Efficient Distributed Coin-tossing Protocols

Hamidreza Amini Khorasgani
Department of Computer Science, Purdue University, USA
haminikh@purdue.edu

Hemanta K. Maji
Department of Computer Science, Purdue University, USA
hmaji@purdue.edu

Himanshi Mehta
Department of Computer Science, Purdue University, USA
mehta142@purdue.edu

Mingyuan Wang
Department of Computer Science, Purdue University, USA
wang1929@purdue.edu

Abstract

Consider a distributed coin-tossing protocol among n processors, where each processor takes turns to broadcast a single message. After each processor broadcasts her message, the outcome $\in \{0,1\}$ is a deterministic function of all the messages. Let $X \in (0,1)$ be the probability of the outcome being head. An eavesdropping adversary who monitors this protocol can intervene once by restarting the processor who has just sent her message. Increasing the number of processors n enables reducing the change in the outcome probability distribution that this adversary can effect. Given a target tolerance ε, our objective is to use the minimum number of processors ensuring that the adversary mentioned above can only change the outcome distribution by at most ε.

A historically prominent protocol in this scenario, when $X = 1/2$, is the "majority protocol" (for example, Blum–1983, Awerbuch, Blum, Chor, Goldwasser, and Micali–1985, Cleve–1986), where each party broadcasts an independent and uniformly random coin, and the outcome is the majority of the broadcast coins. More generally, threshold protocols output heads if the number of processors that broadcast heads exceeds a particular threshold. Recently, Khorasgani, Maji, and Mukherjee (2019) prove the existence of coin-tossing protocols that achieve the same tolerance as the threshold protocols using a smaller number of processors. However, their protocol is not computationally efficient.

Towards this objective, for any $X \in (0,1)$ and $n \in \mathbb{N}$, this paper presents computationally efficient coin-tossing protocols approximating the new protocols of Khorasgani, Maji, and Mukherjee (2019). The running time of this protocol is linear in the accuracy parameter of this approximation, which can be set arbitrarily small.

Keywords and phrases Distributed Coin-tossing Protocols, Information-theoretic Security, Computationally Efficient

Contents

1 Introduction

Coin-tossing problem is one of the most fundamental problem in theoretical computer science. There is a vast literature of influential works [KKL88, LLS89, BL89, CI93, Fei99, GKP15] that studies this problem through diverse notions of securities. In this paper, we study its security against *strong adaptive adversaries* as introduced by Goldwasser, Kalai, and Park [GKP15].[1] Specifically, we consider the following motivating problem, which is first proposed by Cleve and Impagliazzo [CI93].

Representative Motivating Problem. Consider a distributed coin-tossing protocol among n processors. The protocol consists of n rounds, where at round i, the i^{th} processor broadcasts her message. After each processor broadcasts her message, the outcome of the protocol $\in \{0,1\}$ is a deterministic function of n broadcast messages. An honest execution of this protocol shall have expected outcome $X \in (0,1)$, namely, a *bias-X distributed coin-tossing protocol*. An eavesdropping adversary, who monitors the execution of this protocol, might intervene in the following manner. After round i, upon seeing the first i messages, the adversary decides whether to corrupt the i^{th} processor by *restarting* it. That is, the adversary forces the i^{th} processor to re-sample a new message. Throughtout the protocol, the adversary can corrupt at most one processor. The *insecurity* of a distributed coin-tossing protocol, in the presence of this adversary, is the maximum change in the distribution of the outcome that the adversary can cause. The objective, given the bias X and number of processors n, is to design a bias-X distributed coin-tossing protocol among n processors that attains the least amount of insecurity.

Threshold Protocols. Despite many negative results [CI93, BHMO18, KMM19], the only known protocol is (essentially) the threshold protocol [Blu83, ABC+85, Cle86]. Suppose $X \in (0,1)$ such that there exists a threshold $t \in \{0,1,\ldots,n+1\}$ satisfying $\sum_{i=t}^{n} \binom{n}{i} \cdot 2^{-n} = X$. In a threshold coin-tossing protocol, the processor i broadcasts an independent and uniformly random bit $C_i \in \{0,1\}$, where $1 \leqslant i \leqslant n$. The outcome of the protocol is 1 if (and only if) $\sum_{i=1}^{n} C_i \geqslant t$. One can argue that the adversary mentioned above can increase the expected outcome by $\binom{n}{t-1} \cdot 2^{-(n+1)}$.[2] Likewise, the adversary can also decrease the expected outcome by $\binom{n}{t} \cdot 2^{-(n+1)}$. Consequently, the adversary can alter the expected outcome by $\frac{1}{2^{n+1}} \cdot \max\left\{\binom{n}{t-1}, \binom{n}{t}\right\}$; that is, the threshold protocol is $\frac{1}{2^{n+1}} \cdot \max\left\{\binom{n}{t-1}, \binom{n}{t}\right\}$-*insecure*.

Khorasgani-Maji-Mukherjee Protocols. Recently, Khorasgani, Maji, and Mukherjee [KMM19] define new coin-tossing protocols. The protocol description, as well as their insecurity, is inductively defined using an appropriate geometric transformation (see Section 3). These protocols have shown the potential to achieve lower insecurity than the threshold protocols using an identical number of processors. Table 1 illustrates this reduction in insecurity for the representative example of $n = 5$ and all $X \in (0, 1/2]$ realizable by threshold protocols.[3] Before this work, it is unknown how to efficiently implement these

[1] That is, the adversary gets to see a processor's message before it decides whether to corrupt this processor or not.

[2] Intuitively, if an adversary aims to increase the expected outcome, she shall restart a processor whose message is 0. The timing of this attack, e.g., whether she restarts the first or the second processor she sees with 0 message, is inconsequential. Effectively, the increase in the expected outcome is attributed to the scenarios, where an honest execution ends up having exactly $t-1$ number of 1 messages (just below the threshold). In these scenarios, by re-sampling one 0 message, there is a 50% chance that the outcome is flipped from 0 to 1. For the rest scenarios, the re-sampling attack on one processor is ineffective. Hence, an adversary can increase the expected outcome by $\binom{n}{t-1} \cdot 2^{-(n+1)}$.

[3] Bias-X coin-tossing protocols are equivalent to bias-$(1-X)$ coin-tossing protocols. Therefore, it suffices to study $X \in (0, 1/2]$.

X	$\varepsilon_{\text{Thresh}}$	ε_{KMM}
1/32	$5/64 \approx 0.078125$	$0.0217\ldots$
6/32	$10/64 \approx 0.15625$	$0.0923\ldots$
16/32	$10/64 \approx 0.15625$	$0.1415\ldots$

Table 1 The insecurity of bias-X coin-tossing protocols, where $X \in (0, 1/2]$ and $n = 5$. Column $\varepsilon_{\text{Thresh}}$ presents the insecurity of the threshold protocol [Blu83, ABC+85, Cle86], and column ε_{KMM} presents the insecurity of the Khorasgani-Maji-Mukherjee protocol [KMM19].

protocols.

This paper, for any $X \in (0, 1)$ and $n \in \mathbb{N}$, presents a computationally efficient protocol approximating the n-processor bias-X Khorasgani-Maji-Mukherjee coin-tossing protocol.

1.1 Our Contributions

For a bias-X distributed coin-tossing protocol Π, let $\varepsilon(\Pi)$ denote the insecurity of protocol Π. Recall that the insecurity of a protocol is the maximum change an adversary can cause by restarting (at most) one processor after seeing her message. Let $\Pi_{\mathsf{KMM}}(n, X)$ represent the Khorasgani-Maji-Mukherjee protocol with n processors and bias-X [KMM19] (See Section 3). Our contributions are both theoretical and experimental.

Theoretical Results. Theoretically, for any integer $n > 0$ and $X \in (0, 1)$, we give a computationally efficient bias-X distributed coin-tossing protocol among n processors. Our protocol, denoted by $\Pi_{\mathsf{Our}}(n, X, \delta)$, is parametrized by an accuracy parameter δ. Intuitively, the smaller δ is, the more accurately our protocol $\Pi_{\mathsf{Our}}(n, X, \delta)$ approximates the KMM-protocol $\Pi_{\mathsf{KMM}}(n, X)$. The running time of our protocol is linearly dependent on the accuracy parameter δ. The theoretical results are summarized in the following main theorem.

▶ **Theorem 1.** *For any number of processors $n \in \mathbb{N}$, bias $X \in [0, 1]$, and accuracy parameter $\delta \in [0, 1]$, there exists a computationally efficient n-processor bias-X coin-tossing protocol $\Pi_{\mathsf{Our}}(n, X, \delta)$ such that the following bound holds for its insecurity.*

$$\varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right) < \varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right) + n\delta.$$

Furthermore, the time complexity of the next-message generation of processor i, where $1 \leqslant i \leqslant n$, in the protocol $\Pi_{\mathsf{Our}}(n, X, \delta)$ is linear in $(n - i)/\delta$.

Note that, Khorasgani et al. [KMM19] show that the insecurity of their protocol satisfies $\varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right) \geqslant \sqrt{\frac{1}{2(n+1)}} \cdot X(1 - X)$. Therefore, when the accuracy parameter is sufficiently small, then the insecurity of our protocol is $(1 + \mathrm{o}(1))$ multiplicative factor close to the insecurity of the corresponding KMM-protocol. For instance, if we set the accuracy parameter $\delta = \mathrm{o}\left(\frac{X(1-X)}{n^{3/2}}\right)$, this implies that

$$\varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right) \leqslant (1 + \mathrm{o}(1)) \cdot \varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right).$$

▶ Remark. We emphasize that, one can employ an additional optimization that shifts (nearly) the entire computational cost to an offline precomputation phase, which is independent of the bias of the coin-tossing protocol. More concretely, any processor participating in n-processor coin-tossing protocols performs a precomputation step, which has linear time-complexity in n/δ, irrespective of the bias of the coin-tossing protocol execution in the future. During any particular instance of n'-processor coin-tossing protocol execution (irrespective of its bias),

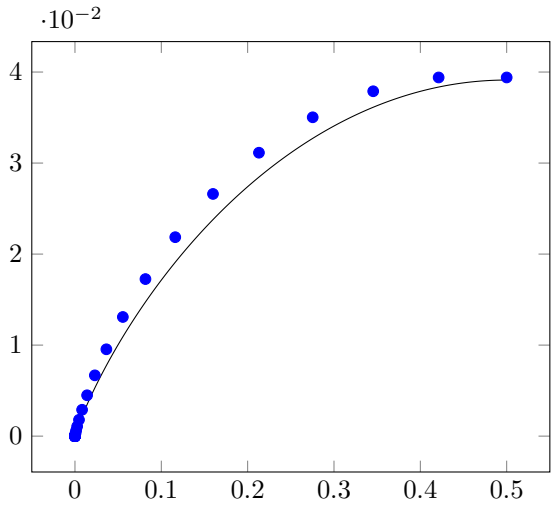

Figure 1 For $n = 101$, the blue marks denote the insecurity of bias-X coin-tossing protocols that are implementable using a threshold protocol. Versus the plot of the insecurity of our protocol, i.e., $\varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right)$, when $X \in [0, 1/2]$ and $\delta = 10^{-6}$. The data of this plot can be found at https://www.cs.purdue.edu/homes/hmaji/papers/data/A-101.csv and https://www.cs.purdue.edu/homes/hmaji/papers/data/thresh.csv, respectively.

where $n' < n$, this processor can generate her next-message of the coin-tossing protocol in *constant* time. Therefore, one precomputation step enables the participation in an arbitrary number of coin-tossing protocol instances involving $n' < n$ processors; thus, enabling the amortization of the computational cost of the precomputation step over multiple coin-tossing instances.

Experimental Results. Experimentally, we implement our protocol and show that the insecurity of our protocol is observably smaller than the insecurity of threshold protocols.

As a representative example, Figure 1 plots the insecurity of our protocol, for $n = 101$ processors and $X \in [0, 1/2]$ with accuracy parameter $\delta = 10^{-6}$.[4] Figure 1 also plots the insecurity of all bias-X coin-tossing protocols that can be implemented using a threshold protocol. Note that the insecurity of our protocol is less than the insecurity of threshold protocol. This reduction in insecurity is prominent specially when X is far from 0 and $1/2$.

Finally, our experiments uncover an exciting phenomenon. As Figure 2 indicates, our experimental results show that the insecurity of our protocols for $X = 1/2$ tends towards the insecurity of the majority protocol, as n tends to infinity. This experiment lends support to the conjecture that the majority protocol is the optimal secure coin-tossing protocol as $n \to \infty$.

1.2 Relevant Prior Works

Secure coin-tossing (or, randomized selection [GGL91, SV05, GVZ06], in general) is one of the most fundamental cryptographic primitives. Historically, designing coin-tossing protocols in the information-theoretic setting for diverse notions of security has been closely associated with some of the most highly influential research in theoretical computer science and extremal

[4] The insecurity of bias-X coin-tossing protocols, where $X \in (1/2, 1]$, is identical to the insecurity of bias-$(1 - X)$ coin-tossing protocols. So, it suffices to consider bias-X protocols, where $X \in [0, 1/2]$.

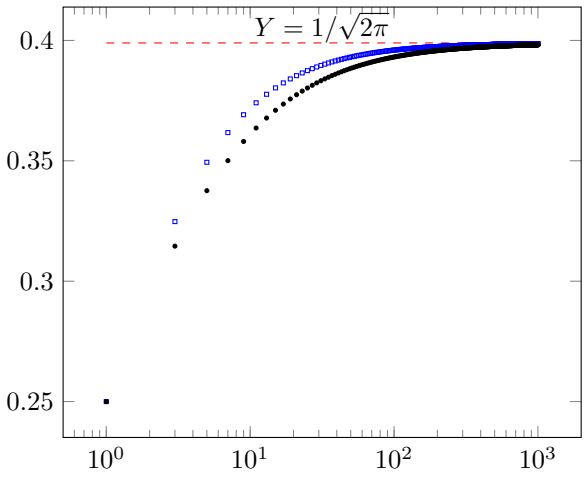

Figure 2 For $n \in \{1, 3, \ldots, 1001\}$, the blue squares show the plot of $\sqrt{n} \cdot \varepsilon_{\text{Maj}}(n)$, the insecurity of the majority coin-tossing protocol. The red dashed line shows the limit of the insecurity of majority protocol using Stirling's approximation, when $n \to \infty$. The black dots show the plot of $\sqrt{n} \cdot \varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right)$, where $X = 1/2$ and $\delta = 10^{-6}$. The graph uses log scale on the X-axis. The data for this plot can be found at https://www.cs.purdue.edu/homes/hmaji/papers/data/maj.csv

combinatorics. In this section, we highlight a few representative coin-tossing protocols for various security notions. This list is not intended to be exhaustive, which is beyond the scope of this paper.

Firstly, one can consider a *static* adversary. Such an adversary corrupts a processor before the protocol begins. During the execution of the protocol, when this particular processor is supposed to speak, she may restart it. In this case, the following protocol guarantees that the adversary can increase/decrease the expected outcome only by $\Theta(1/n)$.[5] In this protocol, each processor broadcasts an independent bit that is 0 with probability c/n and 1 with probability $(1 - c/n)$, and the outcome of the protocol is the AND of all the broadcast bits. We choose c such that $(1 - c/n)^n = 1/2$. For this protocol, one can verify that whichever processor the adversary corrupts, the insecurity is always $c/(2n)$.

Next, one can consider an *adaptive* adversary. That is, the adversary decides on whether to corrupt a processor after she see this processor's message. This is the model we consider in this paper. Historically, majority protocols (or threshold protocol, in general) are the only known protocol [Blu83, ABC+85, Cle86], which is $\Theta(1/\sqrt{n})$ insecure. It is also shown to be asymptotically optimal [CI93, KMM19] up to a constant factor. However, whether majority is *the optimal* protocol remains unknown. Recently, Khorasgani et al. [KMM19] shows the existence of protocols whose insecurity has the potential to be lower than majority. The exact insecurity of their protocols is not very well understood. Moreover, their protocol is also not efficiently implementable.

Another interesting security model for coin-tossing protocols is where the adversary is *rushing*. For a rushing adversary, she gets to see every processor's message before deciding

[5] This insecurity is known to be optimal up to a constant factor. To see this, note that the expected outcome before the protocol begins is $1/2$ and after the protocol ends is $\in \{0, 1\}$. Therefore, the expected outcome "jumps" by $1/2$ during the execution of the protocol. Intuitively, by an averaging argument, there must exists a processor whose message results in a jump of $\Omega(1/n)$ in the expected outcome. Therefore, one verify that a static adversay who corrupts this processor can change the distribution of the outcome by $\Omega(1/n)$.

to intervene. This notion of security is motivated particularly from the consideration of designing election schemes where changing one voter's vote has minimal effect on the overall outcome of the result. The first setting that has been well studied is one where the adversary statically corrupts a processor and can arbitrarily set her broadcast message after observing the messages of all other processors. This problem is the well-known problem of *influence of variables on boolean functions*. When each processor broadcasts a uniformly random bit, the *tribes* function is asymptotically optimal protocol [KKL88]. That is, processors are divided into a number of partitions called tribes, each of size $\Theta(\log n - \log\log n)$. The outcome is 1 if (and only if) there exists a tribe such that the messages of all the processors in this tribe are 1. One can verify that tribes function is $\Theta\left(\frac{\log n}{n}\right)$ insecure against static rushing adversaries. However, characterizing the exact optimal protocol in this setting is an open problem. Furthermore, the case when parties do not broadcast uniformly random bits remains relatively not well-understood [BKK+92, FHH+19].

For static rushing adversaries, there is also a large body of works that considers how many processors an adversary needs to corrupt to complete bias the outcome.[6] The work of [KKL88] shows that, for any function, an adversary needs to corrupt at most $\Theta\left(\frac{n}{\log n}\right)$ processors to completely bias the outcome. As one can verify, in this setting, tribe functions is no longer optimally secure. In fact, it is highly insecure, since an adversary only needs to corrupt an entire tribe ($\Theta(\log n)$ number of processors) to force the outcome to be 1. Majority function is also not asymptotically optimal insecure as an adversary only needs to corrupt $\Theta(\sqrt{n})$ processors. Ajtai and Linial [AL93] shows that there exists boolean functions that are resilient to $\Theta\left(\frac{n}{\log^2 n}\right)$, which almost matches the upper bound. Their construction are randomized and recently made explicit by Chattopadhyay and Zuckerman [CZ16]. Finally, if one relaxes the setting such that processors can interact and send multiple messages, the upper bound of [KKL88] no longer holds. The elegant "baton passing" protocol [Sak89] and "lightest bin" protocol [Fei99] are known to be resilient to up to αn corruptions, where $\alpha \in (0, 1/2)$.

In the model of adaptive rushing adversaries, Lichtenstein, Linial, and Saks [LLS89] first showed that if every party sends a uniform bit, majority protocol (more generally, threshold protocols) achieves the optimal security. Kalai, Komargodski, and Raz [TKKR18] showed that when parties may send arbitrarily long messages, an adaptive rushing adversary can corrupt (at most) $\sqrt{n}\operatorname{polylog} n$ parties to fix the output completely. Very recently, Haitner and Karidi-Heller [HK20] extended this result to the setting where every party takes multiple turns to speak.

Lastly, Goldwasser, Kalai, and Park [GKP15] proposed a strong adaptive adversary. That is, the adversary first sees every processor's message and, then, adaptively chooses which processor to corrupt to set its broadcast message arbitrarily. This security notion is closely related to the vertex isoperimetric inequalities [Har66] for the boolean hypercube.

2 Preliminaries

We use $\mathsf{H.M.}(a, b)$ to represent the harmonic mean of a and b, i.e., $\mathsf{H.M.}(a, b) := 2ab/(a + b)$.

▶ **Definition 1** (Asymptotic Equivalence)**.** Two functions $f(x)$ and $g(x)$ are *asymptotically*

[6] That is, to ensure the expected outcome to be either $o(1)$ or $1 - o(1)$.

equivalent, denoted by $f(x) \sim g(x)$, if the following holds.

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 1.$$

2.1 Definitions for Curves

Throughout this paper, we refer to functions of the form $f : [0,1] \to [0,1]$ as curves.

▶ **Definition 2** (L_∞ norm for curves)**.** Let C and D be two curves. We define the distance between C and D as

$$\|C - D\|_\infty := \sup_{x\in[0,1]} |C(x) - D(x)|.$$

One could verify that the L_∞ norm satisfies the triangle inequality, i.e., $\|C + D\|_\infty \leqslant \|C\|_\infty + \|D\|_\infty$.

▶ **Definition 3** (Convex upwards)**.** A curve C is said to be convex upwards if for all $x_0, x_1 \in [0,1]$ and $\alpha \in [0,1]$, we have

$$C\left(\alpha x_0 + (1-\alpha)x_1\right) \geqslant \alpha \cdot C\left(x_0\right) + (1-\alpha) \cdot C\left(x_1\right).$$

▶ **Definition 4** (Partial Ordering)**.** Let C and D be two curves on domains D_1 and D_2 respectively. Then we say $C \preceq D$, if for all $x \in D_1 \cap D_2$, we have $C(x) \leqslant D(x)$.

▶ **Definition 5** (Lipschitz Condition)**.** A curve C is said to satisfy Lipschitz condition with constant c if for any $x_0, x_1 \in [0,1]$, we have

$$|C(x_0) - C(x_1)| \leqslant c \cdot |x_0 - x_1|.$$

3 KMM Protocol

For all $n \in \mathbb{N}$ and $X \in (0,1)$, let $\Pi_{\mathsf{KMM}}(n, X)$ represent the n-processor bias-X coin-tossing protocol introduced in [KMM19]. Let $A_n(X)$ be a function with domain $[0,1]$ that *upper bounds* the insecurity of $\Pi_{\mathsf{KMM}}(n, X)$. One can define $\Pi_{\mathsf{KMM}}(n, X)$ and $A_n(X)$ inductively as follows. (We refer the readers to [KMM19] for more intuitions and details regarding this definition.)

Base Case. For $n = 1$, (essentially) the only 1-processor protocol is that the processor broadcasts messages corresponding to outcome being 1 with probability X and 0 with probability $1 - X$. Let this protocol be $\Pi_{\mathsf{KMM}}(1, X)$. One can verify that the insecurity of this protocol is upper bounded by $A_1(X) := X(1 - X)$.

Inductive Definition for $n > 1$. Assume that we already know the function $A_{n-1}(X)$ and the protocols $\Pi_{\mathsf{KMM}}(n-1, X)$, for all $X \in [0,1]$. We inductively define the value $A_n(x)$ and the protocol $\Pi_{\mathsf{KMM}}(n, x)$, for any particular $x \in [0,1]$, as follows.

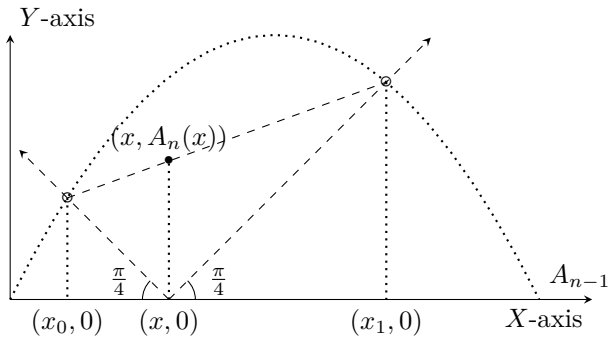

Figure 3 A pictorial summary of the definition of curve A_n. Given curve A_{n-1}, this figure shows how A_n is defined at x. Probabilities p_0 and p_1 (in the definition of protocol $\Pi_{\mathsf{KMM}}(n, x)$) are obtained by solving $p_0 + p_1 = 1$ and $p_0 x_0 + p_1 x_1 = x$.

1. Let $x_0 \in [0, x]$ be the (unique) solution of the equation $Z + A_{n-1}(Z) = x$.
2. Let $x_1 \in [x, 1]$ be the (unique) solution of the equation $Z - A_{n-1}(Z) = x$.
3. Define $A_n(x) := \mathsf{H.M.}\,(A_{n-1}(x_0), A_{n-1}(x_1))$.
4. The protocol $\Pi_{\mathsf{KMM}}(n, x)$ is defined as follows.

a. Define

$$p_0 := \frac{A_{n-1}(x_1)}{A_{n-1}(x_0) + A_{n-1}(x_1)}, \text{ and}$$
$$p_1 := \frac{A_{n-1}(x_0)}{A_{n-1}(x_0) + A_{n-1}(x_1)}.$$

b. Processor 1 either broadcasts 0 with probability p_0, or broadcasts 1 with probability p_1.
c. If the first message is 0, then the remaining processors $\{2, 3, \ldots, n\}$ implement the protocol $\Pi_{\mathsf{KMM}}(n-1, x_0)$. Otherwise, if the first message is 1, then the processors $\{2, 3, \ldots, n\}$ implement the protocol $\Pi_{\mathsf{KMM}}(n-1, x_1)$.

Figure 3 represents a pictorial summary of this definition. The geometric transformation, referred to as T, used to define $A_n(X)$ from $A_{n-1}(X)$ underlying steps 1, 2, and 3 in the protocol above is defined as follows.

▶ **Definition 6** (Geometric Transformation [KMM19])**.** The geometric transformation T takes as an input a curve C and outputs another curve $T(C)$. For any $x_0 \in [0, 1]$, $T(C)(x_0)$ is defined as follows. Let x_L be the point of intersection of the line $y = x_0 - x$ with the curve C and X_R be the point of intersection of the line $y = x - x_0$ with the curve C. Then,

$$T(C)\,(x_0) := \mathsf{H.M.}(x_L, x_R).$$

In general, geometric transformation T is applicable to any convex upwards curve C, where $C(0) = C(1) = 0$. Khorasgani et al. [KMM19] show that this transformation preserves symmetry and convexity. That is, if the original curve C is symmetric around $x = 1/2$ (resp., convex upwards), so is $T(C)$.

Given Definition 6, curve A_n is exactly curve $T\,(A_{n-1})$.

4 Our Computationally Efficient Protocol

In this section, we give a computationally efficient implementation of the KMM protocol. Note that, given the inductive definition of A_n, it remains elusive whether one can find a closed form representation of A_n.[7] Therefore, it is still open if there are computationally efficient implementations of the KMM protocol.

Our protocol $\Pi_{\mathsf{Our}}(n, X)$. In our protocol, rather than computing the curve A_n exactly, we use an appropriate approximation of it. In particular, we show how to compute an approximation curve $\widetilde{A}_{n,\delta}$, which guarantees that $\|A_n - \widetilde{A}_{n,\delta}\|_\infty \leqslant n\delta$. Note that the accuracy parameter $\delta > 0$ can be choosen arbitrarily small. And the running time of our protocol shall have a linearly dependence on $1/\delta$. Given approximate curve $\widetilde{A}_{n,\delta}$, we shall define our protocol $\Pi_{\mathsf{Our}}(n, X)$ in a similar manner as the definition of Π_{KMM}. The base case, i.e., $n = 1$, is defined identically to the KMM protocol. When $n > 1$, our protocol is defined by the following figure.

1. Let $x_0 \in [0, x]$ be the (unique) solution of the equation $Z + \widetilde{A}_{n-1,\delta}(Z) = x$.
2. Let $x_1 \in [x, 1]$ be the (unique) solution of the equation $Z - \widetilde{A}_{n-1,\delta}(Z) = x$.
3. Define

$$p_0 := \frac{\widetilde{A}_{n-1,\delta}(x_1)}{\widetilde{A}_{n-1,\delta}(x_0) + \widetilde{A}_{n-1,\delta}(x_1)}, \text{ and}$$
$$p_1 := 1 - p_0.$$

4. The protocol $\Pi_{\mathsf{Our}}(n, x, \delta)$ is defined as follows:
  a. Processor 1 either broadcasts 0 with probability p_0, or broadcasts 1 with probability p_1.
  b. If the first message is 0, then the remaining processors $\{2, 3, \ldots, n\}$ implement the protocol $\Pi_{\mathsf{Our}}(n - 1, x_0, \delta)$.
  c. Otherwise, if the first message is 1, then the processors $\{2, 3, \ldots, n\}$ implement the protocol $\Pi_{\mathsf{Our}}(n - 1, x_1, \delta)$.

Approximate curve $\widetilde{A}_{n,\delta}$. For simplicity, assume $1/\delta \in \mathbb{N}$. In our protocol, we approximate the curve $A_n(X)$ using a piece-wise linear curve $\widetilde{A}_{n,\delta}(X)$. The end points of the piece-wise linear curve $\widetilde{A}_{n,\delta}(X)$ is always defined on $x = 0, \delta, \ldots, i \cdot \delta, \ldots, 1$. Therefore, curve $\widetilde{A}_{n,\delta}(X)$ is uniquely determined by points $\left(i\delta,\ \widetilde{A}_{n,\delta}(i \cdot \delta)\right)$, for $i = 0, 1, \ldots, 1/\delta$. This allows us to define the curves by samples stored as the following array

$$\widetilde{S}_{n,\delta} := \left[\widetilde{A}_{n,\delta}(0), \widetilde{A}_{n,\delta}(\delta), \widetilde{A}_{n,\delta}(2\delta), \ldots, \widetilde{A}_{n,\delta}(1)\right].$$

For the base case of $n = 1$, the curve A_1 is explicitly defined by the function $A_1(X) := X(1 - X)$. Therefore, we directly sample from A_1 and linearly interpolate them. We prove that the piece-wise linear curve $\widetilde{A}_{1,\delta}$ is a close approximation of A_1 (refer to Lemma 3). This step is

[7] Khorasgani et al. [KMM19] proved closed form upper-bound and lower-bound of A_n. However, these bounds are not tight.

implemented by invoking the subroutine Sample&Linearize (A_1).

Input: Curve C
Output: $\widetilde{C}_\delta$
for $x \in \{0, \delta, \ldots, i\delta, \ldots, 1-\delta\}$ **do**
  Out.append(linearly interpolate $C(x)$, $C(x+\delta)$)
end
return Out

Algorithm 1: Sample&Linearize (C)

Inductively, we obtain $\widetilde{A}_{n,\delta}$ from the geometric transformation of $\widetilde{A}_{n-1,\delta}$. Note that, given $\widetilde{A}_{n-1,\delta}$, it is hard to compute $T\left(\widetilde{A}_{n-1,\delta}\right)$ precisely everywhere. Therefore, we only compute $T\left(\widetilde{A}_{n-1,\delta}\right)$ precisely at $x = 0, \delta, \ldots, i \cdot \delta, \ldots, 1$. And we let the linear interpolation of these samples be $\widetilde{A}_{n,\delta}$. We prove that if $\widetilde{A}_{n-1,\delta}$ is close to A_{n-1}, $\widetilde{A}_{n,\delta}$ will also be close to A_n (refer to Lemma 2 and Lemma 3). This step is summarized in the subroutine Transform&Linearize $\left(\widetilde{A}_{n-1,\delta}\right)$.

Input: $\widetilde{A}_{n-1,\delta}$
Output: $\widetilde{A}_{n,\delta}$
for $x \in \{0, \delta, \ldots, i\delta, \ldots, 1-\delta\}$ **do**
  Solve x_0 as the solution of $Z + \widetilde{A}_{n-1,\delta}(Z) = x$
  Solve x_1 as the solution of $Z - \widetilde{A}_{n-1,\delta}(Z) = x$
  $\widetilde{A}_{n,\delta}(x) \leftarrow$ H.M. $\left(\widetilde{A}_{n-1,\delta}(x_0), \widetilde{A}_{n-1,\delta}(x_1)\right)$
end
return Sample&Linearize $\left(\widetilde{A}_{n,\delta}\right)$

Algorithm 2: Transform&Linearize $\left(\widetilde{A}_{n-1,\delta}\right)$

Overall, the following algorithm summarize how we compute $\widetilde{A}_{n,\delta}$ using subroutines Sample&Linearize $(\cdot)$ and Transform&Linearize $(\cdot)$.

Input: $n \in \mathbb{N}$, $\delta \in (0,1)$
Output: $\widetilde{A}_{n,\delta}$
$\widetilde{A}_{1,\delta} \leftarrow$ Sample&Linearize (A_1, δ)
for $i \in \{2, \ldots, n\}$ **do**
  $\widetilde{A}_{i,\delta} \leftarrow$ Transform&Linearize $\left(\widetilde{A}_{i-1,\delta}\right)$
end
return $\widetilde{A}_{n,\delta}$

Algorithm 3: Algorithm to compute $\widetilde{A}_{n,\delta}$

Main results. We prove that the insecurity of our computationally efficient protocol $\Pi_{\mathsf{Our}}(n,x)$ that uses the approximate curve $\widetilde{A}_{n,\delta}$ is close to the insecurity of Π_{KMM}. Our results are summarized in the following theorem, which states that the insecurity of our protocol is at most $n\delta$ more than that of KMM protocol.

▶ Theorem 1 Restated (Efficient Secure Coin-tossing). *For any number of processors $n \in \mathbb{N}$, bias $X \in [0,1]$, and accuracy parameter $\delta \in [0,1]$, there exists a computationally efficient n-processor bias-X coin-tossing protocol $\Pi_{\mathsf{Our}}(n, X, \delta)$ such that the following bound holds.*

$$\varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right) < \varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right) + n\delta.$$

Furthermore, the time complexity of the next-message generation of processor i, where $1 \leqslant i \leqslant n$, in the protocol $\Pi_{\mathsf{Our}}(n, X, \delta)$ is linear in $(n-i)/\delta$.

We defer the proof this theorem to Section 6. This theorem gives us the following corollary, which states that the insecurity of our protocol is equivalent to the insecurity of KMM protocol, if we set the accuracy parameter δ to be sufficiently small.

▶ **Corollary 1.** *The insecurity of our (computationally efficient) protocol is asymptotically equivalent to* KMM *protocol when the accuracy parameter δ is set to $o\left(\frac{X(1-X)}{n\cdot\sqrt{n}}\right)$. Quantitatively, the following holds.*

$$\varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right) \sim \varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right).$$

The above follows by the lower bounds proved in Khorasgani et al. [KMM19]: $\varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right) = \Omega\left(\frac{X(1-X)}{\sqrt{n}}\right)$. Therefore, if we set the accuracy parameter δ to be $o\left(\frac{X(1-X)}{n\cdot\sqrt{n}}\right)$, the insecurity of our (computationally efficient) protocol is bounded by $\varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right) + o\left(\frac{X(1-X)}{\sqrt{n}}\right) = \varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right) + o(\varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right))$. That is, $\varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right) \sim \varepsilon\left(\Pi_{\mathsf{KMM}}(n, X)\right)$.

5 Useful Lemmas

In this section, we summarize some useful properties on the geometric transformation and the curves A_n and $\widetilde{A}_{n,\delta}$. This shall be helpful for the proof of the main theorem.

Firstly, [KMM19] proves that if a curve is "nice", then the transformation of this curve is also "nice". In particular, we have the following definition and imported lemma.

▶ **Definition 7** (Nice Curves). A curve C on $[0,1]$ is said to be *nice* if it satisfies (1) C is convex upwards on $[0,1]$; (2) C is symmetric along $x = 1/2$ axis; (3) Points $(0,0)$ and $(1,0)$ lie on C; and (4) C satisfies the Lipschitz condition with constant 1, i.e., for any $x_0, x_1 \in [0,1]$,

$$|C(x_0) - C(x_1)| \leqslant |x_0 - x_1|.$$

▶ **Imported Lemma 1.** *[KMM19] If C is a nice curve, $T(C)$ is also nice.*

Follows easily from this imported lemma, all the curves that we are interested in are nice curves.

▶ **Lemma 1** (The curves we consider are nice). *For any $n \in \mathbb{N}$ and $\delta \in (0,1)$ such that $1/\delta \in \mathbb{N}$, A_n and $\widetilde{A}_{n,\delta}$ are nice.*

Proof of Lemma 1. Trivially, $A_1 := X(1-X)$ is a nice curve. Hence, by Imported Lemma 1, $A_n = T^{n-1}(A_1)$ is also nice. Furthermore, if any curve C is nice, one can easily verify that $\mathsf{Sample\&Linearize}\left(C, \delta\right)$ is also nice. Therefore, one can inductively prove that $\widetilde{A}_{n,\delta}$ is nice since

$$\widetilde{A}_{n,\delta} = \mathsf{Sample\&Linearize}\left(T\left(\widetilde{A}_{n-1,\delta}\right), \delta\right). \qquad \blacktriangleleft$$

Our next lemma states that the transformation preserves the *partial ordering* and also the *closeness* of curves.

▶ **Lemma 2** (Transformations preserves partial ordering and closeness). *For any two nice curves C and D such that $C \preceq D$, we have*

$$T(C) \preceq T(D),$$

and

$$\|T(C) - T(D)\|_\infty \leqslant \|C - D\|_\infty.$$

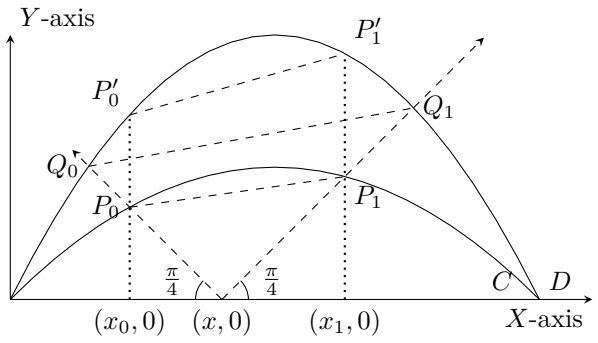

Figure 4 A pictorial summary of the proof of Lemma 2.

Proof of Lemma 2. As shown in Figure 4, consider a point $x \in [0, 1]$. Let P_0 and P_1 be the points of intersection of the curve C with lines $Y = x - X$ and $Y = X - x$. Let Q_0 and Q_1 be the corresponding points for curve D. Let P_0' be the point of intersection of the line $X = x_0$ and curve D. Similarly, let P_1' be the point of intersection of the line $X = x_1$ and curve D. We will complete the proof by proving that segment Q_0Q_1 passes through the trapezium $P_0P_1P_1'P_0'$. By concavity and transitivity, we have $Q_0Q_1 \preceq Q_0P_1' \preceq Q_0P_0'$. This implies, $Q_0Q_1 \preceq P_0'P_1'$, i.e., Q_0Q_1 lies below $P_0'P_1'$.

Consider the convex figure formed by the two rays at $(x, 0)$. By convexity, secant Q_0Q_1 lies above P_0P_1 i.e. $Q_0Q_1 \succeq P_0P_1$. Thus line Q_0Q_1 goes through the trapezium $P_0P_1P_1'P_0'$. Therefore,

$$T(C)(x_0) \leqslant T(D)(x_0).$$

Since $T(D)(x_0), T(C)(x_0)$ lie on Q_0Q_1 and P_0P_1 respectively, we get the following

$$|T(D)(x_0) - T(C)(x_0)| \leqslant \max\{P_0P_0', P_1P_1'\} \leqslant \|C - D\|_\infty.$$

The above expression holds for all $x \in [0, 1]$, thus

$$T(C) \preceq T(D),$$

and

$$\|T(D) - T(C)\|_\infty \leqslant \|C - D\|_\infty.$$ ◀

Lastly, we have the following lemma, which claims that the linear interpolation of a nice curve is very close to itself.

▶ **Lemma 3** (Linearizing of a nice curve is δ-close to itself)**.** *For any nice curve C, we have*

$$\mathsf{Sample\&Linearize}\,(C, \delta) \preceq C,$$

and

$$\|C - \mathsf{Sample\&Linearize}\,(C, \delta)\|_\infty \leqslant \delta.$$

Proof of Lemma 3. Since C is convex upwards, it is obvious that $\mathsf{Sample\&Linearize}\,(C, \delta)$ is below C.

To prove $\|C - \mathsf{Sample\&Linearize}\,(C, \delta)\|_\infty \leqslant \delta$, we use Figure 5 for intuition. Let $\widetilde{C}$ be $\mathsf{Sample\&Linearize}\,(C, \delta)$. Note that $\widetilde{C}(x) = C(x)$, when $x = k \cdot \delta$ for $k \in \{0, 1, \ldots, 1/\delta\}$.

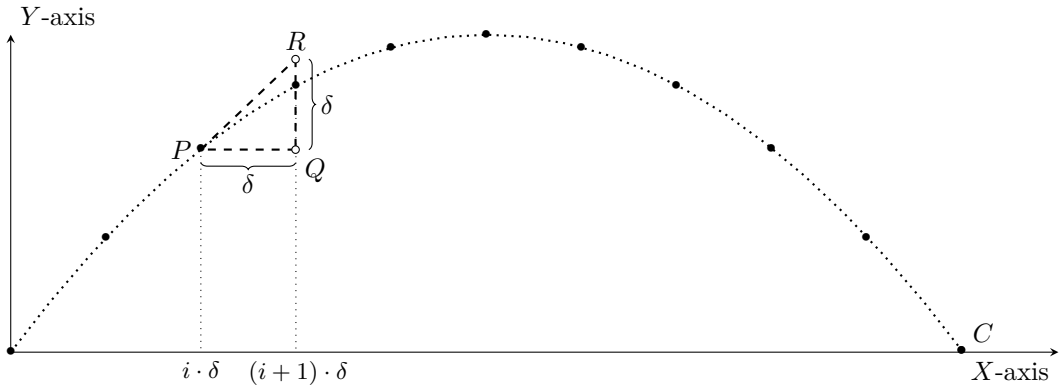

Figure 5 A pictorial summary of the proof of Lemma 3.

Consider the interval $[i\delta, (i+1)\delta]$ where $(i+1)\delta \in (0, \frac{1}{2})$. Since C is a nice curve, which implies that C is convex upwards, symmetric along $x = 1/2$ axis, and satisfies the Lipschitz condition with constant 1, C must be non-decreasing on interval $[i\delta, (i+1)\delta]$. Moreover, both curve $\widetilde{C}$ and curve C restricted to interval $[i\delta, (i+1)\delta]$ will lie within the triangle PQR (isosceles right triangle with two sides of length δ). Thus, the maximum vertical separation between them is bounded by $QR = \delta$. This holds for any index i. Thus, we get

$$\|C - \widetilde{C}\|_\infty \leqslant \delta. \quad \blacktriangleleft$$

6 Proof of Theorem 1

We are now fully equipped to prove our main theorem. The proof follows mainly from the following claims.

Firstly, we have the following claim, which states that $\widetilde{A}_{n,\delta}$ is an underestimate of A_n.

▶ **Claim 1.** $\widetilde{A}_{n,\delta} \preceq A_n$.

The next claim states that the insecurity of our protocol is close to $\widetilde{A}_{n,\delta}(X)$.

▶ **Claim 2.** *The insecurity of our protocol, i.e.,* $\varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right)$*, is upper-bounded by* $\widetilde{A}_{n,\delta}(X) + n\delta$.

Proof of Theorem 1 assuming Claim 1 and Claim 2. We have

$$\begin{aligned} \varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right) &\leqslant \widetilde{A}_{n,\delta}(X) + n\delta && \text{(Claim 2)} \\ &\leqslant A_n(X) + n\delta && \text{(Claim 1)} \\ &= \Pi_{\mathsf{KMM}}(n, X) + n\delta, \end{aligned}$$

which proves the statement on the insecurity of our protocol. Next, we discuss the running time aspect of our protocol.

Time Complexity of Our Protocol. We now show how the desired runtime is achieved. We first discuss some implementation details of the algorithms which would be crucial for optimizing the time complexity. As mentioned, *curves* in our algorithms are implicitly

piece-wise line segments such that their X-projection is of length δ. And recall that we store $\widetilde{A}_{i,\delta}$ as

$$\widetilde{S}_{i,\delta} := \left[\widetilde{A}_{i,\delta}(0), \widetilde{A}_{i,\delta}(\delta), \widetilde{A}_{i,\delta}(2\delta), \ldots, \widetilde{A}_{i,\delta}(1)\right].$$

In the algorithm $\mathsf{Transform\&Linearize}\left(\widetilde{A}_{i-1,\delta}\right)$, fix any $x = u\delta$, we compute x_0 and x_1 in the following manner. To compute x_0 one needs to identify the unique index $u_0 \in \{0, 1, \ldots, u\}$ such that

$$u_0\delta + \widetilde{S}_{i-1,\delta}[u_0] \leqslant u\delta$$
$$(u_0+1)\delta + \widetilde{S}_{i-1,\delta}[u_0+1] > u\delta$$

Once the index u_0 is identified, we obtain the value of x_0 and $\widetilde{A}_{i-1,\delta}(x_0)$ by solving a linear equation. Similarly, we can compute x_1.

For $i > 1$, the computation of u_0 corresponding to every $u \in \{0, 1, \ldots, 1/\delta\}$ can be optimized. Suppose we have an array $\mathsf{ptr}_{i-1}[u] = u_0$ establishing this mapping. Note that this mapping is *non-decreasing*. Therefore, one can compute this mapping in $\Theta(1/\delta)$ time. Processors can pre-compute the arrays $\widetilde{S}_{1,\delta}, \ldots, \widetilde{S}_{n,\delta}$ and the pointer arrays $\mathsf{ptr}_1, \ldots, \mathsf{ptr}_{n-1}$ in an offline precomputation step. Thereafter, the next-message generation takes only a constant time. ◀

Finally, we complete the proof by proving Claim 1 and Claim 2.

Proof of Claim 1. We can prove this claim by induction. For the base case, by Lemma 3,

$$\widetilde{A}_{1,\delta} := \mathsf{Sample\&Linearize}\,(A_1, \delta) \preceq A_1.$$

For the inductive step, note that

$$\begin{aligned}
& \widetilde{A}_{n,\delta} \preceq A_n \\
\implies\ & T(\widetilde{A}_{n,\delta}) \preceq T(A_n) && \text{(Lemma 2)} \\
\implies\ & \mathsf{Sample\&Linearize}\left(T(\widetilde{A}_{n,\delta}), \delta\right) \preceq T(A_n) && \text{(Lemma 3)} \\
\implies\ & \widetilde{A}_{n+1,\delta} \preceq A_{n+1} && \text{(By definition)}
\end{aligned}$$

This completes the proof. ◀

Proof of Claim 2. We prove this claim by induction on n.

For the base case, i.e., $n = 1$, one can verify that the insecurity of our protocol is $\varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right) = X(1-X) = A_1(X)$. By Lemma 3, we have

$$\|A_1 - \widetilde{A}_{1,\delta}\|_\infty = \|A_1 - \mathsf{Sample\&Linearize}\,(A_1, \delta)\|_\infty \leqslant \delta.$$

Hence, $\varepsilon\left(\Pi_{\mathsf{Our}}(n, X, \delta)\right) \leqslant \widetilde{A}_{1,\delta}(X) + \delta$.

Next, we prove the inductive step. Consider our protocol with $n+1$ processors and expected output X. Let $x_0 \in [0, 1]$ satisfy

$$X - x_0 = \widetilde{A}_{n,\delta}(x_0)$$

and $x_1 \in [0, 1]$ satisfy

$$x_1 - X = \widetilde{A}_{n,\delta}(x_1).$$

If the first message is 0, then the remaining protocol becomes our protocol with n processors and expected output x_0. Similarly, if the first message is 1, then the remaining protocol becomes our protocol with n processors and expected output x_1. Conditioned on first message being 0, if the adversary restarts the first processor upon seeing the first message 0, this causes a deviation of $X - x_0$, or identically, $\widetilde{A}_{n,\delta}(x_0)$. Or the adversary will continue on the first message. In this case, by our inductive hypothesis, in the remaining protocol the adversary can cause a deviation of at most $\widetilde{A}_{n,\delta}(x_0) + n\delta$. To summarize, when the first message is 0, the adversary can deviate the protocol by at most $\widetilde{A}_{n,\delta}(x_0) + n\delta$. Analogously, when the first message is 1, the adversary can deviate the protocol by at most $\widetilde{A}_{n,\delta}(x_1) + n\delta$. Therefore, the insecurity of our protocol is upper-bounded by the convex combination of $\widetilde{A}_{n,\delta}(x_0) + n\delta$ and $\widetilde{A}_{n,\delta}(x_1) + n\delta$, which is

$$\frac{x_1 - X}{x_1 - x_0} \cdot \left(\widetilde{A}_{n,\delta}(x_0) + n\delta\right) + \frac{X - x_0}{x_1 - x_0} \cdot \left(\widetilde{A}_{n,\delta}(x_1) + n\delta\right) = T\left(\widetilde{A}_{n,\delta}\right)(X) + n\delta.$$

By Lemma 3, we have

$$T\left(\widetilde{A}_{n,\delta}\right)(X) \leqslant \widetilde{A}_{n+1,\delta}(X) + \delta.$$

Therefore,

$$\varepsilon\left(\Pi_{\mathsf{Our}}(n+1, X, \delta)\right) \leqslant \left(\widetilde{A}_{n+1,\delta}(X) + \delta\right) + n\delta = \widetilde{A}_{n+1,\delta}(X) + (n+1) \cdot \delta.$$

This completes the proof of the inductive step and hence the entire proof. ◀

References

ABC+85 Baruch Awerbuch, Manuel Blum, Benny Chor, Shafi Goldwasser, and Silvio Micali. How to implement bracha's o (log n) byzantine agreement algorithm. *Unpublished manuscript*, 1985. 3, 4, 6

AL93 Miklós Ajtai and Nathan Linial. The influence of large coalitions. *Combinatorica*, 13(2):129–145, 1993. 7

BHMO18 Amos Beimel, Iftach Haitner, Nikolaos Makriyannis, and Eran Omri. Tighter bounds on multi-party coin flipping via augmented weak martingales and differentially private sampling. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 838–849, Paris, France, October 7–9, 2018. IEEE Computer Society Press. doi:10.1109/FOCS.2018.00084. 3

BKK+92 Jean Bourgain, Jeff Kahn, Gil Kalai, Yitzhak Katznelson, and Nathan Linial. The influence of variables in product spaces. *Israel Journal of Mathematics*, 77(1-2):55–64, 1992. 7

BL89 Michael Ben-Or and Nathan Linial. Collective coin flipping. *Advances in Computing Research*, 1989. 3

Blu83 Manuel Blum. How to exchange (secret) keys (extended abstract). In *15th Annual ACM Symposium on Theory of Computing*, pages 440–447, Boston, MA, USA, April 25–27, 1983. ACM Press. doi:10.1145/800061.808775. 3, 4, 6

CI93 Richard Cleve and Russell Impagliazzo. Martingales, collective coin flipping and discrete control processes (extended abstract). 1993. 3, 6

Cle86 Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th Annual ACM Symposium on Theory of Computing*, pages 364–369, Berkeley, CA, USA, May 28–30, 1986. ACM Press. doi:10.1145/12130.12168. 3, 4, 6

CZ16 Eshan Chattopadhyay and David Zuckerman. Explicit two-source extractors and resilient functions. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM Symposium on Theory of Computing*, pages 670–683, Cambridge, MA, USA, June 18–21, 2016. ACM Press. doi:10.1145/2897518.2897528. 7

Fei99 Uriel Feige. Noncryptographic selection protocols. In *40th Annual Symposium on Foundations of Computer Science*, pages 142–153, New York, NY, USA, October 17–19, 1999. IEEE Computer Society Press. doi:10.1109/SFFCS.1999.814586. 3, 7

FHH+19 Yuval Filmus, Lianna Hambardzumyan, Hamed Hatami, Pooya Hatami, and David Zuckerman. Biasing Boolean functions and collective coin-flipping protocols over arbitrary product distributions. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *ICALP 2019: 46th International Colloquium on Automata, Languages and Programming*, volume 132 of *LIPIcs*, pages 58:1–58:13, Patras, Greece, July 9–12, 2019. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.58. 7

GGL91 Oded Goldreich, Shafi Goldwasser, and Nathan Linial. Fault-tolerant computation in the full information model (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science*, pages 447–457, San Juan, Puerto Rico, October 1–4, 1991. IEEE Computer Society Press. doi:10.1109/SFCS.1991.185405. 5

GKP15 Shafi Goldwasser, Yael Tauman Kalai, and Sunoo Park. Adaptively secure coin-flipping, revisited. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP 2015: 42nd International Colloquium on Automata, Languages and Programming, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 663–674, Kyoto, Japan, July 6–10, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-47666-6_53. 3, 7

GVZ06 Ronen Gradwohl, Salil Vadhan, and David Zuckerman. Random selection with an adversarial majority. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 409–426, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany. doi:10.1007/11818175_25. 5

Har66 Lawrence H Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, 1(3):385–393, 1966. 7

HK20 Iftach Haitner and Yonatan Karidi-Heller. A tight lower bound on adaptively secure full-information coin flip. *FOCS*, 2020. 7

KKL88 Jeff Kahn, Gil Kalai, and Nathan Linial. The influence of variables on Boolean functions (extended abstract). In *29th Annual Symposium on Foundations of Computer Science*, pages 68–80, White Plains, NY, USA, October 24–26, 1988. IEEE Computer Society Press. doi:10.1109/SFCS.1988.21923. 3, 7

KMM19 Hamidreza Amini Khorasgani, Hemanta K. Maji, and Tamalika Mukherjee. Estimating gaps in martingales and applications to coin-tossing: Constructions and hardness. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 333–355, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-36033-7_13. 3, 4, 6, 8, 9, 10, 12

LLS89 David Lichtenstein, Nathan Linial, and Michael Saks. Some extremal problems arising from discrete control processes. *Combinatorica*, 9(3):269–287, 1989. 3, 7

Sak89 Michael E. Saks. A robust noncryptographic protocol for collective coin flipping. *SIAM J. Discrete Math.*, 2(2):240–244, 1989. 7

SV05 Saurabh Sanghvi and Salil P. Vadhan. The round complexity of two-party random selection. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 338–347, Baltimore, MA, USA, May 22–24, 2005. ACM Press. doi:10.1145/1060590.1060641. 5

TKKR18 Yael Tauman Kalai, Ilan Komargodski, and Ran Raz. A lower bound for adaptively-secure collective coin-flipping protocols. In *32nd International Symposium on Distributed Computing (DISC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. 7