hat{t}_{2j-1}^a\right) = 0$, then P_1 outputs a_{2j-2} and halts. Otherwise, P_1 reconstructs a_{2j-1} using the shares $a_{2j-1}^{(1)}$ and $\hat{a}_{2j-1}^{(2)}$.

2. P_1 sends a pair of shares to P_2:
   (a) P_1 sends $\left(b_{2j-1}^{(1)}, t_{2j-1}^b\right)$ and $\left(b_{2j}^{(1)}, t_{2j}^b\right)$ to P_2.
   (b) P_2 receives $\left(\hat{b}_{2j-1}^{(1)}, \hat{t}_{2j-1}^b\right)$ and $\left(\hat{b}_{2j}^{(1)}, \hat{t}_{2j}^b\right)$ from P_1. If $\mathsf{Vrfy}_{k_{2j-1}^b}\left(2j-1||\hat{b}_{2j-1}^{(1)}, \hat{t}_{2j-1}^b\right) = 0$, then P_2 outputs b_{2j-2} and halts. Otherwise, P_2 reconstructs b_{2j-1} using the shares $b_{2j-1}^{(1)}$ and $b_{2j-1}^{(2)}$.
   (c) If $\mathsf{Vrfy}_{k_{2j}^b}\left(2j||\hat{b}_{2j}^{(1)}, \hat{t}_{2j}^b\right) = 0$, then P_2 outputs b_{2j-1} and halts. Otherwise, P_2 reconstructs b_{2j} using the shares $b_{2j}^{(1)}$ and $b_{2j}^{(2)}$.

In round $r+1$ do:

1. P_2 sends a share to P_1:
   (a) P_2 sends $\left(a_{2r}^{(2)}, t_{2r}^a\right)$ to P_1.
   (b) P_1 receives $\left(\hat{a}_{2r}^{(2)}, \hat{t}_{2r}^a\right)$ from P_2. If $\mathsf{Vrfy}_{k_{2r}^a}\left(2r||\hat{a}_{2r}^{(2)}, \hat{t}_{2r}^a\right) = 0$, then P_1 outputs a_{2r-1} and halts. Otherwise, P_1 reconstructs a_{2r} using the shares $a_{2r}^{(1)}$ and $\hat{a}_{2r}^{(2)}$.
2. P_1 and P_2 output the values a_{2r} and b_{2r}, respectively, and halt.

Functionality $\mathsf{ShareGen}_r$

Input: Security parameter 1^n, and a polynomially-sampleable distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$.

Computation:

1. Choose $i^* \in \{1, \ldots, 2r\}$ uniformly at random.
2. Define values $a_1, \ldots, a_{2r}$ and $b_1, \ldots, b_{2r}$ as follows:
   - For $1 \leq i \leq i^* - 1$ sample $a_i \leftarrow \mathcal{D}_1$ and $b_i \leftarrow \mathcal{D}_2$ independently.
   - Sample $(c^{(1)}, c^{(2)}) \leftarrow \mathcal{D}$, and for $i^* \leq i \leq r$ let $(a_i, b_i) = (c^{(1)}, c^{(2)})$.
3. For $1 \leq i \leq 2r$, choose $\left(a_i^{(1)}, a_i^{(2)}\right)$ and $\left(b_i^{(1)}, b_i^{(2)}\right)$ as random secret shares of a_i and b_i, respectively.
4. Compute $k_1^a, \ldots, k_{2r}^a, k_1^b, \ldots, k_{2r}^b \leftarrow \mathsf{Gen}(1^n)$. For $1 \leq i \leq 2r$, let $t_i^a = \mathsf{Mac}_{k_i^a}\left(i||a_i^{(2)}\right)$ and $t_i^b = \mathsf{Mac}_{k_i^b}\left(i||b_i^{(1)}\right)$.

Output:

1. Party P_1 receives the values $a_1^{(1)}, \ldots, a_{2r}^{(1)}, \left(b_1^{(1)}, t_1^b\right), \ldots, \left(b_{2r}^{(1)}, t_{2r}^b\right)$, and $k^a = (k_1^a, \ldots, k_{2r}^a)$.
2. Party P_2 receives the values $\left(a_1^{(2)}, t_1^a\right), \ldots, \left(a_{2r}^{(2)}, t_{2r}^a\right)$, $b_1^{(2)}, \ldots, b_{2r}^{(2)}$, and $k^b = (k_1^b, \ldots, k_{2r}^b)$.

Figure 3: The ideal functionality $\mathsf{ShareGen}_r$ of the generalized two-party protocol.

4.2 Proof of Security

We remind the reader that $\mathrm{SD}(\mathcal{D}, \mathcal{D}_1 \otimes \mathcal{D}_2)$ denotes the statistical distance between the joint distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ and the direct-product of the two marginal distributions $\mathcal{D}_1$ and $\mathcal{D}_2$. We prove the following theorem which implies Theorems 1.1 and 1.2:

Theorem 4.1. *For any polynomially-sampleable distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ and polynomial $r = r(n)$, if protocol π securely computes $\mathsf{ShareGen}_r$ with abort, then $\mathsf{Sampling}_r$ is $\frac{\mathrm{SD}(\mathcal{D}, \mathcal{D}_1 \otimes \mathcal{D}_2)}{2r}$-secure.*

Proof. As in the proof of Theorem 3.1 we prove the security of the protocol in a hybrid model where a trusted party for computing $\mathsf{ShareGen}_r$ with abort is available. For every polynomial-time hybrid-model adversary $\mathcal{A}$ corrupting P_1 and running $\mathsf{Sampling}_r$ in the hybrid model, we show that there exists a polynomial-time ideal-model adversary $\mathcal{S}$ corrupting P_1 in the ideal model with access to a trusted party that samples from $\mathcal{D}$ such that the *statistical* distance between these two executions is at most $\frac{\mathrm{SD}(\mathcal{D},\mathcal{D}_1\otimes\mathcal{D}_2)}{2r} + \nu(n)$, for some negligible function $\nu(n)$. The proof for the case that P_2 is corrupted is essentially identical, and therefore is omitted. For simplicity we ignore the aspect of message authentication, and assume that the only malicious behavior of $\mathcal{A}$ is early abort. This does not result in any loss of generality, since there is only a negligible probably of forging an authentication tag.

On input $(1^n, \mathsf{aux})$ the ideal-model adversary $\mathcal{S}$ invokes the hybrid-model adversary $\mathcal{A}$ on $(1^n, \mathsf{aux})$ and queries the trusted party who sends to the parties a sample (c_1, c_2) drawn from the joint distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$. At this point $\mathcal{S}$ receives the value c_1 which was sent to the corrupted P_1. $\mathcal{S}$ simulates the trusted party computing the $\mathsf{ShareGen}_r$ functionality by sending $\mathcal{A}$ independently and uniformly chosen shares $a_1^{(1)}, \ldots, a_{2r}^{(1)}, b_1^{(1)}, \ldots, b_{2r}^{(1)}$. If $\mathcal{A}$ aborts at this point then $\mathcal{S}$ outputs $\mathcal{A}$'s output and halts. Otherwise, $\mathcal{S}$ chooses $i^* \in \{1, \ldots, 2r\}$ uniformly at random, and proceeds by sending $\mathcal{A}$ shares $a_1^{(2)}, \ldots, a_{i^*+1}^{(2)}$ (in the order defined by the rounds of the protocol), where the shares are defined as follows:

1. For every $i \in \{1, \ldots, i^* - 1\}$, $\mathcal{S}$ samples a random $a_i \leftarrow \mathcal{D}_1$ and sets $a_i^{(2)} = a_i^{(1)} \oplus a_i$.

2. For every $i \leq \{i^*, \ldots, r\}$, $\mathcal{S}$ sets $a_{i^*}^{(2)} = a_{i^*}^{(1)} \oplus c_1$ where c_1 is the value received from the trusted party.

If at some point during the simulation $\mathcal{A}$ aborts, then $\mathcal{S}$ outputs $\mathcal{A}$'s output and halts.

We now consider the joint distribution of the adversaries view and the output of the honest party P_2 in the ideal model and in the hybrid model, and show that the statistical distance between the two distributions is as most $\frac{\mathrm{SD}(\mathcal{D},\mathcal{D}_1\otimes\mathcal{D}_2)}{2r}$. As in the proof of Theorem 3.1, note that the adversary's view is always identically distributed in both cases, and therefore we only need to consider the distribution of P_2's output given the adversary's view. There are two cases to consider, depending on whether i^* is even or odd.

Case 1: $i^* = 2j^*$ for some $j^* \in \{1, \ldots, r\}$. In this case P_2 learns its output in round j^* and P_1 learns its output in round $j^* + 1$, and we show that the two distributions are identical. There are two cases to consider:

1. $\mathcal{A}$ aborts before round $j^* + 1$. In both models if $\mathcal{A}$ aborts before round $j^* + 1$ then he does not receive the share $a_{i^*}^{(2)} = a_{2j^*}^{(2)}$ since this share is sent by P_2 only in round $j^* + 1$. Therefore, $\mathcal{A}$'s view is independent of P_2's output.

2. $\mathcal{A}$ aborts in round $j^* + 1$ or does not abort. In this case in both models $\mathcal{A}$ learns c_1 and P_2 outputs c_2, where (c_1, c_2) are sampled from the joint distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$.

Case 2: $i^* = 2j^* - 1$ for some $j^* \in \{1, \ldots, r\}$. In this case both parties learn their outputs in round j^* but P_1 learns its output first. Informally, P_1 can bias P_2's output only by aborting in round j^* after receiving P_2 messages for this round. More formally, there are three cases to consider:

1. $\mathcal{A}$ aborts before round j^*. In this case the distributions are identical: in both models the view of the adversary is the sequence of shares, and the sequence of messages up to the round in

which $\mathcal{A}$ aborted, and the output of P_2 is a random sample from $\mathcal{D}_2$ that is independent of $\mathcal{A}$'s view.

2. $\mathcal{A}$ aborts in round j^*. In this case $\mathcal{A}$'s view is identical in both models, but the distributions of P_2's output given $\mathcal{A}$'s view are not identical. In the ideal model, P_2 outputs the value c_2 that is correlated to the value c_1 that was revealed to $\mathcal{A}$ by $\mathcal{S}$ in round i^* (i.e., (c_1, c_2) is sampled from the joint distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$). In the hybrid model, however, the output of P_2 is the value b_{i^*-1} which is a random sample from $\mathcal{D}_2$ that is independent of $\mathcal{A}$'s view. Thus, in this case the statistical distance between the two distributions is $\mathrm{SD}(\mathcal{D}, \mathcal{D}_1 \otimes \mathcal{D}_2)$. However, this case occurs with probability at most $1/2r$ since in both cases i^* is odd with probability exactly $1/2$ and is independent of $\mathcal{A}$'s view until this round (that is, the probability that $\mathcal{A}$ aborts in round j^* is at most $1/r$).

3. $\mathcal{A}$ aborts in round $j^* + 1$ or does not abort. In this case the distribution are identical: in both models $\mathcal{A}$ learns c_1 and P_2 outputs c_2, where (c_1, c_2) are sampled from the joint distribution $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$.

This implies that the statistical distance between the two distributions is at most $\frac{\mathrm{SD}(\mathcal{D}, \mathcal{D}_1 \otimes \mathcal{D}_2)}{2r}$ and concludes the proof of the theorem. ■

We conclude this section by showing that Theorem 4.2 is tight for the coin-flipping functionality: there exists an efficient adversary that can bias the output of the honest party by essentially $1/(4r)$. This adversary is a natural generalization of the adversary presented at the end of Section 3.

Claim 4.2. *In protocol* $\mathsf{Sampling}_r$ *instantiated with the distribution* $\mathcal{D}$ *that outputs the values* $(0, 0)$ *and* $(1, 1)$ *each with probability* $1/2$*, there exists an efficient adversarial party* P_1^* *that can bias the output of* P_2 *by* $\frac{1-2^{-r}}{4r}$.

Proof. Consider the adversarial party P_1^* that completes the pre-processing phase, and then halts in the first round $j \in \{1, \ldots, r\}$ for which $a_{2j-1} = 0$. We denote by Abort the random variable corresponding to the round in which P_1^* aborts, where $\mathsf{Abort} = \perp$ if P_1^* does not abort. In addition, we denote by c_2 the random variable corresponding to the output bit of P_2. Notice that if i^* is even, then P_2 outputs 1 with probability $1/2$. Now, suppose that $i^* = 2j^* - 1$ for some $j^* \in \{1, \ldots, r\}$, then there are two cases to consider:

- If P_1^* aborts in round $j \leq j^*$ then P_2 outputs a random bit.
- If P_1^* does not abort then P_2 always outputs 1.

Therefore, for every $j^* \in \{1, \dots, r\}$ it holds that

$$
\begin{aligned}
&\Pr\left[c_2 = 1 \mid i^* = 2j^* - 1\right] \\
&\quad = \sum_{j=1}^{j^*} \Pr\left[\mathsf{Abort} = j \mid i^* = 2j^* - 1\right] \Pr\left[c_2 = 1 \mid \mathsf{Abort} = j \wedge i^* = 2j^* - 1\right] \\
&\quad\quad + \Pr\left[\mathsf{Abort} = \perp \mid i^* = 2j^* - 1\right] \Pr\left[c_2 = 1 \mid \mathsf{Abort} = \perp \wedge i^* = 2j^* - 1\right] \\
&\quad = \sum_{j=1}^{j^*} \Pr\left[a_1 = \cdots = a_{2j-3} = 1, a_{2j-1} = 0\right] \Pr\left[c_2 = 1 \mid \mathsf{Abort} = j \wedge i^* = 2j^* - 1\right] \\
&\quad\quad + \Pr\left[a_1 = \cdots = a_{2j^*-3} = a_{2j^*-1} = 1\right] \Pr\left[c_2 = 1 \mid \mathsf{Abort} = \perp \wedge i^* = 2j^* - 1\right] \\
&\quad = \sum_{j=1}^{j^*} \frac{1}{2^j} \cdot \frac{1}{2} + \frac{1}{2^{j^*}} \cdot 1 \\
&\quad = \frac{1}{2} + \frac{1}{2^{j^*+1}} \ .
\end{aligned}
$$

This implies that

$$
\begin{aligned}
\Pr\left[c_2 = 1\right] &= \Pr\left[i^* \text{ is even}\right] \Pr\left[c_2 = 1 \mid i^* \text{ is even}\right] \\
&\quad + \sum_{j^*=1}^{r} \Pr\left[i^* = 2j^* - 1\right] \Pr\left[c_2 = 1 \mid i^* = 2j^* - 1\right] \\
&= \frac{1}{2} \cdot \frac{1}{2} + \sum_{j^*=1}^{r} \frac{1}{2r} \left(\frac{1}{2} + \frac{1}{2^{j^*+1}} \right) \\
&= \frac{1}{2} + \frac{1 - 2^{-r}}{4r} \ .
\end{aligned}
$$

■

5 Open Problems

Identifying the minimal computational assumptions. Blum's coin-flipping protocol, as well as its generalization that guarantees bias of $O(1/\sqrt{r})$, can rely on the existence of any one-way function. We showed that the optimal trade-off between the round complexity and the bias can be achieved assuming the existence of oblivious transfer, a complete primitive for secure computation. A challenging problem is to either achieve the optimal bias based on seemingly weaker assumptions (e.g., one-way functions), or to demonstrate that oblivious transfer is in fact essential.

Identifying the exact trade-off. The bias of our protocol almost exactly matches Cleve's lower bound: Cleve showed that any r-round protocol has bias at least $1/(8r + 2)$, and we manage to achieve bias of at most $1/(4r - c)$ for some constant $c > 0$. It will be interesting to eliminate the multiplicative gap of $1/2$ by either improving Cleve's lower bound or by improving our upper bound. We note, however, that this cannot be resolved by improving the security analysis of our protocol since there exists an efficient adversary that can bias our protocol by essentially $1/(4r)$ (see Section 4), and therefore our analysis is tight.

Efficient implementation. Our protocol uses a general secure computation step in the preprocessing phase. Although asymptotically optimal, the techniques used in general secure computation often have a large overhead. Hence, it would be helpful to find an efficient sub-protocol to compute the $\mathsf{ShareGen}_r$ functionality that can be used in a practical implementation.

References

[1] D. Aharonov, A. Ta-Shma, U. V. Vazirani, and A. C. Yao. Quantum bit escrow. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 705–714, 2000.

[2] N. Alon and M. Naor. Coin-flipping games immune against linear-sized coalitions. *SIAM Journal on Computing*, 22(2):403–417, 1993.

[3] A. Ambainis. A new protocol and lower bounds for quantum coin flipping. *Journal of Computer and System Sciences*, 68(2):398–416, 2004.

[4] A. Ambainis, H. Buhrman, Y. Dodis, and H. Rohrig. Multiparty quantum coin flipping. In *Proceedings of the 19th Annual IEEE Conference on Computational Complexity*, pages 250–259, 2004.

[5] B. Averbuch, M. Blum, B. Chor, S. Goldwasser, and S. Micali. How to implement Bracha's $O(\log n)$ byzantine agreement algorithm. Manuscript, 1985.

[6] 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*, pages 1–10, 1988.

[7] M. Ben-Or and N. Linial. Collective coin flipping. *Randomness and Computation*, pages 91–115, 1989.

[8] M. Blum. Coin flipping by telephone - A protocol for solving impossible problems. In *Proceedings of the 25th IEEE Computer Society International Conference*, pages 133–137, 1982.

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

[10] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 364–369, 1986.

[11] R. Cleve and R. Impagliazzo. Martingales, collective coin flipping and discrete control processes. `http://www.cpsc.ucalgary.ca/~cleve/pubs/martingales.ps`, 1993.

[12] U. Feige. Noncryptographic selection protocols. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 142–153, 1999.

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

[14] D. Gordon and J. Katz. Partial fairness in secure two-party computation. Cryptology ePrint Archive, Report 2008/206, 2008.

[15] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 413–422, 2008.

[16] S. D. Gordon and J. Katz. Rational secret sharing, revisited. In *Proceedings on the 5th International Conference on Security and Cryptography for Networks*, pages 229–241, 2006.

[17] J. Y. Halpern and V. Teague. Rational secret sharing and multiparty computation. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 623–632, 2004.

[18] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

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

[20] J. Katz. On achieving the "best of both worlds" in secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of computing*, pages 11–20, 2007.

[21] G. Kol and M. Naor. Cryptography and game theory: Designing protocols for exchanging information. In *Proceedings of the 5th Theory of Cryptography Conference*, pages 320–339, 2008.

[22] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, 2003.

[23] T. Moran and M. Naor. Basing cryptographic protocols on tamper-evident seals. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, pages 285–297, 2005.

[24] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[25] A. Russell and D. Zuckerman. Perfect information leader election in $\log^* n + O(1)$ rounds. *Journal of Computer and System Sciences*, 63(4):612–626, 2001.

[26] M. Saks. A robust noncryptographic protocol for collective coin flipping. *SIAM Journal on Discrete Mathematics*, 2(2):240–244, 1989.

[27] M. N. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.