PASTRAMI: Privacy-preserving, Auditable, Scalable & Trustworthy Auctions for Multiple Items

Michał Król
City, University of London
United Kingdom
michal.krol@city.ac.uk

Alberto Sonnino
University College London
United Kingdom
alberto.sonnino@ucl.ac.uk

Argyrios Tasiopoulos
University College London
United Kingdom
argyrios.tasiopoulos@ucl.ac.uk

Ioannis Psaras
University College London
United Kingdom
i.psaras@ucl.ac.uk

Etienne Rivière
ICTEAM, UCLouvain
Belgium
etienne.riviere@uclouvain.be

Abstract

Decentralised cloud computing platforms enable individuals to offer and rent resources in a peer-to-peer fashion. They must assign resources from multiple sellers to multiple buyers and derive prices that match the interests and capacities of both parties. The assignment process must be decentralised, fair and transparent, but also protect the privacy of buyers.

We present PASTRAMI, a decentralised platform enabling trustworthy assignments of items and prices between a large number of sellers and bidders, through the support of multi-item auctions. PASTRAMI uses threshold blind signatures and commitment schemes to provide strong privacy guarantees while making bidders accountable. It leverages the Ethereum blockchain for auditability, combining efficient off-chain computations with novel, on-chain proofs of misbehaviour. Our evaluation of PASTRAMI using Filecoin workloads show its ability to efficiently produce trustworthy assignments between thousands of buyers and sellers.

CCS Concepts • **Security and privacy** → **Trusted computing**; **Distributed systems security**;

Keywords blockchains, security and privacy, trusted computing, distributed systems

ACM Reference Format:
Michał Król, Alberto Sonnino, Argyrios Tasiopoulos, Ioannis Psaras, and Etienne Rivière. 2020. PASTRAMI: Privacy-preserving, Auditable, Scalable & Trustworthy Auctions for Multiple Items. In *21st International Middleware Conference (Middleware '20), December 7–11, 2020, Delft, Netherlands.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3423211.3425669

Middleware '20, December 7–11, 2020, Delft, Netherlands

ACM ISBN 978-1-4503-8153-6/20/12…$15.00
https://doi.org/10.1145/3423211.3425669

1 Introduction

Sharing economy systems allow individuals to rent or "share" their resources to other individuals in a peer-to-peer fashion. Multiple platforms already implement this concept towards a decentralised form of cloud computing. They allow renting the machines of other users to run large-scale computations (e.g. Golem [51], iExec [1], Pando [37] or SONM [2]) or for storing data (e.g. Storj [36] or Filecoin [35]). These platforms often build on blockchains as a global source of trust and as an enabler of decentralised payments. For instance, the decentralised file storage platform Filecoin [35] encrypts and scatters end-users' files fragments across multiple storage machines and rewards them with periodic token payments.

One of the main challenges faced by sharing economy platforms is the difficulty of setting up fair allocations and prices between large numbers of sellers and buyers of goods or services (*items* in the rest of this paper). They need to set up decentralised mechanisms to *globally* decide on the assignment of items from sellers to buyers, and the prices that apply to the transactions. The interests of buyers and sellers are, indeed, in conflict: A buyer wishes to buy an adequate item at a minimal price, while a seller wants to sell its item(s) at the highest possible price. Direct negotiations between the two parties, as used currently in Filecoin [35], have poor scalability: Each buyer may have to individually contact and negotiate with an increasing number of storage nodes as competition for items increases. Furthermore, direct negotiation does not provide transparency and cannot guarantee any fairness: there are no guarantees that a seller or buyer settles the most interesting of all possible deals.

Multi-item auctions are a sound basis for the assignment of items and prices in decentralised exchange platforms. They allow buyers, or *bidders*, to announce the price they are willing to pay for an item (their *valuation*) across many items from multiple sellers. Valuations may be for a *specific* item or for

any one of the items that satisfy a number of constraints. We denote the latter case as a *general bid*. We present an example in Figure 1, inspired by Filecoin: Bidder 1 is willing to pay up to 40 $ for a node with *at least* 50 GB of storage space, while bidder 5 specifically bids for the three available storage nodes. Based on these specific and general bids, a multi-item auction algorithm can derive an assignment (e.g., bidder 2 is assigned node 2) and prices for the transactions (e.g., bidder 1 has to pay 30 $, less than the announced 40 $). The assignment of items and prices depend on a global measure of goodness that is specific to the multi-item auction algorithm used. In this example, resources are assigned to bidders who "value them" the most while deciding on a price following rules of supply and demand.

The pricing mechanism is a critical component of an exchange platform and all participants should be convinced of its *correctness*. A single malicious auction operator could easily influence the assignment to maximise its own gain. At the same time, verification should not require all participants to reproduce the entire computation.

Transparency and verification come, however, in apparent tension with the need to preserve the *privacy* of auctions' participants. First, *sealed-bid auctions* require to keep bids private until all of them are submitted. This prevents late bidders from taking advantage from the already revealed information [27]. Second, it is important to keep bidders' identity private. For instance, revealing that a company recently rented significant amounts of storage may expose operation details of this company to its competitors.

Finally, it is necessary to compute auction results in an *efficient* and *scalable* manner. Auction algorithms experience exponential increase in execution time and require storing significant volumes of data with increasing size of the input parameters. In sharing economy systems, potentially thousands of buyers must be assigned to sellers on a regular basis.

Decentralised auctions have been realised previously using Secure Multi-Party Computations (MPC) [3, 8]. MPC-based auctions provide strong privacy and correctness guarantees, but introduce significant computational overhead and can be executed only between a limited number of parties. It means that the computations have to be performed by a small number of actors that must be trusted by all of the auction participants, something that is difficult to realise in practice.

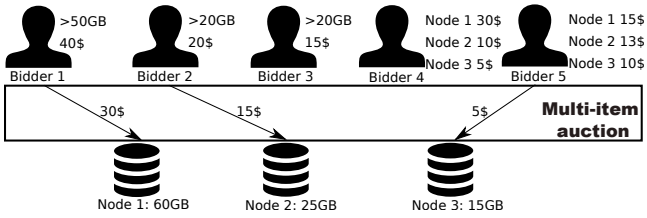

Figure 1. Example of general (bidders 1–3) and specific (bidders 4 & 5) bids for multiple items in the context of a decentralised storage cloud, and of the assignment of items and prices decided by a multi-item auction algorithm.

Blockchain platforms with support for executable smart contracts are an appealing solution for building auditable auctions, thanks to the immutability of their decentralised ledger [7, 23, 24]. However, direct auction execution using a smart contract impairs privacy and is at odds with scalability: The costly assignment calculation has to be repeated by all miners, leading to large overheads. General platforms for verifiable computations such as Truebit [50] or Arbitrum [29] reduce the overhead, but require multiple verifiers to repeat full programs and do not offer sufficient privacy protection. While many specific auctions systems try to increase bidders' privacy or offload heavy computations off-chain, they require a trusted third party [7], secure hardware [56] or involve significant computational overhead [24].

Contributions. We present the design and evaluation of PASTRAMI, a framework for *private*, *secure* and *efficient* multi-item auctions. PASTRAMI leverages the Vicrey-Dutch algorithm (VDA) [41, 42] to determine an assignment between bidders and items and derive optimal prices[1], and can be easily adapted to support other forms of auctions. PASTRAMI extends VDA with support for general bids: Bidders can provide a valuation for any item whose description matches a set of predicates. General bids are automatically transformed in a number of per-item bids, as manipulated by the VDA algorithm.

PASTRAMI leverages the Ethereum blockchain for transparency, enabling any party to audit and if necessary invalidate the outcome of an auction. To ensure that bidders only provide faithful valuations, PASTRAMI binds bids to the Ethereum cryptocurrency: Buyers cannot bid higher than the amount of money submitted to a deposit. Funds are automatically deducted from their accounts if they acquire an item.

PASTRAMI protects privacy by hiding submitted bids as well as the identity of bidders using threshold blind signatures. In contrast with related work, PASTRAMI does not rely on a single trusted third party, but rather on a set of multiple, decentralised authorities. Each user can define their own set of trusted parties and is protected from a definable subset of authorities becoming malicious.

High efficiency in PASTRAMI results from two key design choices. First, we run the heavy VDA computation off-chain and allow any node to submit an auction solution in the form of an assignment between items and bidders and a price vector. Second, to avoid re-running auctions or perform costly on-chain verification we develop simple verification techniques in the form of *proofs of misbehaviour*. Users can generate such proofs without re-running the VDA algorithm, and for

[1] Vickrey-Dutch Auctions are multi-item, sealed-bid auctions that maximise a measure of *social welfare* for their assignments (*i.e.,* sum of the differences between initial bids and paid prices). VDA assigns the price of the *second highest* bid to each item, incentivising truthful valuations.

a fraction of its computational cost. The PASTRAMI smart contract can check the validity of these proofs and discard faulty assignments. Both providers of solutions and submitters of proofs of misbehaviour are held accountable for their actions by linking virtuous and malicious behaviours with monetary rewards and penalties.

We implement PASTRAMI and deploy it over Ethereum, using workloads derived from Filecoin. PASTRAMI ensures correctness, protects bidder's privacy and scale to large of numbers bidders and items. Our evaluation shows the ability to derive optimal assignments between 10,000 users and 100 Filecoin storage nodes with less than 8 seconds of CPU time on a regular laptop and enable auditing the result in less than half a second. The comparison with Verifiable Sealed-bid Auctions [24] shows 4 orders of magnitude lower execution time, 2.5x reduction in contract deployment costs, and up to an 8x reduction in cost per bidder. PASTRAMI source code is publicly available for the scientific community and easy to adapt to support any sharing-economy platform[34].

Outline. The rest of this paper is structured as follows. Section 2 presents our system, its adversary model, and details our objectives. Section 3 presents an overview of PASTRAMI, and Section 4 discusses background. We describe our design divided into a preparation phase and an execution phase in Sections 5 and 6. We provide a discussion and a security analysis in Section 7, and present our evaluation results in Section 8. We finally cover related work in Section 9 and conclude in Section 10.

2 PASTRAMI Design Goals

We start by defining our system model, notations and assumptions. We follow up by specifying our target properties for correctness, but also privacy and ease of deployment. These properties are summarised in Table 1. We describe system components using Filecoin as a running example. However, our platform can be used in any decentralised system requiring an assignment between buyers and sellers [1, 2, 36, 37, 51].

2.1 System Model

We consider the following actors:

- **Bidders** are users wishing to buy goods or services, denoted by the generic term *item*, e.g., Filecoin end-users willing to store their files;
- **Sellers** are users offering such items, e.g., Filecoin storage nodes;
- A set of **authorities** jointly issues credentials allowing bidders to anonymously participate in auctions. For instance, in Filecoin this role can be taken by blockchain validators or a distinctive set of entities trusted by system participants.

We emphasise that none of the authorities is trusted individually by the users. Instead, users trust a set of authorities *as a whole* similarly to an honest majority in blockchains or Byzantine fault-tolerant protocols [15].

PASTRAMI accepts item description submitted by sellers and sealed-bids commitments (*bids*) expressed by bidders. A bid expresses a *valuation*, *i.e.,* the maximal price that the bidder is willing to pay for an item. Each bid is backed by a deposit of that valuation. This deposit is expressed in coins that cannot be double-spent [30] and is guaranteed by the authorities. Based on the set of all bids, PASTRAMI outputs a price vector and an assignment between bidders and items. This assignment must be the output of a globally-known algorithm. PASTRAMI employs the Vickrey-Dutch multi-item auction algorithm, but can be adapted to different types of single-item, multi-item and even combinatorial auctions.

In the example of Filecoin, storage nodes (sellers) advertise their capacities (*i.e.,* storage space, available bandwidth, lease duration) together with a minimum price they are willing to accept. End-users (buyers) submit their requirements and a maximum amount they are willing to pay (*i.e.,* "I am willing to pay 10 tokens for a month of using a 80 GB, high-bandwidth storage node"). PASTRAMI automatically allocates end-users to the most suitable storage nodes and derives prices following rules of supply and demand.

2.2 Notations and Assumptions

Cryptographic assumptions. PASTRAMI inherits the same cryptographic assumptions as its underlying BLS blind signature scheme we present in Section 4, which requires groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order p with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ and satisfying (*i*) *Bilinearity*, (*ii*) *Non-degeneracy*, and (*iii*) *Efficiency*. PASTRAMI uses type-3 pairings because of their efficiency [25]; and thus relies on the XDH assumption which implies the difficulty of the Computational co-Diffie-Hellman (co-CDH) problem in $\mathbb{G}_1$ and $\mathbb{G}_2$, and the difficulty of the Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}_1$ [11].

Threshold assumptions. PASTRAMI assumes that at least t-out-of-n authorities are honest and available at all time (with $t > n/2$). If more than t authorities collude, they can forge signatures or stall the system, but cannot break unlinkability and de-anonymize users.

Communication assumptions. The PASTRAMI actors interact through the direct exchange of messages as well as through a blockchain supporting executable *smart contracts*. Our implementation uses Ethereum [13]. We assume the following about communications:

- **Authorities:** PASTRAMI authorities are collectively responsible for issuing signatures to bidders. They do not need to communicate with each other in order to issue these signatures. However, practical implementations of threshold signature schemes initially require either using the distributed key generation protocol of Kate *et al.* [31],

Economic Properties	
Efficiency	bidders get items they value the most
Incentive Compatibility	bidders benefit from revealing their bids
Individual Rationality	bidders and sellers benefit from participation
Budget Balance	derived prices exceed minimum prices
Correctness	
Bids Binding	bids cannot be changed once committed
Public Auditability	participants are able to verify its correctness
Fairness	users deviating from the protocol are penalised
Privacy	
Hidden Minimum Price	minimum prices are private until the end of the auction
Bidders' Privacy	bidders are unlinkable to their bids
Bids Privacy	bids are private until the end of the auction
Deployment	
Non-Interactivity	bidders are not required to stay online
Openness	anyone can become a bidder or a seller
Distributed authority	PASTRAMI does not rely on a single trusted party
Scalability	support for large number of items and bidders

Table 1. Target properties of PASTRAMI.

which requires (*i*) weak synchrony for liveness (but not for safety), and (*ii*) at most one third of dishonest authorities; or using the distributed key generation protocol of Gennaro *et al.* [26] which requires (*i*) synchrony, and (*ii*) an honest majority. Key generation is, however, run only once (or rarely) and resulting keys can be used in multiple auctions.

- **Authorities-Users:** Users wait for *t*-out-of-*n* replies (in any order of arrival) and aggregate them into a consolidated signature; thus PASTRAMI implicitly assumes an asynchronous setting between users and authorities.
- **Bidders-Sellers:** Bidders do not interact or trust each other. The same applies to sellers. A bidders and a seller need only communicate directly to implement a deal, after an auction final assignment is available.

2.3 Target Properties for PASTRAMI

PASTRAMI aims at providing the following properties:

Auctions' economics. PASTRAMI assigns resources to the bidders that value them the most guaranteeing *Efficiency*. In addition, PASTRAMI guarantees *Incentive Compatibility* meaning that bidders and sellers benefit from revealing their true valuations and can only lose by artificially increasing or decreasing those valuations. This property is linked to *Individual Rationality*, which ensures that both bidders and sellers are incentivised to participate in the auctions. Finally, it targets *Budget Balance* by making sure that the payments submitted cover sellers' compensations (*i.e.,* derived prices are superior to minimum prices submitted by sellers).

Correctness. PASTRAMI verifies that auction assignment are computed correctly and that all participants follow the protocol. To avoid malicious manipulation, bidders cannot change their bids once they are committed (*Bids Binding*). Furthermore, anyone can verify the correct execution of any auction by inspecting the public information available on the ledger (*Public Auditability*). PASTRAMI achieves *Fairness* by making sure that bidders are financially penalised if they deviate from the protocol, but cannot be financially penalised if they follow it correctly.

Privacy. PASTRAMI aims to only disclose the minimum amount of information required to verify the correctness of the auction. The minimum prices submitted by sellers are kept private from the bidders until the end of the auction (*Hidden Minimum Price*); and bids submitted to the system are kept private until the end of the auction (*Bids Privacy*). Those two properties ensure that bidders submit their bids without any knowledge of what others do, preventing price manipulation (*i.e.,* submitting bids slightly above the minimum price or the current highest bid). PASTRAMI also keeps bidders anonymous, ensuring that bidders are unlinkable to their bids and that the identity of the assigned bidder is revealed only to the seller at the end of the auction.

Deployment. The process of submitting items, bids and calculating a result can take a significant amount of time. Many participants may not be able to stay connected during the entire duration of an auction. To facilitate large scale deployments and support large number of users, bidders are not required to interact with each other providing *Non-Interactivity*. Furthermore, PASTRAMI targets *Openness*, meaning that anyone can act as bidder or seller and the system is resilient to censorship. Our platform support the *Distributed Authority* property and does not rely on a single trusted 3rd party at any point. Finally, PASTRAMI achieves high *Scalability* and supports large number of users and items while maintaining low overall cost of running auctions for the platform and its participant.

3 PASTRAMI Overview

Figure 2 presents an overview of PASTRAMI. An auction is divided in two subsequent phases, a *preparation phase* (steps 1 to 5) and an *execution phase* (steps 6 and 7).

Preparation phase. A set of distributed authorities starts by publishing their aggregated verification key along with any associated parameters, and sets up a smart contract on the blockchain (step 1). In order to create privacy-preserving accounts, bidders contact this smart contract and pay a maximum amount they are willing to bid as a deposit (step 2). The authorities observe the smart contract, and generate cryptographic material required by bidders to participate in the auction. Bidders then contact each authority, retrieve parts of the material and combine these parts locally into cryptographic credentials (step 3). Those credentials allow bidders to participate in auctions while guaranteeing strong privacy and ensuring that bidders are backed by a deposit. Bids remain unlinkable to bidders's real identities or to cryptographic material issued by the authorities.

An auction begins with sellers submitting descriptions of their items/services to PASTRAMI. To guarantee that their activities are profitable, sellers can specify a hidden minimum price for which they are willing to trade their items, and are guaranteed that the users assigned during the auction have bid for at least that price (step 4).

In step 5, bidders submit their sealed bids to the smart contracts using their credentials. Bidders can submit two types of bids. Regular bids express the interest of the bidder for a specific item, in a form of a price *valuation*. *General* bids allow, on the other hand, bidders to specify their interest in *any* one of the items satisfying some constraints. Auction algorithms only support single-item evaluation, henceforth PASTRAMI automatically derives a vector of single-item valuations for all items matching the constraints set in a general bid. Once all the information is submitted to the blockchain, bids[2] and minimum prices are revealed.

Execution phase. Instead of performing costly assignment calculations on-chain, the smart contract allows a dedicated node[3] to use the information stored on the blockchain to perform computations off-chain and submit a solution to the platform (step 6). The calculation of a solution off-chain does not suffer from the overhead associated with distributed and privacy-preserving solutions such as MPC (Section 9). A solution contains an assignment between sellers and bidders and derived prices for each of the assigned items. A score is associated to this assignment, representing the *social welfare*, a measure of the quality of the compromise between buyers and sellers' interests.

Once submitted, the solution can be contested within a specified amount of time. Any user, bidder or seller, may verify the solution off-chain and submit a proof of misbehaviour (step 7). PASTRAMI uses lightweight verification that is much faster than recomputing the auction result. The smart contract can efficiently test the validity of each proof of misbehaviour and reject the contested solution. Alternatively, if no valid proof of misbehaviour is received by the end of the time period, the solution is marked as final. Bidders can contact their corresponding sellers, prove their identity and claim acquired items.

In the next sections, we start by detailing the background constructions used by PASTRAMI, followed by the details of the preparation and execution phases.

4 Background

Smart Contracts on Blockchains. The concept of a blockchain was introduced by Bitcoin [44] as a decentralised, append-only ledger that provides a global ordering of financial transactions, to prevent funds being spent twice (the *double-spending* attack).

[2]The revealed bids are unlinkable to bidders who expressed them.

[3]We use a dedicated node for simplicity. In practice, any node can calculate a solution and submit it to the chain.

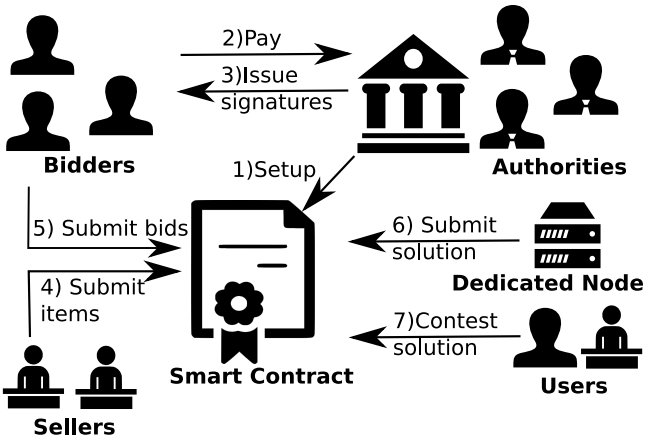

Figure 2. PASTRAMI overview.

Blockchain platforms such as Ethereum [13] provide script languages to allow users to execute more complex programs on the blockchain, called *smart contracts*. Executing a transaction calling a method of an Ethereum smart contract has an associated *gas* cost that is proportional to the number of instructions and the amount of storage required by the transaction. The monetary value of gas varies depending on the load on the network, and is paid for using Ether, Ethereum's built-in currency.

PASTRAMI leverages the high-integrity data structure provided by a blockchain, and uses it for accountancy, auditability, and time reference (*e.g.,* time is defined as a difference in block height in the chain).

Threshold Blind Signatures. Blind signatures are digital signatures allowing users to hide (blind) the content of a message from the signer. The user can then locally unblind the signature, which can then be publicly verified as traditional digital signatures [16]. Most blind signature schemes entrust a *single* authority with a master signing key, allowing a malicious authority to forge any signature. To overcome this limitation, PASTRAMI relies on *threshold blind signatures* [10] that distribute the signing process between multiple authorities.

We give below the high-level definition of a threshold blind signature scheme. For the sake of simplicity, we use in this description a key generation algorithm BS.TTPKeyGen as executed by a single trusted third party. This protocol can, however, be executed in a distributed way as illustrated by Gennaro *et al.* [26] under a synchrony assumption, and as illustrated by Kate *et al.* [32] under a weak synchrony assumption. All algorithms receive the security parameter λ as an input but we show it explicitly only for Setup. Appendix A provides the full cryptographic construction of the blind BLS signature scheme.

❖ **BS.Setup(1^λ) → (*params*):** defines the system parameters *params* with respect to the security parameter λ (these parameters are publicly available).

❖ **BS.TTPKeyGen($params, t, n$) → (x, y):** run by the authorities to generate a secret key x and a verification key y from the public parameters $params$.

❖ **BS.PrepareBlindSign(m) → $(\tilde{h})$:** run by the user to request a blind signature over the cryptographic material $\tilde{h}$ embedding the message m.

❖ **BS.BlindSign($x_i, \tilde{h}$) → $(\tilde{\sigma}_i)$:** run by each authority i to issue a partial blind signature $\tilde{\sigma}_i$ (using their private key x_i) over the user-provided cryptographic material $\tilde{h}$.

❖ **BS.Unblind($r, \tilde{\sigma}_i$) → (σ_i):** run by the user to unblind the signature $\tilde{\sigma}_i$ and recover the signature σ_i (using the blinding factor r).

❖ **BS.AggSig($\sigma_1, \ldots, \sigma_t$) → (σ):** run by the user to aggregate t partial signatures $(\sigma_1, \ldots, \sigma_t)$ into a consolidated signature σ.

❖ **BS.Verify(y, m, σ) → $(true/false)$:** run by any third party verifier to check the validity of a signature σ over the message m (using the aggregated public key y).

4.1 Multi-item Auction Execution

We consider a single smart contract that sets up an auction for a set of $\mathcal{I} = \{1, 2, ..., I\}$ items. A set of $\mathcal{B} = \{1, 2, ..., B\}$ bidders are interested into the offered items of the contract where each bidder $b \in \mathcal{B}$ is expressing their preference for each item $i \in \mathcal{I}$ via a *private monetary valuation*, denoted by $v_{bi} \geq 0$. We focus on a *unit-demand* setting where each bidder is interested in acquiring *at most a single* item[4]. That is, we consider a *null item*, denoted by 0, that is assigned to the bidders who fail to acquire an item in the auction. Note that the valuation of each bidder for the null item is zero, *i.e.,* $v_{b0} = 0 \; \forall b \in \mathcal{B}$. For notational convenience let $\mathcal{I}^* = \mathcal{I} \cup \{0\}$.

A *feasible* allocation X assigns an item $x_b \in \mathcal{I}^*$ to each bidder $b \in \mathcal{B}$ such that for $b \neq b' \; x_b \cap x_{b'} = \emptyset$ or $x_b \cap x_{b'} = 0$, meaning that only the null item 0 can be allocated to more than one bidder. A feasible allocation is consider *efficient*, denoted by X^*, if there is no allocation X such that $\sum_{b \in \mathcal{B}} v_{bx_b} > \sum_{b \in \mathcal{B}} v_{bx_b^*}$.

Auctions are pricing mechanisms that lead to a feasible allocation of items. In detail, an auction associates each item $i \in \mathcal{I}$ to a price p_i forming the *price vector* $p = (p_0, p_1, ..., p_I)$. The seller of an item i can set a *reservation price* $r_i \geq 0$ that is defined as the minimum price that it is willing to offer the item for, restricting the auction's assigned prices for the specific item to $p_i \geq r_i$. Given a price vector p, the interest of each bidder is limited to the items that return the highest valuation after reducing the corresponding price. That is, the demand correspondence $D_b(p)$ for bidder $b \in \mathcal{B}$ is given by $D_b(p) = \{i \in \mathcal{I}^* : v_{bi} - p_i \geq v_{bj} - p_j \forall j \in \mathcal{I}^*\}$.

A price vector p^* is an *equilibrium price vector* if it derives a feasible assignment X where an item i remains unassigned at price $p_i^* = r_i$ or it is uniquely assigned to a bidder $b \in \mathcal{B}$, *i.e.,* $x_b = i$ and $x_b \neq x_{b'} \forall b' \in \mathcal{B} \backslash \{b\}$, where $x_b \in D_b(p)$ at price $p_i^* \geq r_i$. That is, the pair (X, p^*) is an *equilibrium allocation* if p^* is an equilibrium price vector. The set of competitive price vectors is non-empty and forms a complete lattice [47], meaning that there is a unique minimal element that is referred as the Vickrey-Clarke-Groves (VCG) price vector and is denoted by p^{VCG}. Furthermore, the corresponding assignment to p^{VCG} price vector is efficient, denoted by X^{VCG}.

This result essentially states that when the price vector p^{VCG} is applied, an item is assigned to a user at the lowest possible price over any possible equilibrium price vector, *i.e.,* any feasible assignment that respects bidders' demand correspondence. In other words, there is no other equilibrium than VCG that bidders would prefer, since it satisfies their demand in a stable way at the lowest possible price by deriving the highest possible *net* valuations (*i.e.,* the difference between the bid and the obtained price). This has the implication that under an auction that assigns items according to VCG equilibrium, bidders have no incentives to interfere with the smooth operation of the mechanism; therefore is to the best of their interest to be *truthful* upon the declaration of valuations for each item of the auction.

There are plenty of auction mechanisms that derive the VCG equilibrium in polynomial time. We use the Multi-Item Unit Demand Vickrey-Dutch Auction (VDA) [41, 42] as the underlying auction of PASTRAMI[5].

5 Preparation Phase

We present the protocol behind the preparation phase followed by a formal description of the algorithms.

5.1 Preparation Phase Protocol

Setup. A set of authorities execute the Setup method of the PASTRAMI smart contract; they provide their public keys as well as any other scheme parameters (or policy) as the number of authorities and the threshold parameter; publish a unique identifier for the auction id, and initiate a timer T (expressed in a number of blocks). Once created, any seller can add a description of an item to be sold and a commitment to a minimum price r_i.

Commit. Bidders execute Commit by paying a deposit of d coins to the PASTRAMI smart contract, and specifying cryptographic material embedding a fresh account address $addr$ they own, the auction unique identifier id, a random number k, and their bid b. The contract emits an event instructing the authorities to issue a blind signature using this

[4] A bidder interested in buying multiple items may participate multiple times to the same auction under different, un-linkable identities.

[5] Alternative mechanisms include the Vickrey-English and Vickrey-English-Dutch auctions which are also polynomial although are coming at a higher complexity compared to VDA.

cryptographic material. The value d represents the number of coins the bidder wishes to bid. To mitigate traffic analysis, d should be limited to a specific set of possible values, similar to cash denominations. Each authority monitors the PASTRAMI smart contract, and issues a partial blind signature to the user—either on chain or off-chain—upon detecting the request (signature requests are processed only if bidders paid a deposit of d coins to the smart contract and correctly executed Commit). Authorities use a different set of keys for each cash denomination (*e.g.,* if the user deposits $d = 5$, the authorities issue the blind signature using a key pair (sk_5, pk_5)). Bidders locally unblind and aggregate all partial signatures into a consolidated signature. All account addresses in this protocol must be fresh in order to not leak information about the identity of the users; users can privately add coins to an address using a coin tumbler [6, 12, 28, 39, 40, 46, 48, 52].

Reveal. After the timer expires, sellers open the commitment to their minimum price. Bidders execute Reveal and submit their unblinded signatures to the contract using their fresh account address *addr*. The contract accepts the bid only if it is submitted by the same address *addr* as the one embedded in the signature, if the auction *id* matches the current auction, and if it has not already seen the random number k; the contract keeps track of all the numbers k it has seen thus far to prevent double-spending. The contract interprets the value associated with each signature based on their signing key. Including *addr* in the blind signature and verifying that it is used to submit the bid prevents malicious actors from intercepting the bid during Reveal and spending it on behalf of victim users.

5.2 Algorithm Constructions

We present our algorithms. BS-prefixed algorithms are from the threshold blind BLS signature scheme (Section 4).

Interactions between users and smart contract. We first describe the algorithms allowing users to interact with the PASTRAMI smart contract. We prefix bidders' functions (executed off-chain by the client's software) with a capital B, and smart contract functions (executed on-chain by the PASTRAMI smart contract) with a capital C. All algorithms have implicitly access to the address identifying the account of the bidder who sends the transaction. Symbol || denotes string concatenation.

❖ **Setup(**1^λ**) →** (*params*)**:**
▹ Defines the system parameters *params* with respect to the security parameter λ (these parameters are public).
Output $\mathsf{BS.Setup}(1^\lambda)$; the smart contract creates an empty spent-list $\mathcal{L}$ in memory, and initializes a timer T.

❖ **Commit(**$d, addr, id, k, b, T$**) → ()**:
▹ Can only be executed if the timer T has not expired; commit to the bid b along with the parameters $addr$, id, k by sending d coins to the contract.

❖ $\mathsf{B.Commit}(d, addr, id, k, b) \rightarrow (\tilde{h})$:
Compute $m = (addr||id||k||b)$,
and $\tilde{h} = \mathsf{BS.PrepareBlindSign}(m)$; send $\tilde{h}$ along with d coins to the appropriate smart contract instance.

❖ $\mathsf{C.Commit}(\tilde{h}, T) \rightarrow ()$: if the timer T has not expired, emit an event telling the authorities to issue a blind signature to the bidder; otherwise ignore.

❖ **Reveal(**m, σ, y, T**) → ()**:
▹ Reveal the bid by revealing the unblinded signature.
❖ $\mathsf{B.Reveal}(m, \sigma) \rightarrow ()$: submit (m, σ) to the smart contract using the address $addr$.

❖ $\mathsf{C.Reveal}(m, \sigma, y) \rightarrow ()$: parse $m = (addr||id||k||b)$; save the bid b in memory if and only if the sender address is $addr$, the auction id matches the current auction, $k \notin \mathcal{L}$, and if $\mathsf{BS.Verify}(y, m, \sigma) = true$; then append k to $\mathcal{L}$. Otherwise ignore.

Interactions between bidders and authorities. We describe the algorithms allowing users to interact with the authorities. We prefix off-chain bidders' functions with a capital B, and off-chain authority functions with a capital A.

❖ **Issue(**$sk, \tilde{h}$**) →** $(\tilde{\sigma})$**:**
▹ The authorities issue a blind signature to the user over the cryptographic material $\tilde{\sigma}$.
❖ $\mathsf{A.Issue}(sk, \tilde{h}) \rightarrow (\tilde{\sigma}_i)$: each authority returns to the user $(\tilde{\sigma}_i) = \mathsf{BS.BlindSign}(sk, \tilde{h})$.
❖ $\mathsf{B.Issue}(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_t) \rightarrow (\sigma)$: for each $\tilde{\sigma}_i$, compute $\tilde{\sigma} = \mathsf{BS.AggCred}(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_t)$; output $(\sigma) = \mathsf{BS.Unblind}(r, \tilde{\sigma})$.

5.3 Expressing valuations

Regular valuations are submitted as price vectors without any additional support. To enable general bids, we enhance item descriptions submitted by sellers by a set of characteristics $C = \{1, 2, ..., C\}$. Each characteristic represents a single property of the submitted item. In the example of FileCoin, these properties could be the storage size, the service duration, or the seller reputation. For simplicity, let all characteristics be expressed as integers.

A general bid uses a set of constraints for each characteristic f_b and a maximum budget d_b instead of a private monetary valuation for each item v_{bi}. Each bidder is interested uniquely in items fulfilling all the submitted constraints. Furthermore, we use the constraint vector to automatically derive private monetary valuations required by the auction algorithm $derive(f_b) = v_{bi}$ (see Section 4.1).

The constraint vector of a general bid, while less flexible than single item valuation allows to radically decrease the amount of information submitted to the blockchain and enhance bidders' privacy. Finally, the constraint vector is reusable and can be submitted without knowing the list of offered items, reducing further the overhead of our solution.

6 Execution Phase

At this time, all the information to determine an assignment is available on the blockchain. This assignment is calculated by the dedicated node. The first step is to derive bidders' valuations from the *general* bids revealed in Reveal (Figure 3). The derivation function assigns the specified maximum budget v to all the items fulfilling the requirements $v_{bi} = v$. If at least one characteristic is lower than a specified constraint the assigned valuation is set to 0, *i.e.,* $v_{bi} = 0$. Once valuation vectors are computed, the dedicated node uses an unmodified version of the Vickrey-Dutch auction algorithm [41, 42].

In order to incentivise correct behaviour, the dedicated node must first provide a collateral d_s. This collateral is returned once the solution is marked as final. We calculate the minimum amount of the deposit in Section 8.

A submitted solution consists of an assignment of items to bidders X, a price vector p and a score s. p indicates the established price of each item, while s represents the social welfare score (*i.e.,* the sum of net valuations). By definition, VCG equilibrium dictates the solution with the highest social welfare, where all items are universally allocated (Section 4).

Once a solution is stored on-chain, bidders and sellers have a predetermined amount of time t to verify it and submit a proof of misbehaviour. If none is received after time t the solution is marked as final.

6.1 Verification and Proofs of Misbehaviour

Given an assignment X, a price vector p and a score s, an auction solution is correct if:

- every user is satisfied with their assigned item at a given price, *i.e.,* the solution is an equilibrium;
- no price is lower than the reservation price;
- the declared score is correct;
- every bidder pays for their assigned item the minimum possible price over all possible equilibriums, which characterises the VCG equilibrium.

Equilibrium misbehaviour. A user verifies whether the bidders got assigned the most optimal items (with the highest net valuation, *i.e.,* bidder's valuation after subtracting the item's price). Let the assigned item for a bidder $b \in \mathcal{B}$ be $x_b = i$. For each bidder we have to check if there is another item $\exists j \in \mathcal{I} \backslash \{i\}$ that the bidder would prefer at its current price, *i.e.,* $v_{bi} - p_i < v_{bj} - p_j$; if it is the case, it is an indication that the proposed assignment is not an equilibrium. In a nutshell, the verification consists of a single optimization step of the VDA algorithm. The process detects all the possible misconducts and uses a fraction of the cost required to compute the correct solution. Users who detect the existence of such an item can submit a proof of misbehaviour using the wrongAssignment smart contract method, pointing out the bidder b and the alternative item j associated to a higher net valuation. Upon reception of this proof the smart contract derives bidder's b current net valuation, $v_{bi} - p_i$ as well as the proposed alternative net valuation $v_{bj} - p_j$ before comparing the two. The entire operation requires only 2 subtractions and a single comparison and therefore its monetary cost is negligible, as we show in Section 8.

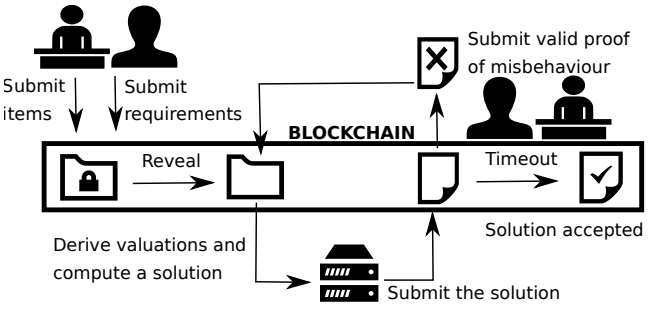

Figure 3. The smart contract offloads heavy computations and accepts computed solutions that can be contested by other users.

Price misbehaviour. A user proceeds by verifying whether items are offered at a valid price, *i.e.,* greater than their reservation price. If it is not the case, any user can point out an index in p with a price lower than the reservation price using the wrongPrice method. The whole operation requires just a single comparison executed on the smart contract.

Score misbehaviour. Each candidate solution contains a score s, corresponding to the sum of net valuations. Users checks its validity in terms of $\sum_{b \in \mathcal{B}} v_{bx_b} - p_{x_b} = s$. This is done by iterating through the assigned items and their price vector while computing the sum of net valuations. If the score is incorrect a user invokes wrongScore.

To verify the proof, the smart contract has to compute the score on-chain, which may be expensive. However, the cost for this calculation will reimbursed by the collateral submitted by the dedicated node if the score was indeed incorrect.

VCG equilibrium misbehaviour. So far, we explained how to verify that a candidate solution is an equilibrium, but not that it is the VCG equilibrium. A VCG equilibrium is defined as the one maximising the sum of bidders' net valuations, *i.e.,* $\sum_{b \in \mathcal{B}} v_{bx_b} - p_{x_b}$. Users compute a set of excess demand and supply. This procedure consists of running a *single* iteration of the Vickrey-Dutch algorithm and can be efficiently performed in polynomial time (Figure 8). A non-empty set indicates that the solution is not a VCG equilibrium with the highest possible score and is thus incorrect.

Proofs submissions timescale. If no valid proof of misbehaviour is submitted by the deadline, the solution is marked as final and the auction finishes. The majority of blockchain platforms do not support scheduling invocation in the future. However, it can be realised by a public function that checks if enough time has passed before marking the solution as final. The submitted solution indicates items assigned to each

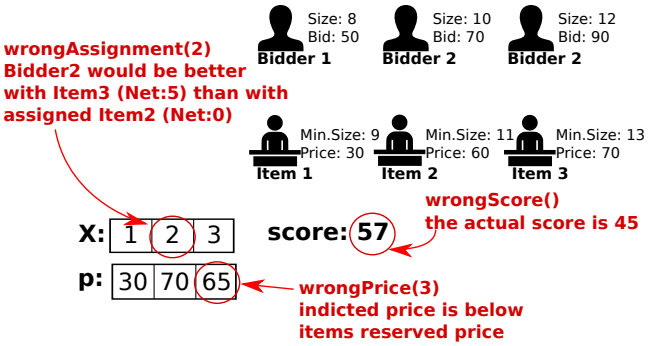

Figure 4. An incorrect solution with different submitted proofs of misbehaviour.

bidder together with the price to pay that is lower than v. Bidders that correctly followed the protocol but get no assigned items may withdraw all of their deposited coins by calling the Withdraw function of the PASTRAMI contract. Recall that PASTRAMI applies a Vickrey-Dutch auction mechanism [54]; the winner of the auction is the bidder with the highest bid v, and pays the price of the second highest bid, v'. Bidders with assigned items are, therefore, able to call Withdraw to withdraw $(v - v')$ coins. Bidders with assigned items can now contact sellers and prove their identity by signing any message using the private key used to participate in the auction (see Section 5.1).

Misbehaviour example. Figure 4 presents an example of an incorrect solution submitted to the smart contract for an auction with 3 bidders and 3 items. The assignment $\mathbf{X}$ is not an equilibrium. Net valuation (the difference between the bid and the price) of Bidder 2 for its currently allocated Item 2 equals to 0 and is lower than for Item 3 (net valuation of 5). A valid proof of misbehaviour would point to the user, allocated Item 2 and alternative Item 3, allowing the smart contract to calculate and compare both valuations efficiently. Furthermore, the price (65) of Item 3 is incorrect as it is below the reservation price specified by the seller (70). A valid proof of misbehaviour simply points to the invalid price allowing the smart contract to verify it against the reservation price and the bid expressed by the assigned bidder. Finally, the declared score is incorrect, as the sum of net evaluations is 45 $(20 + 0 + 25)$.

7 Protocol Analysis and Discussion

We argue that PASTRAMI achieves the design goals described in Section 2.

Auction's economic properties. An auction satisfies all such properties only under the condition of *price-taker participants* [43], *i.e.,* both bidders and sellers have no impact on bids expressed by other users. In the multi-item auctions we consider, it has been proven that Vickrey auctions possesses all these desired attributes based on the assumption of "sealed bids", where neither bidders nor sellers have information about the state of the auction [54]. PASTRAMI through its privacy properties provides a technical implementation of this "sealed bids" assumption, and prevents price manipulations.

Our proofs of misbehaviours cover the full spectrum of potential misbehaviours in terms of proposed solutions. wrongAssignment and wrongPrice limit a user to submit only valid equilibrium solutions respecting minimum prices. That is, the mechanism cannot be unfair to either a participating bidder, who is happy with its assigned items at the given price, or a seller, whose item can only be offered at more than its reservation price. Given the ability to filter solutions that do not fall into the valid equilibrium category, the involved score-based system elicits the VCG equilibrium by allowing anyone to submit the optimal solution with the highest score. At the same time, an incorrect score can be contested using wrongScore. In other words, in PASTRAMI the dedicated node cannot exploit the mechanism by submitting an invalid and/or unfair solution while it is incentivised to submit the VCG equilibrium.

Correctness. PASTRAMI is implemented as a smart contract; its correct execution can be verified by any third party, taking advantage of the *Public Auditability* of the blockchain.

Bidders are bound to their bids as they are first required to commit to their bid, and then to open the commitment by unblinding the signature providing *Bids binding*. Bidders cannot open the commitment to another value than the previously committed, as this would require forging the underlying signature. The users commit to their bid during the commit phase, thus they cannot modify it after observing the bids of other users.

No single authority can issue a signature and steal all the coins in the smart contract—the threshold property of the signature implies that adversaries need to corrupt an arbitrarily large set of authorities for this attack to be possible. Bidders cannot participate in the auction (Section 5) without paying a deposit to receive a valid signature. The timer forces the bidders to commit to their bid before revealing it, preventing any third party from seeing other bidder's bid before committing to a value. Bidders dropping out after committing a bid (and never revealing it) are financially penalized as they cannot withdraw their coins. Once issued, signatures can only be used for the intended auction since they embedded an identifier *id* unique to this auction. As a result, users that do not follow the protocol (*i.e.,* first commit, and then reveal) lose the associated coins, guaranteeing *Fairness*.

Privacy. Upon submission of item descriptions, sellers provide a commitment to their minimum prices instead of the actual values. The minimum price is revealed only during the reveal phase when no additional bids are accepted (*Hidden Minimum Price*).

PASTRAMI takes advantage of the unlinkability property of the underlying blind signature to break the link between

the commit and reveal phase, provided that bidders use fresh addresses. As a result, bidders can submit bids on auctions without revealing their identity enabling *Bidders' privacy*.

Bids are kept private until the timer expires; the blindness property of the underlying signature scheme implies that no information about the bid is revealed while committing to a bid (*Bids privacy*).

Deployment. During the preparation phase (Section 5), Bidders only interact with a subset of the authorities; during the execution phase (Section 6), they only interact with the smart contract; during the execution phase, bidders only interact with the sellers or with the smart contract to withdraw their coins (*Non-interactivity*).

The decentralized nature of the blockchain makes the PASTRAMI smart contract resilient to censorship and guarantees *Openness*. Furthermore, a small subset of authorities cannot block the issuance of blind signatures—the service is guaranteed to be available as long as at least a threshold number of authorities are available.

PASTRAMI does not introduce any single trusted third party; the PASTRAMI contract is executed on a decentralized smart contract platform, and allows threshold issuance by *Distributed authorities*.

PASTRAMI offloads the most CPU-intensive computation to off-chain nodes, decreasing the operations performed on the Smart Contract. The cost of almost all Smart Contract methods is not influenced by increasing the number of items and/or bidders. The only two exceptions consist of submitSolution and wrongScore. The former requires increased on-chain storage when the number of participants increases. However, this cost can be covered from a small commission on system execution. The latter is invoked only when an incorrect solution is proposed and its cost is completely covered by the misbehaving node (*Scalability*). The off-chain solution calculation does not introduce any overhead compared to the original and optimal version of the algorithm. Finally, the verification done by users requires orders of magnitude lower calculation time than calculating the solution itself. We provide an extensive evaluation of all the PASTRAMI components in Section 8.

Extension to other types of auctions. PASTRAMI can be easily extended to support both single-item and combinatorial auctions. Single-item auctions can be seen as a special case of multi-item auctions (with one item only) and does not require any further modifications. The main difference between multi-item and combinatorial auctions lies in the computations' complexity. While it is possible to deterministically derive an optimal solution when using multi-item auction, it has been shown that combinatorial auctions can be modelled as the set packing problem, meaning that they are NP-hard and there is no polynomial-time algorithm for finding the optimal allocation [38]. However, PASTRAMI does not perform auction assignment computations on-chain. The score used could be computed in a similar way to our current design, but with a different set of constraints. Submitting solutions to a combinatorial auction can be seen as a proof of useful work, where better solutions replace those with lower scores. Most of the changes are applied to the off-chain component, to implement algorithms computing assignments and verifying solutions submitted by others.

Non-rational attackers. Non-rational bidders can spend resources to influence the market, *i.e.,* by buying the majority of available storage when they do not need it. Such behavior would temporarily inflate the prices for the scarce resource following the rules of supply and demand. However, PASTRAMI guarantees that the bids are binding (the attacker will pay for the acquired storage) and all bids are sealed/private (the attacker has no way of precisely targeting specific prices/bidders). Non-rational behaviors cannot break these guarantees, and PASTRAMI provides the same level of resilience to market-based attacks as VDA it employs (*i.e.,* bidding strategies are intrinsic to open-market environments). Finally, higher prices paid to workers would encourage additional sellers to join the network, increase the supply and restore the price equilibrium in the long run.

8 Implementation & Evaluation

We evaluate the PASTRAMI prototype on a desktop device for the user side and Ethereum for the Smart Contract platform, and compare it with an implementation of Verifiable Sealed-bid Auction (VSA) [24]. VSA provides similar auditability guarantees, follows a similar to ours approach running heavy computations off-chain, but supports only single-item auctions[6]. We are not aware of any system providing similar guarantees as ours, including privacy, for multi-item auctions. Note that general, verifiable computation platforms require validators to repeat the computations [29], do not provide sufficient privacy protection [50] or expierience multiple orders of magnitude higher complexity [33]

We implement PASTRAMI's off-chain component in Python and the Smart Contract part in Solidity [34]. We use Web3py [20] for communication between both modules. The implementation consists of 1000 Python LoC and 410 Solidity LoC including a simple command-line user interface. We rely on threshold BLS signatures [10] which are blinded using a construction introduced by Boldyreva [9]. This construction provides unlinkability between the signing and verification process and preserves users' anonymity. We use a slightly modified version of the original blind BLS signature of Boldyreva in order to make it compatible with type-3 pairings, which are efficient [25] and supported by Ethereum as pre-compiled contracts (for optimal Ate pairing checks) on the elliptic curve alt_bn128 [14].

[6]This section still evaluates a multi-item version of our system

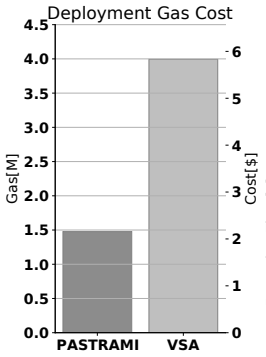

Figure 5. Deployment cost.

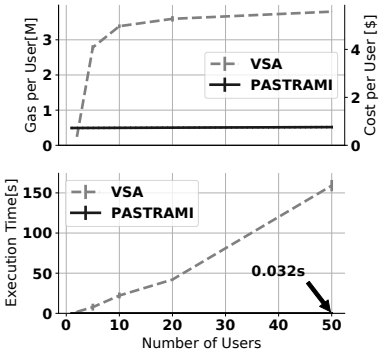

Figure 6. Execution cost.

Operation	Gas	USD
Commit	26,590	0.15
Reveal	364,456	2.06
submitItem	43,556	0.3
RevealMinPrice	52,378	0.361
submitSolution	5,068 + 408*	0.155 + 0.003*
wrongAssign-ment	45,572	0.282
wrongScore	18,048 + 6,494*	0.112 + 0.04*
wrongPrice	35,714	0.221

Table 2. Cost of invoking PASTRAMI functions. *per bidder/item

Setup. We deploy the PASTRAMI Smart Contract on the Ethereum Ropsten Testnet. The application listens for events generated on-chain. It can automatically compute a solution after the reveal phase, or verify solutions submitted by others. All off-chain computations are performed on a Dell Latitude 5590 laptop with an Intel Core i7-8650U CPU and 16 GB of RAM.

The VSA [24] implementation uses two Ethereum Smart Contracts and a client application written in C# [22].

Comparison with VSA. We start by investigating the cost of deploying both platforms on Ethereum (Figure 5). Due to much lower complexity and smaller contract size, PASTRAMI incurs a 2.5 times lower deployment cost than VSA.

We continue by performing single-item auctions with increasing number of users, as VSA only supports this type of auctions. We measure the gas cost for each bidder (Figure 6 top). The PASTRAMI Smart Contract performs only a simple set of operations, related to submitting and revealing bids. It keeps the cost per user low and almost constant when the number of participants increases. In contrast, VSA executes the auction algorithm on-chain, resulting in much higher cost, with a per-bidder costs that raises with the number of bidders.

Finally, we compare the execution time of the off-chain components for all the users and the node calculating solution in PASTRAMI (Figure 6 bottom). Even though PASTRAMI needs to calculate an assignment off-chain, the required operations are much simpler than the costly zero-knowledge proofs used by VSA. It results in much lower computational load and much lower impact of an increasing number of users.

Smart Contract Evaluation. We investigate the cost of invoking each function of the PASTRAMI Smart Contract on Ethereum. We use Remix [21] to calculate this cost in gas and ETH Gast Station [18] to convert it to ETH and to an indicative amount in USD[7]. For the conversion, we use Ethereum Slow mode which increases confirmation time, but decreases monetary cost of all the operations. Table 2 presents the results. We split the number into fixed part (not influenced by the number of bidders/items) and dynamic part expressed as an additional cost per item/bidder. For simplicity, we assume the same number of bidder and items participating in the auction. The presented cost includes both the transaction and the execution cost.

The Commit algorithm is cheap as it only performs basic checks and emits an event asking the authorities to issue a blind signature to the user; our implementation assumes the signature issuance happens off-chain. The Reveal algorithm is more expensive as it performs a signature verification, involving elliptic curve pairing checks. The analogical algorithms for submitting and revealing item description involves only simple commitments and much lower monetary cost.

Submitting a solution with submitSolution consists only of storing the solution on-chain. This involves a small static cost and small cost per item for the storage space. wrongAssignment and wrongPrice use indexes passed as input parameters and perform only simple mathematical operations (comparison and subtractions). Finally, wrongScore requires iterating through the assignment and re-calculating the score in order to verify whether the declared score is correct. The minimum collateral submitted by the dedicated node should be greater than the cost of the most expensive method being part of the proofs of misbehaviour. PASTRAMI automatically computes this value based on information from Table 2 and the size of the auction and does not accept solutions with lower collateral.

Filecoin deployment. As an example of the use of auctions in a decentralised utility platform, we extract data about storage nodes from the Filecoin testnet [4, 5]. At the core of Filecoin lies a *Storage Market* that acts as an exchange point where clients and miners can advertise their requests and offer storage space. Currently, Filecoin maintains an on-chain order book storing a public list of active storage miner offers and client orders. It rely on a direct, peer-to-peer off-chain assignment between users and storage providers. Once two parties reach an agreement, they upload a signed deal to the blockchain.

[7] Measured on November 30th 2019.

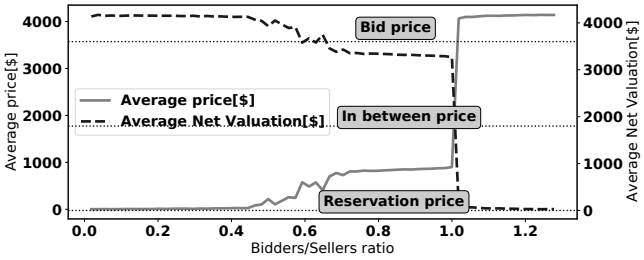

Figure 7. Prices and net valuation derived by PASTRAMI.

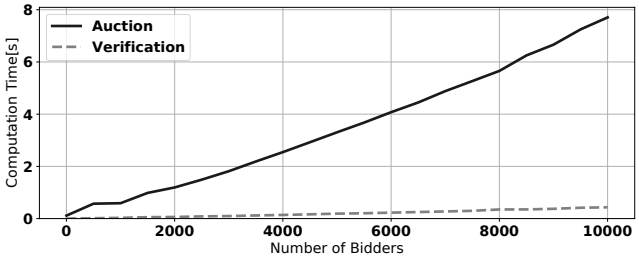

Figure 8. Execution time of auction and verification algorithms.

We query the Filecoin network for all the miners offering storage together with their features. With the current API, we were able to get the current storage price per GB and storage duration, but the available storage capacity is currently unavailable [45].

We perform multi-item Vickrey-Dutch auctions using different numbers of bidders. We convert Filecoin prices to USD per GB per month. The bids are derived from the Google Cloud Storage price [49] and randomized storage duration such that bidders willing to rent storage space for longer pay a lower price per month.

Figure 7 presents the evaluation of average price and average net valuation in USD. Dotted lines indicate simpler mechanisms where bidders manually contact advertised workers and agree to pay reservation price, valuation price or value in between. We observe that the auction algorithm is able to adapt to changes in the supply/demand ratio and derive the most optimal prices. When the number of bidders exceeds the number of items, the price raises to the average valuation and does not increase further, as we assume that at this point, users will switch to cloud storage providers.

Finally, we measure the time required to calculate a solution and verify it with increasing number of users (Figure 8). Even for 10,000 users, PASTRAMI is able to derive an optimal solution in less than 8s. Furthermore, the verification process proves to be efficient and completes within 500ms even for the largest auction.

9 Related Work

We cover related work on private computation over blockchains, auction platforms, and decentralised cloud platforms. We summarize the most related solutions and their security features in Table 3.

Several frameworks aim at hiding private data submitted to a public ledger. Hawk [33] divides Smart Contracts into public and private parts and secure private input using zero-knowledge proofs, but requires a centralized trusted manager to operate. ShadowEth [56] allows processing confidential Smart Contract data using Trusted Execution Environments (TEE). However, such a scheme requires users to trust the hardware vendor and can expose the system to TEE's vulnerabilities [17, 53, 55].

Blass and Kerschbaum [7] propose Strain, a system that preserves sealed-bid auctions privacy against malicious participants. Strain uses a two-party comparison protocol based on Secure Multi-Party Computation (MPC), but has a flaw that reveals the order of bids. Furthermore, running protocols involving MPC on blockchain is not efficient due to the extensive computations and the number of rounds involved. Galal and Youssef [24] present a protocol ensuring public verifiability, privacy of bids, and fairness. However, the solution scales badly with the number of bidders and relies on random number retrieved from blockchains that are not proven to be secure. This scheme was later improved using zk-SNARKS [23], but it still relies on a centralized party for zero-knowledge proofs and does not protect bidders' identities. An alternative approach was proposed by Bogetoft *et al.* [8]. Their system uses MPC to perform auctions on encrypted bids. However, such a scheme reveals the final assignment between bidders and items and does not provide transparency.

Filecoin [35] does not implement an automated system assigning clients to storage nodes. Users are required to chose storage nodes on a one-to-one basis and offers are publicly posted on the blockchain. Other industrial systems such as Golem [51], iExec [1] or SONM [2] either do not specify their requester—worker assignment technique or rely on similar, non-transparent solutions. All those platform could use PASTRAMI to increase their level of security and automatically determine optimal price for services.

10 Conclusion

We presented PASTRAMI, a system for determining an assignment between providers and users and deriving optimal prices in a decentralised environment, while preserving the privacy and accountability of participants.

PASTRAMI enables a transparent use of Vickey-Dutch multi-item auctions to derive prices, assignments and maximise social wellness. It secures users bids as well as the identity of the bidders using blind signatures. In contrast with previous work, PASTRAMI does not rely on a trusted third party to issue signatures, but rather on a set of entities that can

System	Bids Privacy	Bidders' Privacy	Non-Interactivity	Distributed Authority	Trusted Hardware	Public Auditability
ShadowEth [56]	✓	✗	✓	●	Intel SGX [19]	✓
Hawk [33]	✓	✗	✓	◒	None	✓
Strain [7]	✓	✗	✗	○	None	✓
Galal *et al.* [23]	✓	✗	✗	○	None	✓
Bogetoft *et al.* [8]	✓	✗	✓	●	None	✗
Filecoin [35]	✗	✗	✗	●	None	✓
Galal *et al.* [24]	✓	✗	✓	○	None	✓
PASTRAMI	✓	✓	✓	●	None	✓

Table 3. Comparison of properties achieved by related systems. The decentralisation property reads as follows; ○ : relies on a trusted third party, ◒ : relies on a trusted third party for only one (or some) of the properties described in Section 2, ● : does not rely on any trusted third party.

be freely chosen by users. These distributed authorities issue only partial signatures that are merged locally by each users, protecting the system from a subset of malicious authorities.

To enable scalability, heavy computations are performed off-chain and submitted to a smart contract. An assignment for an auction can be challenged by proofs of misbehaviour submitted by users to this smart contract. We have shown how PASTRAMI can be deployed on Ethereum blockchain and adapted to shared economy systems with the example of Filecoin, and our evaluation shows its reduced costs compared to a system with similar auditability guarantees.

Acknowledgments

This work is partially supported by the Belgian FNRS project DAPOCA (33694591), EPSRC INSP Early Career Fellowship under grant agreement number EP/M003787/1 and Facebook Calibra.

A Blind BLS signatures

We recall the cryptographic construction of the blind BLS signature [9] used in our implementation. For the sake of simplicity, we describe below a key generation algorithm BS.TTPKeyGen as executed by a trusted third party; this protocol can however be executed in a distributed way as illustrated by Gennaro *et al.* [26] under a synchrony assumption, and as illustrated by Kate *et al.* [32] under a weak synchrony assumption.

❖ **BS.Setup**$(1^\lambda) \rightarrow (params)$**:** Choose a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with order p, where p is an λ-bit prime number. Let g_1 be a generator of $\mathbb{G}_1$, and g_2 a generator of $\mathbb{G}_2$. The system parameters are $params = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2)$.

❖ **BS.TTPKeyGen**$(params, t, n) \rightarrow (x, y)$**:** Pick a polynomial v of degree $t-1$ with coefficients in $\mathbb{F}_p$, and set $(x, y) = (v(0), g_2^{v(0)})$. Issue to each authority $i \in [1, \ldots, n]$ a secret key $x_i = v(i)$, and publish their verification key $y_i = g_2^{x_i}$.

❖ **BS.PrepareBlindSign**$(m) \rightarrow (\tilde{h})$**:** pick a random $r \leftarrow \mathbb{F}$; output $\tilde{h} = H^*(m)^r \in \mathbb{G}_1$.

❖ **BS.BlindSign**$(x_i, \tilde{h}) \rightarrow (\tilde{\sigma}_i)$**:** output $\tilde{\sigma}_i = \tilde{h}^{x_i}$.

❖ **BS.Unblind**$(r, \tilde{\sigma}) \rightarrow (\sigma)$**:** output $\sigma = \tilde{\sigma}^{(-r)}$.

❖ **BS.AggSig**$(\sigma_1, \ldots, \sigma_t) \rightarrow (\sigma)$**:** Output $\sigma = \prod_{i=1}^{t} \sigma_i^{l_i})$, where l is the Lagrange coefficient:

$$l_i = \left[\prod_{j=1, j\neq i}^{t} (0-j)\right]\left[\prod_{j=1, j\neq i}^{t} (i-j)\right]^{-1} \mod$$

❖ **BS.Verify**$(y, m, \sigma) \rightarrow (true/false)$**:** compute $h = H^*(m)$; output *true* if $e(h, y) = e(\sigma, g_2)$; otherwise output *false*.

References

[1] 2018. iExec Whitepaper. https://iex.ec/whitepaper/iExec-WPv3.0-English.pdf.

[2] 2018. SONM: Decentralized Fog Computing Platform. https://sonm.com.

[3] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2014. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 443–458.

[4] Juan Benet. 2014. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561* (2014).

[5] Juan Benet, David Dalrymple, and Nicola Greco. 2018. *Proof of replication*. Technical Report. Technical report, Protocol Labs, July 27, 2017. https://filecoin. io/proof-of-replication. pdf. Accessed June.

[6] George Bissias, A. Pinar Ozisik, Brian N. Levine, and Marc Liberatore. 2014. Sybil-Resistant Mixing for Bitcoin. In *Workshop on Privacy in the Electronic Society (WPES '14)*. ACM, 149–158.

[7] Erik-Oliver Blass and Florian Kerschbaum. 2018. Strain: A secure auction for blockchains. In *European Symposium on Research in Computer Security*. Springer, 87–110.

[8] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. 2009. Secure multiparty computation goes live. In *Intl. Conf. on Financial Cryptography and Data Security*. Springer, 325–343.

[9] Alexandra Boldyreva. 2003. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In *International Workshop on Public Key Cryptography*. Springer, 31–46.

[10] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *Eurocrypt*, Vol. 2656. Springer, 416–432.

[11] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. *ASIACRYPT* (2001), 514–532.

[12] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. 2014. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *Financial Cryptography 2014*.

[13] Vitalik Buterin et al. 2013. Ethereum white paper.

[14] Vitalik Buterin and Christian Reitwiessner. 2017. Ethereum Improvement Proposal 197. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-197.md.

[15] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[16] David Chaum. 1983. Blind signatures for untraceable payments. In *Advances in cryptology*. Springer, 199–203.

[17] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. 2019. SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 142–157.

[18] Concourse Open Community. [n. d.]. ETH Gas Station. https://www.ethgasstation.info.

[19] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[20] Ethereum foundation. [n. d.]. A python interface for interacting with the Ethereum blockchain and ecosystem. https://github.com/ethereum/web3.py.

[21] Ethereum foundation. [n. d.]. Remix. https://remix.ethereum.org.

[22] Hisham S. Galal. [n. d.]. Auctioneer source code. https://github.com/HSG88/AuctionContract/tree/master/Auctioneer.

[23] Hisham S Galal and Amr M Youssef. 2018. Succinctly Verifiable Sealed-Bid Auction Smart Contract. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 3–19.

[24] Hisham S Galal and Amr M Youssef. 2018. Verifiable sealed-bid auction on the ethereum blockchain. In *International Conference on Financial Cryptography and Data Security*. Springer, 265–278.

[25] Steven D Galbraith, Kenneth G Paterson, and Nigel P Smart. 2008. Pairings for cryptographers. *Discrete Applied Math.* 156, 16 (2008), 3113–3121.

[26] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure distributed key generation for discrete-log based cryptosystems. In *Eurocrypt*, Vol. 99. Springer, 295–310.

[27] Michael Harkavy, J Doug Tygar, and Hiroaki Kikuchi. 1998. Electronic Auctions with Private Bids.. In *USENIX Workshop on Electronic Commerce*.

[28] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. 2016. TumbleBit: An untrusted Bitcoin-compatible anonymous payment hub. In *NDSS 2017*.

[29] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. 2018. Arbitrum: Scalable, private smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1353–1370.

[30] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending fast payments in bitcoin. In *ACM conference on Computer and communications security*. ACM, 906–917.

[31] Aniket Kate, Yizhou Huang, and Ian Goldberg. 2012. Distributed Key Generation in the Wild. *IACR Cryptology ePrint Archive* 2012 (2012), 377.

[32] Aniket Kate, Yizhou Huang, and Ian Goldberg. 2012. Distributed Key Generation in the Wild. Cryptology ePrint Archive, Report 2012/377. https://eprint.iacr.org/2012/377.

[33] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 839–858.

[34] Michał Król, Alberto Sonnino, and Tasiopoulos Argyrios. 2020. Pastrami source code. https://github.com/harnen/FilecoinPricingMechanism.

[35] Protocol Labs. 2018. *FileCoin: A Decentralized Storage Network*. Technical Report.

[36] Storj Labs. 2018. Storj: A Decentralized Cloud Storage NetworkFramework. "https://storj.io/storj.pdf".

[37] Erick Lavoie, Laurie J. Hendren, Frederic Desprez, and Miguel Correia. 2019. Pando: Personal Volunteer Computing in Browsers. In *20th ACM/IFIP International Conference on Middleware*.

[38] Daniel Lehmann, Rudolf Müller, and Tuomas Sandholm. 2006. The winner determination problem. *Combinatorial auctions* (2006), 297–318.

[39] Gregory Maxwell. 2013. CoinJoin: Bitcoin privacy for the real world. https://bitcointalk.org/index.php?topic=279249.

[40] Sarah Meiklejohn and Rebekah Mercer. 2018. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies* 2018, 2 (2018), 105–121.

[41] Debasis Mishra and David C Parkes. 2004. *Multi-item Vickrey-Dutch auction for unit-demand preferences*. Technical Report. Technical Report Harvard EconCS Technical Report, Harvard University.

[42] Debasis Mishra and David C Parkes. 2009. Multi-item Vickrey-Dutch auctions. *Games and Economic Behavior* 66, 1 (2009), 326–347.

[43] Roger B Myerson and Mark A Satterthwaite. 1983. Efficient mechanisms for bilateral trading. *Journal of economic theory* 29, 2 (1983), 265–281.

[44] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[45] Filecoin project. [n. d.]. Feature request: Clients can get estimates of miner storage. https://github.com/filecoin-project/go-filecoin/issues/2991.

[46] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2014. CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin. In *ESORICS (2) (Lecture Notes in Computer Science)*, Vol. 8713. Springer, 345–364.

[47] Lloyd S Shapley and Martin Shubik. 1971. The assignment game I: The core. *International Journal of game theory* 1, 1 (1971), 111–130.

[48] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, and George Danezis. 2018. Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers. *arXiv preprint arXiv:1802.07344* (2018).

[49] Google Cloud Storage. [n. d.]. Pricing details by storage class. https://cloud.google.com/storage/pricing-summary/.

[50] Jason Teutsch, Loi Luu, and Christian Reitwiessner. 2016. Truebit: A verification and storage solution for blockchains.

[51] The Golem Project. 2016. Golem Whitepaper. https://golem.network/doc/Golemwhitepaper.pdf.

[52] Luke Valenta and Brendan Rowan. 2015. Blindcoin: Blinded, Accountable Mixes for Bitcoin. In *Financial Cryptography and Data Security*, Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff (Eds.). 112–126.

[53] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium*.

[54] William Vickrey. 1961. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance* 16, 1 (1961), 8–37.

[55] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM.

[56] Rui Yuan, Yu-Bin Xia, Hai-Bo Chen, Bin-Yu Zang, and Jan Xie. 2018. ShadowEth: Private Smart Contract on Public Blockchain. *Journal of Computer Science and Technology* 33, 3 (2018), 542–556.