Scalable Off-Chain Auctions

Mohsen Minaei
Visa Research

Ranjit Kumaresan
Visa Research

Andrew Beams
Visa Research

Pedro Moreno-Sanchez
IMDEA Software Institute
Visa Research
MPI-SP

Yibin Yang
Georgia Institute of Technology

Srinivasan Raghuraman
Visa Research
MIT

Panagiotis Chatzigiannis
Visa Research

Mahdi Zamani
Visa Research

Duc V. Le
Visa Research

***Abstract*—Blockchain auction plays an important role in the price discovery of digital assets (e.g. NFTs). However, despite their importance, implementing auctions directly on blockchains such as Ethereum incurs scalability issues. In particular, the on-chain transactions scale poorly with the number of bidders, leading to network congestion, increased transaction fees, and slower transaction confirmation time. This lack of scalability significantly hampers the ability of the system to handle large-scale, high-speed auctions that are common in today's economy.**

In this work, we build a protocol where an auctioneer can conduct sealed bid auctions that run entirely off-chain when parties behave honestly, and in the event that k bidders deviate (e.g., do not open their sealed bid) from an n-party auction protocol, then the on-chain complexity is only $O(k)$. This improves over existing solutions that require $O(n)$ on-chain complexity, even if a single bidder deviates from the protocol. In the event of a malicious auctioneer, our protocol still guarantees that the auction will successfully terminate. We implement our protocol and show that it offers significant efficiency improvements compared to existing on-chain solutions. Our use of zkSnark to achieve scalability also ensures that the on-chain contract and other participants do not learn anything about the bidders' identities and their respective bids, except for the winner and the winning bid amount.

I. Introduction

In an online auction, sellers advertise the sale of arbitrary assets and buyers can place bids as the price they are willing to pay for such assets. Online auctions are widely used in the current world economy, moving billions of dollars in exchange for goods and services [1], [2]. However, online auctions rely on the trustworthiness of the auctioneer to correctly run the auction. Blockchains with smart contract capabilities (e.g., Ethereum) have lately been leveraged to add transparency to the process: The auction's logic can be implemented as a smart contract that, when deployed on the blockchain, is in charge of *(i)* receiving the asset from the seller; *(ii)* receiving bids while the bidding interval is open; *(iii)* after the bid interval is closed, selecting the winning bid according to the type of auction (e.g., highest bid in first price sealed bid auctions); and *(iv)* transferring the winning bid to the seller whereas the corresponding bidder obtains the auctioned asset.

Although blockchain-based auctions are a promising alternative to online auctions, there are several obstacles to ensuring the security and privacy of the participants. Apart from *correctness* (i.e., seller gets highest bid while corresponding bidder gets the auctioned good), a bidder should not know other bids before committing to its own (*bid privacy*). Moreover, from the security point of view, bidders should not be able to change their committed bids (*bid binding*) whereas the auctioneer should not be able to give undue advantage to malicious bidders (*non-malleability*). The auction should terminate even in the presence of malicious participants (*liveness*), that must get penalized if they deviate from the protocol (*financial fairness*). Finally, for practicality, bidders should not need to interact with each other (*non-interactivity*) and the overall on-chain cost[1] should not depend on the number of bidders, in the optimistic case (*efficiency*).

Simultaneously achieving the aforementioned properties is challenging. For instance, correctness requires comparison across all bids to determine the winning one, while privacy mandates that the actual bid values remain hidden from bidders and blockchain observers. An off-the-shelf multi-party protocol among bidders to compute the auction functionality while preserving the required privacy guarantees would violate the non-interactivity requirement.

In addition to security and privacy, scalability is an efficiency requirement of utmost importance. Transaction processing in decentralized blockchains is highly limited to few transactions per second, and doing intensive cryptographic operations on-chain (such as commit and reveal for sealed bid auctions) are likely to be very expensive and impractical. Ideally, one would want the entire auction to be conducted off-chain (excluding the asset transfer from seller to winning bidder). However, doing so would sacrifice transparency since

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230410
www.ndss-symposium.org

[1]We distinguish between *on-chain costs* (gas fees for blockchain transactions, bounded by block limits) and *off-chain costs* (computation/communication outside blockchain, essentially cheap). The bottleneck is on-chain cost due to limited block space and high fees.

a malicious auctioneer can violate, for e.g., auction correctness, and get away with it.

Our System in a Nutshell. We designed a robust system where an auctioneer coordinates the communication between bidders and the smart contract. The auctioneer is considered fully malicious for security properties: it cannot steal users' funds or abort the auction without being punished financially. Moreover, our protocol offers bid privacy, that is, bids are hidden even from the auctioneer during the bidding phase. Finally, in the optimistic case, where the auctioneer does not deviate, our protocol also offers *post-auction privacy*. This property is an improvement over existing online bidding protocols, such as eBay [3] or OpenSea [4], which have no privacy since bid information (e.g., bid amounts, bidder addresses, and the bid history) is always publicly visible.

In the presence of such an auctioneer, our system is designed in stages as follows. First, at the creation stage, the seller agrees on the auction parameters and the auction good (e.g., an NFT) with the auctioneer. The auctioneer establishes the auction by deploying a smart contract in the blockchain with the mentioned auction setup information. This information includes the collateral amount the auctioneer commits to pay if the auction fails. Second, during the bidding stage, each bidder commits the fully sealed (even to the eyes of the auctioneer) bid to the auctioneer. During this stage, both parties agree on a collateral amount that will be forfeited if either party misbehaves. At the end of this stage, each bidder gets a bid's inclusion confirmation from the auctioneer, who in turn pushes the list of bids to the contract in an accumulated form for efficiency. Fixing the set of fully sealed bids at this point helps to achieve bid binding and bid privacy. After the bid interval is finished, the opening stage permits two actions from bidders: Either a bidder opens their sealed bid to the auctioneer if it was included in the contract in the previous step; or a bidder challenges the auctioneer about the lack of their sealed bid in the contract. In the former case, the auction enters the settle stage, which we overview later. In the latter case, the contract's logic is such that it can deterministically decide the cheating party and financially punish them. This functionality helps to achieve financial fairness.

During the final stage, called settle stage, the auctioneer interacts with the smart contract to indicate the winning bid along with a (zero-knowledge) proof attesting the veracity of the winning conditions (e.g., the winning bid indeed is the one with the highest value in a first price sealed bid auction). The auction ends with the bid being transferred to the winning bidder, who in turns gets the auctioned asset, whereas the rest of the bidders get refunded. The system ensures auction correctness and maintains non-interactivity among bidders.

Our Contributions. In this work, we propose the first scalable system capable of handling sealed-bid auctions with over $1,000$ bidders. We build a protocol where an auctioneer can conduct sealed bid auctions that run entirely off-chain when parties behave honestly, and in the event that k bidders deviate (e.g., do not open their sealed bid) from an n-party auction protocol, then the on-chain complexity is only $O(k)$. This improves over existing solutions that require $O(n)$ on-chain complexity even if a single bidder deviates from the protocol (see Section II).

Our implementation, deployed on a private EVM chain using Hyperledger Besu, demonstrates strong scalability. We evaluated our protocol against three baseline approaches: (i) naive on-chain auctions, (ii) off-chain auctions without zk-SNARKs, and (iii) Riggs-TC (CCS'23), the current state-of-the-art in on-chain auctions [5]. Our protocol is the only solution that successfully scales beyond $1,024$ bidders on Ethereum, while Riggs-TC reaches the block gas limit with just 20 bidders. Moreover, the use of zkSnark ensures that the on-chain contract and other participants do not acquire any information about the bidders' identities and their respective bids, except for the winner and the winning bid amount. Finally, we analyze our protocol in the Universal Composability framework. We provide an ideal functionality modeling of the auction and show that, under the right assumptions, our protocol achieves UC security. Also, we use game-theoretic analysis to demonstrate that our protocol is robust against *rational* participants.

II. Related Work

On-chain Auction. Sealed bid auctions can be implemented directly on Layer-1 as in [6]–[19]. The auction contract will accept sealed bids until a certain deadline, `registrationDeadline`. Following this, and until another deadline, `auctionDeadline`, the contract accepts the opening of the sealed bids. After the `auctionDeadline` has passed, anyone can invoke the auction contract to perform an atomic swap of the NFT asset (to the winning bidder) and the amount corresponding to the highest bid (to the seller).

Note that bidders will be required to post collateral to the auction contract. The reason for this is two-fold. First, it ensures that the bidder has enough money to cover the purchase of the NFT in case it wins the auctions. Second, this collateral can also be used to punish bidders who refuse to open their sealed bids. This is important since otherwise an adversary can launch the following *malleability* attack without incurring any penalty. A malicious user can impersonate multiple bidders with bids ranging from 1 through `maxPrice` and then open only the bid which is one more than the highest honest bid. Alternatively, a malicious seller can similarly impersonate multiple bidders and wait to see the highest bid and then decide whether to open a higher bid or not.

We note that the main drawback of the above solution is that its on-chain complexity is proportional to the number of bidders (who have to submit their sealed bids and the corresponding openings), even if all parties are honest.

Using State Channels. To minimize on-chain complexity, one could use state channels [20]–[25] to implement the auction off-chain. Parties off-chain decide on the contract source code of the auction contract and the salt that they are going to use to deploy the auction contract via the `CREATE2` opcode.[2] Note

[2]https://legacy.ethgasstation.info/blog/what-is-create2/

TABLE I: On-chain complexity comparison with other works. Here n denotes the number of bidders.

Auction Protocol	All participants are honest	k bidders deviate	Malicious Auctioneer	Privacy
On-chain Auction	$O(n)$	$O(n)$	n/a	No
State Channels	$O(1)$	$O(n)$	n/a	No
Auction on (zk)-rollup	$O(n)$	$O(n)$	n/a	No
Ours	$O(1)$	$O(k)$	$O(n)$	Yes

that the contract is not deployed yet, but when deployed, it will be created at a deterministic address thanks to `CREATE2`. Now, parties exchange transactions to the auction contract and attain consensus off-chain on these. If all parties behave honestly, then the only on-chain footprint of the auction execution is the exchange of the NFT to the winning bidder. However, if some party misbehaves, then the auction contract is deployed on-chain, and all the transactions that were exchanged off-chain are then played back on-chain. There are many subtle details that we omit here, but the key takeaway is that even if one bidder misbehaves, the entire auction needs to be carried out on-chain. Thus, the worst case on-chain complexity in this case is $O(n)$, where n denotes the number of bidders.

Using Rollups. A natural Layer-2 solution would be to "roll up" the straw man solution (either in an optimistic rollup or a ZK rollup) [26]–[28]. However, in such solutions, a malicious sequencer can deny an honest bidder from opening its commitment. This would result in the honest bidder losing its collateral. The only way to avoid honest bidders from losing money would be to continue the execution (i.e., dispute resolution) on Layer-1, however, this would result in a worst-case $O(n)$ on-chain complexity. Note that in optimistic and ZK rollups, rolling up the solution still results in on-chain complexity $O(n)$ even in the optimistic case, since the transaction data is dependent on n.

Existing Off-chain Solutions May Still Require $O(n)$ On-chain Cost. While existing off-chain approaches like state channels and rollups reduce computational overhead, they fundamentally fail to achieve true scalability due to their dispute resolution mechanisms. *State channels* require posting all channel states on-chain during disputes, for a 1,000-bidder auction, even a single malicious participant forces all 1,000 bidder interactions on-chain. *Rollups* must post complete transaction data on-chain for verification, resulting in $O(n)$ data availability costs regardless of dispute scenarios. In contrast, our protocol achieves $O(k)$ on-chain complexity by isolating only the k misbehaving bidders' actions on-chain, while honest bidders remain entirely off-chain. This design difference enables practical auctions with $1,000+$ participants where existing solutions reach prohibitive gas costs.

Non-sealed Bid Auctions. Typical NFT sales, e.g., via OpenSea [4], are conducted through non-sealed bid English auctions directly on Layer-1. This has the advantage that the seller does not need to set a max bid price, and the NFT could be sold potentially for a large sum of money. On the other hand, not conducting a sealed bid auction opens up various attack vectors, such as insider trading. Since bidders can submit bids multiple times, this also increases the total amount of on-chain activity during auction time, thereby increasing the gas price for regular users (not participating in the auction).

Other Blockchain Auctions. Ethereum Name Service (ENS) [29], which allows users to register human-readable domain names that can be used to interact with Ethereum contracts, uses auctions to auction off newly released domain names to the highest bidder. Typical DeFi protocols often use auctions to determine the price of assets or to distribute tokens to users. For example, in a liquidity auction, users can bid on the price of an asset, and the protocol will use the bids to determine the asset's price. In a token distribution auction, users can bid on tokens, and the protocol will distribute the tokens to the highest bidders.

Our work: Using a Programmable Payment Channel. Our approach relies on a new notion called programmable payment channel (PPC) [30], which can facilitate any off-chain computations between two participants sharing a channel. In this work, we assume there is an untrusted hub that has a PPC with each participant. In the scenarios when all participants act honestly, our design can achieve an $O(1)$ on-chain cost, similar to the efficiency of state channels. However, diverging from multiparty state channels, our protocol leverages pairwise state channels (implemented via PPC) to decrease on-chain disputes, making such interventions primarily necessary only when dealing with malicious parties. Moreover, our construction embeds sealed bids within a Merkle tree to optimize on-chain storage costs. However, this approach has a drawback. Should bidders diverge from the expected behavior, removing their bids comes at a computational cost. Specifically, if k bidders deviate, the system requires $O(k)$ operations to exclude these participants. Nevertheless, this is still more efficient compared to other existing alternatives. As our system does depend on an untrusted auctioneer, a malicious auctioneer could induce $O(n)$ on-chain complexity if they decline to collaborate. However, there is an asymmetry concerning on-chain tasks that deters such actions. This is because the auctioneer would be compelled to engage in a challenge-response mechanism with n other bidders.

Comparison with Rigg [5]. Tyagi *et al.* [5] proposed an on-chain auction protocol, Rigg, designed to finalize auctions even when up to $n-1$ bidders collude. This is enabled by timed commitments for sealed bids, allowing honest users to open bids if dishonest ones refuse to. Although the protocol is theoretically sound, practical implementation on platforms like Ethereum faces significant challenges in terms of gas cost and fairness. Firstly, the gas costs associated with the Rigg protocol are deemed prohibitive. Rigg requires extensive non-interactive zero-knowledge verification throughout all stages

(bid collection, bid forced opening), resulting in gas costs ranging from $1.9m$ to $4m$ per user (c.f. Section VI). This cost is substantially higher than that of simpler on-chain auctions. Secondly, the effectiveness of the forced opening mechanism on blockchain systems is questionable. This concept relies on honest users solving timed commitments to open dishonest users' bids and earn collateral that is locked. However, these users face the risk of being front-run during transaction submission, potentially leading to the theft of their rewards by others who did not solve the required puzzles. Additionally, similar to optimistic rollups, Rigg encounters an inherent incentive challenge. Honest users may find themselves performing computations without compensation if bidders decide to open their bids. In contrast, our approach utilizes an untrusted hub for conducting auctions, necessitating minimal on-chain dispute resolution, primarily when confronting malicious parties. Consequently, our construction eliminates the need for costly on-chain NIZK verifications at various auction stages as well as costly off-chain proof generations for all parties.

III. Preliminaries

A. Cryptographic Building Blocks

Notation. We denote by 1^λ the security parameter and by $\texttt{negl}(\lambda)$ a negligible function in λ. We express a pair of public and private keys by $(\mathsf{pk}, \mathsf{sk})$. We use $\mathbb{Z}_{\geq a}$ to denote the set of integers that are greater or equal to a, $\{a, a+1, \dots\}$. We let PPT denote probabilistic polynomial time. We use $[k]$ to denote the set, $\{1, \dots, k\}$ We use a shaded area i, j, k to denote the private inputs in the relation $st : \{(a, b, c; i, j, k) : f(a, b, c, i, j, k) = \text{“}\texttt{True}\text{”}\}$. We use $st[a, b, c \dots]$ to denote those fixed and public values of an instance of the relation st.

Collision-Resistant Hash Functions. A family H of hash functions is collision-resistant, iff for all PPT $\mathcal{A}$ given $h \xleftarrow{\$} H$, the probability that $\mathcal{A}$ finds x, x', such that $h(x) = h(x')$ is negligible. We refer to the cryptographic hash function h as a fixed function, $h : \{0,1\}^* \to \{0,1\}^\lambda$. For the formal definitions of the cryptographic hash function family, we refer the reader to [31].

Digital Signature. A cryptographic digital signature allows the verification of the authenticity and integrity of a digital message or transaction.

Definition 1 (Digital Signature)**.** A digital signature scheme, Σ, with a message space $\mathcal{M}$ and a signature space, $\mathcal{S}$ consists of three algorithms:

- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \texttt{KeyGen}(1^\lambda)$: The *probabilistic* generation algorithm takes as input the security parameter and outputs a pair $(\mathsf{sk}, \mathsf{vk})$ of secret key and verification key.
- $\sigma \leftarrow \texttt{Sign}(m, \mathsf{sk})$ for any $m \in \mathcal{M}$: takes as input a private key sk and a message m from the message space $\mathcal{M}$ and outputs a signature σ in the signature space $\mathcal{S}$.
- $0/1 \leftarrow \texttt{SigVerify}(\sigma, m, \mathsf{vk})$ takes as input a public key vk, a message m, and a signature σ, and outputs the validity of the signature, $b \in \{0, 1\}$.

We require the signature scheme Σ to satisfy the *correctness* and the *existential unforgeability* properties of a digital signature scheme.

Commitment Scheme. A commitment scheme allows an entity to commit to a value while keeping it hidden, with the option of later revealing the value. A commitment scheme contains two rounds: committing and revealing. During the committing round, a client commits to selected values while concealing them from others. During the revealing round, the client can choose to reveal the committed value.

Definition 2 (Commitment Scheme)**.** A commitment scheme consists of two algorithms:

- $\mathsf{cm} \leftarrow \texttt{P}_{\texttt{com}}(m, r)$ takes a message m and a secret randomness r as inputs and returns the commitment cm.
- $0/1 \leftarrow \texttt{V}_{\texttt{com}}(m, r, \mathsf{cm})$ accepts a message m, a commitment cm and a decommitment value r as inputs, and returns 1 if the commitment is opened correctly and 0 otherwise.

A commitment scheme should satisfy the properties of *hiding*, *binding*, *non-malleable*, and *non-interactive*. For the formal definitions of these properties, we refer readers to [32].

zkSnark. A zero-knowledge Succinct Non-interactive ARgument of Knowledge (zkSnark) is a "succinct" non-interactive zero-knowledge proofs (NIZK) for arithmetic circuit satisfiability. For a field $\mathbb{F}$, an arithmetic circuit C takes as inputs elements in $\mathbb{F}$ and outputs elements in $\mathbb{F}$. We adopt a similar definition from Zerocash [33] to define the arithmetic circuit satisfiability problem. An arithmetic circuit satisfiability problem of a circuit $C : \mathbb{F}^n \times \mathbb{F}^h \to \mathbb{F}^l$ is captured by the relation $st_C : \{(x, \mathsf{wit}) \in \mathbb{F}^n \times \mathbb{F}^h : C(x, \mathsf{wit}) = 0^l\}$, with the language $\mathcal{L}_C = \{x \in \mathbb{F}^n \mid \exists\ \mathsf{wit} \in \mathbb{F}^l\ s.t\ C(x, \mathsf{wit}) = 0^l\}$.

Definition 3 (zkSnark [33])**.** A zero-knowledge Succinct Non-interactive ARgument of Knowledge for arithmetic (zkSnark) circuit satisfiability is a triple of efficient algorithms $(\texttt{Setup}, \texttt{Prove}, \texttt{Verify})$:

- $(\mathsf{ek}, \mathsf{vk}) \leftarrow \texttt{Setup}(1^\lambda, C)$ takes as input the security parameter and the arithmetic circuit C, outputs an evaluation key ek, and a verification key vk.
- $\pi \leftarrow \texttt{Prove}(\mathsf{ek}, x, \mathsf{wit})$ takes as input the evaluation key ek and $(x, \mathsf{wit}) \in R_C$, outputs a proof π for the statement $x \in \mathcal{L}_C$
- $0/1 \leftarrow \texttt{Verify}(\mathsf{vk}, x, \pi)$ takes as input the verification key vk, the public input x, the proof, π, outputs 1 if π is valid proof for $x \in \mathcal{L}_C$.

A zkSnark requires *Correctness, Soundness*, *Zero-knowledge*, and *Simulation Extractability* properties.

Merkle Tree. In this work, we are interested in the Merkle tree as an authenticated data structure for set membership.

Definition 4 (Merkle Tree [34])**.** A Merkle tree is an authenticated data structure using a collision-resistant hash function h. A Merkle tree consists of four algorithms that work as follows:

- $\mathsf{root} \leftarrow \texttt{Init}(1^\lambda, X)$ takes the security parameter and a list $X = (x_1, \dots, x_n)$ as inputs and outputs a root, root.

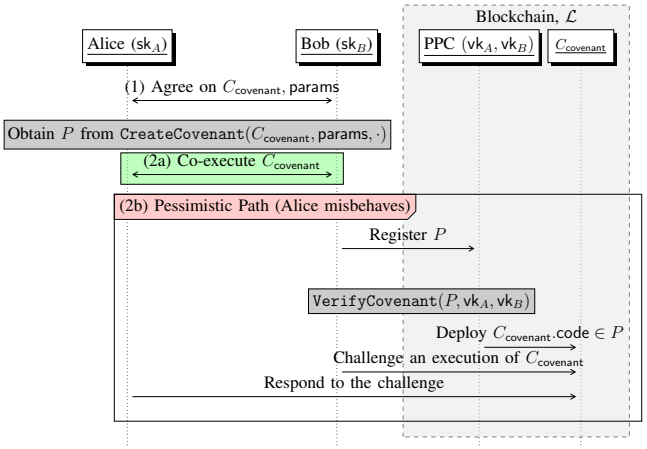


Fig. 1: An overview of creating and executing promises in PPC. Step (2a) indicates the optimistic path, while step (2b) indicates a pessimistic path where an on-chain resolution is needed.

- $\mathsf{path}_i \leftarrow \mathtt{Prove}(i, x, X)$ takes an element $x \in \{0,1\}^*, 1 \leq i \leq n$ and $X = (x_1, \ldots, x_n)$ as inputs, and outputs the proof path_i, a list of internal nodes inside the tree, that can be used to recompute the root.
- $0/1 \leftarrow \mathtt{Verify}(i, x_i, \mathsf{root}, \mathsf{path}_i)$ takes an element, $x_i \in \{0,1\}^*$, an index $1 \leq i \leq n, \mathsf{root} \in \{0,1\}^\lambda$ and a proof path as inputs. The algorithm outputs 1 if path is correctly verified and 0 otherwise. The verification time is logarithmic in the size of the list X.
- $\mathsf{root}' \leftarrow \mathtt{Update}(i, x, X)$ takes an element $x \in \{0,1\}^*, 1 \leq i \leq n$ and X as inputs, and outputs $\mathsf{root}' = \mathtt{Init}(1^\lambda, X')$ where X' is X but $x_i \in X$ is replaced by x.

A Merkle tree should satisfy the properties of *correctness* and *security*. For the formal definitions of these properties, we refer to the cryptography introduction book of Boneh and Shoup [34].

Efficient Replace. The update algorithm described previously needs the entire set X to be able to recalculate the root. Nevertheless, it is feasible to update the root without knowing the entire set. Specifically, we can update the root in $O(\log(|X|))$ operations using only the information about the membership of the node that one wants to replace and the current root. This update will allow an efficient on-chain update of the Merkle tree.

- $\mathsf{root}'/\perp \leftarrow \mathtt{Replace}(i, x, \mathsf{root}, \mathsf{path}_i, x')$: takes as input the index i, the old element x and its membership proof path_i, and the new element, x' that we want to put in the i-th position. The algorithm verifies the membership of both x in the old root using path_i, abort otherwise. Once the verification returns 1, it recomputes the root root' using x' and path_i.

In our protocol, this allows us to maintain the auction protocol's continuity, even if certain bidders decline to disclose their bids. Furthermore, since the depth, d, of the Merkle tree sets the maximum number of bidders that can participate (i.e., 2^d), we can fix this number before the auction begins and treat the cost of replacement as $O(1)$.

B. Programmable Payment Channel (PPC) and State Channel

Programmable Payment Channel (PPC). A PPC [30] is a payment channel between Alice and Bob where either user (e.g., Alice) can authorize a *promise* for a one-way payment to the counterparty (e.g., Bob) conditioned on the logic of a *program* (code). In the case that Alice issues the promise, Bob can redeem such promise either optimistically or pessimistically. In the optimistic path, the contract logic is correctly executed off-chain and in the end, Alice provides Bob with a *receipt* that credits Bob's balance by the promised amount. In the pessimistic path (e.g., Alice is unresponsive), Bob can unilaterally *execute* the promise's program on-chain and claim the promise's balance. Several promises can be created and executed (off-chain) during the lifetime of the PPC for one-way payments from Alice to Bob and vice versa.

2-Party State Channel from PPC. In order to execute an arbitrary two-party smart contract off-chain (i.e., sharing a state channel), PPC must enable both parties to issue two interlocked promises that together encompass the smart contract's logic. The concept of interlocked promises means that the state and logic of Alice's promise can depend on the state and logic of another promise from Bob, and vice versa. Interlocked promises enable any party to claim the payment amount associated to both promises if the other party misbehaves. This mechanism thereby encourages both parties to adhere to the rules and minimizes the potential for disputes. In summary, PPC can be used to fully realize off-chain state channels as shown in [30], allowing any two parties to execute arbitrary smart contracts off-chain. Next we abstract the concept of interlocked promises, called *covenant*, and illustrate it in Fig. 1. We refer to [30] for an explanation of how to compile any two-party covenant contract into two interlocked promises. For completeness, we refer the detailed construction of PPC supporting covenants to the full version of this paper [35].

IV. Building off-chain Auction from PPC: An Overview

A. An Overview of Our Solution: Auction protocol from PPC

System Model. The system consists of an untrusted hub, hub, a set R of *registered* users and a blockchain $\mathcal{L}$ supporting smart contracts. We assume that a PPC exists between hub and each $u \in R$, thereby following the *hub-and-spoke model*. In our system, any registered user can register as a bidder or as a seller in any auction. The hub acts as the auctioneer. We denote the set of *bidders* to be $B = \{\mathsf{bidder}_1, \ldots, \mathsf{bidder}_n\} \subseteq R$, and define the *seller* to be $\mathsf{seller} \in R$.

Threat Model. We separate our assumptions into cryptographic and economic categories: *(i) Cryptographic Assumptions:* We assume that the cryptographic primitives (cf.III) are secure. Adversaries are computationally bounded. *(ii) Economic Assumptions:* Participants are rational actors who seek to maximize their utility and will not perform actions that

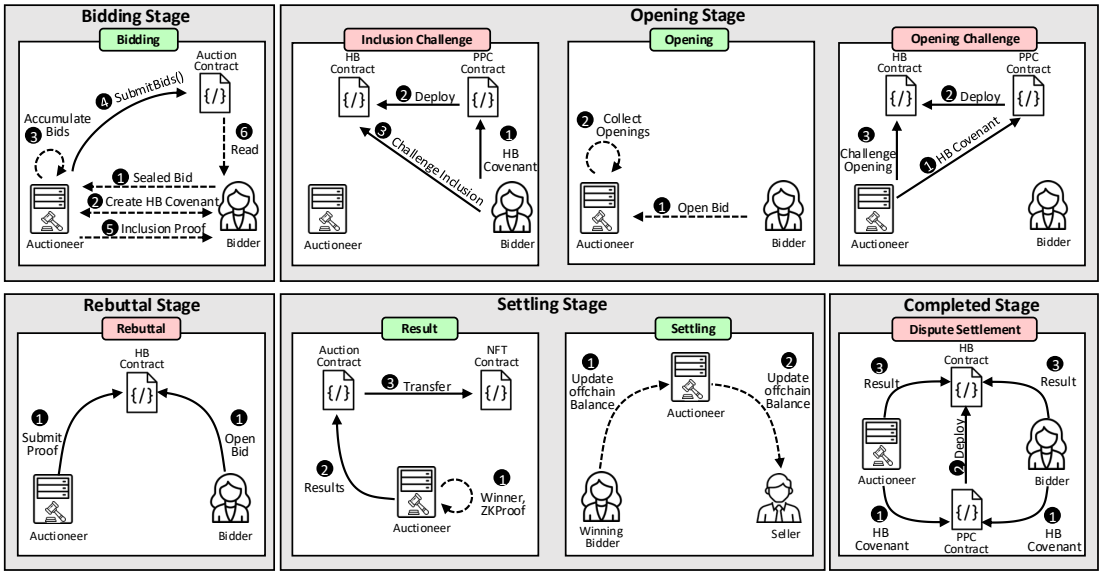


Fig. 2: An overview of the auction protocol. The solid line denotes on-chain actions, and the dashed line denotes off-chain actions. The dispute settlement between Seller and Hub is identical to the dispute settlement between seller and bidder.

result in net financial loss. The collateral amounts (maxBid) are sufficiently large to disincentivize malicious behavior, and gas costs for dispute resolution are non-negligible but much smaller than collateral penalties. We assume the blockchain operates as a secure bulletin board that ensures transaction ordering and finality.

Auction Contract Overview. Our auction contract consists of four primary functions: `Start`, `SubmitSealedBids`, `RevealWinner`, `DeclareFailed`. The auction contract itself advances through various stages: *Bidding*, *Opening*, *Rebuttal*, *Settle*, and *Completed*. The purposes of these functions are relatively self-explanatory. The `Start` function can only be invoked by the hub to initiate the auction. The `SubmitSealedBids` function can only be triggered by the hub to submit a succinct Merkle root value representing all submitted sealed bids. The `RevealWinner` takes as input the winning bid and the zkSnark proof from the hub, proving that the submitted amount is indeed the highest bid. Additionally, the `RevealWinner` function removes all unopened bids before verifying the zkSnark proof. If the auction fails due to the hub's failure to invoke any of the previously described functions, the seller retains the option to call the `DeclareFailed` function to reclaim the auctioned item, such as an NFT.

Seller and Hub: Create the Auction. The seller and hub agree on the parameters (e.g., maximum bid amount and item) of the auction and ensure that the seller will receive a payment or get the NFT back. After reaching an agreement, the hub deploys the auction contract (C_{auction}) containing all agreed-upon parameters. Once the auction contract is deployed, hub and seller have to agree on the details of the covenant contract $C_{\mathsf{HSCovenant}}$. The $C_{\mathsf{HSCovenant}}$ enforces two possible outcomes of the auction for the seller: *(i)* If the auction fails, hub will pay the maximum bid amount to the seller[3]; *(ii)* If there is a winner, hub will send the seller the amount specified in the sealed bid previously submitted by the winning bidder. Once both the seller and hub agree on this contract ($C_{\mathsf{HSCovenant}}$), they will agree on a covenant based on such a contract. Once a valid covenant is obtained, the hub starts the auction via an *on-chain* call that triggers the transfer of the NFT to the auction contract address, effectively placing the NFT in custody for the auction's duration. This action marks the completion of the creation stage.

Bidders and Hub: Bidding, Opening and Rebuttal. The *Bidding* stage is divided into two phases: (1) bidders submit their sealed bids to the hub; and (2) the hub accumulates all the bids and submits them to the auction contract. To do this, hub and bidders have to agree on the covenant contract, $C_{\mathsf{HBCovenant}}$, whose logic enforces the different parties to follow the protocol: *(i)* The hub must include the sealed bid in the accumulated bids and provide an inclusion proof for it; *(ii)* If a bidder does not open their sealed bid, they will be penalized with an amount agreed in the promise; *(iii)* If the bidder is the winner, she will pay the amount committed in the sealed bid.

Once both parties agree on the covenant contract, they obtain the agreed covenant by both signing on the contract detail and its parameters. During the *Opening* stage, either Hub or Bidder can challenge the other party, if they misbehave (i.e., do not provide an inclusion proof or do not open a sealed bid). When challenged, either party can respond to the challenge during the *Rebuttal* stage via on-chain function calls.

Bidders, Seller, Hub: Settling the Auction. At the end of the auction, after the hub announces the winner, the auction winner

[3]In addition, the seller will get back its auctioned item via the auction contract (see Fig. 4 for details).

will pay the hub the amount that she previously committed in the sealed bid during the *Bidding* stage. The rest of bidders get their bids back. Finally, the seller will either receive a payment from the winning bidder via the hub or a penalty from the hub and reclaim the NFT if the auction fails. This can be done either off-chain during the *settle* stage or on-chain during the *completed* stage.

Fig. 2 gives a high-level overview of our protocol.

B. Desired Properties and Threat Model

Desired Properties. The proposed system should provide the following properties:

- **(P1) Auction Correctness**: Our protocol must ensure correctness by unequivocally designating the highest bidder as the winner, guaranteeing a seller payout based on the highest bid amount.
- **(P2) Privacy**: Our protocol should maintain *bid-privacy*, concealing bid amounts of participants until the opening phase. In an optimistic scenario without on-chain challenges, only the winning bid should be disclosed, ensuring *post-auction privacy*.
- **(P3) Efficiency**: For an honest execution of the protocol, the cost should be lower than alternative solutions outlined in Section II. Specifically, in an optimistic case, our auction should demand no interactions among bidders (i.e., *non-interactivity*). Moreover, we require the on-chain cost to be independent of the number of bidders.
- **(P4) Liveness**: Our protocol achieves liveness if it remains operational even when a fraction of bidders abort the process or deviate from it.
- **(P5) Security**: Our auction is secure if it satisfies the following two properties. Firstly, it must be *non-malleable*, preventing a malicious hub from colluding with other bidders to place bids that depend on the bids of honest bidders (e.g., hub cannot generate $\mathsf{cm}' = \mathtt{P}_{\mathtt{com}}(m+1, r)$ from $\mathtt{P}_{\mathtt{com}}(m, r)$). Secondly, the auction should ensure *bid binding*, preventing bidders from altering their bids after submitting their sealed bids.
- **(P6) Financial Fairness**: If any participant deviates, they will be penalized, and honest parties will be refunded.

Threat Model. We assume that the cryptographic primitives (cf. Section III-A) are secure. We consider adversaries to be computationally bounded. Moreover, we assume the correct execution of the smart contract on the blockchain. Users are presumed to have continuous access to read the blockchain state and write to the blockchain. Furthermore, we assume that the adversary can always read all transactions issued to the contract, while the transactions are propagating on the P2P network, and afterward when they are permanently recorded on the blockchain.

V. Off-chain PPC Auction Construction

A. Protocol Setup

Prior to initiating the auction protocol, a *one-time* trusted setup is necessary to securely generate all the public parameters required for the cryptographic building blocks. In Section VII, we discuss how one can mitigate this trusted setup. Specifically, the cryptographic building blocks are as follows.

Cryptographic Parameters. The setup algorithm samples hash functions $h_{2p} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ from collision-resistant hash families. h_{2p} will be used to initialize the Merkle tree used in our auction contract. The hub decides on the commitment scheme $\Gamma = (\mathtt{P}_{\mathtt{com}}, \mathtt{V}_{\mathtt{com}})$ for the auction protocol. In our auction, the sealed bid amount will be computed as $\mathsf{bid} = \mathtt{P}_{\mathtt{com}}(\mathsf{amt}; r)$. In the beginning, the auction contract stores the root of a Merkle tree, $\mathsf{root}_{\mathtt{bids}}$, initialized from a list of empty leaves and the collision-resistant hash function, h_{2p}. When a bidder makes a bid, the corresponding leaf of this Merkle tree will be computed as $\mathsf{leaf} = h_{2p}(\mathsf{bidder}, \mathsf{bid})$. Finally, one needs to define the statement for zkSnark. In our auction protocol, to reveal the winner, the hub needs to prove the following claims (*i*) the revealed winner (winner) participated in the auction during the bidding stage, (*ii*) the amount committed by the winner is the highest amount among *all* bids. Once the setup (cf. Fig. 3) is finished, Hub can start sharing the cryptographic parameters, $\mathsf{params}_{\mathtt{Crypto}}$, with all participants.

B. Auction Protocol

Auction Creation (Hub ↔ Seller). Before the auction begins, the Hub and Seller jointly establish the auction's parameters and cryptographic specifications. Once agreed, the Hub deploys the auction contract, $C_{\mathtt{auction}}$, on-chain incorporating these predefined parameters (cf. Fig. 4). Subsequently, a covenant, as defined in Fig. 5, is established to ensure either a payout to the seller or reimbursement in the event of an auction failure. With the covenant in place, the Seller authorizes the auction contract to facilitate the transfer of the auction item, allowing the Hub to start the auction. This action signals the start of the bidding and associated stages, as visually represented in Fig. 6.

Once the hub starts the auction, bidders can obtain $\mathsf{params}_{\mathtt{crypto}}$ and $\mathsf{params}_{\mathtt{auction}}$ from $C_{\mathtt{auction}}$ and initiate a protocol with the hub to place bids. This protocol progresses

$\mathtt{AuctionSetup}(1^\lambda)$

1 : Sample $h_{2p} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$
2 : Choose $d \in \mathbb{Z}_{>0}$
3 : Let $X = \{x_1, \ldots, x_{2^d}\}$ where $x_i = 0$ for all $x_i \in X$
4 : Let T be a Merkle tree instance parameterized with h_{2p} :
  − Run $\mathsf{root}_{\mathtt{bids}} = T.\mathtt{Init}(1^\lambda, X)$,
5 : Let $\Gamma = (\mathtt{P}_{\mathtt{com}}, \mathtt{V}_{\mathtt{com}})$ be a commitment scheme.
6 : Construct $C_{\mathtt{winner}}$ for the statement described in Eq. (1)
7 : Let Π be a zkSnark instance :
  − Run $(\mathsf{ek}_{\mathtt{winner}}, \mathsf{vk}_{\mathtt{winner}}) \leftarrow \Pi.\mathtt{Setup}(1^\lambda, C_{\mathtt{winner}})$
8 : Return $\mathsf{params}_{\mathtt{Crypto}} = (\mathbb{F}, h_{2p}, \Gamma, T, \Pi,$
  $\mathsf{ek}_{\mathtt{winner}}, \mathsf{vk}_{\mathtt{winner}}, \mathsf{root}_{\mathtt{bids}})$

Fig. 3: Auction's cryptographic parameters setup

Start() /*Hub starts the auction*/

1 : Require msg.sender = hub
2 : Invoke nftAddress.TransferFrom(seller, address(this), tokenID)
3 : Set $\mathsf{T}_{\mathsf{start}} \leftarrow$ block.time

SubmitSealedBids($\overline{\mathsf{numBids}}, \overline{\mathsf{root}_{\mathtt{bids}}}$) /*Bidding Stage*/

1 : Require msg.sender = hub
2 : Require GetStage() = "Bidding" $\wedge$ accSubmitted = "False"
/ if there is 0 bids, require $\overline{\mathsf{root}_{\mathtt{bids}}}$ to be equal default $\mathsf{root}_{\mathtt{bids}}$
3 : **if** $\overline{\mathsf{numBids}} = 0$: Require $\mathsf{root}_{\mathtt{bids}} = \overline{\mathsf{root}_{\mathtt{bids}}}$
4 : Set $(\mathsf{numBids}, \mathsf{root}_{\mathtt{bids}}) \leftarrow (\overline{\mathsf{numBids}}, \overline{\mathsf{root}_{\mathtt{bids}}})$
5 : Set accSubmitted $\leftarrow$ "True"

RevealWinner($\overline{\mathsf{amt}_{\mathtt{winner}}}, \overline{\mathsf{winner}}, \pi, \mathsf{UnopenedBids}$)

1 : Require msg.sender = hub
2 : **if** numBids = 0 : Set resultSubmitted $\leftarrow$ "True"
3 : **else** :
4 : Require GetStage() = "Settle" $\wedge$ resultSubmitted = "False"
5 : Invoke UpdateRoot(UnopenedBids) / Remove unopened bids
6 : Require:
7 : Π.Verify($\mathsf{vk}_{\mathtt{winner}}, [\mathsf{root}_{\mathtt{bids}}, \mathsf{winner}, \mathsf{amt}_{\mathtt{winner}}, \mathsf{numBids}], \pi) = 1$
8 : Set $(\mathsf{amt}_{\mathtt{winner}}, \mathsf{winner}) \leftarrow (\overline{\mathsf{amt}_{\mathtt{winner}}}, \overline{\mathsf{winner}})$
9 : Invoke nftAddress.TransferFrom(address(this), winner, tokenID)
10 : Set resultSubmitted $\leftarrow$ "True"

DeclareFailed() /*Auction Failed*/

1 : Require GetStage() = "Completed"
2 : **if** accSubmitted = "False" $\vee$ resultSubmitted = "False" :
3 : auctionFailed $\leftarrow$ "True"
4 : **if** auctionFailed $\vee$ numBids = 0 :
5 : Invoke nftAddress.TransferFrom(address(this), seller, tokenID)

GetStage()

1 : / Durations = $[\mathsf{T}_{\mathsf{bidding}}, \mathsf{T}_{\mathsf{opening}}, \mathsf{T}_{\mathsf{rebuttal}}, \mathsf{T}_{\mathsf{settle}}]$*/
2 : **if** $\mathsf{T}_{\mathsf{start}} = 0$: **return** "NotStarted"
3 : **if** block.time $< (\mathsf{T}_{\mathsf{start}} + \mathsf{T}_{\mathsf{bidding}})$: **return** "Bidding"
4 : **if** block.time $< (\mathsf{T}_{\mathsf{start}} + \mathsf{T}_{\mathsf{bidding}} + \mathsf{T}_{\mathsf{opening}})$: **return** "Opening"
5 : **if** block.time $< (\mathsf{T}_{\mathsf{start}} + \mathsf{T}_{\mathsf{bidding}} + \mathsf{T}_{\mathsf{opening}} + \mathsf{T}_{\mathsf{rebuttal}})$:
6 : **return** "Rebuttal"
7 : **if** block.time $< \mathsf{T}_{\mathsf{start}} + \sum$Durations $\wedge$ resultSubmitted = "False" :
8 : **return** "Settle"
9 : **return** "Completed"

UpdateRoot(UnopenedBids) /*Remove Unopened Bids*/

1 : Require msg.sender = hub
2 : $[(i_j, \mathsf{bidder}_j, \mathsf{bid}_j, \mathsf{path}_j, C_{\mathsf{HBCovenant}}.\mathsf{addr}, \sigma_j)]_{j \in B'} \leftarrow$ UnopenedBids
3 : For each $j \in [k]$: / Replace unopened bids with a default value.
/ Covenant is authorized
- Require SigVerify($\mathsf{bidder}_j, C_{\mathsf{HBCovenant}}.\mathsf{addr}, \sigma_j) = 1$
/ Hub challenged previously
- Require $C_{\mathsf{HBCovenant}}$.openingChallenged = "True"
/ Bidder did not open
- Require $C_{\mathsf{HBCovenant}}$.openingResolved = "False"
- Compute $\mathsf{leaf}_{i_j} = h_{2p}(\mathsf{bidder}_j, \mathsf{bid}_j)$
/ remove unopened bid
- $\mathsf{root}_{\mathtt{bids}} \leftarrow T$.Replace($i_j, \mathsf{leaf}_j, \mathsf{root}_{\mathtt{bids}}, \mathsf{path}_j, 0$)
- numBids $\leftarrow$ numBids $- 1$

Fig. 4: Auction Contract, $C_{\mathtt{auction}}$ initialized with $\mathsf{params} = (\mathsf{params}_{\mathtt{crypto}}, \mathsf{params}_{\mathtt{auction}})$ where $\mathsf{params}_{\mathtt{crypto}}$ is parameters generated during the setup phase (cf. Section V-A) and $\mathsf{params}_{\mathtt{auction}} = (\mathsf{seller}, \mathsf{hub}, \mathsf{tokenID}, \mathsf{nftAddress}, \mathsf{Durations}, \mathsf{maxBid})$ includes mutually agreed-upon parameters between hub and seller. In the code above, address(this) refers to the address of the auction contract.

Resolve() /*Update seller's balance according to Auction's outcome*/

1 : Require Auction.GetStage() = "Completed"
2 : **if** Auction.auctionFailed = "True" : **return** maxBid
3 : **return** Auction.$\mathsf{amt}_{\mathtt{winner}}$;

Fig. 5: A covenant contract, $C_{\mathsf{HSCovenant}}$, between Hub and Seller, initialized with $\mathsf{params}_{\mathsf{HSCovenant}} = (\mathsf{maxBid}, C_{\mathtt{auction}}.\mathsf{addr}, \mathsf{salt})$.

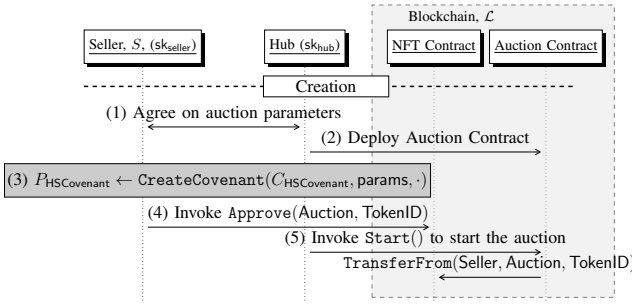


Fig. 6: PPC Auction: Subprotocol between seller and hub.

through different stages. In the following, we describe in detail the interaction between the hub and bidders in each stage.

Bidding (Hub $\leftrightarrow$ Bidders). In the first step, the bidder and the hub define parameters for the $C_{\mathsf{HBCovenant}}$ contract, with the bidder registering their sealed bid within these parameters. Specifically, the bidder selects a bid amount, commits to this amount. This contract ensures both parties follow the auction protocol. Once the hub verifies the logic and parameters and confirms their validity, they proceed. Then, both the hub and the bidder produce a covenant from the $C_{\mathsf{HBCovenant}}$ contract as defined in Fig. 7, ensuring that the hub will include the bidder's bid and provide the inclusion proof, while the bidder

commits to revealing their bid by a specified stage or facing penalties.

/*Bidder challenges inclusion proof and hub responds*/

ChallengeInclusion()

1 : Require:
2 : - inclusionResolved = "True"
3 : - msg.sender = bidder_i
4 : - Auction.GetStage() = "Opening"
5 : Set inclusionChallenged ← "True";

RespondInclusion(i, path_i)

1 : Compute $\mathsf{leaf}_i = h_{2p}(\mathsf{bidder}_i, \mathsf{bid}_i)$
2 : Require:
3 : - msg.sender = hub
4 : - Auction.GetStage() ∈ {"Opening", "Rebuttal"}
5 : - T.Verify($i, \mathsf{leaf}_i, \mathsf{Auction.root}_{\mathtt{bids}}, \mathsf{path}_i$) = 1
6 : - $i \leq$ numBids
7 : Set inclusionResolved ← "True";

/*Hub challenges opening, and bidder responds*/

ChallengeOpening(i, path_i)

1 : Compute $\mathsf{leaf}_i = h_{2p}(\mathsf{bidder}_i, \mathsf{bid}_i)$
2 : Require:
3 : - msg.sender = hub
4 : - Auction.GetStage() = "Opening"
5 : - T.Verify($i, \mathsf{leaf}_i, \mathsf{Auction.root}_{\mathtt{bids}}, \mathsf{path}_i$) = 1
6 : Set openingChallenged ← "True"
7 : Set inclusionResolved ← "True";

RespondOpening(amt_i, r_i)

1 : Require:
2 : - msg.sender = bidder_i
3 : - Auction.GetStage() ∈ {"Opening", "Rebuttal"}
4 : - $\mathtt{V}_{\mathtt{com}}(\mathsf{amt}_i, r_i, \mathsf{bid}_i) = 1$
5 : Set openingResolved ← "True"

/*Update bidder's or hub's balances*/

Resolve()

1 : Require Auction.GetStage() = "Completed"
// punish hub if auction fails
2 : **if** Auction.auctionFailed = "True" : **return** $(2 \cdot \mathsf{maxBid}, 0)$
// punish if the hub misbehaves
3 : **if** inclusionChallenged = "True" ∧ inclusionResolved = "False" :
4 : **return** $(2 \cdot \mathsf{maxBid}, 0)$
// punish if the bidder misbehaves
5 : **if** openingChallenged = "True" ∧ openingResolved = "False" :
6 : **return** $(0, 2 \cdot \mathsf{maxBid})$
// update balance if the bidder is the winner
7 : **if** Auction.winner = bidder :
8 : **return** $(\mathsf{maxBid} - \mathsf{Auction.amt}_{\mathtt{winner}}, \mathsf{maxBid} + \mathsf{Auction.amt}_{\mathtt{winner}})$
// refund both parties if auction fails
9 : **return** (maxBid, maxBid)

Fig. 7: A covenant contract, $C_{\mathsf{HBCovenant}}$, between Hub and Bidder, initialized with $\mathsf{params}_{\mathsf{HBCovenant}} = (\mathsf{maxBid}, \mathsf{bidder}_i, \mathsf{hub}, C_{\mathtt{auction}}.\mathsf{addr}, \mathsf{bid}_i, \mathsf{salt}, \sigma_i)$. Here $\mathsf{Auction} = C_{\mathtt{auction}}.\mathsf{addr}$.

Opening (Hub ↔ Bidders). After the bidding phase, the protocol advances to the opening stage. During this phase, the hub's responsibility is to provide the inclusion proof to the bidder, confirming the presence of the bidder's sealed bid within the Merkle root. If the hub fails to provide this proof, the bidder can initiate an on-chain challenge using the covenant contract (see Fig. 7). Assuming all goes smoothly, the bidder proceeds to reveal their bid to the hub. However, if the bidder fails or refuses to disclose their bid, the hub can also employ the covenant to initiate an on-chain challenge against

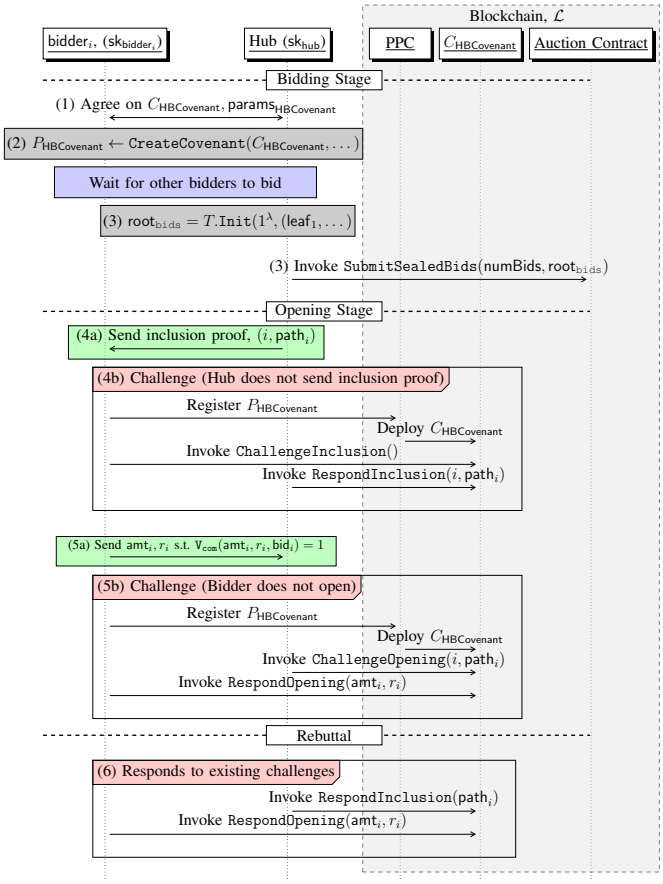


Fig. 8: PPC Auction: Subprotocol between bidder and hub.

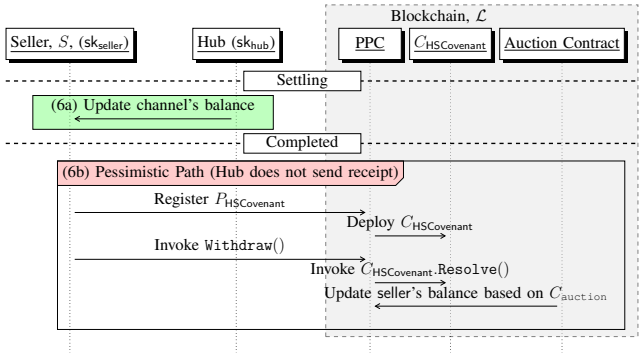


Fig. 9: PPC Auction: Protocols between seller and hub. Settling stage: step (6a) indicates the optimistic path where Hub is honest and sends the receipt and step (6b) is when the hub does not send the receipt

the bidder.

Rebuttal (Hub ↔ Bidders). In the rebuttal stage, both parties are granted additional time to tackle any challenges that may have arisen during the opening phase. This phase essentially functions as a buffer, affording either the hub or the bidder the opportunity to respond adequately to challenges related to inclusion proof and bid openings.

Settling (Hub): Announcing Winner. After the rebuttal stage concludes, the hub must reveal the winner through the `RevealWinner()` call of the auction contract. This step is straightforward if all bidders open their bids. However, if some bidders refuse to open their bids, the hub will be unable to generate a zkSnark proof on the existing $\mathsf{root}_{\mathtt{bids}}$, as the zkSnark defined in Eq. (1) requires all bids to be opened. To ensure the continuity of our protocol, the hub needs to perform the following operations:

(i) Removing Unopened Bids. The hub must reveal the set of unopened bids, UnopenedBids, on-chain to update the Merkle root accordingly. Crucially, to prevent a malicious hub from actively excluding certain bidders, the hub must provide the address of the covenant (i.e., $C_{\mathsf{HBCovenant}}$) deployed on-chain for each unopened bid, along with the bidder's signature to prove prior authorization of the covenant address. Once the auction contract verifies the signature, it checks that the hub had previously challenged the bidder and received no response (i.e., openingChallenged = "`True`" and openingResolved = "`False`"). It then updates the Merkle tree root by replacing the unopened bid with 0.

(ii) Revealing Winner via zkSnark Proof. With all unopened bids removed, the hub can compute the updated root, $\mathsf{root}_{\mathtt{bids}}$, containing only opened bids, and issue a valid zero-knowledge proof, π, for the statement described in Eq. (1). Once the proof is verified by $C_{\mathtt{auction}}$, only the winner and the winning amount will be accepted, and the NFT will be transferred to the winner. In the event of tied highest bids, our circuit can be configured to output either the first or the last highest bidder.

Settling (Seller ↔ Hub, Hub ↔ Bidders): Updating Channel Balances Off-chain. During this stage, each pair (hub and seller, hub and bidder) can honestly update the PPC's balances according to the outcome of the auction (in PPC [30], this update is done through a signed message named a receipt). However, if any party refuses to settle off-chain, the other party can resolve this on-chain in the *completed* stage.

Settling (Seller ↔ Hub, Hub ↔ Bidders): Updating Channel Balances On-chain. Ideally, the protocol should have finished after the *settled* stage, and there should not be any on-chain action after settlement. However, malicious parties may refuse to update the channel balances. Hence, the honest party needs to resolve this during the *completed* stage by registering the previously agreed covenant with the PPC contract. Then, he can settle through the `Resolve()` function call.

Finally, the protocol's high-level flow is described in Fig. 8 and Fig. 9, with detailed description available in Section A.

C. Security Analysis

UC Security Analysis. We analyze our protocol in the UC framework with formal cost-complexity trade-offs. Our formal theorem is stated in terms of a hybrid world involving hybrid ideal functionalities $\mathcal{F}_{\mathsf{ledger}}$ (modeling the ledger) and $\mathcal{F}_{\mathsf{SC}}$ (modeling a 2-party state channels functionality) and we refer the reader to [30] for definitions of these. Recall that [30] shows how the (pairwise) state channel functionality $\mathcal{F}_{\mathsf{SC}}$ can be emulated via PPC channels. All the formalization and proofs can be found in the full version of the paper [35].

Theorem 1. Assume that H is a collision-resistant hash function, Σ is a UC-secure digital signature scheme, com is a UC-secure commitment scheme, and Π is UC-secure zkSnark. Then, our auction protocol UC-realizes $\mathcal{F}_{\mathsf{auction}}$ in the $(\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{ledger}})$-hybrid world with the following explicit cost-adversary trade-offs:

- *Honest Hub, Honest Bidders:* 4 on-chain calls (Auction Deployment, Start, SubmitSealedBids, RevealWinner), with total gas complexity $O(1)$.
- *Honest Hub, k Malicious Bidders:* $4 + O(k)$ on-chain calls (base operations plus k covenant deployments, $\leq k$ challenge resolutions or potentially $\leq k$ bid removals, and k resolve transactions).
- *Malicious Hub:* $\leq 4 + O(k)$ on-chain calls where k represents censored or excluded bidders, each requiring covenant deployment and potential dispute resolution.

The $O(k)$ bound includes all mandatory protocol calls and scales only with misbehaving participants, not total auction size n.

Game Theoretic Analysis. We analyze our protocol using established game-theoretic frameworks, specifically extensive form games with perfect information [36], [37] and the subgame perfect Nash equilibrium (SPNE) solution concept [36]. This approach follows recent blockchain protocol analyses [38], [39]. Our analysis focuses primarily on the Hub-Bidder interactions, as these encompass the core auction mechanics, including bidding, inclusion proofs, bid openings, and associated challenge-response mechanisms, which present the most significant strategic complexity. While the Hub-Seller interaction is crucial for the auction setup and final settlement, its outcomes are directly governed by the pre-established $C_{\mathsf{HSCovenant}}$, offering a comparatively simpler strategic space focused on end states rather than multi-stage auction dynamics between hub and bidder. We apply backward induction to derive the unique SPNE, demonstrating incentive compatibility. Due to space constraints, we defer the formal analysis to the full version of this paper [35].

Theorem 2. Consider the Hub-Bidder auction game derived from the protocol execution flow in Fig. 13, modeled as an extensive form game with perfect information. Let players be expected utility maximizers with risk-neutral preferences. Define the following economic parameters:

- $\mathsf{maxBid} > 0$: collateral locked by each participant
- $\mathsf{amt} \in (0, \mathsf{maxBid}]$: bidder's true bid amount

- $y > 0$: aggregate cost of on-chain dispute resolution (gas fees, opportunity costs)
- Economic rationality condition: $\mathsf{maxBid} > \max(\mathsf{amt}, y)$

Let $\Sigma^* = (\sigma_H^*, \sigma_B^*)$ be the strategy profile where:

- Hub strategy σ_H^*: Submit sealed bids on-chain (`SubmitSealedBids`), provide inclusion proofs, reveal winner with valid zkSNARK (`RevealWinner`)
- Bidder strategy σ_B^*: Submit sealed bid, open bid, accept settlement

Under the economic rationality condition, Σ^* constitutes the unique Subgame Perfect Nash Equilibrium (SPNE) of the auction game. Furthermore, this strategy achieves *(i) Incentive Compatibility*: Deviations result in collateral loss (maxBid) plus dispute costs (y), making honest behavior strictly dominant. *(ii) Efficiency*: The equilibrium path involves zero on-chain dispute costs *(iii) Robustness*: The equilibrium is strict and persists under small perturbations in cost parameters

VI. Evaluation

A. Parameters and PPC Implementation

Choices of cryptographic primitives. We use Groth's zk-Snark [40] due to its efficiency in terms of proofs' size and the verifier's cost. For cryptographic hash functions, we use Poseidon hash function [41] for h_{2p}. Arithmetic circuits using Poseidon hash yield a lower number of constraints and operations when compared to arithmetic circuits relying on other hash functions (i.e. SHA-256, Keccak). Moreover, We use ECDSA for our signature scheme. Finally, the commitment scheme and the Merkle tree can be directly instantiated using Poseidon hash functions.

Software. For the arithmetic circuit construction, we use the `Circom` library [42] to construct the circuit, C_{winner}, described in Equation (1). We use Groth's zkSnark proof system implemented by the `snarkjs` library [43] to perform the trusted setup for obtaining the proving and evaluation keys during the auction setup (cf. Fig. 3) and to generate the prover and verifier programs, as well as to compute the witness. We deploy our auction protocol to a private PoW EVM chain running on Hyperledger Besu v23.1.0. Our auction smart contracts consist of 1156 lines of Solidity code. We also have a Java application and Python client for the off-chain protocol, which have 8461 and 1528 lines of code, respectively.

Hardware. We conducted our experiments on a MacBook Pro equipped with a 2.6GHz 6-Core Intel Core i7 and 16 GB of memory.

Implementation of Programmable Payment Channel (PPC). We have implemented a Programmable Payment Channel (PPC) as outlined in Section IV. This auction process takes place off-chain and is secured by collateral held in PPC established between the hub and participating parties. The initial cost of deploying and establishing a PPC between the hub and each participating party is $3,243,988$ gas. This represents a one-time cost that not only covers multiple auctions (up to the available channel funds) but also facilitates other off-chain applications. In an optimistic scenario where no disputes arise in any auctions or other applications, the closure of the channel will consume $146,908$ gas, spreading the cost across all off-chain transactions.

B. Performance

Off-chain costs: zkSnark setup and proving cost. We focused solely on zk-SNARK proving time, as it remains the auctioneer's most significant overhead. Additional operations, such as Merkle tree generation and bid verification, require approximately 51 ms for $1,024$ bidders. We evaluated our auction protocol at various tree depths. Note that greater tree depths allow for accommodating a larger number of bidders. Table II presents the zkSnark setup cost for the statement in Eq. (1). Note that these costs are off-chain, while the on-chain costs remain constant regardless of the tree depth.

On-chain costs: Hub, sellers, and bidders. Table III shows all on-chain costs for the hub, seller, and bidders in both optimistic and pessimistic cases. In this optimistic case, bidders do not have to issue any on-chain transactions; therefore, we do not include it in the table. For the pessimistic case, we consider the cost of registering a covenant on the PPC contract for participants.

Comparison With Other Auction Protocols. To validate our theoretical claims, we compared our protocol against five alternative approaches. We implemented and evaluated the following protocols: *(1) Our off-chain auction protocol* with zkSNARK-based winner opening. *(2) Off-chain auction without zkSNARKs:* A variant of our construction where the hub reveals the winner without leveraging zero-knowledge proofs. *(3) Naive on-chain auction:* A straightforward implementation where bidders submit and open their bids entirely on-chain. *(4) Rigg-RP [5]:* An on-chain commit-reveal protocol that uses customized NIZKs to prove facts about committed values. *(5) Rigg-TC [5]:* We extrapolated the reported gas costs to estimate total system costs for varying numbers of bidders. *(6) On-chain auction with general-purpose zkSNARKs:* A naive on-chain implementation that uses general-purpose zero-knowledge proofs for proving facts about committed bids.

Figure 10 presents a detailed comparison of gas costs across all these protocols.

Performance with Malicious Bidders. To assess the robustness and cost-efficiency of our protocol under adversarial conditions, we evaluated its performance in the presence of malicious bidders. Figure 11 illustrates the total on-chain gas cost for an auction with 1024 participants as the percentage

TABLE II: Off-chain Costs: zkSnark Setup and Proving. For a tree of a depth d, it can support up to 2^d bidders.

Tree depth	#Constraints (C_{winner})	One-time Setup	Key Sizes	Proving Time
4	$14,444$	$33.213s$	7.9MB	$3.071s$
6	$54,626$	$55.983s$	30.0MB	$6.142s$
8	$213,896$	$268.5s$	118MB	$17.493s$
10	$849,518$	$1001.3s$	468MB	$61.347s$

TABLE III: On-chain gas costs for participants in our protocol.

Operations	On-chain Cost	Invoking Party	Case
Deploying cryptographic libraries	4,507,969	hub	One-time Setup
Deploying $C_{\texttt{auction}}$	1,449,003	hub	Optimistic
`approve` NFT Transfer	49,233	seller	Optimistic
`Start` Auction	92,612	hub	Optimistic
`SubmitSealedBids`	87,138	hub	Optimistic
`RevealWinner`	763,444	hub	Optimistic
Registering $C_{\mathsf{HSCovenant}}$	459,140	hub or seller	Pessimistic
Registering $C_{\mathsf{HBCovenant}}$	3,029,708	hub or bidder	Pessimistic

of malicious bidders increases. In the optimistic case with zero malicious bidders, the gas cost is minimal, as nearly all operations are conducted off-chain. As bidders deviate from the protocol (e.g., by refusing to open their bids), they trigger on-chain dispute resolutions, causing the total gas cost to rise. Majority of the gas usage is due to the registration of $C_{\mathsf{HBCovenant}}$ for each malicious bidder (see Table III). The figure demonstrates that this increase is linear, directly corresponding to the $O(k)$ on-chain complexity, where k is the number of deviators.

Crucially, our protocol demonstrates a significant performance advantage over previous solutions across a wide range of realistic adversarial conditions. When only a small fraction of bidders are malicious ($k \ll n$), our protocol maintains near-optimal $O(1)$ performance, providing orders of magnitude gas savings compared to existing $O(n)$ solutions. As the comparison against the Riggs [5] protocol benchmark shows, our system is more gas-efficient even when a substantial majority of bidders are malicious. The total gas cost does eventually surpass the Riggs benchmark at an extremely high percentage of deviators, as seen in the 80% scenario, as the cumulative cost of resolving each of the k disputes exceeds the cost of a single, larger on-chain process. This demonstrates that for all but the most severe failure cases, our protocol provides superior economic and scalability advantages, establishing it as a highly practical solution for real-world auctions. Furthermore, this cost analysis directly extends to a scenario where the Hub performs a DoS attack on different percentages of bidders, as the primary recovery cost for each affected bidder is the on-chain deployment of the $C_{\mathsf{HBCovenant}}$.

VII. Discussion

Setting a maxBid for Collateral. In sealed-bid auction systems (e.g., Rigg [5]) that do not use a fixed maxBid for collateral, a critical vulnerability emerges: the collateral each bidder locks directly reveals their maximum possible bid.

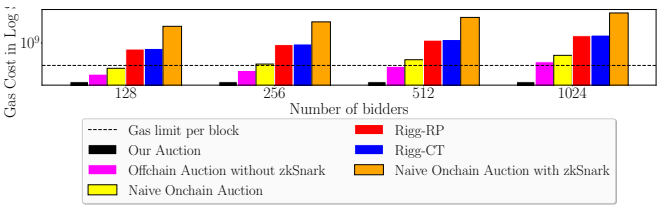


Fig. 10: Total gas costs for different approaches in log scale.

This fundamentally breaks *bid-privacy*, as any observer can determine the exact upper bound of each bidder's potential bid simply by viewing their collateral amount. As shown in Fig. 12a, this approach leaves bidders vulnerable to strategic exploitation. Competitors can confidently outbid them based solely on the collateral they've committed. In our example, B_2 sets its bid one unit higher than B_1 to ensure it outbids it.

To mitigate this issue, we implement a uniform maxBid, as shown in Fig. 12b. Here, all bidders lock the same amount of collateral in their covenants, ensuring no individual bidder leaks information about their maximum bid. This uniformity preserves *bid-privacy* by preventing other participants from inferring any details about competing bids. However, this solution introduces a tradeoff: some bidders may need to lock more collateral than they would prefer (or able to), temporarily restricting their funds' availability for other purposes.

For clarity, throughout this paper we present hub's collateral with each bidder as maxBid. The hub's collateral with bidders can be adjusted to any reasonable value that ensures that the *rational* hub does not deviate from the protocol. However, bidders' collaterals must fully cover their committed bids, and likewise, the hub's collateral with the seller must fully cover the winning amount to ensure *financial fairness*.

Trusted Setup in zkSnark. Our protocol is designed to be compatible with any zkSnark scheme, providing flexibility in addressing the trusted setup requirement. The use of Groth16's zkSnark requires a trusted setup to generate the evaluation and proving keys for the circuit. We chose Groth16 for its superior on-chain verification efficiency, which is critical for minimizing gas costs in our auction protocol. While our construction requires only one setup for all auctions, relying on a trusted third party for this setup is undesirable because if a malicious third party can generate the keys, she can forge arbitrary valid proofs without knowing the witness. There are two possible ways to deal with this limitation. One possible solution is to employ a multi-party computation (MPC) setup, where multiple users contribute shares to the trusted setup. Several works [44]–[46] have proposed protocols for such trusted setups, demonstrating that as long as at least one participant is honest, the zkSnark instance remains secure. MPC ceremonies are a well-established solution deployed in production systems including Zcash [47], Filecoin, and Tornado Cash. Importantly, this is a one-time overhead that

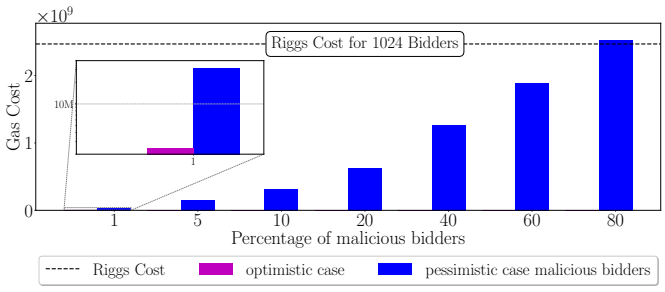
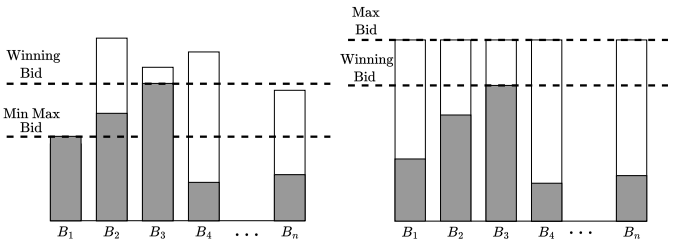


Fig. 11: Total gas costs for different percentages of malicious bidders. Total number of bidders is set to 1024 bidders.



(a) Without maxBid (b) With maxBid

Fig. 12: Comparison of bid-privacy with and without a set maxBid. The white rectangles are the collateral locked by the bidders and the gray rectangles represent the actual bid.

amortizes across all future auctions, making it orthogonal to our core contribution of scalable off-chain auctions. For a detailed discussion of the concrete costs involved in MPC setups, we refer the reader to the recent systematization of knowledge by [48]. A second approach is to use universal setup zkSnark constructions [49]–[52] that can accommodate any circuits within a bounded size from a pre-generated common reference string. These constructions offer the advantage of a universal setup that can be easily integrated into our system in the future. While transparent SNARKs eliminate the trusted setup requirement entirely, they typically incur significantly higher verification costs on-chain. Universal setup schemes like PLONK offer a middle ground with moderate verification overhead compared to Groth16, which may be acceptable for applications prioritizing setup flexibility over gas optimization.

Attack Vectors and Provided Mitigation Strategies. Our sealed-bid auction protocol is designed to resist several potential attack vectors that plague traditional auction systems: *(i) Insider Trading Prevention:* Unlike public auction formats, our protocol ensures bid privacy throughout the bidding phase, preventing all parties (including the hub) from learning bid amounts during bidding. This eliminates front-running and bid manipulation opportunities that exist in transparent auction systems. *(ii) Hub Censorship Resistance:* Malicious hubs attempting to selectively exclude bids are prevented through our covenant mechanism. If the hub censors bids, honest bidders can unilaterally trigger on-chain dispute resolution via their $C_{\mathsf{HBCovenant}}$ contracts, forcing the hub to face financial penalties. *(iii) Collusion Mitigation:* The financial penalties through maxBid collateral requirements and our commit-then-reveal structure make coordinated collusion attacks economically irrational. Each participating bidder must lock significant collateral, making large-scale collusion prohibitively expensive. *(iv) Bid Binding Enforcement:* Once bidders commit to their sealed bids during the bidding phase, they cannot modify or withdraw them, preventing bid manipulation based on partial auction information. *(v) Hub Bid Manipulation Prevention:* The hub cannot selectively modify or exclude bids after observing bid openings because it must commit to the Merkle root of all collected bids during the bidding phase, before any bidder reveals their bid values. This commitment mechanism ensures the hub cannot adaptively manipulate the auction outcome based on revealed bid information.

Practical Significance of Large-Scale Auction Capacity. Current blockchain auction protocols face scaling limitations due to gas costs, with existing solutions reaching practical limits with tens of bidders due to transaction overhead. This technical constraint creates a participation ceiling that may not reflect natural auction demand.

Our protocol's $1,000+$ bidder capacity addresses this limitation by reducing on-chain overhead. For context, blockchain platforms like OpenSea [4] rarely observe auctions exceeding $1,000$ bidders, suggesting our capacity covers the practical range of auction participation. Our protocol also allows scaling beyond $1,000$ bidders through Merkle tree depth adjustment if required.

Duration Lengths. In Section V, we discuss the different stages of the auction protocol: *bidding*, *opening*, *rebuttal*, and *settling*. However, specific timings for these stages were not provided due to their dependence on factors such as the number of bidders, blockchain block creation rate, and the communication delay between the bidders and the hub. Here, we provide insights into these factors.

For timer management, our protocol uses the `GetStage()` function with `block.time` for phase enforcement, which is secure despite minor timestamp manipulation potential due to: (1) consensus constraints preventing arbitrary timestamps, and (2) coarse granularity where stage durations span many blocks, making small timestamp variations negligible. Alternatively, `block.number` could provide even stronger guarantees through strictly monotonic block counting. The *bidding* and *settling* stages involve a single on-chain transaction each, while the *opening* and *rebuttal* stages do not require any on-chain transaction in the optimistic case (i.e., no party deviates from the protocol) and can have up to n (number of bidders) on-chain transactions[4] in the worst-case scenario. Thus, the duration of all stages will rely on the time taken for transaction finality. Additionally, the *bidding* and *opening* stages involve multiple off-chain interactions between bidders and hub. During the bidding stage, there are four message exchanges for each bidder, and in the opening stage, there is one message exchange.[5]. Thus, the timing of these stages will be dependent on the network delay between the hub and bidders for message exchange. Furthermore, the preprocessing time to create transactions for the auction contract needs to be taken into account. Specifically, in the *bidding* stage, the hub accumulates sealed bids and provides inclusion proof for each bid. In our experiments, we were able to process 10,000 bids in less than a second. However, while the off-chain generation of zkSnark proofs in the *settling* stage does not affect the on-chain cost, it heavily depends on the number of bidders impacting the overall timing.

[4]These on-chain transactions for the opening and rebuttal stage can be submitted simultaneously.

[5]Hub can process the messages of different bidders concurrently.

ACKNOWLEDGEMENT

We would like to thank the reviewers for their helpful feedback. This work is part of the grant CEX2024-001471-M/funded by MICIU/AEI/10.13039/501100011033; and of the grant PID2022-142290OB-I00, funded by MCIN/AEI/10.13039/501100011033/ FEDER, UE

REFERENCES

[1] "eBay revenue and usage statistics," https://www.businessofapps.com/data/ebay-statistics/.

[2] J. Chen, G. Scott, and A. Bellucco-Chatham, "Dutch auction: Understanding how it's used in public offerings," https://www.investopedia.com/terms/d/dutchauction.asp.

[3] "ebay," https://www.ebay.com/, 2023.

[4] "Opensea," https://opensea.io/, 2023.

[5] N. Tyagi, A. Arun, C. Freitag, R. Wahby, J. Bonneau, and D. Mazières, "Riggs: Decentralized sealed-bid auctions," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1227–1241. [Online]. Available: https://doi.org/10.1145/3576915.3623182

[6] E. Blass and F. Kerschbaum, "Strain: A secure auction for blockchains," in *European Symposium on Research in Computer Security, ESORICS 2018*, 2018.

[7] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 839–858.

[8] A. Kiayias, H.-S. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 705–734.

[9] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference*, ser. Lecture Notes in Computer Science, vol. 8617. Springer, 2014, pp. 421–439.

[10] J. C. Schlegel and A. Mamageishvili, "On-chain auctions with deposits," *CoRR*, vol. abs/2103.16681, 2021.

[11] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *Financial Cryptography and Data Security*, 2020, pp. 423–443.

[12] B. Chen, X. Li, T. Xiang, and P. Wang, "SBRAC: blockchain-based sealed-bid auction with bidding price privacy and public verifiability," *J. Inf. Secur. Appl.*, vol. 65, p. 103082, 2022.

[13] M. Król, A. Sonnino, A. G. Tasiopoulos, I. Psaras, and E. Rivière, "PASTRAMI: privacy-preserving, auditable, scalable & trustworthy auctions for multiple items," in *Middleware '20: 21st International Middleware Conference*, 2020, pp. 296–310.

[14] T. Constantinides and J. Cartlidge, "Block auction: A general blockchain protocol for privacy-preserving and verifiable periodic double auctions," in *2021 IEEE International Conference on Blockchain*. IEEE, 2021.

[15] H. Desai and M. Kantarcioglu, "SECAUCTEE: securing auction smart contracts using trusted execution environments," in *2021 IEEE International Conference on Blockchain*. IEEE, 2021, pp. 448–455.

[16] H. Desai, M. Kantarcioglu, and L. Kagal, "A hybrid blockchain architecture for privacy-enabled and accountable auctions," in *IEEE International Conference on Blockchain*. IEEE, 2019, pp. 34–43.

[17] H. S. Galal and A. M. Youssef, "Verifiable sealed-bid auction on the ethereum blockchain," in *Financial Cryptography and Data Security - FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, CuraCcao, March 2, 2018, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 10958. Springer, 2018, pp. 265–278.

[18] ——, "Trustee: Full privacy preserving vickrey auction on top of ethereum," in *Financial Cryptography and Data Security - FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 11599. Springer, 2019, pp. 190–207.

[19] J. Xiong and Q. Wang, "Anonymous auction protocol based on time-released encryption atop consortium blockchain," *CoRR*, vol. abs/1903.03285, 2019.

[20] "State channels - ethereum.org," https://ethereum.org/en/developers/docs/scaling/state-channels/.

[21] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Financial Cryptography and Data Security*. Springer International Publishing, 2019, pp. 508–526.

[22] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 949–966.

[23] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 625–656.

[24] J. Coleman, L. Horne, and L. Xuanji, "Counterfactual: Generalized state channels," *Acessed: http://l4.ventures/papers/statechannels.pdf*, vol. 4, p. 2019, 2018.

[25] T. Close, "Nitro protocol," *Cryptology ePrint Archive*, 2019.

[26] P. McCorry, C. Buckland, B. Yee, and D. Song, "Sok: Validating bridges as a scaling solution for blockchains," *Cryptology ePrint Archive*, 2021.

[27] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain scaling using rollups: A comprehensive survey," *IEEE Access*, vol. 10, 2022.

[28] B. Yee, D. Song, P. McCorry, and C. Buckland, "Shades of finality and layer 2 scaling," *arXiv preprint arXiv:2201.07920*, 2022.

[29] P. Xia, H. Wang, Z. Yu, X. Liu, X. Luo, and G. Xu, "Ethereum name service: the good, the bad, and the ugly," *CoRR*, vol. abs/2104.05185, 2021.

[30] Y. Yang, M. Minaei, S. Raghuraman, R. Kumaresan, and M. Zamani, "Off-chain programmability at scale," Cryptology ePrint Archive, Paper 2023/347, 2023.

[31] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *International workshop on fast software encryption*. Springer, 2004, pp. 371–388.

[32] G. D. Crescenzo, J. Katz, R. Ostrovsky, and A. Smith, "Efficient and non-interactive non-malleable commitment," in *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, ser. EUROCRYPT '01. Berlin, Heidelberg: Springer-Verlag, 2001, p. 40–59.

[33] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Symposium on Security and Privacy*. IEEE, 2014, pp. 459–474.

[34] D. Boneh and V. Shoup, "A graduate course in applied cryptography," *Draft 0.5*, 2020.

[35] M. Minaei, R. Kumaresan, A. Beams, P. Moreno-Sanchez, Y. Yang, S. Raghuraman, P. Chatzigiannis, M. Zamani, and D. V. Le, "Scalable off-chain auctions," Cryptology ePrint Archive, Paper 2023/1454, 2023. [Online]. Available: https://eprint.iacr.org/2023/1454

[36] M. J. Osborne and A. Rubinstein, "A course in game theory," mIT press.

[37] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1991.

[38] L. Aumayr, Z. Avarikioti, M. Maffei, and S. Mazumdar, "Securing lightning channels against rational miners," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 393–407. [Online]. Available: https://doi.org/10.1145/3658644.3670373

[39] M. Scaffino, L. Aumayr, Z. Avarikioti, and M. Maffei, "Alba: The dawn of scalable bridges for blockchains," in *Proceedings of the Network and Distributed System Security Symposium*, ser. NDSS '25, 2025.

[40] J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2016, pp. 305–326.

[41] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," Cryptology ePrint Archive, Report 2019/458, 2019.

[42] Iden3, "Circom: Circuit compiler for zksnark," https://github.com/iden3/circom.

[43] ——, "Snarkjs: Javascript and pure web assembly implementation of zksnark schemes," https://github.com/iden3/snarkjs.

[44] S. Bowe, A. Gabizon, and M. D. Green, "A multi-party protocol for constructing the public parameters of the pinocchio zk-snark," in *Financial Cryptography and Data Security*, 2019, pp. 64–77.

[45] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure sampling of public parameters for succinct zero knowledge proofs," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 287–304.

[46] S. Bowe, A. Gabizon, and I. Miers, "Scalable multi-party computation for zk-snark parameters in the random beacon model," Cryptology ePrint Archive, Report 2017/1050, 2017.
[47] A. Miller and S. Bowe, "Zcash MPC Setup," https://www.zfnd.org/blog/powers-of-tau/.
[48] F. Wang, S. Cohney, and J. Bonneau, "SoK: Trusted setups for powers-of-tau strings," Cryptology ePrint Archive, Paper 2025/064, 2025. [Online]. Available: https://eprint.iacr.org/2025/064
[49] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, 2019, p. 2111–2128.
[50] M. Campanelli, D. Fiore, and A. Querol, "Legosnark: Modular design and composition of succinct zero-knowledge proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, 2019, p. 2075–2092.
[51] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Report 2019/953, 2019.
[52] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in *Advances in Cryptology – EUROCRYPT 2020*, A. Canteaut and Y. Ishai, Eds. Cham: Springer International Publishing, 2020, pp. 738–768.

APPENDIX

APPENDIX A
DETAILED PROTOCOL

In this section, we detail all the missing protocols. In particular, the protocol setup, subprotocol between hub, and subprotocol between seller and between hub and bidders.

A. Protocol Setup

Prior to initiating the auction protocol, a one-time trusted setup is to securely generate all public parameters for the cryptographic components used in our protocol. Specifically, the cryptographic components include the following.

Proving Statement. In our auction protocol, to reveal the winner, the hub needs to prove the following claims (*i*) the revealed winner (winner) participated in the auction during the bidding stage, (*ii*) the amount committed by the winner is the highest amount among *all* bids.

$$
\begin{aligned}
&st_{\texttt{Winner}} : \{\mathsf{root}_{\texttt{bids}}, \mathsf{winner}, \mathsf{amt}_{\texttt{winner}}, \mathsf{numBids};\\
&\quad w, \{j, \mathsf{leaf}_j, \mathsf{path}_j, \mathsf{bidder}_j, \mathsf{amt}_j, r_j\}_{j\in[\mathsf{numBids}]}\} :\\
&\quad \text{// leaves are computed correctly}\\
&\quad \mathsf{bid}_j = \mathtt{P}_{\mathtt{com}}(\mathsf{amt}_j, r_j)\\
&\quad \wedge\, \mathsf{leaf}_j = h_{2p}(\mathsf{bidder}_j, \mathsf{bid}_j) \text{ for } j \in [\mathsf{numBids}] \wedge\\
&\quad \text{// bids are included in the computation of } \mathsf{root}_{\texttt{bids}} \text{ previously}\\
&\quad T.\mathtt{Verify}(j, \mathsf{leaf}_j, \mathsf{root}_{\texttt{bids}}, \mathsf{path}_j) = 1 \wedge\\
&\quad \text{// winner has the highest amount}\\
&\quad \mathsf{winner} = \mathsf{bidder}_w\\
&\quad \wedge\, \mathsf{amt}_{\texttt{winner}} = \mathsf{amt}_w = \max(\{\mathsf{amt}_j\}_{j\in[\mathsf{numBids}]})\}
\end{aligned}
\tag{1}
$$

In the case of equal highest bid amounts, we can design the circuit for this statement to output the first or last bidder with the highest bid amount.

B. Subprotocol between Hub and Seller

The protocol between hub and seller work as follows.

(1) Auction parameters agreement. Before starting the auction, Hub and Seller agree on the details of the auction contract, $C_{\texttt{auction}}$ (cf. Fig. 4), the cryptographic parameters ($\mathsf{params}_{\texttt{crypto}}$), the auction parameters, $\mathsf{params}_{\texttt{auction}}$, consisting of the seller's address, seller, the hub's address, hub, the address of the item, tokenID, the address of the NFT contract, nftAddress, the durations of different bidding stages, $\mathsf{Durations} = (\mathsf{T}_{\mathsf{bidding}}, \mathsf{T}_{\mathsf{opening}}, \mathsf{T}_{\mathsf{rebuttal}}, \mathsf{T}_{\mathsf{settle}})$, and the max bid amount, maxBid. At the end of this step, both parties know the contract, $C_{\texttt{auction}}$ and the parameters, $\mathsf{params} = (\mathsf{params}_{\texttt{crypto}}, \mathsf{params}_{\texttt{auction}})$

(2) Hub deploys the auction contract initialized with agreed parameters. Once both parties agree to the parameters, the Hub deploys the auction contract on-chain and initializes it with the agreed-upon parameters (i.e., $\mathsf{params}_{\texttt{crypto}}, \mathsf{params}_{\texttt{auction}}$).

(3) Create a covenant from the contract, $C_{\mathsf{HSCovenant}}$. Once the auction contract is deployed, both hub and seller run `CreateCovenant` protocol on the contract code $C_{\mathsf{HSCovenant}}$ (cf. Fig. 5) and parameters for the contract $\mathsf{params}_{\mathsf{HSCovenant}} = (\mathsf{maxBid}, C_{\texttt{auction}}.\mathsf{addr}, \mathsf{salt})$ to obtain the $P_{\mathsf{HSCovenant}}$ which serves as an authorization from both parties on the code and the parameters for the contract. The covenant $P_{\mathsf{HSCovenant}}$ guarantees either a payout for the seller if there is a winner, or a reimbursement for the seller with the amount maxBid by the hub if the auction fails.

(4) Seller approves NFT transfer. Once the covenant is created between the seller and the hub, the seller can approve the auction contract to transfer the token, tokenID, to itself to start the auction.

(5) Hub starts the auction. Upon receiving approval from the seller, the hub can start the auction anytime by invoking `Start`(). Upon the `Start`() event, nftAddress will transfer the ownership of tokenID to the auction contract, and the auction moves to the subprotocol between the hub and the bidders (i.e., *Bidding*, *Opening*, and *Rebuttal* stages).

Steps (1)-(5) capture the *creation* stage in our protocol. Figure 6 gives a pictorial illustration of how these steps work.

(6) Settlement between Seller and Hub. Once the auction is finished, hub updates the channel balance according to the auction's outcome. As we described in Section III-B and Fig. 1, in PPC, this step can have two possible paths:

- (6a: Optimistic Path) Hub follows the protocol. It can either pay the seller the same amount as the highest bidder's bid if there is one, or pay the amount maxBid if the auction fails. Note that this channel update can be done efficiently by issuing a signature on the channel state (i.e., receipt). Upon receiving this receipt, the seller can always register this receipt with the PPC contract to update the channel state.
- (6b: Pessimistic Path) Hub refuses to follow the protocol by providing wrong receipt or seller ignores the receipt. If the hub refuses to send the receipt, the seller can choose

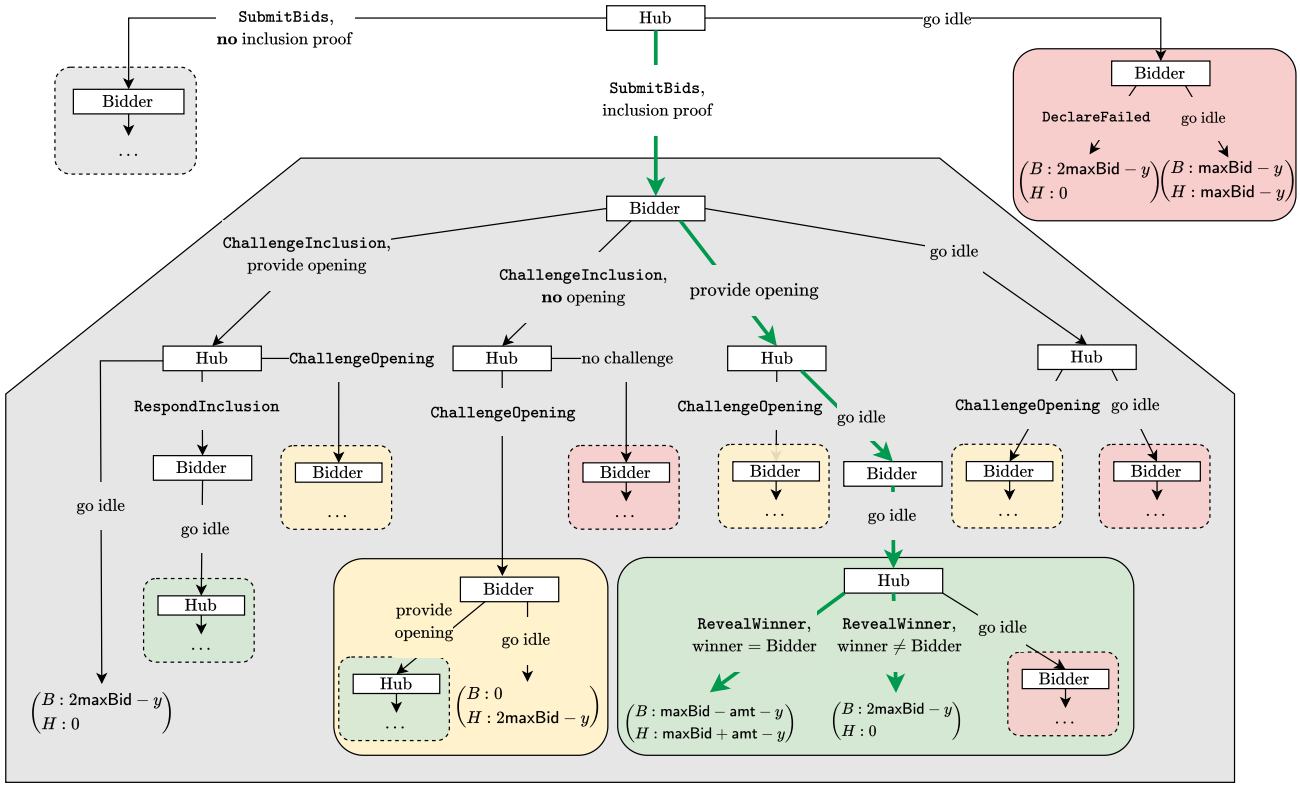


Fig. 13: The game between Hub and Bidder. The green path traces the ideal execution (optimistic path) where no deviations occur. For improved readability, colored dotted boxes represent collapsed summaries of the detailed interactions shown in the solid boxes of the same color. y denotes the aggregate cost associated with initiating or responding to on-chain challenges via the covenant contract (including gas fees and opportunity costs) is non-negative ($y \geq 0$).

to register the covenant $P_{\mathsf{HSCovenant}}$ with the PPC contract and request payment directly on-chain according to the outcome of the auction (i.e., $C_{\texttt{auction}}$). This happens when the auction is in the completed stage.

We provide a detailed protocol on how hub and seller agree on on the creation of the auction in Fig. 14, and Fig. 9 outlines how this settlement step works.

C. Subprotocol between Hub and Bidders

Bidding Stage. This stage is for bidders to place bids with the hub and mutually agree on the parameters for the covenant contract with the hub.

(1) Parameters agreement (e.g., sealed bid) for the bidder-hub contract, $C_{\mathsf{HBCovenant}}$. This step allows each bidder to agree with the hub on the parameters for the $C_{\mathsf{HBCovenant}}$ (cf. Fig. 7), and register the sealed bid as one of these parameters. $C_{\mathsf{HBCovenant}}$ makes sure both parties follow all subsequent steps of the protocol.

The bidder chooses an amount amt_i from $[0, \ldots, \mathsf{maxBid}]$, samples a nonce, r_i, and computes $\mathsf{bid}_i = \mathsf{P}_{\mathsf{com}}(\mathsf{amt}_i, r_i)$. The bidder specifies the logic of $C_{\mathsf{HBCovenant}}$ initialized with the following parameters, $\mathsf{params}_{\mathsf{HBCovenant}} = (\mathsf{maxBid}, \mathsf{bidder}_i, \mathsf{hub}, C_{\texttt{auction}}.\mathsf{addr}, \mathsf{bid}_i, \mathsf{salt}, \sigma_i)$ where maxBidand $C_{\texttt{auction}}$.addr are from $C_{\texttt{auction}}$, σ_i is a valid signature on the address of $C_{\mathsf{HBCovenant}}$. This address can be computed using the CREATE2 opcode. Finally, the bidder sends $(C_{\mathsf{HBCovenant}}, \mathsf{params}_{\mathsf{HBCovenant}})$ to the hub. Once the hub verifies the logic of $C_{\mathsf{HBCovenant}}$ and $\mathsf{params}_{\mathsf{HBCovenant}}$, the hub and bidder move to the next step.

(2) Create a covenant from the contract $C_{\mathsf{HBCovenant}}$. Both hub and bidder will run CreateCovenant() on $C_{\mathsf{HBCovenant}}$ initialized with $\mathsf{params}_{\mathsf{HBCovenant}}$ to obtain $P_{\mathsf{HBCovenant}}$. This covenant ensures two things: *(i)* hub will agree to include the bidder's bid and provide the inclusion proof or get punished otherwise, and *(ii)* the bidder will open the sealed bid eventually before the end of the *rebuttal* stage or get punished otherwise.

(3) Hub accumulates sealed bids and update the auction contract. Before the bidding stage finishes, hub will accumulate all submitted sealed bids into a Merkle tree where the leaf is $\mathsf{leaf}_i = h_{2p}(\mathsf{bidder}_i, \mathsf{bid}_i)$ and submit the $\mathsf{root}_{\mathsf{bids}}$ to the auction contract (cf. function SubmitSealedBids()). This step minimizes the data to be included on-chain.

Remark. It should be noted that our protocol requires that the collateral must be equivalent to the maxBid, an important element of the covenant's parameters. This requirement is to impose a financial penalty on the bidder in case of any deviation from the established protocol.

Opening Stage. Once the *bidding* stage is over, the protocol advances to the *opening* stage. During this stage, the hub has

Create Auction Protocol

Preliminaries. seller wishes to auction its NFT tokenID in contract nftAddress, with the following parameters: maxBidprice for the maximum bid and Durations that identifies the different stages of our protocol.
Let $\mathcal{L}$ be the blockchain that both seller and hub use to post transactions, $C_{\mathsf{HSCovenant}}$ to be the agreed covenant that hub and seller agree on (see Fig. 5), $C_{\texttt{auction}}$ be the publicly available contract Fig. 4, $\mathsf{params}_{\texttt{crypto}}$ be the public cryptographic parameters used the Auction contract shown in Fig. 3, $\texttt{CreateCovenant}()$ be an idealized protocol that allows both hub and seller to agree on the covenant. $\texttt{CreateCovenant}()$ can be instantiated using two interlocked promises in PPC.

Protocol.

1) seller begins by performing the following steps:
   a) $\mathsf{params}_{\texttt{auction}} \leftarrow (\mathsf{hub}, \mathsf{seller}, \mathsf{nftAddress}, \mathsf{tokenID}, \mathsf{maxBid}, \mathsf{Durations})$;
   b) Send [**CreateAuction**, $(\mathsf{params}_{\texttt{auction}}, \mathsf{params}_{\texttt{crypto}})$] to hub.
2) Upon receiving [**CreateAuction**, $(\mathsf{params}_{\texttt{auction}}, \mathsf{params}_{\texttt{crypto}})$] from a seller, hub performs the following steps:
   a) Verify all parameters in $\mathsf{params}_{\texttt{auction}}$ and $\mathsf{params}_{\texttt{crypto}}$;
   b) $\mathsf{params} \leftarrow (\mathsf{params}_{\texttt{auction}}, \mathsf{params}_{\texttt{crypto}})$;
   c) $\mathcal{L}.\texttt{DeployContract}(C_{\texttt{auction}}, \mathsf{params})$;
   d) Upon $C_{\texttt{auction}}$ is deployed at $C_{\texttt{auction}}.\mathsf{addr}$, Hub samples $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{\lambda}$ and sets $\mathsf{params}_{\mathsf{HSCovenant}} = (\mathsf{maxBid}, C_{\texttt{auction}}.\mathsf{addr}, \mathsf{salt})$;
   e) Participate in $\texttt{CreateCovenant}(C_{\mathsf{HSCovenant}}, \mathsf{params}_{\mathsf{HSCovenant}}, \cdot)$ with seller;
   f) Once $\texttt{CreateCovenant}()$ outputs a valid covenant $P_{\mathsf{HSCovenant}} = (C_{\mathsf{HSCovenant}}.\mathsf{code}, \mathsf{params}_{\mathsf{HSCovenant}}, \sigma_{\mathsf{hub}}, \sigma_{\mathsf{seller}})$, Hub waits for the seller to approve the transfer. Abort otherwise.
3) Upon receiving valid $[P_{\mathsf{HSCovenant}}]$ from $\texttt{CreateCovenant}()$ protocol, seller performs the following steps:
   a) Invoke $\mathsf{nftAddress}.\texttt{Approve}(\mathsf{tokenID}, \mathsf{Auction})$;
4) When $\mathsf{nftAddress}.\texttt{getApproved}(\mathsf{tokenID}) = \mathsf{Auction}$, hub invokes $\mathsf{Auction}.\texttt{Start}()$;

Fig. 14: Detailed subprotocol between seller and hub

to provide the inclusion proof to the bidder to prove that her bid has been included in the Merkle root, and once the bidder receives this proof, she has to open her bid to the hub. In particular, this stage works as follows.

(4) Hub sends bidder the inclusion proof. In this step, the hub must prove the bidder that her bid has been included. In this step, there can be two possible scenarios:

- (4a: Optimistic Path) Both Hub and Bidder follow the protocol. Hub sends back the Merkle path, path_i, proving that $\mathsf{leaf}_i = h_{2p}(\mathsf{bidder}_i, \mathsf{bid}_i)$ is included in the computation of $\mathsf{root}_{\texttt{bids}}$. Once the bidder receives the proof and verifies the validity of the proof, she advances to the next step.
- (4b: Pessimistic Path) Either one of them does not follow the protocol. In this case, bidder can register $P_{\mathsf{HBCovenant}}$ with the PPC contract. Upon receiving a valid covenant, the PPC contract will deploy $C_{\mathsf{HBCovenant}}$ on-chain, and at this point, bidder_i can directly challenge (cf. $\texttt{ChallengeInclusion}()$) the hub to provide the inclusion proof on-chain (via $\texttt{RespondInclusion}()$).

(5) Bidder sends hub the opening of the sealed bid. In this step, the bidder has to open her previous sealed bid by sending the hub the opening (i.e., amt, r). There can be two possible scenarios:

- (5a: Optimistic Path) Both parties follow the protocol. In particular, if the bidder sends a valid opening (amt_i, r_i) (i.e. $\mathtt{V}_{\mathtt{com}}(\mathsf{amt}_i, r_i, \mathsf{bid}_i) = 1$), hub advances to the next step.
- (5b: Pessimistic Path) If one of them decides not to follow the protocol. In this case, the hub can register $P_{\mathsf{HBCovenant}}$ with the PPC contract. Upon receiving a valid covenant, the PPC contract will deploy $C_{\mathsf{HBCovenant}}.\mathsf{code}$ on-chain. Once $C_{\mathsf{HBCovenant}}$ is on-chain, the hub can challenge the bidder, bidder_i, on the opening of bid_i (cf. $\texttt{ChallengeOpening}()$), and the bidder will have to respond to the challenge on-chain (via $\texttt{RespondOpening}()$).

Rebuttal Stage. We note that upon receiving any challenge during the *opening* stage, both parties can either respond immediately or respond during the *rebuttal* stage. Hence, the goal of this *rebuttal* stage is to give both parties enough time to reply to any existing on-chain challenges (cf. $\texttt{ChallengeInclusion}$ or $\texttt{ChallengeOpening}$).

(6) Rebuttal to any existing on-chain challenges. During the *rebuttal* stage, either hub or bidder can respond to any existing on-chain challenges (cf. $\texttt{RespondInclusion}()$ and $\texttt{RespondOpening}()$) from the other party.

Step (1)-(6) captures three stages of our protocol, namely, the *bidding, opening* and *rebuttal* stage. Figure 8 illustrates the protocol between the hub and bidders.

Remark. To safeguard against auction failure due to zero participants, one can require that the seller initiates the auction by placing a sealed bid as the first bidder. This approach also allows the seller to set a minimum price for the item up for trade. Without loss of generality, we define $\mathsf{seller} = \mathsf{bidder}_0$.

Finally, in Figure 15, we described in detail the protocol between Bidders and Hub.

Subprotocol between Hub and Bidders

Preliminaries. After creating the Auction, bidders and hub begin the Bidding protocol. Without loss of generality, here we focus on the interactions of a single bidder, bidder_i and hub, hub. Let $\mathcal{L}$ be the chain that both parties seller and hub use to post transactions, $C_{\mathsf{HBCovenant}}$ to be the condition/logic of the covenant that hub and bidder want to agree on, known by both parties (see Fig. 7), $C_{\mathtt{PPC}}$ be the contract of the PPC channel between hub and bidder, $C_{\mathtt{auction}}$ be the publicly available contract Fig. 4, $\mathsf{params} = (\mathsf{params}_{\mathtt{auction}}, \mathsf{params}_{\mathtt{crypto}})$ to be the public parameters known by both hub and bidder after the deployment $C_{\mathtt{auction}}$, maxBid to be the value that both hub and bidder have previously agreed to be paid to bidder in case of an auction failure, $\mathtt{CreateCovenant}()$ be an idealized protocol that allows both hub and bidder to agree on the covenant. $\mathtt{CreateCovenant}()$ can be instantiated using two interlocked promises in PPC.

Protocol.

1) bidder waits till $C_{\mathtt{auction}}.\mathtt{GetStage}() =$ "`Bidding`" and begins by performing the following steps:
   a) Sample $\mathsf{salt}, r_i \leftarrow \{0,1\}^\lambda$
   b) Compute: $C_{\mathsf{HBCovenant}}.\mathsf{addr} = H(\mathsf{0xFF}, C_{\mathtt{PPC}}.\mathsf{addr}, \mathsf{salt}, C_{\mathsf{HBCovenant}}.\mathsf{code})$
   c) Sign: $\sigma_i \leftarrow \mathtt{Sign}(C_{\mathsf{HBCovenant}}.\mathsf{addr}, \mathsf{sk}_{\mathsf{bidder}_i})$
   d) Chose $\mathsf{amt}_i \in [0, \mathsf{maxBid}]$, $\mathsf{bid}_i = \mathtt{P}_{\mathtt{com}}(\mathsf{amt}_i, r_i)$
   e) Set: $\mathsf{params}_{\mathsf{HBCovenant}} = (\mathsf{maxBid}, \mathsf{bidder}_i, \mathsf{hub}, C_{\mathsf{Auction}}.\mathsf{addr}, \mathsf{bid}_i, \mathsf{salt}, \sigma_i)$
   f) Participate in $\mathtt{CreateCovenant}(C_{\mathsf{HBCovenant}}, \mathsf{params}_{\mathsf{HBCovenant}}, \cdot)$ with the hub;
   g) Once $\mathtt{CreateCovenant}()$ outputs a valid covenant $P_{\mathsf{HBCovenant}}$, bidder waits for hub to return the inclusion proof. Abort, otherwise.
2) Upon receiving all $[P_{\mathsf{HBCovenant},i}]$ from the $\mathtt{CreateCovenant}()$ protocol with bidder, bidder_i, hub performs the following steps:
   a) $(\mathsf{maxBid}, \mathsf{bidder}_i, \mathsf{hub}, C_{\mathsf{Auction}}.\mathsf{addr}, \mathsf{bid}_i, \mathsf{salt}, \sigma_i) \leftarrow \mathsf{params}_{\mathsf{HBCovenant}}$;
   b) Compute $\mathsf{leaf}_i = h_{2p}(\mathsf{bidder}_i, \mathsf{bid}_i)$;
   c) Compute $\mathsf{root}_{\mathtt{bids}} = T.\mathtt{Init}(1^\lambda, X = (\mathsf{leaf}_1, \ldots, 0, \ldots))$ where $|X| = \mathsf{numBids}$
   d) Invoke $\mathtt{SubmitSealedBids}(\mathsf{numBids}, \mathsf{root}_{\mathtt{bids}})$;
   e) Compute path_i for bidder_i, send [**InclusionProof**, (i, path_i)] to bidder_i.
3) Upon receiving [**InclusionProof**, (i, path_i)] from a hub, bidder_i performs the following steps:
   a) Retrieve $\mathsf{root}_{\mathtt{bids}}$ from $C_{\mathtt{auction}}$;
   b) Compute $\mathsf{leaf}_i = h_{2p}(\mathsf{bidder}_i, \mathsf{bid}_i)$
   c) If $T.\mathtt{Verify}(i, \mathsf{leaf}_i, \mathsf{root}_{\mathtt{bids}}) =$ "`False`": challenge hub by registering $P_{\mathsf{HBCovenant}}$ with the PPC contract, then invoke $\mathtt{ChallengeInclusion}()$ on $C_{\mathsf{HBCovenant}}$;
   d) Upon receiving a valid opening, send [**Opening**, (amt_i, r_i)] to the hub.
4) Upon receiving [**Opening,** (amt_i, r_i)] from a bidder_i, hub performs the following steps:
   a) If $\mathtt{V}_{\mathtt{com}}(\mathsf{amt}_i, r_i, \mathsf{bid}_i) =$ "`False`": challenge the bidder by registering $P_{\mathsf{HBCovenant}}$ with the PPC contract, then invoke $\mathtt{ChallengeOpening}()$ on $C_{\mathsf{HBCovenant}}$;
   b) Upon receiving a valid opening, hub stores m_i, r_i, and wait for the rebuttal period to be over.

Fig. 15: Bidder and Hub bidding protocol