On Time and Order in Multiparty Computation

Pablo Azar[1], Shafi Goldwasser[2], and Sunoo Park[1]

[1]MIT
[2]MIT and the Weizmann Institute of Science

Abstract

The availability of vast amounts of data is changing how we can make medical discoveries, predict global market trends, save energy, improve our infrastructures, and develop new educational strategies. One obstacle to this revolution is the willingness of different entities to share their data with others.

The theory of secure multiparty computation (MPC) seemingly addresses this problem in the best way possible. Namely, parties learn the minimum necessary information: the value of the function computed on their joint data and nothing else. However, the theory of MPC does not deal with an important aspect: *when* do different players receive their output? In time-sensitive applications, the timing and order of output discovery may be another important deciding factor in whether parties choose to share their data via the MPC.

In this work, we incorporate time and order of output delivery into the theory of MPC. We first extend classical MPC to *ordered MPC* where different players receive their outputs according to an order which in itself is computed on the inputs to the protocol, and to refine the classical notions of guaranteed output delivery and fairness to require instead *ordered output delivery* and *prefix-fairness*. We then define *timed-delay MPCs* where explicit time delays are introduced into the output delivery schedule. We show general completeness theorems for ordered MPCs and timed-delay MPCs. We also introduce a new primitive called *time-line puzzles*, which are a natural extension of classical timed-release crypto, in which multiple events can be serialized in time.

Next, we show how ordered MPC can give rise to MPCs which are provably "worth" joining, in competitive settings where relative time of output discovery may deter parties from joining the protocol. We formalize a model of collaboration and design a mechanism in which n self-interested parties can decide, based on their inputs, on an ordering of output delivery and a distribution of outputs to be delivered in the mandated order. The mechanism guarantees a higher reward *for all participants* when joining an ordered MPC or declares that such a guarantee is impossible to achieve. We show a polynomial time algorithm to compute the mechanism for a range of model settings.

1 Introduction

The availability of vast amounts of data is changing how we can make medical discoveries, predict global market trends, save energy, improve our infrastructures, and develop new educational strategies. When parts of the data are held by disjoint and *competing* parties, such as companies competing for market share or research laboratories competing for scientific credit, the parties face a clear dilemma – *is it better to share data or proceed independently of each other?*

The theory of multiparty computation (MPC) sheds little light on this quandary. Although MPC makes it possible to compute a function on the union of many parties' data without having to disclose the data fully, it gives no clue as to whether it is in the parties' interest to compute the function in the first place: that is, whether the gain that a party A makes by learning the function value outweighs the loss incurred by a competitor party learning the function value (based on A's data) as well.

In this paper, we incorporate an important new aspect into the theory of MPC: the dimension of time and order of output delivery. We then show an application of the resulting theory to mechanisms for collaboration which are provably "worth" joining for all players, in a formal sense that is captured in a model of data-centered scientific collaboration.

We chose to focus on the order of output delivery as a crucial aspect of our data-sharing mechanisms, since we believe that this is an important and primarily unaddressed issue in data-based collaborations. For example, in the scientific research community, data sharing can translate to losing a prior publication date. An interesting real-life case is the 2003 Fort Lauderdale meeting on large-scale biological research [Wel03], in which the participating researchers recognized that "pre-publication data release can promote the best interests of [the field of genomics]" but "might conflict with a fundamental scientific incentive – publishing the first analysis of one's own data". Researchers at the meeting agreed to adopt a set of principles by which although data is shared upon discovery, researchers hold off publication until the original holder of the data has published a first analysis. Being a close-knit community in which reputation is key, this was a viable agreement. However, more generally, they state that "incentives should be developed by the scientific community to support the voluntary release of [all sorts of] data prior to publication".

We proceed with an informal description of our results on time and order in MPC in section 1.1, then give an overview of our model and results in the setting of data-centered collaborations in section 1.2.

1.1 Extending MPC and fairness notions to incorporate order and time

In the MPC landscape, fairness is the one notion that addresses the idea that either all parties participating in an MPC should benefit, or none should. The thesis of our paper is that *fairness is not enough*. In certain settings, the ordering of when outputs are delivered may be just as important.

1.1.1 Ordered MPC

We extend the classical notion of MPC to *ordered MPC*, where players receive their outputs in order which can depend on the inputs to the protocol: that is, the outputs are delivered to the n players according to a particular permutation $\pi : [n] \to [n]$, which is computed during the MPC and may itself be a function of the players' inputs. In the latter case, the "ordering function" can be considered to be a parameter to the MPC along with the functionality to be computed.

We refine the classical notions of guaranteed output delivery and fairness to require instead *ordered output delivery* and *prefix-fairness*, where either no players receive an output, or those who do strictly belong to a prefix of the mandated order. We adapt the standard privacy requirement for the ordered MPC setting: we require that no player learns more information about the other players' inputs than can be learned from his own input and output[1], *and his own position in the output delivery order*. We show that ordered output delivery can be achieved when the faulty players

[1]This is formalized in the usual simulation-based paradigm.

are in minority, and prefix-fairness can always be achieved, regardless of the number of faulty players under the same assumptions underlying the classical MPC constructions (such as [GMW87]).

Definition (informal). A protocol Π achieves *ordered output delivery* if $\forall i, j$ such that $\pi(i) < \pi(j)$, Π delivers output to player i before player j, where π is the permutation induced by the players' inputs. We say that Π achieves *prefix-fairness* if $\forall i, j$ such that $\pi(i) < \pi(j)$, if player j receives his output then so will player i.

Fairness in MPC requires that either all or none of the players receive outputs. Without honest majority, fairness cannot be achieved in general [Cle86]. The situation with prefix-fairness is different: it can be achieved for any number of honest players under standard (computational MPC) assumptions.

Theorem (informal). Given any standard MPC protocol (such as [GMW87]), there is a secure ordered MPC protocol that achieves ordered output delivery for honest majority, and is prefix-fair even if the honest players are in minority.

1.1.2 Proportional fairness

Interestingly, using an ordered MPC protocol as a subroutine, we achieve a probabilistic guarantee on fairness in (traditional) MPC that "scales" with the number of faulty players, which we name *proportional fairness*. Essentially, proportional fairness has a probabilistic guarantee of fairness that "degrades gracefully" as the number of malicious parties t grows, as follows.

Definition (informal). Let n be the number of parties, and let Fair denote the event that either all parties receive their outputs before protocol termination, or no parties receive their outputs before protocol termination. For any PPT adversary $\mathcal{A}$ that corrupts up to t parties, we say that a protocol achieves *proportional fairness* if for $t < n/2$, $\Pr[\mathsf{Fair}] = 1$ and for $t \geq n/2, \Pr[\mathsf{Fair}] \geq (n-t)/n$.

Theorem (informal). Given any standard MPC protocol (such as [GMW87]), there is an MPC protocol that securely computes any function correctly and privately and is proportionally fair for any number of faulty players.

1.1.3 Timed-delay MPC

Analyzing more closely the question, discussed earlier, of whether collaboration is worthwhile, it becomes clear that we may sometimes need not only ordering of the MPC outputs, but also to introduce a *delay* between deliveries. This can enable players to have time to reap the advantage of receiving an output earlier than other players. We define *timed-delay MPC* as an ordered MPC with the additional guarantee that in between players receiving their outputs, there will be a delay in time, and we construct protocols to realize this MPC definition.

Theorem (informal). Given any MPC protocol secure in the presence of honest majority, there is a timed-delay MPC protocol that securely computes any function with fairness in the presence of an honest majority.

The above theorem makes no additional assumptions to achieve time-delay, since it essentially enforces time passage by adding *empty* message exchange rounds. The drawback of this is that all (honest) parties continue interacting in the protocol, until the last party has received his output:

that is, the protocol duration is as long as the sum of the delays. It would be more desirable if parties could stop participating after they receive their own output, or even that the MPC protocol would serve as a "pre-computation phase" after which the computed data could be non-interactively released with delays.

Time-line puzzles. In order to achieve this feature, we turn to "timed-release crypto" which was introduced by [May93] and constructed first by Rivest, Shamir and Wagner [RSW96]. Informally, a time-lock puzzle is a primitive which allows "locking" of data, such that it will only be released after a certain time delay. We extend the concept of time-lock puzzles to a novel primitive which we call a *time-line puzzle*, which allows locking of multiple data items with different delays into a single puzzle. These can be useful for locking together many data items with different delays for a single recipient, or for locking data items with different delays for a group of recipients. In the latter case, an assumption is required on the similarity of computing power of different parties.

Using time-line puzzles, we construct a second timed-delay MPC protocol, which (as above) achieves security and fairness for honest majority, and has the additional feature that the protocol duration essentially does not depend on the delay lengths: the players' interaction terminates even before the first player gets his output.

Theorem (informal). Given a secure time-line puzzle and any MPC protocol secure in the presence of honest majority, and assuming that the differences in parties' computing power are bounded (pairwise) by a logarithmic factor, there is a timed-delay MPC protocol that securely computes any function with complete fairness in the presence of an honest majority.

Time-line puzzles can be implemented either using the modular exponentiation function in a way quite similar to [RSW96] (which requires making the same computational assumptions as they do, about inherent sequentiality of the modular squaring function), or using the black-box assumption that inherently sequential hash functions exist (i.e. the random oracle assumption) as suggested by Mahmoody, Moran and Vadhan [MMV13]. Furthermore, it was recently shown by [Bit+] how to build time-lock puzzles based on indistinguishability obfuscation and a complexity-theoretic assumption on the existence of worst-case non-parallelizing languages, and their construction is extensible to time-lines.

1.2 A model for data-centered collaborations

We present a formal model of data-sharing collaborations in which n self-interested players can jointly decide whether they are "better off" collaborating via an MPC or not. Our model setup assumes an underlying metric in which the quality of an "outside option" (that is, parties' utility when they work on their own and do not collaborate) and the quality of players' outputs (when they do collaborate via MPC) can be compared.

We present our results using the terminology of *scientific* collaborations and publications, but we emphasize that the importance of "beating a competitor to the punch" is certainly not exclusive to this domain. It also exists, say, in financial enterprises where timing of investments and stock trading can translate to large financial gains or losses.

Our model captures the following stylized features of scientific collaboration.

- Different participants may share data, but each will learn a different result.
- Results will be delivered in order and will improve on each other.

- Earlier output delivery can be worth more. Thus, players may have to agree in which order they want to receive their output.
- Groups can learn from other groups' publications. Thus, data-sharing protocols must take into account that information leaks as publications appear.

We define and design mechanisms for sharing data in our model. Suppose that each research group i has a dataset x_i, and the ideal outcome of a collaboration is learning the output of a function $f(x_1, ..., x_n)$. Let $y = f(x_1, ..., x_n)$. The output of the mechanism on players' inputs $x_1, ..., x_n$ will consist of: an ordering in which the players should receive their outputs, and a sequence of random variables (corresponding to players' outputs) $y_1, ..., y_n$ such that $\mathbb{E}[y_1] = \mathbb{E}[y_2] = ... = \mathbb{E}[y_n] = f(x_1., , , x_n)$. Since n results will be published, and results must strictly improve on each other, the data-sharing mechanism that the players agree on cannot simply output y to all the players, since then the first research group to obtain y would simply publish it, and pre-empt all future publications. Thus, any data-sharing mechanism must reveal the information slowly and in order. The mechanism's goal is to either guarantee a higher reward *for all participants* when joining an ordered (or timed-delay) MPC, or declare that such a guarantee is impossible to achieve.

The reason why players may choose not to collaborate is that they may think they might get more credit by publishing the results they can get on their own. That is, player i has some constant value α_i that he assigns to not collaborating, In the case of collaboration, he gets a reward $R_i(\sigma, y_1, ..., y_n)$ which may depend not only on the assigned estimates $y_1, ..., y_n$ but also on the order of publication σ. Player i will collaborate if and only if $R_i(\sigma, y_1, ..., y_n) \geq \alpha_i$. When each player wants to collaborate as long as all other players are collaborating, we say that a *collaborative Nash equilibrium* exists.[2]

One delicate issue with such a protocol is that, when the protocol reveals y_i to player i, she is not necessarily forced to publish result y_i. Instead, she may publish a result z_i with variance $Var(z_i) < Var(y_i)$ (that is, a "better" result) if she has extra information in addition to y_i. This extra information may leak to a future player j, inducing j to incorporate that into her own publication, etc.

In order to give meaningful results, we bound the amount of information players can learn through these side channels. The way we bound the information will make a big difference on whether sharing data in our model is computationally efficient or intractable. Informally, if a data-sharing protocol gives player i an estimate y_i with variance $Var(y_i)$, then player i may publish any result z_i with variance in the interval $[Var(y_i) - \lambda, Var(y_i)]$, where λ is a *learning bound* that may depend on previous players' publications z_j (assuming player j gets her result before i) as well as the order in which these results were revealed.

We show two main results in our model. The first theorem shows that, when the amount that player i learns from player j's publication is bounded by some number μ_j *which only depends on* j *and not* i, then there exists a feasible data-sharing mechanism.

Theorem (informal). Assume that when player j publishes her result, all players i who have not published yet can reduce the variance of their own estimate by at most μ_j. Then *there exists*

[2]Note that, in our model, the players only have two actions, collaboration or non-collaboration. We implicitly assume that players cannot lie about their data x_i once they have chosen to collaborate, since this data can be verified by peer review. Furthermore, the payoffs α_i from not collaborating are known, and can model how "good" the data of team i is. A higher α_i corresponds to better individual data, which allows for better results from not collaboration. This allows data sharing mechanisms to give greater credit to players with better data (as measured by α_i) since otherwise these players will not collaborate.

a polynomial time data-sharing mechanism that outputs an ordering of players σ and estimates $(y_1, ..., y_n)$ such that collaboration is a Nash equilibrium. If no such ordering and estimates exist, then the mechanism outputs `NONE`.

The algorithm for sharing data is non-trivial, and uses the fact that minimum weight perfect matchings can be found in polynomial time [Edm65]. In this result, we assume there is a bound μ_j on the amount that *any* player can learn from player j's publication. It may seem reasonable to make this bound depend also on the "learner" i – however, it turns out that this additional complexity makes collaboration intractable.

Theorem (informal). Assume that when player j publishes her result, all players i who have not published yet can reduce the variance of their own estimates by at most μ_{ij}. *Then the problem of deciding whether there exists a collaborative Nash equilibrium is NP-complete.*

We formally describe our data-sharing model and results in section 4. We conclude the introduction with a couple of remarks.

Example. As mentioned earlier, there are other settings besides scientific collaboration, in which ordering of MPC outputs makes a difference. Another simple application where players may want to reveal the outputs in a predetermined order is collaboration between companies. Assume that we have n companies seeking to develop a new medical treatment. Concretely, if each company's data is x_i, then the new treatment is $y = f(x_1, \ldots, x_n)$ where $f(\cdot, \ldots, \cdot)$ is a function known to all the parties.

Why would the companies want to reveal a partial answer y_i at a different period in time to each party? One possibility is that each company i contributes a different amount to the discovery of the drug. Thus, companies that contribute "more" will agree to collaborate only if they can get some priority in the use of the new treatment. The companies with the largest contributions will be able to have a monopoly on the treatment (and, after multiple firms receive their y_i, an oligopoly).

Remark. We recognize that in a collaboration, different entities may be asymmetric in the amount, type, or quality of data they hold[3]. Instead of imposing the restriction of trying to "reward parties based on the quality of their contribution", our mechanism guarantees that each player in an MPC will receive a reward which improves on the reward he can obtain without participating.

1.3 Other related work

The problem of how to make progress in a scientific community has been studied in other contexts. Banerjee, Goel and Krishnaswamy [BGK14] consider the problem of partial progress sharing, where a scientific task is modeled as a directed acyclic graph of subtasks. Their goal is to minimize the time for all tasks to be completed by selfish agents who may not wish to share partial progress.

Kleinberg and Oren [KO11] study a model where researchers have different projects to choose from, and can work on at most one. Each researcher i has a certain probability of being able to solve a problem j, and she gets a reward w_j if she is the only person to solve it. If multiple researchers solve the problem, they study how to split the reward in a socially optimal way. They show that assigning credit asymmetrically can be socially optimal when researchers seek to maximize individual reward,

[3]For example, a hospital with more patients will clearly have more patient data than a small facility, and yet access to data of small but homogenous or rare communities can at times be more valuable than access to larger heterogenous sets of data.

and they suggest implementing a "Matthew Effect", where researchers who are already credit-rich should be allocated more credit than in an even-split system. Interestingly, this is coherent with the results of our paper, where it is socially optimal to obfuscate data so that researchers who are already "ahead" (in terms of data), end up "ahead" in terms of credit.

Cai, Daskalakis and Papadimitriou [CDP14] study the problem of incentivizing n players to share data, in order to compute a statistical estimator. Their goal is to minimize the sum of rewards made to the players, as well as the statistical error of their estimator. In contrast, our goal is to give a decentralized mechanism through which players can pool their data, and distribute partial information to themselves in order so as to increase the utility of every collaborating player.

Boneh and Naor [BN00] construct timed commitments that can be "forced open" after a certain time delay, and discuss applications of their timed commitments to achieve fair two-party contract signing (and coin-flipping) under certain timing assumptions including bounded network delay and the [RSW96] assumption about sequentiality of modular exponentiation.

2 Ordered MPC and Notions of Fairness

We introduce formal definitions of ordered MPC and associated notions of fairness and ordered output delivery, and give protocols that realize these notions.

Notation. For a finite set A, we will write $a \leftarrow A$ to denote that a is drawn uniformly at random from A. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. The operation $\oplus$ stands for exclusive-or. The relation $\overset{c}{\approx}$ denotes computational indistinguishability. $\mathsf{negl}(n)$ denotes a negligible function in n, and $\mathsf{poly}(n)$ denotes a polynomial in n. An *efficient* algorithm is one which runs in probabilistic polynomial time (PPT). $\circ$ denotes function composition, and for a function f, we write f^t to denote $\underbrace{f \circ f \circ \cdots \circ f}_{t}$.

Throughout this work, we consider computationally bounded (rushing) adversaries in a synchronous complete network, and we assume the players are honest-but-curious, since any protocol secure in the presence of honest-but-curious players can be transformed into a protocol secure against malicious players [GMW87].

2.1 Definitions

Let f be an arbitrary n-ary function and p be an n-ary function that outputs permutation $[n] \to [n]$. An ordered MPC protocol is executed by n parties, where each party $i \in [n]$ has a private input $x_i \in \{0,1\}^*$, who wish to securely compute $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \in (\{0,1\}^*)^n$ where y_i is the output of party i. Moreover, the parties are to receive their outputs in a particular *ordering* dictated by $p(x_1, \ldots, x_n) = \pi \in ([n] \to [n])$. That is, for all $i < j$, party $\pi(i)$ must receive his output *before* party $\pi(j)$ receives her output.

Following [GMW87], the security of ordered MPC with respect to a functionality f and permutation function p is defined by comparing the execution of a protocol to an ideal process $\mathcal{F}_{\text{Ordered-MPC}}$ where the outputs and ordering are computed by a trusted party who sees all the inputs. An ordered MPC protocol F is considered to be secure if for any real-world adversary $\mathcal{A}$ attacking the real protocol F, there exists an ideal adversary $\mathcal{S}$ in the ideal process whose outputs (views) are indistinguishable from those of $\mathcal{A}$. Note that this implies that no player learns more information

about the other players' inputs than can be learned from his own input and output, *and his own position in the output delivery order.*

IDEAL FUNCTIONALITY $\mathcal{F}_{\text{Ordered-MPC}}$

In the ideal model, a trusted third party T is given the inputs, computes the functions f, p on the inputs, and outputs to each player i his output y_i in the order prescribed by the ordering function. In addition, we model an ideal process adversary $\mathcal{S}$ who attacks the protocol by corrupting players in the ideal setting.

Public parameters. $\kappa \in \mathbb{N}$, the security parameter; $n \in \mathbb{N}$, the number of parties; $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, the function to be computed; and $p : (\{0,1\}^*)^n \to ([n] \to [n])$, the ordering function.

Private parameters. Each player $i \in [n]$ holds a private input $x_i \in \{0,1\}^*$.

1. INPUT. Each player i sends his input x_i to T.
2. COMPUTATION. T computes $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and $\pi = p(x_1, \ldots, x_n)$.
3. OUTPUT. The output proceeds in n sequential rounds. At the start of the j^{th} round, T sends the output value $y_{\pi(j)}$ to party $\pi(j)$, and sends the message $\perp$ to all other parties. When party $\pi(j)$ receives his output, he responds to T with the message ack. (The players who receive $\perp$ are not expected to respond.) Upon receipt of the ack, T proceeds to the $(j+1)^{th}$ round – or, if $j = n$, then the protocol terminates.
4. OUTPUT OF VIEWS. At each sequential round, after receiving his message from T, each party involved outputs a *view* of the computation so far. Each uncorrupted party i outputs y_i if he has already received his output, or $\perp$ if he has not. Each corrupted party outputs $\perp$. Additionally, the adversary $\mathcal{S}$ outputs an arbitrary function of the information that he has learned during the execution of the ideal protocol, which comprises all the messages seen by the corrupt players, and the outputs that they received.

   Let the view outputted by party i in the j^{th} round be denoted by $\mathscr{V}_{i,j}$, and let the view outputted by $\mathcal{S}$ in the j^{th} round be denoted by $\mathscr{V}_{\mathcal{S},j}$. Let $\mathscr{V}^{\mathsf{ideal}}_{\text{Ordered-MPC}}$ denote the collection of all views for all stages:

$$\mathscr{V}^{\mathsf{ideal}}_{\text{Ordered-MPC}} = ((\mathscr{V}_{\mathcal{S},1}, \mathscr{V}_{1,1}, \ldots, \mathscr{V}_{n,1}), \ldots, (\mathscr{V}_{\mathcal{S},n}, \mathscr{V}_{1,n}, \ldots, \mathscr{V}_{n,n}))\,.$$

   (If the protocol is terminated early, then views for rounds which have not yet been started are taken to be $\perp$.)

Definition 2.1 (Security)**.** *A multi-party protocol F is said to securely realize $\mathcal{F}_{\text{Ordered-MPC}}$, if the following conditions hold.*

1. *The protocol includes a description of n check-points $C_1, \ldots, C_n$ corresponding to events during the execution of the protocol.*
2. *Take any* PPT *adversary $\mathcal{A}$ attacking the protocol F by corrupting a subset of players $S \subset [n]$, which outputs an arbitrary function $V_{\mathcal{A},j}$ of the information that it has learned in the protocol execution after each check-point C_j. Let*

$$V^{\mathsf{real}}_{\mathcal{A}} = ((V_{\mathcal{A},1}, V_{1,1}, \ldots, V_{n,1}), \ldots, (V_{\mathcal{A},n}, V_{1,n}, \ldots, V_{n,n}))$$

be the tuple consisting of the adversary $\mathcal{A}$'s outputted views along with the views of the real-world parties as specified in the ideal functionality description. Then there is a PPT *ideal adversary $\mathcal{S}$ which, given access to $\mathcal{F}_{\text{Ordered-MPC}}$ and corrupting the same subset S of players, can output views $\mathcal{V}_{\mathcal{S},j}$ such that $V_{\mathcal{A}}^{\mathsf{real}} \overset{c}{\approx} \mathcal{V}_{\text{Ordered-MPC}}^{\mathsf{ideal}}$.*

In the context of ordered MPC, the standard guaranteed output delivery notion is insufficient. Instead, we define *ordered output delivery*, which requires in addition that all parties receive their outputs in the order prescribed by p.

Definition 2.2 (Ordered output delivery)**.** *An ordered MPC protocol satisfies* ordered output delivery *if for any inputs $x_1, \ldots, x_n$, functionality f, and ordering function p, it holds that all parties receive their outputs before protocol termination, and moreover, if $\pi(i) < \pi(j)$, then party i receives his output before party j receives hers, where $\pi = p(x_1, \ldots, x_n)$.*

We also define a natural relaxation of the fairness requirement for ordered MPC, called *prefix-fairness*. Although it is known that fairness is impossible for general functionalities in the presence of a dishonest majority, we show in the next subsection that prefix-fairness can be achieved even when a majority of parties are corrupt.

Definition 2.3 (Prefix-fairness)**.** *An ordered MPC protocol is* prefix-fair *if for any inputs $x_1, \ldots, x_n$, it holds that the set of parties who have received their outputs at the time of protocol termination (or abortion) is a prefix of $(\pi(1), \ldots, \pi(n))$, where $\pi = p(x_1, \ldots, x_n)$ is the permutation induced by the inputs.*

Prefix-fairness can be useful, for example, in settings where it is more important for one party to receive the output than the other; or where there is some prior knowledge about the trustworthiness of each party (so that more trustworthy parties may receive their outputs first).

2.2 Construction

Ordered MPC is achievable by standard protocols for general MPC, as described in Protocol 1. A little care must be taken during the output stage, as the ordering may be a function of the input data and thus may reveal private information.

PROTOCOL 1

Public parameters. $\kappa \in \mathbb{N}$, the security parameter; $n \in \mathbb{N}$, the number of parties; $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, the function to be computed; and $p : (\{0,1\})^* \to ([n] \to [n])$, the ordering function.

1. Using any general secure MPC protocol (such as [GMW87]), jointly compute[4] the desired function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and permutation $\pi = p(x_1, \ldots, x_n)$ on the players' inputs. Before any outputs are issued to players, proceed to step 2.

2. Each value y_i will be outputted to the corresponding player i, and the outputs will be issued in the order prescribed by permutation π, as follows. The outputs will occur in n rounds, and in the i^{th} round, player $\pi(i)$ will receive his output. Moreover, in the i^{th}

round, all players other than $\pi(i)$ will receive a "fake" output with value $\perp$. All of the values issued to party i during these output rounds will be masked by a secret random value known only to party i, so that it is not revealed to anyone but the recipient who receives his "real" output in which round.

Theorem 2.4. *Protocol 1 securely realizes $\mathcal{F}_{\text{Ordered-MPC}}$. In the case of honest majority, fairness is achieved. If half or more the parties are dishonest, then Protocol 1 achieves prefix-fairness.*

Proof. Take any adversary $\mathcal{A}$ attacking Protocol 1 by corrupting some subset $S \subset [n]$ of parties. Consider any check-point C_j (for $j \in [n]$) of the real protocol. Recall that an adversary's view may be an arbitrary function of the information that he has learned during the execution of the ideal protocol, which comprises all the messages seen by the corrupt players, and the outputs that they received. Let $\mathcal{A}_j(M)$ and $\mathcal{A}_j(M)$ denote the functions that $\mathcal{A}$ outputs after check-point C_j (and before C_{j+1}), where M is the sequence of messages the corrupt players have seen so far (including output messages).

Let M_0 denote the sequence of messages seen by the corrupt players during step 1 ("the computation phase") of Protocol 1. By the security of the underlying general MPC protocol, it holds that any arbitrary function of M_0 can be simulated by an ideal adversary $\mathcal{S}^*$. (We refer to this simulator $\mathcal{S}^*$ later in the proof.) Similarly, define $\mathscr{M}_0 = ()$ to be the (empty) sequence of messages seen by the corrupt players before any outputs are issued in the ideal protocol.

In the real execution: Let M_j denote the sequence of messages seen by corrupt players up to and including check-point C_j. By the definition of the check-point, for any $j \in [n]$, it holds that

$$M_j = \begin{cases} M_{j-1}||(y_{\pi(j)}) & \text{if } j \in S \\ M_{j-1}||(\perp) & \text{otherwise} \end{cases}, \tag{1}$$

where the $||$ operator denotes concatenation of tuples.

In the ideal execution: Let $\mathscr{M}_j$ and $\mathscr{O}_j$ be defined in the corresponding way for the ideal execution. Also with the ideal check-points, it holds that

$$\mathscr{M}_j = \begin{cases} \mathscr{M}_{j-1}||(y_{\pi(j)}) & \text{if } j \in S \\ \mathscr{M}_{j-1}||(\perp) & \text{otherwise} \end{cases}. \tag{2}$$

In order to construct an ideal-world simulator $\mathcal{S}$ for the real-world adversary $\mathcal{A}$, we define some auxiliary algorithms. For $j \in [n]$, define $\mathcal{A}'_j$ to be the following algorithm:

$$\mathcal{A}'_j(M, O) = \mathcal{A}\left(M||(z_{\pi(1)}, \ldots, z_{\pi(j)})\right), \text{ where } z_{\pi(i)} = \begin{cases} y_{\pi(i)} & \text{if } i \in S \\ \perp & \text{otherwise} \end{cases}.$$

By equation 3, for each $j \in [n]$, it holds that $M_j = M||(z_{\pi(1)}, \ldots, z_{\pi(j)})$. Hence, $\mathcal{A}'_j(M_0, O_0) = \mathcal{A}_j(M_j, O_j)$. We define a ideal adversary $\mathcal{S}$ which simulates $\mathcal{A}$ as follows. $\mathcal{S}$ simulates each view $\mathcal{A}_j(M_j, O_j)$ outputted by the real-world adversary $\mathcal{A}$ by using the simulator $\mathcal{S}^*$ to compute $\mathcal{A}'_j(M_0, O_0)$. This is possible since:

- $\mathcal{S}^*$ can simulate any function of (M_0, O_0), as remarked above; and

- $\mathcal{S}$ knows $\mathscr{M}_j$ in the ideal execution, which (by equation 4) contains exactly the $z_{\pi(1)}, \ldots, z_{\pi(j)}$ that need to be hard-wired into the function $\mathcal{A}'_j$ that is to be computed.

□

2.3 A fairness notion that scales with number of corruptions

Since standard fairness is impossible for general functionalities in the presence of a dishonest majority, it is natural to ask what relaxed notions of fairness are achievable. One such notion is the prefix-fairness which was defined above. An alternative interesting definition could require fairness to "degrade gracefully" with the number of malicious parties t: that is, the "fairness" guarantee becomes weaker as t increases. A new definition that captures this idea is given below.

Definition 2.5 (Proportional fairness)**.** *An MPC protocol is* proportionally fair *if the following two properties hold. Let n be the number of parties, and let* Fair *denote the event that either all parties receive their outputs before protocol termination, or no parties receive their outputs before protocol termination. For any* PPT *adversary $\mathcal{A}$ that corrupts up to t parties:*

1. *If $t < n/2$,* $\Pr[\mathsf{Fair}] = 1$.
2. *If $t \geq n/2$,* $\Pr[\mathsf{Fair}] \geq (n-t)/n$.

The probabilities are taken over the random coins of the honest parties and $\mathcal{A}$.

We show that using Protocol 1 as a building block, we can obtain a simple new protocol which achieves proportional fairness. The new protocol uses one-time message authentication codes (MACs) as a building block[5]. In describing Protocol 2, without loss of generality, we assume that the output to all players is the same[6].

PROTOCOL 2

Public parameters. $\kappa \in \mathbb{N}$, the security parameter; $n \in \mathbb{N}$, the number of parties; $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ a MAC scheme; and $f : (\{0,1\}^*)^n \to \{0,1\}^*$, the function to be computed.

1. Each party P_i provides as input: his input value $x_i \in \{0,1\}^*$, a signing key $k_i \leftarrow \mathsf{Gen}(1^\kappa)$, and a random string $r_i \leftarrow \{0,1\}^{n^2}$. Let $r = r_1 \oplus \cdots \oplus r_n$. The random string r is used to sample[7]a random permutation $\pi \in [n] \to [n]$. The computation of r and the sampling are performed using any general secure MPC protocol (such as [GMW87]), but the result π is not outputted to any player (it can be viewed as secret-shared amongst the players).

[5]See Appendix B for formal definitions of MACs.

[6]An MPC protocol where a different output y_i is given to each party i can be transformed into one where all players get the same output, as follows. The output in the new protocol is $(y_1, \ldots, y_n) \oplus (r_1, \ldots, r_n)$ where r_i is a random "mask" value supplied as input by party i, and $\oplus$ denotes component-wise XOR. Each party can "unmask" and learn his own output value, but the other output values look random to him.

2. Define functionality f' as follows:

$$f'\left((x_1,k_1),\ldots,(x_n,k_n)\right)=(y,\sigma_1,\ldots,\sigma_n)\,,$$

where $y=f(x_1,\ldots,x_n)$ and $\sigma_i=\mathsf{Sign}_{k_i}(y)$. Use Protocol 1 to securely compute f' with ordering function p, where p is the constant function π.

3. Any player who has received the output $y'=(y,\sigma_1,\ldots,\sigma_n)$ sends y' to all other players. Any player P_i who receives the above message y' accepts y as the correct output value of $f(x_1,\ldots,x_n)$ if and only if $\mathsf{Verify}_{k_i}(y,\sigma_i)=\mathsf{true}$.

The proof below refers to the security of standard MPC, the definition of which may be found in Appendix A.

Theorem 2.6. *Protocol 2 is a secure, correct, and proportionally fair MPC protocol.*

Proof. Let n be the number of parties and t be the number of malicious parties, and let Fair denote the event that either all parties receive their outputs before protocol termination, or no parties receive their outputs before protocol termination.

If $t<n/2$, then by standard MPC results, all parties are guaranteed to receive their output. Security and correctness in this case follow immediately from the security and correctness of Protocol 1. It remains to consider the more interesting case $t\geq n/2$.

Correctness: By the correctness of Protocol 1, any received outputs in step 2 must be correct. We now consider step 3. In order for an incorrect message $(y^*,\sigma_1^*,\ldots,\sigma_n^*)$ in step 3 to be accepted by an honest player P_i, it must be the case that $\mathsf{Verify}_{k_i}(y^*,\sigma_i^*)=\mathsf{true}$. Note that $y=f(x_1,\ldots,x_n)$ is independent of the (honest players') keys k_i. Hence, by the security of the MAC, no adversary (having seen $(y,\sigma_1,\ldots,\sigma_n)$ in step 2) can generate y^*,σ_i^* such that $y^*\neq y$ and $\mathsf{Verify}_{k_i}(y^*,\sigma_i^*)=\mathsf{true}$. Therefore, any output value y^* accepted by an honest player in step 3 must be equal to the correct $y=f(x_1,\ldots,x_n)$ that was outputted in step 2.

Security: By the security of the general MPC protocol used in step 1, all messages exchanged during step 1 can be simulated in the ideal process. No outputs are issued in step 1.

The ordering π is independent of the inputs x_i, so cannot reveal any information about them. Hence, it follows from the ordered MPC security of Protocol 1 (which was proven in Theorem 2.4) that step 2 of Protocol 2 is (standard MPC) secure when f' is the function to be computed. Recall, however, that the function to be computed by Protocol 2 is f, not f'. By the security of Protocol 1, an adversary $\mathcal{S}$ in the ideal process can simulate the output value y. It follows that an adversary $\mathcal{S}'$ in the ideal process in Protocol 2 can simulate the output $(y,\sigma_1,\ldots,\sigma_n)$ given in step 2, by using $\mathcal{S}$ to generate a y^* which is indistinguishable from y, then generating σ_i^* as $\mathsf{Sign}_{k_i^*}(y)$, where $\mathcal{S}$ samples the keys $k_i^*\leftarrow\mathsf{Gen}(1^\kappa)$ independently for each i. By the security of the MAC scheme and since $y\overset{c}{\approx}y^*$, these $\sigma_1^*,\ldots,\sigma_n^*$ are indistinguishable from the $\sigma_i=\mathsf{Sign}_{k_i}(y)$ produced in the real world using the real players' keys k_i. Hence, all messages and outputs in step 2 can be simulated in the ideal process, even when the functionality to be computed is f rather than f'.

[7]Note that this is possible, since the number of possible permutations is $n!$, and the number of possible values of r is $2^{n^2}>n!$.

Finally, in step 3, the only messages transmitted are equal to the output value y' that is outputted in step 2, which we have already shown can be simulated.

PREFIX-FAIRNESS: If step 1 is completed (that is, not aborted prematurely), then by the correctness of the general MPC protocol, $r = r_1 \oplus \cdots \oplus r_n$ has been correctly computed. Then, since honest parties sample their r_i uniformly at random, and we can assume there is at least one honest party, r is uniformly random, and so π is a random permutation. Note that no outputs are issued until step 2 of the protocol, so if any player receives an output, then step 1 must have been completed.

By Theorem 2.4, Protocol 1 is prefix-fair, so the set of players who have received their outputs at the time of protocol termination must be a prefix of $(\pi(1), \ldots, \pi(n))$, where $\pi = p(x_1, \ldots, x_n)$. In particular, if any party has received his output from Protocol 1, then player $\pi(1)$ must have received her output. We consider two cases:

1. *The protocol terminates before any players have received their outputs.* Then Fair occurs.

2. *The protocol terminates after some players have received their outputs.* Then, $\pi(1)$ must have received her output. Moreover, step 1 must have been completed, so π is a random permutation. If $\pi(1)$ is honest, then she will execute step 3, after which all players will know their correct output. That is, if $\pi(1)$ is honest, then Fair must occur. The probability that $\pi(1)$ is honest is $(n-t)/t$, since π is a random permutation. Therefore, Fair occurs with at least $(n-t)/t$ probability.

Hence, for any PPT adversary $\mathcal{A}$ that corrupts up to t parties, the probability (over the random coins of the honest parties and $\mathcal{A}$) that Fair occurs is at least $(n-t)/t$ when $t \geq n/2$. □

3 Timed-Delay MPC

In this section, we implementing *time delays* between different players receiving their outputs. The model is exactly as before, with n players wishing to compute a function $f(x_1, \ldots, x_n)$ in an ordering prescribed by $p(x_1, \ldots, x_n)$ – except that now, there is an additional requirement of a delay after each player receives his output and before the next player receives her output. To realize the timed-delay MPC functionality, we make use of time-lock and time-line puzzles, which are introduced in section 3.2.

3.1 Ideal functionality with time delays

We measure time delay in units of computation, rather than seconds of a clock: that is, rather than making any assumption about global clocks (or synchrony of local clocks)[8], we measure time by the *evaluations of a particular function* (on random inputs), which we call the *clock function*.

IDEAL FUNCTIONALITY $\mathcal{F}_{\text{Timed-Delay-MPC}}$

[8] A particular issue that arises when considering clocks is that it is not clear that we can reasonably assume or prove that clocks are in synchrony between the real and ideal world – but this is necessary to prove security by simulation in the ideal functionality.

In the ideal model, a trusted third party T is given the inputs, computes the functions f, p on the inputs, and outputs to each player i his output y_i in the order prescribed by the ordering function. Moreover, T imposes delays between the issuance of one party's output and the next. In addition, we model an ideal process adversary $\mathcal{S}$ who attacks the protocol by corrupting players in the ideal setting.

Public parameters. $\kappa \in \mathbb{N}$, the security parameter; $n \in \mathbb{N}$, the number of parties; $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, the function to be computed; $p : (\{0,1\}^*)^n \to ([n] \to [n])$, the ordering function; and $G = G(\kappa) \in \mathbb{N}$, the number of time-steps between the issuance of one party's output and the next.

Private parameters. Each player $i \in [n]$ holds a private input $x_i \in \{0,1\}^*$.

1. INPUT. Each player i sends his input x_i to T. If, instead of sending his input, any player sends the message quit, then the computation is aborted.
2. COMPUTATION. T computes $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and $\pi = p(x_1, \ldots, x_n)$.
3. OUTPUT. The output proceeds in n sequential stages. At each stage j, T waits for G time-steps, then sends the j^{th} output, $y_{\pi(j)}$, to party $\pi(j)$.
4. OUTPUT OF VIEWS. There are n *check-points*[9] during the protocol execution, at which the parties output their views. Each check-point $\mathscr{C}_j$ corresponds to the event of outputting a value $y_{\pi(j)}$ to party $\pi(j)$ (there are n distinct such events). Each uncorrupted party i outputs y_i as his view if he has already received his output, or $\perp$ if he has not. Each corrupted party outputs $\perp$. Additionally, the adversary $\mathcal{S}$ outputs an arbitrary function of the information that he has learned during the execution of the ideal protocol, after each check-point. The adversary may output his view at any time after the current check-point and before the next one (or, if the current checkpoint is the last one, then he can output the view any time after the checkpoint).

   Let the vector of all check-points be denoted by $\mathscr{C} = (\mathscr{C}_1, \ldots, \mathscr{C}_n)$. For $i \in [n] \cup \{\mathcal{S}\}$ and $j \in [G]$, let $\mathscr{V}_{i,j}$ be the view outputted by i for the check-point $\mathscr{C}_j$. Let $\mathscr{V}^{\mathsf{ideal}}_{\text{Timed-Delay-MPC}}$ denote the collection of all views for all check-points:

$$\mathscr{V}^{\mathsf{ideal}}_{\text{Timed-Delay-MPC}} = ((\mathscr{V}_{\mathcal{S},1}, \mathscr{V}_{1,1}, \ldots, \mathscr{V}_{n,1}), \ldots, (\mathscr{V}_{\mathcal{S},n}, \mathscr{V}_{1,n}, \ldots, \mathscr{V}_{n,n}))\,.$$

Definition 3.1 (Security)**.** *A multi-party protocol F (with parameters κ, n, f, p, G) is said to securely realize $\mathcal{F}_{\text{Timed-Delay-MPC}}$, if the following conditions hold.*

1. *The protocol includes a description of n check-points $C_1, \ldots, C_n$ corresponding to events during the execution of the protocol.*
2. *Take any* PPT *adversary $\mathcal{A}$ attacking the protocol F by corrupting a subset of players $S \subset [n]$, which outputs an arbitrary function $V_{\mathcal{A},j}$ of the information that it has learned in the protocol execution after each check-point C_j. Let*

$$V^{\mathsf{real}}_{\mathcal{A}} = ((V_{\mathcal{A},1}, V_{1,1}, \ldots, V_{n,1}), \ldots, (V_{\mathcal{A},n}, V_{1,n}, \ldots, V_{n,n}))$$

be the tuple consisting of the adversary $\mathcal{A}$'s outputted views along with the views of the real-world parties as specified in the ideal functionality description. Then there is a PPT *ideal adversary $\mathcal{S}$ which, given access to $\mathcal{F}_{\text{Timed-Delay-MPC}}$ and corrupting the same subset S of players, can output views $\mathscr{V}_{\mathcal{S},j}$ such that $V^{\mathsf{real}}_{\mathcal{A}} \overset{c}{\approx} \mathscr{V}^{\mathsf{ideal}}_{\text{Timed-Delay-MPC}}$.*

[9] The use of checkpoints is introduced to capture the views of players and the adversary at intermediate points in protocol execution.

3. *There exists a "clock function" g such that between any two consecutive checkpoints C_i, C_{i+1} during an execution of F, any one of the parties (in the real world) must be able to locally run $\Omega(G)$ sequential evaluations of g on random inputs. g may also be a* protocol *(involving $n' \leq n$ parties) rather than a function, in which case we instead require that any subset consisting of n' parties must be able to run $\Omega(G)$ sequential executions of g (on random inputs) over the communication network being used for the main multi-party protocol F. Then, we say that F is "clocked by g".*

A simple protocol for securely realizing timed-delay MPC is to implement delays by running G "dummy rounds" of communication (amongst all players) in between issuing outputs to different players.

PROTOCOL 3

Public parameters. $\kappa \in \mathbb{N}$, the security parameter; $n \in \mathbb{N}$, the number of parties; $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, the function to be computed; $p : (\{0,1\})^* \to ([n] \to [n])$, the ordering function; and $G = \mathsf{poly}(\kappa)$, the number of time-steps between the issuance of one party's output and the next.

1. Using any general secure MPC protocol (such as [GMW87]) jointly compute the desired function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and permutation $\pi = p(x_1, \ldots, x_n)$ on the players' inputs. Before any outputs are issued to players, proceed to step 2.

2. Each value y_i will be outputted to the corresponding player i, and the outputs will be issued in the order prescribed by permutation π. The outputs will occur in n rounds, and in the i^{th} round, player $\pi(i)$ will receive his output. Moreover, in the i^{th} round, all players other than $\pi(i)$ will receive a "fake" output with value $\perp$.

   In each round, before the output (and fake outputs) are issued to any player, the players run G "dummy rounds" of communication. A dummy round of communication is a "mini-protocol" defined as follows (let this mini-protocol be denoted by g_{dum}):

   - each player initially sends the message challenge to every other player;
   - each player responds to each challenge he receives with a message response.

Check-points. There are n check-points. For $i \in [n]$, the check-point C_i is the event of outputting $y_{\pi(i)}$ to party $\pi(i)$.

Theorem 3.2. *In the honest-but-curious setting with honest majority, Protocol 3 securely realizes* $\mathcal{F}_{\text{Timed-Delay-MPC}}$.

Proof. Take any adversary $\mathcal{A}$ attacking Protocol 1 by corrupting some subset $S \subset [n]$ of parties. Consider any check-point C_j (for $j \in [n]$) of the real protocol. Recall that an adversary's view may be an arbitrary function of the information that he has learned during the execution of the ideal protocol, which comprises all the messages seen by the corrupt players, and the outputs that they received. Let $\mathcal{A}_j(M)$ and $\mathcal{A}_j(M)$ denote the functions that $\mathcal{A}$ outputs after check-point C_j (and

before C_{j+1}), where M is the sequence of messages the corrupt players have seen so far (including output messages).

Let M_0 denote the sequence of messages seen by the corrupt players during step 1 ("the computation phase") of Protocol 1. By the security of the underlying general MPC protocol, it holds that any arbitrary function of M_0 can be simulated by an ideal adversary $\mathcal{S}^*$. (We refer to this simulator $\mathcal{S}^*$ later in the proof.) Similarly, define $\mathscr{M}_0 = ()$ to be the (empty) sequence of messages seen by the corrupt players before any outputs are issued in the ideal protocol.

The only messages that are sent in the real protocol *apart from* the output messages $y_{\pi(j)}$ are the "dummy" messages $\mathsf{challenge}$ and $\mathsf{response}$, which are fixed messages that do not depend on the players' inputs. To be precise, between each pair of checkpoints, each player (including corrupt players) will send $n-1$ $\mathsf{challenge}$ messages, receive $n-1$ $\mathsf{challenge}$ messages, and send $n-1$ $\mathsf{response}$ messages. Let $\mathsf{DummyTuple}$ denote the tuple consisting of these $2n-2$ $\mathsf{challenge}$ messages and $n-1$ $\mathsf{response}$ messages.

In the real execution: Let M_j denote the sequence of messages seen by corrupt players up to and including check-point C_j. By the definition of the check-point, for any $j \in [n]$, it holds that

$$M_j = \begin{cases} M_{j-1}||\mathsf{DummyTuple}||(y_{\pi(j)}) & \text{if } j \in S \\ M_{j-1}||\mathsf{DummyTuple}||(\perp) & \text{otherwise} \end{cases}, \tag{3}$$

where the $||$ operator denotes concatenation of tuples.

In the ideal execution: Let $\mathscr{M}_j$ and $\mathscr{O}_j$ be defined in the corresponding way for the ideal execution. For the ideal check-points, it holds that

$$\mathscr{M}_j = \begin{cases} \mathscr{M}_{j-1}||(y_{\pi(j)}) & \text{if } j \in S \\ \mathscr{M}_{j-1}||(\perp) & \text{otherwise} \end{cases}. \tag{4}$$

In order to construct an ideal-world simulator $\mathcal{S}$ for the real-world adversary $\mathcal{A}$, we define some auxiliary algorithms. For $j \in [n]$, define $\mathcal{A}'_j$ as follows:

$$\mathcal{A}'_j(M, O) = \mathcal{A}\left(M||z_{\pi(1)}||\ldots||z_{\pi(j)}\right),$$
$$\text{where } z_{\pi(i)} = \begin{cases} \mathsf{DummyTuple}||(y_{\pi(i)}) & \text{if } i \in S \\ \mathsf{DummyTuple}||(\perp) & \text{otherwise} \end{cases}.$$

By equation 3, for each $j \in [n]$, it holds that $M_j = M||(z_{\pi(1)}, \ldots, z_{\pi(j)})$. Hence, $\mathcal{A}'_j(M_0, O_0) = \mathcal{A}_j(M_j, O_j)$. We define a ideal adversary $\mathcal{S}$ which simulates $\mathcal{A}$ as follows. $\mathcal{S}$ simulates each view $\mathcal{A}_j(M_j, O_j)$ outputted by the real-world adversary $\mathcal{A}$ by using the simulator $\mathcal{S}^*$ to compute $\mathcal{A}'_j(M_0, O_0)$. This is possible since:

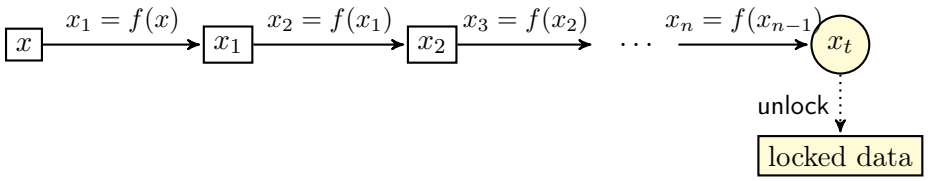
- $\mathcal{S}^*$ can simulate any function of (M_0, O_0), as remarked above; and
- $\mathcal{S}$ knows $\mathscr{M}_j$ in the ideal execution, which (by equation 4) contains exactly the $z_{\pi(1)}, \ldots, z_{\pi(j)}$ that need to be hard-wired into the function $\mathcal{A}'_j$ that is to be computed.

Finally, it remains to show that condition 3 of the security definition (Definition 3.1) is satisfied. The players are literally running g over the MPC network G times in between issuing outputs, so it is clear that condition 3 holds. □

One downside of the simple solution above is that it requires all (honest) parties to be online and communicating until the last player receives his output. To address this, we propose an alternative solution based on timed-release cryptography, at the cost of an additional assumption that all players have comparable computing speed (within a logarithmic factor).

3.2 Time-lock and time-line puzzles

The delayed release of data in MPC protocols can be closely linked to the problem of "timed-release crypto" in general, which was introduced by [RSW96] with their proposal of *time-lock puzzles*. We assume time-lock puzzles with a particular structure (that is present in all known implementations): namely, the passage of "time" will be measured by sequential evaluations of a function ($\mathsf{TimeStep}$). Unlocking a t-step time-lock puzzle can be considered analogous to following a chain of t pointers, at the end of which there is a special value x_t (e.g. a decryption key) that allows retrieval of the locked data.



Definition 3.3 (Time-lock puzzle scheme)**.** *A* time-lock puzzle scheme *is a tuple of* PPT *algorithms* $T = (\mathsf{Lock}, \mathsf{TimeStep}, \mathsf{Unlock})$ *as follows:*

- $\mathsf{Lock}(1^\kappa, d, t)$ *takes parameters* $\kappa \in \mathbb{N}$ *the security parameter,* $d \in \{0,1\}^\ell$ *the data to be locked, and* $t \in \mathbb{N}$ *the number of steps needed to unlock the puzzle, and outputs a time-lock puzzle* $P = (x, t, b, a) \in \{0,1\}^n \times \mathbb{N} \times \{0,1\}^{n''} \times \{0,1\}^{n'}$ *where* $\ell, n, n', n'' = \mathsf{poly}(\kappa)$.
- $\mathsf{TimeStep}(1^\kappa, x', a')$ *takes parameters* $\kappa \in \mathbb{N}$ *the security parameter, a bit-string* $x' \in \{0,1\}^n$*, and auxiliary information* a'*, and outputs a bit-string* $x'' \in \{0,1\}^n$.
- $\mathsf{Unlock}(1^\kappa, x', b')$ *takes parameters* $\kappa \in \mathbb{N}$ *the security parameter, a bit-string* $x' \in \{0,1\}^n$*, and auxiliary information* $b' \in \{0,1\}^{n'}$*, and outputs some data* $d' \in \{0,1\}^\ell$.

To unclutter notation, we will sometimes omit the initial security parameter of these functions (writing e.g. simply $\mathsf{Lock}(d, t)$). We now define some auxiliary functions. For a time-lock puzzle scheme $T = (\mathsf{Lock}, \mathsf{TimeStep}, \mathsf{Unlock})$ and $i \in \mathbb{N}$, let $\mathsf{IterateTimeStep}_i^T$ denote the following function:

$$\mathsf{IterateTimeStep}^T(i, x, a) = \underbrace{\mathsf{TimeStep}(\mathsf{TimeStep}(\ldots(\mathsf{TimeStep}(x, a), a)\ldots), a)}_{i}.$$

Define $\mathsf{CompleteUnlock}^T$ to be the following function:

$$\mathsf{CompleteUnlock}^T((x, t, b, a)) = \mathsf{Unlock}(\mathsf{IterateTimeStep}^T(t, x, a), b),$$

that is, the function that should be used to unlock a time-lock puzzle outputted by Lock.

The following definitions formalize correctness and security for time-lock puzzle schemes.

Definition 3.4 (Correctness)**.** *A time-lock puzzle scheme* $T = (\mathsf{Lock}, \mathsf{TimeStep}, \mathsf{Unlock})$ *is* correct *if the following holds (where* κ *is the security parameter):*

$$\Pr_{(x,t,b,a)\leftarrow\mathsf{Lock}(d,t)}\left[\mathsf{CompleteUnlock}^T((x,t,b,a)) \neq d\right] \leq \mathsf{negl}(\kappa).$$

For an algorithm $\mathcal{A}$, let the run-time of $\mathcal{A}$ on input inp be denoted by $\mathsf{time}_{\mathcal{A}}(\mathsf{inp})$. If $\mathcal{A}$ is probabilistic, the run-time will be a distribution over the random coins of $\mathcal{A}$. Note that the exact run-time of an algorithm will depend on the underlying computational model in which the algorithm is run. In this work, all algorithms are assumed to be running in the same underlying computational model, and our definitions and results hold regardless of the specific computational model employed.

Definition 3.5 (Security)**.** *Let* $T = (\mathsf{Lock}, \mathsf{TimeStep}, \mathsf{Unlock})$ *be a time-lock puzzle scheme.* T *is* secure *if it holds that: for all* $d, d' \in \{0,1\}^\ell, t = \mathsf{poly}(\kappa)$*, if there exists an adversary* $\mathcal{A}$ *that solves the time-lock puzzle* $\mathsf{Lock}(d,t)$*, that is,*

$$\Pr[\mathcal{A}(\mathsf{Lock}(d,t)) = d] = \varepsilon \text{ for some non-negligible } \varepsilon,$$

then for each $j \in [t-1]$*, there exists an adversary* $\mathcal{A}_j$ *such that*

$$\Pr[\mathcal{A}_j(\mathsf{Lock}(d',j)) = d'] \geq \varepsilon, \text{ and}$$

$$\Pr[\mathsf{time}_{\mathcal{A}}(\mathsf{Lock}(d,t)) \geq (t/j)\cdot \mathsf{time}_{\mathcal{A}_j}(\mathsf{Lock}(d',j)) \mid \mathcal{A}(\mathsf{Lock}(d,t)) = d] > 1 - \mathsf{negl}(\kappa).$$

3.2.1 Time-line puzzles.

We now introduce the more general, novel definition of *time-line* puzzles, which can be useful for locking together many data items with different delays for a single recipient, or for locking data for a group of people. In the latter case, it becomes a concern that computation speed will vary between parties: indeed, the scheme will be unworkable if some parties have orders of magnitude more computing power than others, so some assumption is required on the similarity of computing power among parties. This issue is discussed further in the next section.

Definition 3.6 (Time-line puzzles)**.** *A* time-line puzzle scheme *is a family of* PPT *algorithms* $\mathcal{T} = \{(\mathsf{Lock}_m, \mathsf{TimeStep}_m, \mathsf{Unlock}_m)\}_{m\in\mathbb{N}}$ *as follows:*

- $\mathsf{Lock}_m(1^\kappa, (d_1,\ldots,d_m), (t_1,\ldots,t_m))$ *takes parameters* $\kappa \in \mathbb{N}$ *the security parameter,* $(d_1,\ldots,d_m) \in (\{0,1\}^\ell)^m$ *the data items to be locked, and* $(t_1,\ldots,t_m) \in \mathbb{N}^m$ *the number of steps needed to unlock each data item (respectively), and outputs a puzzle*

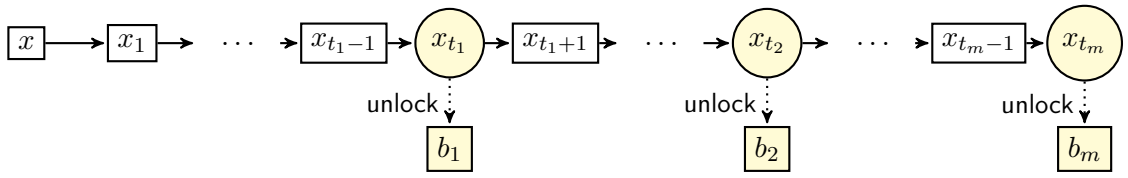
  $$P = (x, (t_1,\ldots,t_m), (b_1,\ldots,b_m), a) \in \{0,1\}^n \times \mathbb{N} \times (\{0,1\}^{n''})^m \times \{0,1\}^{n'}$$

  where $n, n', n'' = \mathsf{poly}(\kappa)$*, and* a *can be thought of as auxiliary information.*

- $\mathsf{TimeStep}_m(1^\kappa, x', a')$ *takes parameters* $\kappa \in \mathbb{N}$ *the security parameter, a bit-string* $x' \in \{0,1\}^n$*, and auxiliary information* a'*, and outputs a bit-string* $x'' \in \{0,1\}^n$*.*

- $\mathsf{Unlock}_m(1^\kappa, x', b')$ *takes parameters* $\kappa \in \mathbb{N}$ *the security parameter, a bit-string* $x' \in \{0,1\}^n$*, and auxiliary information* $b' \in \{0,1\}^{n'}$*, and outputs some data* $d' \in \{0,1\}^\ell$*.*

In terms of the "pointer chain" analogy above, solving a time-line puzzle may be thought of as following a pointer chain where not one but many keys are placed along the chain, at different locations $t_1, \ldots, t_m$. Each key x_{t_i} in the pointer chain depicted below enables the "unlocking" of the locked data b_i: for example, b_i could be the encryption of the i^{th} data item d_i under the key x_{t_i}.



Using similar notation to that defined for time-lock puzzles: for a time-line puzzle scheme $\mathcal{T}$, let $\mathsf{IterateTimeStep}_m^{\mathcal{T}}$ denote the following function:

$$\mathsf{IterateTimeStep}_m^{\mathcal{T}}(i, x, a) = \underbrace{\mathsf{TimeStep}_m(\mathsf{TimeStep}_m(\ldots(\mathsf{TimeStep}_m(x, a), a)\ldots), a)}_{i}.$$

Define $\mathsf{CompleteUnlock}_{m,i}^{\mathcal{T}}$ to be the following function:

$$\mathsf{CompleteUnlock}_{m,i}^{\mathcal{T}}((x, t_i, b_i, a)) = \mathsf{Unlock}_m(\mathsf{IterateTimeStep}_m^{\mathcal{T}}(t_i, x, a), b_i),$$

that is, the function that should be used to unlock the i^{th} piece of data locked by a time-line puzzle which was generated by Lock_m. We now define correctness and security for time-line puzzle schemes.

Definition 3.7 (Correctness)**.** *A time-line puzzle scheme $\mathcal{T}$ is* correct *if for all $m = \mathsf{poly}(\kappa)$ and for all $i \in [m]$, it holds that*

$$\Pr_{(x,\vec{t},\vec{b},a)\leftarrow\mathsf{Lock}_m(\vec{d},\vec{t})}\left[\mathsf{CompleteUnlock}_i^{\mathcal{T}}((x, t_i, b_i, a)) \neq d_i\right] \leq \mathsf{negl}(\kappa),$$

where κ is the security parameter, $\vec{d} = (d_1, \ldots, d_m)$, and $\vec{t} = (t_1, \ldots, t_m)$.

Security for time-line puzzles involves more stringent requirements than security for time-lock puzzles. We define security in terms of two properties which must be satisfied: *timing* and *hiding*. The timing property is very similar to the security requirement for time-lock puzzles, and gives a guarantee about the relative amounts of time required to solve different time-lock puzzles. The hiding property ensures (informally speaking) that the ability to unlock any given data item that is locked in a time-line puzzle does not imply the ability unlock any others. The security definition (Definition 3.8, below) refers to the following security experiment.

The experiment $\mathsf{HidingExp}_{\mathcal{A},\mathcal{T}}(\kappa)$

1. $\mathcal{A}$ outputs $m = \mathsf{poly}(\kappa)$ and data vectors $\vec{d}_0, \vec{d}_1 \in (\{0,1\}^\ell)^m$ and a time-delay vector $\vec{t} \in \mathbb{N}^m$.
2. The challenger samples $(\beta_1, \ldots, \beta_m) \leftarrow \{0,1\}^m$, computes the time-line puzzle $(x, \vec{t}, \vec{b}, a) = \mathsf{Lock}_m(1^\kappa, ((d_{\beta_1})_1, \ldots, (d_{\beta_m})_m), \vec{t})$, and sends (x, a) to $\mathcal{A}$.

3. $\mathcal{A}$ sends a query $i \in [m]$ to the challenger. The challenger responds by sending b_i to $\mathcal{A}$. This step may be repeated up to $m-1$ times. Let I denote the set of queries made by $\mathcal{A}$.
4. $\mathcal{A}$ outputs $i' \in [m]$ and $\beta' \in \{0,1\}$.
5. The output of the experiment is 1 if $i' \notin I$ and $\beta' = \beta_{i'}$. Otherwise, the output is 0.

Definition 3.8 (Security)**.** *Let $\mathcal{T} = \{(\mathsf{Lock}_m, \mathsf{TimeStep}_m, \mathsf{Unlock}_m)\}_{m\in\mathbb{N}}$ be a time-line puzzle scheme. T is* secure *if it satisfies the following two properties.*

- Timing: *For all $m = \mathsf{poly}(\kappa)$ and $\vec{d}, \vec{d'} \in (\{0,1\}^\ell)^m$ and $\vec{t} = (t_1, \ldots, t_m)$, if there exists an adversary $\mathcal{A}$ that solves any one of the puzzles defined by the time-line, that is,*

$$\Pr[\mathcal{A}(\mathsf{Lock}_m(\vec{d}, \vec{t})) = d_i] = \varepsilon \text{ for some non-negligible } \varepsilon \text{ and some } i \in [m],$$

then for all $j \in [t_i - 1]$ and all $\vec{t'} \in [t_m]^m$, there exists an adversary $\mathcal{A}_{j,t'}$ such that

$$\Pr[\mathcal{A}_{j,t'}(\mathsf{Lock}_m(\vec{d'}, \vec{t'})) = d_j] \geq \varepsilon, \text{ and}$$

$$\Pr[\mathsf{time}_{\mathcal{A}}(\mathsf{Lock}_m(\vec{d}, \vec{t})) \geq (t'_j/t_i) \cdot \mathsf{time}_{\mathcal{A}_{j,t'}}(\mathsf{Lock}_m(\vec{d}, \vec{t'})) \mid \mathcal{A}(\mathsf{Lock}_m(\vec{d}, \vec{t})) = d_i] > 1 - \mathsf{negl}(\kappa).$$

- Hiding: *For all* PPT *adversaries $\mathcal{A}$, it holds that*

$$\Pr[\mathsf{HidingExp}_{\mathcal{A},\mathcal{T}}(\kappa) = 1] \leq 1/2 + \mathsf{negl}(\kappa).$$

3.2.2 Black-box construction from inherently-sequential hash functions

Definition 3.9 (Inherently-sequential hash function)**.** *Let $\mathcal{H}_\kappa = \{h_s : \{0,1\}^\kappa \to \{0,1\}^\kappa\}_{k\in\{0,1\}^n}$ for $n = \mathsf{poly}(\kappa)$ be a family of functions and suppose that evaluating $h_s(r)$ for $r \leftarrow \{0,1\}^\kappa$ takes time $\Omega(T)$. $\mathcal{H}_\kappa$ is said to be* inherently-sequential *if evaluating $h_s^t(r)$ for $s \leftarrow \{0,1\}^n, r \leftarrow \{0,1\}^\kappa$ takes time $\Omega(t \cdot T)$, and the output of $h_s^t(r)$ is pseudorandom.*

The time-line puzzle construction in this section relies on the following assumption about the existence of inherently-sequential functions.

Assumption 1. *There exists a family of functions*

$$\widetilde{\mathcal{H}}_\kappa = \{\widetilde{h}_s : \{0,1\}^\kappa \to \{0,1\}^\kappa\}_{s\in\{0,1\}^n}$$

which is inherently-sequential (where $n = \mathsf{poly}(\kappa)$).

Definition 3.10. BB-TimeLinePuzzle *is a time-line puzzle defined as follows, where $\widetilde{\mathcal{H}}_\kappa$ is the inherently-sequential hash function family from Assumption 1:*

- $\mathsf{Lock}_m(1^\kappa, (d_1, \ldots, d_m), (t_1, \ldots, t_m))$ *takes input data $(d_1, \ldots, d_m) \in \{0,1\}^\kappa$, samples random values $s \leftarrow \{0,1\}^n, x \leftarrow \{0,1\}^\kappa$, and outputs the puzzle*

$$P = \left(x, (t_1, \ldots, t_m), s, \left(d_1 \oplus \widetilde{h}_s^{t_1}(x), \ldots, d_m \oplus \widetilde{h}_s^{t_m}(x)\right)\right).$$

- $\mathsf{TimeStep}_m(1^\kappa, i, x', a')$ *outputs* $\widetilde{h}_{a'}(x')$.
- $\mathsf{Unlock}_m(1^\kappa, x', b')$ *outputs* $x' \oplus b'$.

It is clear that BB-TimeLinePuzzle satisfies correctness, so we proceed to prove security.

Theorem 3.11. *If Assumption 1 holds, then* BB-TimeLinePuzzle *is a secure time-line puzzle.*

Proof. Given a time-line puzzle, in order to correctly output a piece of locked data d_i, the adversary $\mathcal{A}$ must compute the associated mask $\widetilde{h}_s^{t_i}(x)$. This is because

- all components of the puzzle apart from the masked value $d_i \oplus \widetilde{h}_s^{t_i}(x)$ are independent of the locked data d_i, and
- the mask $\widetilde{h}_s^{t_i}(x)$ is pseudorandom (by Assumption 1), so the masked value $d_i \oplus \widetilde{h}_s^{t_i}(x)$ is indistinguishable from a truly random value without knowledge of the mask.

Moreover, by Assumption 1, since $\widetilde{\mathcal{H}}_\kappa$ is an inherently-sequential function family, it holds that there is no (asympotically) more efficient way for a PPT adversary to compute $\widetilde{h}_s^{t_i}(x)$ than to sequentially compute $\widetilde{h}_s$ for t_i iterations. It follows that BB-TimeLinePuzzle is a secure time-line puzzle. □

3.2.3 Concrete construction based on modular exponentiation

In this subsection we present an alternative construction quite similar in structure to the above, but based on a concrete hardness assumption. Note that the [RSW96] time-lock puzzle construction was also based on this hardness assumption, and our time-line puzzle may be viewed as a natural "extension" of their construction.

Assumption 2. *Let* RSA_κ *be the distribution generated as follows: sample two* κ*-bit primes* p, q *uniformly at random and output* $N = pq$*. The family of functions* $\mathcal{H}^{square} = \{h_N : \mathbb{Z}_N \to \mathbb{Z}_N\}_{N \leftarrow \mathsf{RSA}_\kappa}$*, where the index* N *is drawn from distribution* RSA *and* $h_N(x) = x^2 \mod N$*, is inherently-sequential.*

Definition 3.12. Square-TimeLinePuzzle *is a time-line puzzle defined as follows:*

- $\mathsf{Lock}_m(1^\kappa, (d_1, \ldots, d_m), (t_1, \ldots, t_m))$ *takes input data* $(d_1, \ldots, d_m) \in \{0,1\}^\kappa$*, samples random* κ*-bit primes* p, q*, sets* $N = pq$*, and outputs the puzzle*

$$P = \left(x, (t_1, \ldots, t_m), N, \left(d_1 \oplus h_N^{t_1}(x), \ldots, d_m \oplus h_N^{t_m}(x)\right)\right).$$

- $\mathsf{TimeStep}_m(1^\kappa, i, x', a')$ *outputs* $h_{a'}(x') = x'^2 \mod a'$.
- $\mathsf{Unlock}_m(1^\kappa, x', b')$ *outputs* $x' \oplus b'$.

Again, it is clear that Square-TimeLinePuzzle satisfies correctness, so we proceed to prove security.

Theorem 3.13. *If Assumption 2 holds,* Square-TimeLinePuzzle *is a secure time-line puzzle.*

Proof. This follows from Assumption 2 in exactly the same way as Theorem 3.11 follows from Assumption 1, so we refer the reader to the proof of Theorem 3.11. □

An advantage of this construction over BB-TimeLinePuzzle is that the Lock algorithm can be much more efficient. In the case of black-box inherently-sequential hash functions, we can only assume that the values $\widetilde{h}_s^t(x)$ (which are XORed with the data values by the Lock algorithm) are computed by sequentially evaluating $\widetilde{h}_s$ for t iterations – that is, there is a linear dependence on t. However, Lock can implemented much faster with the Square-TimeLinePuzzle construction, as follows. Since p, q are generated by (and therefore, available to) the Lock algorithm, the Lock algorithm can efficiently compute $\phi(N)$. Then, $h_N^t(x)$ can be computed very efficiently by first computing $e = 2^t \mod \varphi(N)$, then computing $h_N^t(x) = x^e \mod N$. Exponentiation (say, by squaring) has only a logarithmic dependence on the security parameter.

Finally, we note that although both of the time-line puzzle constructions presented here lock κ bits of data per puzzle (for security parameter κ), this is not at all a necessary restriction. Using encryption, it is straightforwardly possible to lock much larger amounts of data for any given parameter sizes of the time-line puzzles presented here: for example, one can encrypt the data as $\mathsf{Enc}_k(d)$ using a secure secret-key encryption scheme, then use the given time-line puzzle schemes to lock the key k (which is much smaller than d) under which the data is encrypted. Such a scheme, with the additional encryption step, would be much more suitable for realistic use.

3.3 Time-lock-based construction of timed-delay MPC

Because of the use of time-lock puzzles by different parties in the protocol that follows, we require an additional assumption that all players have comparable computing power (within a logarithmic factor).

Assumption 3. *The difference in speed of performing computations between any two parties $i, j \in [n]$ is at most a factor of $B = O(\log(\kappa))$.*

PROTOCOL 4

Public parameters. $\kappa \in \mathbb{N}$, the security parameter; $n \in \mathbb{N}$, the number of parties; $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, the function to be computed; $p : (\{0,1\})^* \to ([n] \to [n])$, the ordering function; $B = O(\log(\kappa))$, the maximum factor of difference between any two parties' computing power; $G = \mathsf{poly}(\kappa)$, the number of time-steps between the issuance of one party's output and the next; and $T = \{\mathsf{Lock}, \mathsf{TimeStep}, \mathsf{Unlock}\}$ a time-lock puzzle scheme.

1. Using any general MPC protocol (such as [GMW87]) secure in the presence of an honest majority, jointly compute the desired function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and permutation $\pi = p(x_1, \ldots, x_n)$ on the players' inputs. Before any outputs are issued to players, proceed to step 2.

2. Again using the general MPC protocol, compute n time-lock puzzles $P_i = (x_i, t_i, a_i, b_i)$ for $i \in [n]$, each computed as
$$P_i = \mathsf{Lock}(y_i \oplus r_i, t_i),$$
where we define $t_1 = 1$ and $t_{i+1} = (B \cdot G + 1) \cdot t_i$ for $i \in [n-1]$, and each r_i is a random string provided as input by party i.

3. For each $i \in [n]$, output P_i to player i. (Each player receives his tuple at the same time, then recovers his output y_i by solving his time-lock puzzle, and "unmasking" the result by XORing with his random input r_i. We assume each player uses the fastest possible algorithm that gets the correct answer[10].)

Check-points. There are n check-points. For $i \in [n]$, the check-point C_i is the event of party $\pi(i)$ learning his eventual output $y_{\pi(i)}$ (i.e. the point at which he finishes solving his time-lock puzzle).

Theorem 3.14. *If Assumption 3 holds, then Protocol 4 securely realizes* $\mathcal{F}_{\text{Timed-Delay-MPC}}$ *in the honest-but-curious setting with honest majority.*

Proof. Exactly as in the proof of Theorem 3.2, any real-world adversary attacking step 1 of Protocol 4 can be simulated in the ideal world: this follows directly from the security of the underlying general MPC protocol. Moreover (again as in the proof of Theorem 3.2) the output messages (which only occur at check-points) can be simulated in the ideal functionality since they are the same (and occur at the same checkpoints) in the real and ideal functionalities. In order to simulate the real adversary's view entirely, it remains only to prove that the *non-output* messages in the real protocol can also be simulated in the ideal world. The non-output messages are comprised of the time-lock puzzle parameters that are outputted to each player. The data items "locked" by the time-lock puzzles are perfectly indistinguishable from uniformly random, without knowledge of the input random values r_i. Hence, the ideal adversary can perfectly simulate all of these non-output messages by generating n puzzles using Lock, using the same delay values $t_1, \ldots, t_n$ as in the real protocol, but substituting the data items $d_1, \ldots, d_n$ with uniformly random values $d'_1, \ldots, d'_n$.

Now we have shown that the first two conditions of the security definition (Definition 3.1) hold, and it remains to prove that condition 3 is satisfied. Let $\mathcal{A}_i$ denote the algorithm that party i uses to solve his time-lock puzzle, and let the time at which party i learns his answer y_i be denoted by $\tau_i = \mathsf{time}_{\mathcal{A}_i}(P_i)$. By the security of the time-lock puzzles, there exists an algorithm $\mathcal{A}'_i$ that player i could use to solve the puzzle $\mathsf{Lock}(0^\ell, 1)$ in time τ_i/t_i. Moreover, by Assumption 3, it holds that no player can solve the puzzle $\mathsf{Lock}(0^\ell, 1)$ more than B times faster than another player: that is, $\max_i(\tau_i/t_i) \leq B \cdot \min_i(\tau_i/t_i)$. It follows that even the slowest player (call him i^*) would be able to run t_i/B executions of $\mathcal{A}'_{i^*}$ within time τ_i, for any i.

Without loss of generality, assume that the ordering function p is the identity function. Consider any consecutive pair of checkpoints C_i, C_{i+1}. These checkpoints occur at times τ_i and τ_{i+1}, by definition. We have established that in time τ_i, player i^* can run t_i/B executions of $\mathcal{A}'_{i^*}$, and in time τ_{i+1}, he can run t_{i+1}/B executions of $\mathcal{A}'_{i^*}$. It follows that in between the two checkpoints (i.e. in time $\tau_{i+1} - \tau_i$), he can run $(t_{i+1} - t_i)/B$ executions of $\mathcal{A}'_{i^*}$. Substituting in the equation $t_{i+1} = (B \cdot G + 1) \cdot t_i$ from the protocol definition, we get that player i^* can run $G \cdot t_i$ executions of $\mathcal{A}'_{i^*}$ between checkpoints C_i and C_{i+1}. Since $t_i \geq 1$ for all i, this means that i^* can run at least

[10]Without this assumption, any further protocol analysis would not make sense: there can always be a "lazy" player who willfully uses a very slow algorithm to solve his puzzle, who will as a result learn his eventual output much later in the order than he could otherwise have done. The property that we aim to achieve is that every player *could* learn his output at his assigned position in the ordering π, with appropriate delays before and after he learns his output.

G executions of $\mathcal{A}'_{i^*}$ between *any* consecutive pair of checkpoints. Hence, condition 3 holds, and Protocol 4 securely realizes $\mathcal{F}_{\text{Timed-Delay-MPC}}$ clocked by $\mathcal{A}'_{i^*}$. □

A few remarks are in order. In Protocol 4, all the parties can stop interacting as soon as all the puzzles are outputted. When the locking algorithm $\mathsf{Lock}(d, t)$ has run-time that is independent of the delay t, the run-time of Protocol 4 is also independent of the delay parameters. (This is achievable using the [RSW96] time-lock construction, for example.) Alternatively, using a single time-line puzzle in place of the time-lock puzzles in Protocol 4 can improve efficiency, since the time required to generate a time-line puzzle is dependent only on the longest delay t_n, whereas the time required to generate n separate time-lock puzzles depends on the sum of all the delays, $t_1 + \cdots + t_n$.

4 Finding Feasible Orders for Scientific Collaboration

We present a model of scientific collaboration, addressing *whether* it is beneficial for parties to collaborate, and if so, *how*: in particular, since the order of research publication matters, we examine the problem of finding a feasible collaboration order. Once an order is found, the ordered and timed-delay MPC protocols of the previous sections may be used to securely implement beneficial collaborations.

We propose a model of collaboration between n research groups which captures the following features. Groups may pool their data, but each group will publish their own results. Moreover, only results that improve on the "state of the art" may be published. That is, a new result must improve on prior publications. However, more credit is given to earlier publications. Finally, a group will learn not only from pooling their data with other groups, but also from other groups' publications. To formalize these intuitions, we specify a model as follows:

- There is a set $[n]$ of players.
- For each $i \in [n]$, there is a set X_i of possible datasets, and a dataset $x_i \in X_i$ that the player knows. Let X denote $X_1 \times \cdots \times X_n$.
- A set of outputs $Y \subset \mathbb{R}$. $\Delta(Y)$ denotes the set of random variables over Y.
- A function $f : X \to Y$, whose output $y \stackrel{\text{def}}{=} f(x_1, \ldots, x_n)$ the players wish to learn.
- All players commonly know an "initial result" $y_0 \in \Delta(Y)$ such that $\mathbb{E}[y_0] = y$.
- A *collaboration outcome* is given by a permutation $\pi : [n] \to [n]$ and a vector of random variables $(z_{\pi(1)}, \ldots, z_{\pi(n)}) \in \Delta(Y)^n$ such that $\mathbb{E}[z_{\pi(t)}] = y$ and $Var(y_0) > Var(z_{\pi(1)}) > \cdots > Var(z_{\pi(n)})$.
  The intuition behind this condition is that, at time t, player $\pi(t)$ will publish result $z_{\pi(t)}$. This result is unbiased, but has a variance of $Var(z_{\pi(t)})$ A result $z_{\pi(t)}$ is higher quality if it's variance is lower. Since only results that improve on the "state of the art" can be published, we must have that $Var(z_{\pi(t)})$ decreases with the time of publication t.
- If $\omega = (\pi, \mathbf{z})$ is an outcome, the player who publishes at time t obtains a reward $R_t(\pi, \mathbf{z}) = \beta^t[Var(z_{\pi(t-1)}) - Var(z_{\pi(t)})]$ where $\beta \in (0, 1)$ is a discount factor which penalizes later publications, and $Var(z_{\pi(t-1)}) - Var(z_{\pi(t)})$.[11]
- For each player i, we denote by $\alpha_i \geq 0$ their reward from not collaborating. This models the best "outside payoff" that they could receive from publishing on their own. We denote by $A = \mathbb{R}^n_+$ the set of all outside option vectors.

[11] This is motivated by market scoring rules [Han12], where different experts only receive a reward equal to how much they improve existing predictions.

4.1 Data-sharing mechanisms

Definition 4.1. *Given a function $f : X \to Y$, a* data sharing mechanism *is a function $M_f : X \times A \to ([n] \to [n]) \times \Delta(Y)^n$ which takes as inputs a vector $(x_1, \ldots, x_n)$ of datasets and a vector $(\alpha_1, \ldots, \alpha_n)$ of outside options. The outputs from such a mechanism are an ordering π of the players, and a vector of random variables $(y_1, \ldots, y_n)$ such that $\mathbb{E}[y_i] = f(x_1, \ldots, x_n)$.*

One issue with the above definition is that player i may not publish the suggested result y_i, but may instead publish a result z_i with lower variance than y_i. The published result z_i may depend on y_i, x_i, as well as from the publications of players who published before her. We bound the amount that player i can learn from others (and from her own dataset) by introducing *learning bound vectors.*

Definition 4.2. *A* learning bound vector $\{\lambda_{\pi,i}\}_{\pi \in ([n] \to [n]), i \in [n]}$ *is a non-negative vector such that, if $(\pi, (y_1, \ldots, y_n))$ is a collaboration outcome proposed by a data-sharing mechanism, and z_i is the best result that player i knows at time $\pi^{-1}(i)$, then $Var(z_i) \geq Var(y_i) - \lambda_{\pi,i}$. We denote by $\Lambda = \mathbb{R}_+^{n! \times n}$ the set of all learning bound vectors.*

We will henceforth refer to the player who publishes at time t as $\pi(t)$.

Definition 4.3. *Let $\{\lambda_{\pi,i}\}_{\pi \in ([n] \to [n]), i \in [n]}$ be a vector of learning bounds. Let $\pi \in ([n] \to [n])$ be an ordering of the players and let $(y_1, \ldots, y_n)$ be a vector of random variables with $\mathbb{E}[y_i] = y$ and variance $Var(y_i)$. The set of* inferred publications *derived from (π, y) is the set $\mathcal{I}(\pi, \mathbf{y}) = \{(z_1, \ldots, z_n) : Var(y_{\pi(t)}) - \lambda_{\pi,\pi(t)} \leq Var(z_\pi(t)) \leq Var(y_{\pi(t)})\}$.*

The intuition behind the above definition is that the amount of information that player $\pi(t)$ can learn from prior publications is measured by how much she can reduce the variance of her prediction. This reduction in variance is bounded by $\lambda_{\pi,\pi(t)}$. Thus, her eventual publication will be some $z_{\pi(t)}$ with variance between $Var(y_{\pi(t)}) - \lambda_{\pi,\pi(t)}$ and $Var(y_{\pi(t)})$.

In our model, each research group $\pi(t)$ will collaborate only if the credit they obtain from doing so is greater than the "outside option" reward $\alpha_{\pi(t)}$. We want to design a mechanism that guarantees collaboration whenever possible. Our data-sharing mechanisms will depend on $\{\lambda_{\pi,\pi(t)}\}$, so we often write $M(\mathbf{x}, \alpha, \lambda)$ instead of $M(\mathbf{x}, \alpha)$ to emphasize the dependence on the learning bound vector.

Given the above definition, we can define the following equilibrium concept.

Definition 4.4. *Let $(\mathbf{x}, \alpha, \lambda) \in X \times A \times \Lambda$ and $(\pi, \mathbf{y}) = M_f(\mathbf{x}, \alpha, \lambda)$. Let $\mathcal{I}(\pi, \mathbf{y})$ be the set of inferred publications from (π, y). We say that $(\pi, \mathbf{y})$ is a* collaborative Nash equilibrium *if $\forall \mathbf{z} \in \mathcal{I}(\pi, \mathbf{y})$ we have $\beta^t(Var(z_{\pi(t-1)}) - Var(z_{\pi(t)})) \geq \alpha_{\pi(t)}$.*

Our goal is to find data-sharing mechanisms M_f for which collaboration is an equilibrium. Although we will show that this problem is, in general, NP-complete, there is a very large subset of learning vectors for which we can efficiently find a collaborative Nash equilibrium, if one exists. The feasible case corresponds to when there is a bound μ_j on the amount that *any* player can learn from player j's publication. Formally, we define a learning bound vector to be *rank*-1 if it satisfies this property.

Definition 4.5. *We say that a learning vector $\lambda \in \Lambda$ is rank-1 if there exists a non-negative vector $\{\mu_j\}_{j=1}^n$ such that $\lambda_{\pi,\pi(t)} = \sum_{\tau=1}^{t-1} \mu_\tau$. We denote by $\Lambda_1 \subset \Lambda$ the set of all rank-1 learning vectors.*

When λ is a *rank*-1 learning vector, the amount that any player i learns from publication j is bounded above by μ_j. Thus, the total amount that player $\pi(t)$ learns from all prior publications is $\sum_{\tau=1}^{t-1} \mu_\tau$. Below, we show that when λ is a *rank*-1 vector, we can find a collaborative Nash equilibrium in polynomial time.

Theorem 4.6. *There exists a polynomial-time data-sharing mechanism* SHARE-DATA $: X \times A \times \Lambda_1$ *that, given inputs* $(\mathbf{x}, \alpha, \lambda)$ *where* λ *is a rank*-1 *learning vector, outputs a collaborative Nash equilibrium* $(\pi, \mathbf{y})$ *if such an equilibrium exists.*

SHARE-DATA$(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_n, \mu_1, \ldots, \mu_n)$

- Let $y = f(x_1 \ldots, x_n)$ and $\delta_0 = Var(y_0)$.
- Construct a complete weighted bipartite graph $G = (L, R, E)$ where $L = [n], R = [n], E = L \times R$. For each edge (i, t), assign a weight $w(i, t) = \frac{\alpha_i}{\beta^t} + (n - t)\mu_i$
- Let M be the minimum-weight perfect matching on G. For each node $t \in R$, let $\pi(t) \in L$ be the node that it is matched with. If the weight of M is larger than δ_0, output `NONE`. Else, define $\delta_{\pi(n)} = 0$, $y_{\pi(n)} = y$.
- For t from n to 2:
  - Let $\delta_{\pi(t-1)} = \delta_{\pi(t)} + \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{\tau=1}^{t-2} \mu_{\pi(\tau)}$.
  - Let $y_{\pi(t-1)} = y_{\pi(t)} + Normal(0, \delta_{\pi(t-1)} - \delta_{\pi(t)})$.[12]
- Output $\omega = (\pi, (y_{\pi(1)}, \ldots, y_{\pi(n)}))$.

The theorem follows from Lemmas 4.7 and 4.8 and Theorem 4.9, given below.

Lemma 4.7. *If there exists a collaborative Nash equilibrium* $(\pi, \mathbf{y})$ *given* $(\mathbf{x}, \alpha, \lambda)$ *then the following condition holds* $\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n} (n - t)\mu_{\pi(t)} \leq Var(y_0)$ *where* $(\mu_1, \ldots, \mu_n)$ *is a vector such that* $\lambda_{\pi,\pi(t)} = \sum_{\tau=1}^{t-1} \mu_\tau$.

Proof. Let $(\pi, \mathbf{y})$ be a proposed collaborative equilibrium, and let $\mathbf{z} \in \mathcal{I}(\pi, \mathbf{y})$ be a possible vector of inferred publications. For every t, we must have that

$$\beta^t (Var(z_{\pi(t-1)}) - Var(z_{\pi(t)})) \geq \alpha_{\pi(t)}$$

$$Var(z_{\pi(t-1)}) - Var(z_{\pi(t)}) \geq \frac{\alpha_{\pi(t)}}{\beta^t}.$$

The worst case for player $\pi(t)$ is when player $\pi(t-1)$ learns as much as possible from prior publications and player $\pi(t)$ learns as little as possible. That is

$$Var(z_{\pi(t-1)}) = Var(y_{\pi(t-1)}) - \mu_{\pi(1)} - \cdots - \mu_{\pi(t-2)}$$

$$Var(z_{\pi(t)}) = Var(y_{\pi(t)})$$

[12] Here $Normal(0, \delta_{\pi(t-1)} - \delta_{\pi(t)})$ is a normal random variable with mean 0 and variance $\delta_{\pi(t-1)} - \delta_{\pi(t)}$. Note that by construction, the random variables $y_{\pi(1)}, \ldots, y_{\pi(n)}$ are correlated in a way that player $\pi(t)$ cannot learn anything more from observing $y_{\pi(1)}, \ldots, y_{\pi(t-2)}, y_{\pi(t-1)}$ that he didn't know from observing $y_{\pi(t)}$. However, player $\pi(t)$ actually observes $z_{\pi(1)}, \ldots, z_{\pi(t-1)}$ as well as π, which may contain some extra information about $x_{\pi(1)}, \ldots, x_{\pi(t-1)}$ that is not revealed by $y_{\pi(1)}, \ldots, y_{\pi(t-1)}$. Thus, the extra learning captured by $\mathcal{I}(\pi, \vec{y})$ does not come from observing the "independent draws" $y_{\pi(1)}, \ldots, y_{\pi(t-1)}$ (which by construction are not independent), but by observing the leaked information about $x_{\pi(\tau)}$ contained in $z_{\pi(\tau)}$.

In this case, the equilibrium condition becomes $Var(y_{\pi(t-1)}) - \sum_{\tau=1}^{t-2} \mu_{\pi(\tau)} - Var(y_{\pi(t)}) \geq \frac{\alpha_{\pi(t)}}{\beta^t}$. Letting $\delta_{\pi(t)} = Var(y_{\pi(t)})$, we have $\delta_{\pi(t)} - \delta_{\pi(t-1)} \leq -\frac{\alpha_{\pi(t)}}{\beta^t} - \sum_{\tau=1}^{t-2} \mu_{\pi(\tau)}$.

Summing over all t yields $\delta_{\pi(n)} - \delta_{\pi(0)} \leq -\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} - \sum_{t=1}^{n}(n-t)\mu_{\pi(t)}$, where $\delta_{\pi(0)} = Var(y_0)$ is the variance of the prior knowledge (and can be interpreted as the "size of the pie" of credit to be allocated). Flipping the signs in the inequality, the existence of a collaborative equilibrium implies

$$\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n}(n-t)\mu_{\pi(t)} \leq Var(y_0) - \delta_{\pi(n)} \leq Var(y_0)$$

□

Lemma 4.8. *If there exists a permutation π such that $\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n}(n-t)\mu_{\pi(t)} \leq Var(y_0)$, there is a collaborative Nash equilibrium $(\pi, \mathbf{y})$ of the game specified by $(\mathbf{x}, \alpha, \lambda)$.*

Proof. Note that we have that the sufficient conditions for $(\pi, \mathbf{y})$ being a collaborative Nash equilibrium

$$\delta_{\pi(t-1)} - \delta_{\pi(t)} \geq \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{\tau=1}^{t-2} \mu_{\pi(\tau)}$$

for all t, where $\delta_{\pi(t)} = Var(y_{\pi(t)})$.

We need to construct $(y_1, \ldots, y_n)$ such that $(\pi, \mathbf{y})$ is a Nash equilibrium. We construct $\mathbf{y}$ inductively as follows: let $\delta_{\pi(n)} = 0$, and for any t such that $2 \leq t \leq n$, let $\delta_{\pi(t-1)} = \delta_{\pi(t)} + \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{\tau=1}^{t-2} \mu_{\pi(\tau)}$. Note that defining $\{\delta_{\pi(t)}\}_{t=1}^{n}$ in this way, if we set $y_i \leftarrow Normal(0, \delta_i)$ then the sufficient conditions hold for all $t \geq 2$. Finally, to check that the condition holds for $t = 1$, note that $\delta_{\pi(0)} = Var(y_0)$, so the condition becomes $Var(y_0) \geq \delta_{\pi(1)} + \frac{\alpha_{\pi(1)}}{\beta}$. Replacing $\delta_{\pi(t-1)} = \delta_{\pi(t)} + \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{\tau=1}^{t-2} \mu_{\pi(\tau)}$ iteratively, we get $Var(y_0) \geq \sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n}(n-t)\mu_{\pi(t)}$ which is guaranteed by assumption. □

Theorem 4.9. *The algorithm* SHARE-DATA *can be computed in polynomial time, and outputs a collaborative Nash equilibrium if it exists, and it outputs* `NONE` *if no such equilibrium exists.*

Proof. The fact that the algorithm runs in polynomial time is immediate, since additions, comparisons, drawing normal random variables and finding minimum weight matchings in a graph [Edm65] can all be done in (randomized) polynomial time.

By our lemmas, a collaborative Nash equilibrium exists if and only if there exists a permutation π such that

$$\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n}(n-t)\mu_{\pi(t)} \leq Var(y_0).$$

Note that our algorithm constructs a complete bipartite graph $G = (L \cup R, E)$ where the weight on every edge is $w(i,t) = \frac{\alpha_i}{\beta^t} + (n-t)\mu_i$. A matching M on this graph induces a permutation π where, for every $t \in R$, we have $\pi(t) = i$ such that $(i,t) \in M$. The weight of such a matching is

$$\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n}(n-t)\mu_{\pi(t)}.$$

Thus, there exists a collaborative equilibrium if and only if the maximum-weight matching in G has weight less than or equal to $Var(y_0)$. Note that when the weight of the maximum-matching is greater than $Var(y_0)$, our algorithm outputs `NONE`, indicating such an equilibrium does not exist.

Finally, when the weight of the maximum matching is less than or equal to $Var(y_0)$, the algorithm outputs $(\pi, \mathbf{y})$ where $\mathbb{E}[y_{\pi(t)}] = y$ and $Var(y_{\pi(t-1)}) - Var(y_{\pi(t)}) = \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{\tau=1}^{t-2} \mu_{\pi(\tau)}$, so the sufficient conditions for $(\pi, \mathbf{y})$ to be an equilibrium are satisfied. □

Remark. Note that by construction, the random variables $y_{\pi(1)}, \ldots, y_{\pi(n)}$ are correlated in a way that player $\pi(t)$ cannot learn anything more from observing $y_{\pi(1)}, \ldots, y_{\pi(t-2)}, y_{\pi(t-1)}$ that he didn't know from observing $y_{\pi(t)}$. However, player $\pi(t)$ actually observes $z_{\pi(1)}, \ldots, z_{\pi(t-1)}$ as well as π, which may contain some extra information about $x_{\pi(1)}, \ldots, x_{\pi(t-1)}$ that is not revealed by $y_{\pi(1)}, \ldots, y_{\pi(t-1)}$. Thus, the extra learning captured by $\mathcal{I}(\pi, \vec{y})$ does not come from observing the "independent draws" $y_{\pi(1)}, \ldots, y_{\pi(t-1)}$ (which by construction are not independent), but by observing the leaked information about $x_{\pi(\tau)}$ contained in $z_{\pi(\tau)}$.

Finally, one may wonder if we can get an efficient mechanism for learning vectors which are not *rank*-1. We show that this is unlikely, since finding a collaborative Nash equilibrium is NP-complete even under a weak generalization of *rank*-1 learning vectors.

Definition 4.10. *We say that a learning vector $\lambda \in \Lambda$ is rank-2 if there exists a non-negative matrix $\{\mu_{ij}\}_{i,j=1}^n$ such that $\lambda_{\pi,\pi(t)} = \sum_{\tau=1}^{t-1} \mu_{\pi(t),\pi(\tau)}$. We denote by $\Lambda_2 \subset \Lambda$ the set of all rank-2 learning vectors.*

When λ is a *rank*-2 learning vector, the amount that player $\pi(t)$ learns from publication $\pi(\tau)$ is bounded above by $\mu_{\pi(t),\pi(\tau)}$. Thus, the total amount that player $\pi(t)$ learns from all prior publications is $\sum_{\tau=1}^{t-1} \mu_{\pi(t),\pi(\tau)}$. The corresponding necessary condition for a collaborative equilibrium is that there is a permutation π such that

$$\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n} \sum_{s>t} \mu_{\pi(s),\pi(t)} \leq Var(y_0).$$

We show that even checking whether this condition holds is NP-complete.

Theorem 4.11. *Given inputs y_0,α and $\{\mu_{ij}\}_{i,j=1}^n$, it is NP-complete to decide whether there exists π such that*

$$\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n} \sum_{s>t} \mu_{\pi(s),\pi(t)} \leq Var(y_0).$$

Proof. It is clear that the problem is in NP, since we can always guess a permutation π and check in polynomial time whether the condition holds.

To show that the problem is NP-hard, we reduce it to the *minimum weighted feedback arcset problem*. The unweighted version of this problem was shown to be NP-complete by Karp [Kar72], and the weighted version is also NP-complete [ENSS95].

$Minimum - Weight - Feedback - Arcset$

- `INPUTS`: A graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}_{\geq 0}$, a threshold $\gamma \in \mathbb{R}_{\geq 0}$
- `OUTPUT`: Whether or not there exists a set $S \subset E$ of edges which intersects every cycle of G and has weight less than γ.

All that we need to show is that, given a graph G, a set S of edges is a feedback arcset if and only if there exists a permutation π of the vertices of V such that $S = \{(\pi(t), \pi(s)) \in E : s < t\}$.

To see this, note that if π is a permutation and $S = \{(\pi(t), \pi(s)) \in E : s < t\}$ then the set S intersects every cycle of G. This is because, if $C = \{(\pi(i_1), \pi(i_2)), (\pi(i_2), \pi(i_3), \ldots, (\pi(i_k), \pi(i_1))\}$ is a cycle in G, then there must exist s, t such that $s < t$ and $(\pi(t), \pi(s)) \in C$, so S intersects C. Thus, S is a feedback arcset.

Conversely, if S is a feedback arcset, then $G' = (V, E - S)$ is a directed acyclic graph, and we can induce an ordering π on V following topological sort. Any edge $(\pi(t), \pi(s)) \in E - S$ must satisfy $t < s$. Thus, any edge $(\pi(t), \pi(s))$ where $s < t$ must be in S. Thus, given π from the topological sort, we must have $S \supset \{(\pi(t), \pi(s)) \in E : s < t\}$. Since weights are non-negative, the minimal feedback arcset S^* will correspond to a permutation π^* such that $S^* = \{(\pi^*(t), \pi^*(s)) \in E : s < t\}$.

We show how to reduce $Minimum - Weight - Feedback - Arcset$ to our problem. Given $G = (V, E)$, $w : E \to \mathbb{R}_{\geq 0}$ and $t \in \mathbb{R}_{\geq 0}$, let $Var(y_0) = \gamma$, $\alpha = 0$, and $\mu_{ij} = w(i, j)$ if $(i, j) \in E$ and $\mu_{ij} = 0$ otherwise.

Suppose there exists a permutation π such that

$$\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n} \sum_{s>t} \mu_{\pi(s),\pi(t)} \leq Var(y_0).$$

Plugging in our choices of $\alpha, \mu_{ij}, Var(y_0)$, this becomes

$$\sum_{(\pi(s),\pi(t)) \in E : s>t} w(\pi(s), \pi(t)) \leq \gamma.$$

Since the set $S = \{(\pi(s), \pi(t)) \in E : s > t\}$ is a feedback arcset, we have that there exists a feedback arcset with weight less than γ.

Conversely, assume no such permutation π exists. That is,

$$\sum_{(\pi(s),\pi(t)) : s>t} \mu_{\pi(s),\pi(t)} > Var(y_0)$$

for all permutations π. Note that whether s comes before t or vice-versa does not matter, since this inequality holds for all permutations. Thus, we can also write $\sum_{(\pi(s),\pi(t)) : s<t} \mu_{\pi(s),\pi(t)} > Var(y_0)$ for all permutations π. From the argument above, the minimum weight feedback arcset S^* induces a permutation π^* such that $S^* = \{(\pi^*(t), \pi^*(s)) \in E : s < t\}$. The weight of S^* is

$$\sum_{(\pi^*(s),\pi^*(t)) \in E : s<t} w(\pi^*(s), \pi^*(t)) = \sum_{(\pi^*(s),\pi^*(t)) : s<t} \mu_{\pi^*(s),\pi^*(t)} > Var(y_0) = t.$$

Thus, there does not exist a feedback arcset with weight less than or equal to t.

We conclude that if we can efficiently check whether

$$\sum_{t=1}^{n} \frac{\alpha_{\pi(t)}}{\beta^t} + \sum_{t=1}^{n} \sum_{s>t} \mu_{\pi(s),\pi(t)} \leq Var(y_0),$$

then we can efficiently check whether there exists a feedback arcset S with weight less than t. Thus, the feedback arcset problem reduces to ours, and our problem is NP-complete. □

We have shown that in our stylized model of scientific collaboration, it can indeed be very beneficial to *all parties involved* to collaborate under certain ordering functions, and such beneficial orderings can be efficiently computed in certain realistic situations (but probably not in the general case).

Acknowledgements

We would like to thank Yehuda Lindell for an interesting discussion on the nature of fairness in multiparty computation.

References

[BGK14] Siddhartha Banerjee, Ashish Goel, and Anilesh Kollagunta Krishnaswamy. "Re-incentivizing discovery: mechanisms for partial-progress sharing in research". In: *ACM Conference on Economics and Computation, EC '14, Stanford , CA, USA, June 8-12, 2014*. Ed. by Moshe Babaioff, Vincent Conitzer, and David Easley. ACM, 2014, pp. 149–166. ISBN: 978-1-4503-2565-3. DOI: 10.1145/2600057.2602888. URL: http://doi.acm.org/10.1145/2600057.2602888.

[Bit+] Nir Bitansky et al. *Time-Lock Puzzles from Randomized Encodings*. Manuscript.

[BN00] Dan Boneh and Moni Naor. "Timed Commitments". In: *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Springer, 2000, pp. 236–254. ISBN: 3-540-67907-3. DOI: 10.1007/3-540-44598-6_15. URL: http://dx.doi.org/10.1007/3-540-44598-6_15.

[CDP14] Yang Cai, Constantinos Daskalakis, and Christos Papadimitriou. "Optimum Statistical Estimation with Strategic Data Sources". In: *ArXiv e-prints* (2014). arXiv: 1408.2539 [stat.ML].

[Cle86] Richard Cleve. "Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract)". In: *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*. Ed. by Juris Hartmanis. ACM, 1986, pp. 364–369. ISBN: 0-89791-193-8. DOI: 10.1145/12130.12168. URL: http://doi.acm.org/10.1145/12130.12168.

[Edm65] Jack Edmonds. "Paths, trees, and flowers". In: *Canadian Journal of mathematics* 17.3 (1965), pp. 449–467.

[ENSS95] Guy Even, Joseph(Seffi) Naor, Baruch Schieber, and Madhu Sudan. "Approximating minimum feedback sets and multi-cuts in directed graphs". English. In: *Integer Programming and Combinatorial Optimization*. Ed. by Egon Balas and Jens Clausen. Vol. 920. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1995, pp. 14–28. ISBN: 978-3-540-59408-6. DOI: 10.1007/3-540-59408-6_38. URL: http://dx.doi.org/10.1007/3-540-59408-6_38.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. Ed. by Alfred V. Aho. ACM, 1987, pp. 218–229. ISBN: 0-89791-221-7. DOI: 10.1145/28395.28420. URL: http://doi.acm.org/10.1145/28395.28420.

[Han12] Robin Hanson. "Logarithmic Market Scoring Rules For Modular Combinatorial Information Aggregation". In: *The Journal of Prediction Markets* 1.1 (2012), pp. 3–15.

[Kar72] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[KO11] Jon M. Kleinberg and Sigal Oren. "Mechanisms for (mis)allocating scientific credit". In: *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*. Ed. by Lance Fortnow and Salil P. Vadhan. ACM, 2011, pp. 529–538. ISBN: 978-1-4503-0691-1. DOI: 10.1145/1993636.1993707. URL: http://doi.acm.org/10.1145/1993636.1993707.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. "Publicly verifiable proofs of sequential work". In: *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*. Ed. by Robert D. Kleinberg. ACM, 2013, pp. 373–388. ISBN: 978-1-4503-1859-4. DOI: 10.1145/2422436.2422479. URL: http://doi.acm.org/10.1145/2422436.2422479.

[May93] Timothy C. May. *Timed-release crypto*. 1993. URL: http://www.hks.net/cpunks/cpunks-01460.html.

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. *Time-lock puzzles and timed-release crypto*. Tech. rep. 1996.

[Wel03] The Wellcome Trust. *Sharing Data from Large-scale Biological Research Projects: A System of Tripartite Responsibility*. Report of a meeting organized by the Wellcome Trust and held on 14âĂŞ15 January 2003 at Fort Lauderdale, USA. 2003.

A MPC security definition

IDEAL FUNCTIONALITY $\mathcal{F}_{\text{MPC}}$

In the ideal model, a trusted third party T is given the inputs, computes the function f on the inputs, and outputs to each player i his output y_i. In addition, we model an ideal process adversary $\mathcal{S}$ who attacks the protocol by corrupting players in the ideal setting.

Public parameters. $\kappa \in \mathbb{N}$, the security parameter; $n \in \mathbb{N}$, the number of parties; and $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, the function to be computed.

Private parameters. Each player $i \in [n]$ holds a private input $x_i \in \{0,1\}^*$.

1. INPUT. Each player i sends his input x_i to T.
2. COMPUTATION. T computes $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$.
3. OUTPUT. For each $i \in [n]$, T sends the output value y_i to party i.

4. OUTPUT OF VIEWS. After the protocol terminates, each party outputs a *view* of the computation. Each uncorrupted party i outputs y_i if he has received his output, or $\perp$ if not. Each corrupted party outputs $\perp$. Additionally, the adversary $\mathcal{S}$ outputs an arbitrary function of the information that he has learned during the execution of the ideal protocol, which comprises all the messages seen by the corrupt players, and the outputs that they received.

   Let the view outputted by party i be denoted by $\mathscr{V}_i$, and let the view outputted by $\mathcal{S}$ be denoted by $\mathscr{V}_{\mathcal{S}}$. Let $\mathscr{V}^{\mathsf{ideal}}_{\mathrm{MPC}}$ denote the collection of all the views:

$$\mathscr{V}^{\mathsf{ideal}}_{\mathrm{MPC}} = (\mathscr{V}_{\mathcal{S}}, \mathscr{V}_1, \ldots, \mathscr{V}_n)\,.$$

Definition A.1 (Security)**.** *A multi-party protocol F is said to securely realize $\mathcal{F}_{\mathrm{MPC}}$ if for any* PPT *adversary $\mathcal{A}$ attacking the protocol F by corrupting a subset of players $S \subset [n]$, there is a* PPT *ideal adversary $\mathcal{S}$ which, given access to $\mathcal{F}_{\mathrm{MPC}}$ and corrupting the same subset S of players, can output a view $\mathscr{V}_{\mathcal{S}}$ such that*

$$V_{\mathcal{A}} \overset{c}{\approx} \mathscr{V}_{\mathcal{S}},$$

where $V_{\mathcal{A}}$ is the view outputted by the real-world adversary $\mathcal{A}$ (this may be an arbitrary function of the information that $\mathcal{A}$ learned in the protocol execution).

B Message authentication codes

Definition B.1 (Message authentication code (MAC))**.** *A* message authentication code $\mathsf{MAC} = \{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}\}$ *is a tuple of* PPT *algorithms as follows:*

- $\mathsf{Gen}(1^\kappa)$ *takes as input $\kappa \in \mathbb{N}$, the security parameter, and outputs a secret key $k \in \mathcal{K}$.*
- $\mathsf{Sign}_k(m)$ *takes as input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs an authentication tag $\sigma \in \mathcal{U}$.*
- $\mathsf{Verify}_k(m, \sigma')$ *takes a input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and an authentication tag $\sigma \in \mathcal{U}$, and outputs a value in $\{\mathsf{true}, \mathsf{false}\}$.*

We call $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{U}$ the key space*,* message space*, and* tag space*, respectively.*

Definition B.2 (Correctness)**.** *A message authentication code $\mathsf{MAC} = \{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}\}$ satisfies* correctness *if for all $m \in \mathcal{M}$, it holds that*

$$\Pr\left[\mathsf{Verify}_k\left(m, \sigma\right) = \mathsf{false} \mid k \leftarrow \mathsf{Gen}(1^\kappa) \text{ and } \sigma \leftarrow \mathsf{Sign}_k(m)\right] \leq \mathsf{negl}(\kappa).$$

Next, we describe a security experiment that will be used in the security definition that follows.

The experiment $\mathsf{ForgingExp}_{\mathcal{A},\mathsf{MAC}}(\kappa)$

1. The challenger generates a key $k \leftarrow \mathsf{Gen}(1^\kappa)$.
2. $\mathcal{A}$ has oracle access to $\mathsf{Sign}_k(\cdot)$ and $\mathsf{Verify}_k(\cdot)$, and can make polynomially many queries to these.

3. The output of the experiment is 1 if $\mathcal{A}$ made a query (m^*, σ^*) to Verify_k, where
   - $\mathsf{Verify}_k(m^*, \sigma^*) = \mathsf{true}$, and
   - $\mathcal{A}$ did not already query Sign_k on input m^*.

   Otherwise, the output of the experiment is 0.

Definition B.3 (Security)**.** *A message authentication code* $\mathsf{MAC} = \{\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}\}$ *is* secure *if for all* PPT *adversaries* $\mathcal{A}$*, it holds that*

$$\Pr\left[\mathsf{ForgingExp}_{\mathcal{A},\mathsf{MAC}}(\kappa) = 1\right] \leq 1/2 + \mathsf{negl}(\kappa).$$