Truth, Justice, and Secrecy: Cake Cutting Under Privacy Constraints

Yaron Salman[1*], Tamir Tassa[1*], Omer Lev[2], Roie Zivan[2]

[1]The Open University of Israel
[2]Ben-Gurion University of the Negev
sayaron8@post.openu.ac.il, tamirta@openu.ac.il, omerlev@bgu.ac.il, zivanr@bgu.ac.il

Abstract

Cake-cutting algorithms, which aim to fairly allocate a continuous resource based on individual agent preferences, have seen significant progress over the past two decades. Much of the research has concentrated on fairness, with comparatively less attention given to other important aspects. Chen et al. (2010) introduced an algorithm that, in addition to ensuring fairness, was strategyproof—meaning agents had no incentive to misreport their valuations. However, even in the absence of strategic incentives to misreport, agents may still hesitate to reveal their true preferences due to privacy concerns (e.g., when allocating advertising time between firms, revealing preferences could inadvertently expose planned marketing strategies or product launch timelines). In this work, we extend the strategyproof algorithm of Chen et al. by introducing a privacy-preserving dimension. To the best of our knowledge, we present the first private cake-cutting protocol, and, in addition, this protocol is also envy-free and strategyproof. Our approach replaces the algorithm's centralized computation with a novel adaptation of cryptographic techniques, enabling privacy without compromising fairness or strategyproofness. Thus, our protocol encourages agents to report their true preferences not only because they are not incentivized to lie, but also because they are protected from having their preferences exposed.

1 Introduction

For millennia, people have grappled with the challenge of dividing resources—whether land, water, loot, or other valuable commodities—in a way that is seen as fair. The modern formalism of this problem—*cake-cutting*—has existed for only a little over 80 years, yet it has found wide application across domains where multiple agents (human or otherwise) assign different values to different parts of a divisible resource. From scheduling shared computer time to allocating energy from power sources, such problems are all instances of cake-cutting, where the goal is to divide the resource fairly among the agents involved.

Naturally, the issue of what is "fair" has been widely discussed, with the most basic concepts being *proportionality*, in which each agent gets at least $\frac{1}{n}$ of the value of the whole (assuming n agents overall). A stronger concept is an *envy-free* allocation, in which each agent is most satisfied with its allocation and has no reason to switch and get a different agent's allocation. There have been many further fairness concepts, and various algorithms have been suggested, aiming for strong guarantees for as wide as possible family of valuation functions, with as low as possible computational complexity (Brams and Taylor 1996; Robertson and Webb 1998; Brandt et al. 2016).

Although a major open problem in the field—envy-free cake-cutting for $n > 3$ agents—was finally solved in the last decade (Aziz and Mackenzie 2016), there are many other dimensions to the problem. As with voting, an algorithm guaranteeing many properties may be worthless if agents do not report to the algorithm their true valuations. Envy-free allocations of misreported valuations might not be, in practice, envy-free for the genuine, true valuations. Hence, strategyproof algorithms, in which agents are never hurt by being truthful, are highly desirable.

Work on strategyproof algorithms began with Chen et al. (2010, 2013), showing a deterministic algorithm for piecewise uniform valuations (e.g., each agent has times when it can use the computer and times when it cannot), which we denote in this paper as CC_PUV; and a randomized algorithm for piecewise linear valuations. Since that work, there has been additional work on strategyproof algorithms (Mossel and Tamuz 2010; Maya and Nisan 2012; Menon and Larson 2017; Bei et al. 2017; Bei, Huzhang, and Suksompong 2020). Some work explicitly focused on piecewise uniform, piecewise constant and piecewise linear valuations, as it became quite clear that a deterministic strategyproof algorithm may be limited to a subset of these, with a few additional constraints (Aziz and Ye 2014; Bu, Song, and Tao 2023).

However, while strategyproofness is a desired property, it does not, in fact, completely solve the issue of convincing agents to be truthful and thus obtain the algorithm's guarantees. In many cases, agents do not wish to share with others their valuations for reasons that can be regulatory (e.g., data protection laws in healthcare may restrict revealing valuations tied to patient information), personal (e.g., when dividing time to use a game console, one may not wish to reveal to their employer that they are playing during work hours) or due to external effects (e.g., telecom companies bidding on spectrum fear competitors will glean from their bids their

*The first two authors contributed equally to this work.

future plans and directions). In such cases, agents may withhold their preferences even from a strategyproof algorithm if its operation exposes those preferences to other agents, despite this reducing the algorithm's effectiveness (e.g., invalidating envy-freeness guarantees).

In this paper, we propose what we believe to be the **first private cake-cutting protocol**, named PP_CC_PUV. This protocol is a privacy-preserving variant of Chen et al. (2010, 2013)'s algorithm and retains its properties of envy-freeness, Pareto-optimality, and strategyproofness (for piecewise uniform valuations). In fact, this is the first protocol that genuinely allows agents to reveal their true valuations without hesitation. It accomplishes this by using secret sharing and secure multiparty computation (MPC) techniques. Secret sharing is used to hide information such as the preferences of agents, the interim results of the computation performed by the algorithm, and the final allocations, while MPC is used to securely perform the computation of the algorithm without exposing the agents' private information.

Achieving this private variant is not simply a matter of inserting standard cryptographic replacements into the original algorithm; it requires novel adaptations of cryptographic techniques. In addition, the algorithm itself had to be restructured. For example, the algorithm of Chen et al. (2010, 2013) constructs a different graph in each iteration, computes a maximum flow in it, and then derives allocations from it. However, the structure of those graphs in each of the iterations might leak sensitive valuation information. To address this, our protocol replaces the dynamic graph with a fixed graph used across all iterations, with edge weights computed cryptographically. Despite this additional complexity, the private variant incurs only $\mathcal{O}(1)$ computational overhead and at most $\mathcal{O}(n^2)$ additional communication cost.

Beyond privacy, our protocol also allows for the removal of a central coordinating agent to run the algorithm as the participating agents can run it for themselves. This flexibility, requiring no algorithm infrastructure besides the agents, allows cake-cutting algorithms to run on much wider and ad-hoc scenarios, from allocating work shifts between employees (who do not trust the boss to be fair) to computer resource allocation without a pre-existing time-allocation daemon running in the background.

While this is the first private cake-cutting algorithm, we believe that the techniques and ideas it introduces can be extended to "privatize" a broader range of cake-cutting algorithms. Privacy is important in many settings, and we believe that incorporating privacy into the suite of desiderata for resource allocation algorithms is increasingly vital. This is true even when strategyproofness is unnecessary (e.g., for non-strategic agents), as agents may still be reluctant to disclose their valuations to others. A natural direction for future work is to extend these ideas to general-purpose cake-cutting algorithms, which are applicable to large classes of valuation functions and stand to benefit significantly from privacy-preserving adaptations that could expand their practical usability.

Due to space limitations, additional technical details and all pseudocode are provided in the full version of this paper (Salman et al. 2025).

2 Cryptographic Background

Here we briefly review the two fundamental cryptographic techniques underpinning our protocol—secret sharing (Section 2.1) and multiparty computation (MPC) over secret shares (Section 2.2)—and describe the security model and privacy guarantees (Section 2.3).

2.1 Threshold Secret Sharing

Secret sharing schemes (Shamir 1979) are protocols that enable distributing a secret scalar s among a set of parties, $P_1, \ldots, P_n$. Each party, P_i, $i \in [n]$[1], gets a random value $[[s]]_i$, called *a share*, so that some subsets of those shares enable the reconstruction of s, while each of the other subsets of shares reveals no information on s. In its most basic form, called *Threshold Secret Sharing*, there is a threshold value $t \leq n$, and then a subset of shares enables the reconstruction of s iff its size is at least t.

Shamir's t-out-of-n threshold secret sharing scheme (Shamir 1979) operates over a finite field $\mathbb{F}_p$, where $p > n$ is a prime sufficiently large so that all possible secrets may be represented in $\mathbb{F}_p$. It has two procedures—Share and Reconstruct:

- $\text{Share}_{t,n}(s)$. The procedure samples a uniformly random polynomial $f(\cdot)$ over $\mathbb{F}_p$, of degree at most $t-1$, where the constant term is the secret s. The procedure outputs n values, $[[s]]_i = f(i)$, $i \in [n]$, where $[[s]]_i$ is the share given to P_i. The entire set of shares, denoted $[[s]] := \{[[s]]_i : i \in [n]\}$, is called a (t, n)-sharing of s.
- $\text{Reconstruct}_t([[s]])$. The procedure is given any selection of t shares out of $[[s]]$—the (t, n)-sharing of s. It then interpolates a polynomial $f(\cdot)$ of degree at most $t-1$ using the given points and outputs $s = f(0)$. Any selection of t shares will yield the secret s, as t points determine a unique polynomial of degree at most $t-1$. On the other hand, any selection of $t-1$ shares or less reveals nothing about the secret s.

Hereinafter, we set the threshold to be

$$t := \lfloor (n+1)/2 \rfloor . \tag{1}$$

Namely, to reconstruct the secret, at least half of the parties must collaborate. Hence, if the set of n parties has an honest majority, in the sense that more than half of them act honestly, without trying to collude to reconstruct the secret illicitly, the shared secret will remain fully protected.

2.2 Multiparty Computation Over Secret Shares

Let u and v be two secret values in the field $\mathbb{F}_p$, and assume that $P_1, \ldots, P_n$ hold (t, n)-sharings of them, denoted $[[u]] = \{[[u]]_i : i \in [n]\}$ and $[[v]] = \{[[v]]_i : i \in [n]\}$.

We are concerned here with performing secure computations over those shared values in the following sense: if $G(\cdot, \cdot)$ is a publicly known function, the goal is to use given (t, n)-sharings $[[u]]$ and $[[v]]$ in order to compute a (t, n)-sharing of $G(u, v)$, without learning any information on u, v, or $G(u, v)$. We consider here basic functions that enable performing a wide range of more elaborate computations.

[1]For any integer n we let $[n]$ denote the set $\{1, \ldots, n\}$.

Affine combinations. Let $\alpha, \beta, \gamma \in \mathbb{F}_p$ be three values publicly known to all. Then

$$[[\alpha]] + \beta[[u]] + \gamma[[v]] := \{\alpha + \beta[[u]]_i + \gamma[[v]]_i : i \in [n]\}$$

is a proper (t, n)-sharing of $w := \alpha + \beta u + \gamma v$, thanks to the affinity of secret sharing. By writing $[[w]] \leftarrow [[\alpha]] + \beta[[u]] + \gamma[[v]]$ we mean that each party P_i, $i \in [n]$, sets $[[w]]_i \leftarrow \alpha + \beta[[u]]_i + \gamma[[v]]_i$, so that now the parties hold a (t, n)-sharing of $w = \alpha + \beta u + \gamma v$, without needing to interact or to perform any further polynomial computations.

By choosing $\beta = \gamma = 0$, it follows that for any public value α in the field, the set $\{[[\alpha]]_i = \alpha : i \in [n]\}$ constitutes a valid (t, n)-secret sharing of α.

Multiplications. A secure multiplication protocol, $[[w]] \leftarrow [[u]] \cdot [[v]]$, takes (t, n)-sharings of u and v and computes a (t, n)-sharing of $w = u \cdot v$, without revealing to the parties any information on u, v, or $w = uv$. Damgård and Nielsen (2007) designed such a secure multiplication protocol, which was later improved by Chida et al. (2018).

Comparisons. Hereinafter, if $\mathcal{P}$ is a predicate then $1_{\mathcal{P}}$ is a bit that equals 1 if the predicate $\mathcal{P}$ holds and equals 0 otherwise. Then a secure comparison protocol, $[[w]] \leftarrow [[1_{u<v}]]$, takes (t, n)-sharings of u and v and securely computes a (t, n)-sharing of $w = 1_{u<v}$. Nishide and Ohta (2007) proposed such a secure comparison protocol.

Equality. A secure equality testing protocol, $[[w]] \leftarrow [[1_{u=0}]]$, takes a (t, n)-sharing of u and securely computes a (t, n)-sharing of $w = 1_{u=0}$, see (Kogan, Tassa, and Grinshpoun 2023).

Minimum. A secure minimum protocol, $[[w]] \leftarrow \min([[u]], [[v]])$, takes (t, n)-sharings of u and v, and securely computes a (t, n)-sharing of their minimum, $w = \min(u, v) = v + 1_{u<v} \cdot (u - v)$. Maxima can be computed similarly by the identity $\max(u, v) = u + 1_{u<v} \cdot (v - u)$.

OR. A secure OR protocol, $[[w]] \leftarrow [[u]] \vee [[v]]$, takes (t, n)-sharings of two secret bits, u and v, and securely computes a (t, n)-sharing of the OR between them, $w = u \vee v = u + v - u \cdot v$.

Divisions. A secure division protocol, $\langle [[q]], [[r]] \rangle \leftarrow [[u]]/[[v]]$, takes (t, n)-sharings of two secret integers, u and v, and securely computes (t, n)-sharings of the corresponding quotient $q = \lfloor \frac{u}{v} \rfloor$ and remainder $r = u \bmod v$. In Salman et al. (2025, Appendix A.1) we present such a protocol.

2.3 Security Model and Privacy Guarantees

Our protocol operates in a standard MPC setting in which the computation is carried out by several semi-honest servers, with an honest majority ensuring that any colluding subset is smaller than half. Agents may attempt to behave dishonestly, but the protocol incorporates integrity checks that prevent them from submitting valuation data that deviates from the required piecewise-uniform structure. Privacy is guaranteed in an information-theoretic sense: no party—neither agents nor servers—learns anything beyond the outputs explicitly revealed by the protocol. In particular, the agents' valuation functions, all intermediate values (including demands, interval selections, and flow computations), and the structure of the underlying decision processes remain completely hidden. The only information that may be disclosed is the final allocation, which can be revealed either in full or only to the corresponding agents, depending on the chosen visibility mode.

3 Strategyproof Cake Cutting Algorithm

We begin by presenting Chen et al. (2010, 2013)'s cake-cutting algorithm, which we refer to as CC_PUV—the Cake Cutting algorithm for piecewise uniform valuations.

The cake to be divided is modeled as the interval $C = [0, 1] \subset \mathbf{R}$. The set of agents that seek to divide the cake between them is $\mathcal{A} := \{A_i : i \in [n]\}$. Agent A_i has a valuation function $v_i : C \to [0, \infty)$ that is a probability density function over C; i.e., it is measurable, nonnegative, and $\int_C v_i(x)dx = 1$. Given any piece of the cake, $X \subseteq C$, its value to A_i is $v_i(X) := \int_X v_i(x)dx$.

CC_PUV assumes all valuations are piecewise uniform:

Definition 3.1 *A valuation $v : C \to [0, \infty)$ is called piecewise uniform if there exist $\ell \geq 1$ disjoint intervals, $\{I_j \subseteq C : j \in [\ell]\}$, such that $v(x) = c$ for all $x \in B := \bigcup_{j \in [\ell]} I_j$, for $c = \left(\sum_{j \in [\ell]} |I_j|\right)^{-1}$, while $v(x) = 0$ for all $x \in C \setminus B$.*

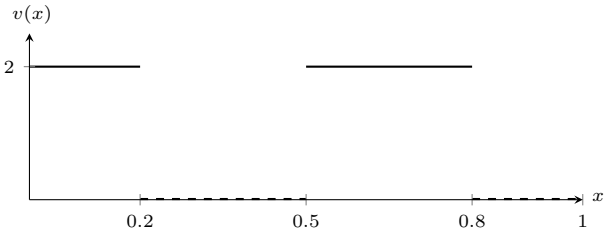

Figure 1: A piecewise uniform valuation with $\ell = 2$ intervals.

Figure 1 illustrates such a function with a support consisting of $\ell = 2$ intervals.

The end points of the support intervals in $v_i(\cdot)$, over all $i \in [n]$, induce a partition of $C = [0, 1]$ into a set of intervals, which we denote by $\mathcal{W}$. Given an interval $I \in \mathcal{W}$ and a subset $S \subseteq \mathcal{A}$ of agents, S desires I if I is included in the support of the valuation of at least one agent in S. If $X \subseteq \mathcal{W}$ then $D(S, X)$ is the subset of all intervals in X that are desired by S, and $\text{avg}(S, X) := \text{Len}(D(S, X))/|S|$ is the average demand of S on X. (If Y is a set of disjoint intervals, then $\text{Len}(Y)$ is the sum of lengths of those intervals.)

The deterministic algorithm of Chen et al. (2010), CC_PUV, operates iteratively. It starts with $X = \mathcal{W}$ and $S = \mathcal{A}$. It then identifies a subset of agents $S' \subseteq S$ (from the current agent set S) with the smallest average demand on the current set of intervals X. The algorithm proceeds to perform an *exact allocation* of $D(S', X)$ among S', giving each agent in S' an equal share of size $\text{avg}(S', X)$ out of those intervals in X that it desires. The algorithm computes these allocations by solving a maximum-flow problem on a capacitated directed graph that encodes the desirability relationships between the intervals in X and the agents

in S'. A detailed explanation of this computation is provided in Section 4.6. The procedure then recurses on the remaining agents and intervals, specifically on $S \setminus S'$ and $X \setminus D(S', X)$, until all agents have been served—that is, until $S = \emptyset$.

4 Privacy-Preserving Cake-Cutting Protocol

In this section, we present our privacy-preserving cake-cutting protocol, which securely implements the cake-cutting algorithm introduced by Chen et al. (2010, 2013). We begin with a high-level overview of our privacy-preserving protocol, PP_CC_PUV—the Privacy-Preserving variant of CC_PUV (Section 4.1). The subsequent sections describe the core computational steps of PP_CC_PUV.

4.1 Overview of PP_CC_PUV

The main challenge in implementing CC_PUV securely is preserving the privacy of the agents' valuations. Our privacy-preserving protocol, Protocol 1 (PP_CC_PUV), addresses this challenge by having each agent distribute secret shares of their piecewise-uniform valuations. It then faithfully emulates CC_PUV by executing carefully tailored MPC subprotocols on these secret shares.

PP_CC_PUV begins with several preparatory steps. First (line 1), the agents execute a subprotocol called SHARINGPRIVATEVALUATIONS, which (a) discretizes the private valuations in a lossless manner, (b) enforces integrity checks to prevent dishonest agents from distributing shares that do not correspond to valid valuations, and (c) distributes secret shares of the private valuations (Section 4.2).

Next (lines 2–3), the secret-shared valuation functions are used to partition the cake $C = [0, 1]$ into a collection of intervals, denoted $\mathcal{W}$, such that each interval lies entirely within the support of v_i or entirely outside it, for every $i \in [n]$. A subprotocol called INTERVALS establishes secret shares of various auxiliary data structures, such as indicator bits specifying, for each agent and interval, whether the agent desires that interval (Sections 4.3 and 4.4).

At this point, PP_CC_PUV enters its core iterative phase (lines 5–8). In each iteration, it executes a subprotocol called ITERATIVEALLOCATION that selects a subset of yet-unserved agents, according to the same selection criterion used by CC_PUV. The key challenge here is to keep the protocol oblivious to which agents have already been served and which intervals remain available (Section 4.5). Once the appropriate subset is (obliviously) selected, the allocation of intervals (or partial intervals) to agents is performed by solving a maximum-flow problem on a graph that encodes the relationships between the relevant agents and intervals. Crucially, the subprotocol ASSIGNCAKETOSELECTEDAGENTS does this while remaining oblivious to the structure of the underlying graph (Section 4.6).

Once the iterative process is complete, the subprotocol FINALSERVING scans all intervals and assigns them to the appropriate agents, based on the computation that took place during the iterative phase (line 9). Depending on the application scenario, the protocol can operate under either restricted visibility, where each agent learns only their own allocated piece, or full visibility, where the complete allocation is disclosed to all agents (Section 4.7).

Protocol 1: PP_CC_PUV

1 Distribute secret shares of private valuations
2 Split the cake into intervals $\mathcal{W}$
3 Compute a $\mathcal{W}$-based representation of the valuations
4 $S \leftarrow [n],\ X \leftarrow \mathcal{W}$
5 **while** $S \neq \emptyset$ **do**
6 Find $S' \subseteq S$ minimizing $\text{avg}(S', X)$ (the average demand of agents in S' over intervals in X)
7 Compute a fair allocation of intervals in $D(S', X)$ (the subset of intervals in X desired by S') to agents in S'
8 $S \leftarrow S \setminus S',\ X \leftarrow X \setminus D(S', X)$
9 Distribute the intervals in $\mathcal{W}$ among the agents

4.2 Secret Sharing the Private Valuation Functions

Here we describe how the agents secret share their private valuation functions. Let ℓ_i denote the number of intervals in v_i's support, $i \in [n]$ and let $\ell = \max_{i \in [n]} \ell_i$. Then the support of v_i can be described by

$$\text{supp}(v_i) = \bigcup_{j=1}^{\ell} I_{i,j}\,, \text{ where } I_{i,j} = [a_{i,j}, b_{i,j})\,,\ j \in [\ell]\,, \text{ and}$$

$$0 \leq a_{i,1} \leq b_{i,1} \leq a_{i,2} \leq b_{i,2} \leq \cdots \leq a_{i,\ell} \leq b_{i,\ell} \leq 1\,. \quad (2)$$

By invoking a secure maximum computation (see Section 2.2), the agents can compute ℓ without disclosing any additional information about ℓ_i, $i \in [n]$. Then, if $\ell_i < \ell$, agent A_i sets the last $\ell - \ell_i$ intervals in v_i's support to the empty interval $[1, 1)$. Such an inflated description of the valuations guarantees that no information is leaked about the number of intervals in the valuations' supports.

The end points of the intervals in those supports, Eq. (2), will be protected by secret sharing. As secret sharing is applied in some finite field, it is essential to convert those real values into integers. To do so, the agents jointly agree upfront on some sufficiently large integer, Q, and then each agent A_i, $i \in [n]$, redefines its end points as follows, $a_{i,j} := \lfloor a_{i,j} \cdot Q \rceil$ and $b_{i,j} := \lfloor b_{i,j} \cdot Q \rceil$, where $\lfloor \cdot \rceil$ denotes the closest integer. By taking $Q = 10^d$, where d is the maximum number of nonzero decimal digits after the decimal point among the numbers $\{a_{i,j}, b_{i,j}\}_{i \in [n], j \in [\ell]}$, such a discretization is lossless. Therefore, henceforth the support of v_i will be described by 2ℓ integers

$$0 \leq a_{i,1} \leq b_{i,1} \leq a_{i,2} \leq b_{i,2} \leq \cdots \leq a_{i,\ell} \leq b_{i,\ell} \leq Q\,. \quad (3)$$

Now, the agents proceed to distribute shares of the 2ℓ integers in Eq. (3), for all $i \in [n]$. Such a procedure must incorporate a computation of $\ell = \max_{i \in [n]} \ell_i$ and of d (as described above), as well as mechanisms to detect possible cheats of dishonest agents.

This is carried out by Subprotocol 4, termed SHARINGPRIVATEVALUATIONS (see Salman et al. (2025, Appendix A.2)). After its completion, the agents hold (t, n)-sharings in $a_{i,j}$ and $b_{i,j}$ for all $i \in [n]$ and $j \in [\ell]$.

4.3 Splitting the Cake to Intervals

Consider the multiset $V := \{a_{i,j}, b_{i,j} : i \in [n], j \in [\ell]\} \cup \{0, Q\}$, of size $m = 2\ell n+2$, consisting of all support boundaries of all valuations, as well as the cake's endpoints. Let $W(1 : m)$ be an array that holds the values in V sorted, namely, $W(1) \leq \cdots \leq W(m)$. Then W defines a collection of $m-1$ disjoint intervals (some of which may be empty) that cover the entire cake:

$$\mathcal{W} := \big\{[W(k), W(k+1)) : k \in [m-1]\big\} \tag{4}$$

Note that $[W(k), W(k+1)) \subset [0, Q]$, and that the actual interval it represents on $C = [0, 1]$ is $[\frac{W(k)}{Q}, \frac{W(k+1)}{Q})$.

After completing Subprotocol SHARINGPRIVATEVALUATIONS, the agents hold (t, n)-sharings of all values in V. They can proceed to compute a secret sharing of $W(k)$, for all $k \in [m]$, by applying a suitable secure protocol that computes the k-th element in a secret-shared dataset, without getting any wiser on the private valuations; in (Salman et al. 2025, Appendix A.3) we describe Subprotocol 5 that performs that computation.

Let $\text{IntervalLen}(k) = W(k+1) - W(k)$ denote the length of the k-th interval, $k \in [m-1]$. Secret shares in those intervals' lengths can be computed by $[[\text{IntervalLen}(k)]] \leftarrow [[W(k+1)]] - [[W(k)]]$. In addition, the protocol maintains secret sharings of the bits $\text{IntervalAvailable}(k)$, $k \in [m-1]$, that indicate whether the k-th interval has not yet been served to any agent. Initially, $[[\text{IntervalAvailable}(k)]] \leftarrow [[1]]$. Once the k-th interval is served, the protocol updates the shares of $\text{IntervalAvailable}(k)$ to reflect the bit update from 1 to 0, in an *oblivious manner*, namely, without knowing that such an update actually took place and to whom that interval was served.

4.4 Encoding the Valuations by Binary Vectors

The intervals in $\mathcal{W}$, Eq. (4), can be used to encode the private valuations by a bit array, $\text{IntervalDesired}(1 : n, 1 : m-1)$, as follows: $\text{IntervalDesired}(i, k) = 1$ iff the k-th interval in $\mathcal{W}$ is desired by A_i. Such desirability holds iff that interval is of positive length and it is contained in v_i's support. Hence,

$$\text{IntervalDesired}(i,k) = \big(1 - 1_{\text{IntervalLen}(k)=0}\big) \cdot \bigvee_{j \in [\ell]} \big(1_{a_{i,j} \leq W(k)} \cdot 1_{W(k+1) \leq b_{i,j}}\big)\,, \tag{5}$$

because $[W(k), W(k+1)) \subseteq \text{supp}(v_i)$ iff $[W(k), W(k+1)) \subseteq [a_{i,j}, b_{i,j})$ for some $j \in [\ell]$.

The agents may compute secret sharings of $\text{IntervalDesired}(i, k)$ for all $i \in [n]$ and $k \in [m-1]$, from the secret sharings that they already hold in $a_{i,j}, b_{i,j}$, $W(k)$, and $\text{IntervalLen}(k)$, using the secure MPC protocols for comparison, multiplication, OR, and equality, as discussed in Section 2.2.

Subprotocol 6 (INTERVALS) in Salman et al. (2025, Appendix A.4) summarizes the interval computations as discussed in Sections 4.3 and 4.4. It computes the secret sharings of the arrays W, IntervalLen, and IntervalDesired, and initializes the secret sharing of the array IntervalAvailable. In addition, it initializes the secret shares of two additional secret arrays that will be used later on: $\text{AllocationDenominator}(1 : n)$, and $\text{IntervalAllocation}(1 : n, 1 : m-1)$. Subprotocol INTERVALS initializes their secret shares to zero. They will be kept secret-shared, and only after all agents have been served they will be used to infer the portion of each of the intervals that would go to any specific agent. Specifically, the fraction

$$\text{Portion}(i,k) := \frac{\text{IntervalAllocation}(i,k)}{\text{AllocationDenominator}(i)} \tag{6}$$

will equal the portion of the k-th interval allocated to agent A_i, $i \in [n]$, $k \in [m-1]$. The update of those arrays will be explained in due time (Section 4.6).

4.5 Selecting a Subset of Agents to Serve

As explained in Section 3, the algorithm CC_PUV functions through an iterative process. It maintains two variables: (1) a subset $S \subseteq [n]$ that represents the agents that have yet to be served a piece of cake; and (b) a subset $X \subseteq \mathcal{W}$ of cake intervals that still have not been served to anyone.

Initially, $S = [n]$ and $X = \mathcal{W}$. In each iteration, the algorithm selects $S' \subseteq S$ that minimizes

$$\text{avg}(S', X) := \text{Len}(S', X)/|S'|\,, \tag{7}$$

where $\text{Len}(S', X)$ is the sum of lengths of all intervals in X that are desired by at least one agent in S'. As can be readily verified, it is given by

$$\text{Len}(S', X) = \sum_{k \in [m-1]} \text{IntervalAvailable}(k) \cdot \left(\bigvee_{i \in S'} \text{IntervalDesired}(i,k)\right) \cdot \text{IntervalLen}(k) \tag{8}$$

For privacy considerations, it is necessary to protect the identity of agents who are being served in each iteration. However, at the same time, it is necessary to consider in each iteration only subsets S' that consist entirely of agents who have not yet been served. To achieve this while keeping the served agents' identities hidden, we introduce the bit array $\text{AgentsServed}(1 : n)$, in which the i-th bit is initially set to 0 and is updated to 1 once A_i has been served. By keeping that array (t, n)-secret-shared, the identities of the agents being served in each iteration remains secret.

Subprotocol 7, termed ITERATIVEALLOCATION (see Salman et al. (2025, Appendix A.5)), carries out the iterative allocation procedure. In each iteration, it searches for a subset S' that minimizes $\text{avg}(S', X)$, and then allocates to each agent in S' its portion of the intervals in X that it desires. We proceed to survey its main features.

In each of its iterations, subprotocol ITERATIVEALLOCATION finds a non-empty subset $S' \subseteq [n]$ that minimizes $\text{avg}(S', X)$, Eq. (7). In order to force the subprotocol to identify a minimizing subset only from among the legal subsets S'—those that consist of agents that have not yet been served—we introduce the marker

$$h(S') := \sum_{i \in S'} \text{AgentsServed}(i)\,. \tag{9}$$

A subset $S' \subseteq [n]$ is legal iff $h(S') = 0$. Define

$$\begin{aligned}\text{Len}^*(S', X) := \text{Len}(S', X) \cdot 1_{h(S')=0} \\ +n(Q+1) \cdot (1 - 1_{h(S')=0}) .\end{aligned} \tag{10}$$

The value $\text{Len}^*(S', X)$ equals $\text{Len}(S', X)$ for legal subsets S', while for illegal subsets it equals $n(Q+1)$. Hence,

$$\text{avg}^*(S', X) := \text{Len}^*(S', X)/|S'| , \tag{11}$$

equals $\text{avg}(S', X)$ (Eq. (7)) for legal subsets S', while it is at least $Q+1$ for illegal subsets. Since for all legal subsets $\text{avg}(S', X)$ is at most Q, any subset S' that minimizes $\text{avg}^*(S', X)$ would be legal. Hence, Subprotocol ITERATIVEALLOCATION finds a non-empty subset $S' \subseteq [n]$ that minimizes $\text{avg}^*(S', X)$, using Eqs. (8)–(11) and the basic secure MPC protocols described in Section 2.2. Then, it calls subprotocol ASSIGNCAKETOSELECTEDAGENTS (which we discuss below) that assigns cake pieces to the agents in the selected subset S'. Finally, Subprotocol ITERATIVEALLOCATION updates the secret shares of AgentsServed, according to the identities of the agents in S' who were just served, as well as the secret shares of IntervalAvailable, according to the intervals that were served in this iteration. Those updates are carried out in an oblivious manner, namely, without revealing the identities of those agents nor the indices of the intervals that were served to them.

4.6 Allocating Cake Pieces to the Agents in the Selected Subset

After identifying a subset of agents S' that minimizes the average demand over the available set of intervals X, algorithm CC_PUV (Chen et al. 2010) computes a fair allocation of the desired intervals in X among the agents in S'.

To do so, it constructs a four-layer directed graph $G(S', X)$: layer 1 has a source node; layer 2 has a node for each interval in X; layer 3 has a node for each agent in S'; and layer 4 has a target node. The graph has an edge from the source node to each of the nodes in layer 2, with a capacity that equals the length of the corresponding interval. There is an edge from each node in layer 2 to each node in layer 3 that corresponds to an agent that desires that cake interval; the capacity of such an edge is the length of the interval from which it emerges. Finally, it has an edge from each node in layer 3 to the target node with a capacity that equals $\text{avg}(S', X)$, Eq. (7). Figure 2 illustrates the graph $G(S', X)$ for a set X of 5 intervals and a set S' of 3 agents, who desire 4 out of the 5 available intervals in X.

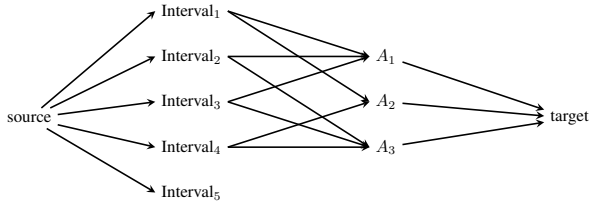

Figure 2: An example of a flow graph $G(S', X)$

The graph has a maximum flow of value $\text{Len}(S', X)$. This flow fully saturates the edges between layers 1 and 2 corresponding to intervals in X desired by S', while all other edges between these layers carry zero flow. The flow across the edges from layer 2 to layer 3 determines a fair allocation of the S'-desired intervals in X among the agents in S'. Finally, all edges between layers 3 and 4 are saturated, ensuring that each agent in S' gets cake pieces of which the sum of lengths is exactly $\text{avg}(S', X) = \text{Len}(S', X)/|S'|$.

Subprotocol 9 (ASSIGNCAKETOSELECTEDAGENTS), see Salman et al. (2025, Appendix A.6), emulates the computation described above while addressing a significant algorithmic challenge: how can one compute a maximum flow in a graph that must remain secret? The graph $G(S', X)$ inherently reveals the sets S' and X, as well as the desirability relationships between them, and therefore cannot be explicitly constructed. To resolve this challenge, the subprotocol operates on a fixed, publicly known four-layer graph, where layer 2 contains a node for each interval in $\mathcal{W}$ and layer 3 contains a node for each agent in $\mathcal{A}$. Although the structure of this graph is static and public (since $|\mathcal{W}| = m-1$ and $|\mathcal{A}| = n$), the edge capacities vary across iterations. These capacities are carefully defined to match those in the underlying algorithm CC_PUV for edges between intervals in X and agents in S', while all other capacities are set to zero. Crucially, the capacities are secret-shared to conceal the topology of the actual subgraph, thereby preserving the privacy of S', X, and the desirability relationships between agents and intervals.

The maximum flow that Subprotocol ASSIGNCAKETOSELECTEDAGENTS computes determines the allocation of (a subset of the) intervals to (a subset of the) agents. Those allocations cannot be disclosed at this stage. Instead, they must be registered and disclosed together with all allocations from all iterations only after the iterative allocation process terminates. To that end, we use the secret-shared arrays IntervalAllocation$(1 : n, 1 : m-1)$ and AllocationDenominator$(1 : n)$, that were initialized in Subprotocol INTERVALS (see Section 4.4). AllocationDenominator(i) registers the size of the subset S' to which A_i belonged when it was served, while IntervalAllocation(i, k) registers the length of the part of the k-th interval allocated to A_i, multiplied by AllocationDenominator(i) (so that, eventually, the portion of the k-th interval allocated to A_i is the fraction in Eq. (6)). After Subprotocol ASSIGNCAKETOSELECTEDAGENTS finds a maximum flow, it updates the secret shares of the relevant entries in those two arrays. For a comprehensive account, see Salman et al. (2025, Appendix A.6).

4.7 Cutting the Cake

Once Subprotocol ITERATIVEALLOCATION terminates, it is necessary to compute the values Portion(i, k) (Eq. (6)) that specify the fraction of the k-th interval allocated to agent A_i, for all $k \in [m-1]$ and $i \in [n]$. Specifically, if $\text{Portion}(i, k) = \alpha$, then A_i receives a subinterval of length $\frac{\alpha}{Q}$ from the interval $[\frac{W(k)}{Q}, \frac{W(k+1)}{Q}) \subset [0, 1)$ (see Eq. (4) and the accompanying discussion). Subprotocol 11, referred

to as FINALSERVING (see Salman et al. (2025, Appendix A.7)), performs this post-processing step by cutting the cake and assigning each agent their corresponding portion.

Let $g(k)$ denote the number of agents that were allocated some non-empty part of the k-th interval, $k \in [m-1]$. Intervals for which $g(k) = 1$ are considered *exclusive*, meaning they are fully assigned to a single agent. Intervals for which $g(k) \neq 1$ are *non-exclusive*; they are either split among multiple agents or unassigned altogether. The subprotocol records this classification in the secret-shared array, ExclusiveInterval$(1 : m-1)$, in which the k-th entry equals the index of the single agent that was assigned the entire k-th interval, when $g(k) = 1$, or 0 otherwise.

After completing the initial loop to determine the type of each interval, the subprotocol proceeds with a second loop over all intervals, $k \in [m-1]$, that assigns each interval to the appropriate agent or agents. The treatment of each of the intervals, $[W(k), W(k+1))$, begins by reconstructing the bit $1_{0<\text{ExclusiveInterval}(k)}$ from its secret shares.

If ExclusiveInterval$(k) = 0$ (namely, if the interval is to be split among more than one agent, or if the interval is assigned to none) the subprotocol proceeds as follows. It maintains a secret-shared variable Position that stores the left end point of the next portion of the interval to be assigned to an agent; it is initialized to the left end point of the interval. Then, the subprotocol scans all agents, $i \in [n]$. If IntervalAllocation$(i, k) > 0$ the subprotocol computes the portion of that interval to be assigned to A_i by computing the fraction in Eq. (6), using the secure MPC division protocol. After computing a secret sharing of AllocationSize—the length of the portion of the k-th interval allocated to A_i—the subprotocol proceeds to compute secret sharings of the two end points of that portion. If, however, A_i was not assigned any portion of that interval, the two end points it would get are *obliviously* set to Q (which stands for the right end point of the cake) so that the portion it gets is empty. At the end, all agents send their shares of those two values to A_i who proceeds to reconstruct them and deduce the actual portion on the interval $[0, 1)$ that it was given. Finally, the secret-shared value Position is moved forward by AllocationSize.

Next, we discuss the case ExclusiveInterval$(k) > 0$ in which the k-th interval is assigned as a whole to a single agent, say A_i. Assume that A_i is assigned a sequence of consecutive whole intervals, say $[W(k), W(j))$ where $k < j \leq m$. A simple treatment of that case would send $j - k$ independent messages to A_i, one for each of the intervals $[W(h), W(h+1))$, $k \leq h < j$, that were assigned to it. However, such an approach is inefficient, as it breaks a single portion, $[W(k), W(j))$, into $j - k$ intervals. Moreover, if A_i is assigned the entire portion $[W(k), W(j))$, there is no need to expose the internal points $W(h)$, $k < h < j$, which correspond to private valuations of the agents. Hence, Subprotocol FINALSERVING is designed to generate a single message for A_i for the entire portion $[W(k), W(j))$.

Note that the subprotocol sends a full set of n messages to all agents, for each interval (or a block of consecutive intervals), even though most of those messages are redundant. That way, PP_CC_PUV maintains *restricted visibility*: namely, each agent is notified only of its own allocated portion, but remains oblivious regarding the allocations of its peers. In settings where a *full visibility* is desired, Subprotocol FINALSERVING can be modified so that instead of sending the shares of the two end points only to the designated agent, it will broadcast those shares, so that all agents get notified of the portions given to each one of them.

5 Properties

PP_CC_PUV offers perfect privacy and is strategy-proof. It preserves CC_PUV's asymptotic runtime complexity and incurs only a quadratic increase in communication complexity. Those properties are stated in Theorems 5.1-5.3. The proofs are given in Salman et al. (2025, Appendix B).

Theorem 5.1 *(Perfect Security) If the set of agents has an honest majority then Protocol* PP_CC_PUV *is perfectly secure.*

Theorem 5.2 *(Strategyproofness) Protocol* PP_CC_PUV *is strategyproof for agents with piecewise uniform valuations.*

Theorem 5.3 *(Overheads) The overheads incurred by* PP_CC_PUV *on top of* CC_PUV *increase the overall runtime by at most a constant multiplicative factor, while the added communication complexity is at most* $\mathcal{O}(n^2)$.

6 Conclusion

We presented a private, strategyproof, and envy-free cake-cutting protocol, building on the non-private algorithm of Chen et al. (2010, 2013). Privacy is a critical consideration in many real-world allocation scenarios; without it, agents may withhold or distort their true preferences out of concern for exposure. By preserving strategyproofness within a privacy-preserving framework, our protocol enables agents to safely and truthfully report their valuations. As a result, the algorithm's envy-freeness holds with respect to the agents' actual preferences. In contrast, an envy-free algorithm that does not guarantee truthful reporting may produce allocations that appear envy-free based on misreported inputs, but fail to be so under the agents' true valuations.

This work represents a first step toward establishing privacy guarantees for cake-cutting algorithms. We believe the methodology introduced here can serve as a foundation for privatizing a broader class of cake-cutting algorithms. While privacy is particularly powerful when combined with strategyproofness—addressing both incentive and confidentiality concerns—it also holds value in non-strategyproof settings. This is especially relevant in environments with non-strategic agents, such as automated systems that are restricted from misreporting, where the absence of privacy alone may discourage participation, regardless of strategic considerations.

References

Aziz, H.; and Mackenzie, S. 2016. A Discrete and Bounded Envy-Free Cake Cutting Protocol for Any Number of Agents. In *FOCS*, 416–427. New Brunswick, New Jersey.

Aziz, H.; and Ye, C. 2014. Cake cutting algorithms for piecewise constant and piecewise uniform valuations. In

Proceedings of the 10th International Conference on Web and Internet Economics (WINE), 1–14. Beijing, China.

Bei, X.; Chen, N.; Huzhang, G.; Tao, B.; and Wu, J. 2017. Cake Cutting: Envy and Truth. In *IJCAI*, 3625–3631. Melbourne, Australia.

Bei, X.; Huzhang, G.; and Suksompong, W. 2020. Truthful fair division without free disposal. *Social Choice and Welfare (SCW)*, 55: 523–545.

Brams, S. J.; and Taylor, A. D. 1996. *Fair Division. From cake-cutting to dispute resolution*. Cambridge University press.

Brandt, F.; Conitzer, V.; Endriss, U.; Lang, J.; and Procaccia, A. D., eds. 2016. *Handbook of Computational Social Choice*. Cambridge University Press.

Bu, X.; Song, J.; and Tao, B. 2023. On existence of truthful fair cake cutting mechanisms. *Artificial Intelligence*, 319: 103904.

Chen, Y.; Lai, J. K.; Parkes, D. C.; and Procaccia, A. D. 2010. Truth, Justice, and Cake Cutting. In *AAAI*, 756–761. Atlanta, Georgia.

Chen, Y.; Lai, J. K.; Parkes, D. C.; and Procaccia, A. D. 2013. Truth, justice, and cake cutting. *Games and Economic Behavior (GEB)*, 77: 284–297.

Chida, K.; Genkin, D.; Hamada, K.; Ikarashi, D.; Kikuchi, R.; Lindell, Y.; and Nof, A. 2018. Fast Large-Scale Honest-Majority MPC for Malicious Adversaries. In *CRYPTO*, 34–64.

Damgård, I.; and Nielsen, J. B. 2007. Scalable and Unconditionally Secure Multiparty Computation. In Menezes, A., ed., *CRYPTO*, 572–590.

Kogan, P.; Tassa, T.; and Grinshpoun, T. 2023. Privacy preserving solution of DCOPs by mediation. *Artificial Intelligence*, 319: 103916.

Maya, A.; and Nisan, N. 2012. Incentive compatible two player cake cutting. In *Proceedings of the 8th Workshop on Internet and Network Economics (WINE)*, 170–183. Liverpool, Great Britain.

Menon, V.; and Larson, K. 2017. Deterministic, Strategyproof, and Fair Cake Cutting. In *IJCAI*, 352–358. Melbourne, Australia.

Mossel, E.; and Tamuz, O. 2010. Truthful Fair Division. In *SAGT*, 288–299. Athens, Greece.

Nishide, T.; and Ohta, K. 2007. Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. In *PKC*, 343–360.

Robertson, Y. J.; and Webb, W. 1998. *Cake-Cutting Algorithms, Be Fair If You Can*. A K Peters.

Salman, Y.; Tassa, T.; Lev, O.; and Zivan, R. 2025. Truth, Justice, and Secrecy: Cake Cutting Under Privacy Constraints. *arXiv*, abs/2511.09882.

Shamir, A. 1979. How to Share a Secret. *Communications of the ACM*, 22(11): 612–613.