Scalable Off-Chain Auctions

Mohsen Minaei[1], Duc V. Le[1], Ranjit Kumaresan[1], Andrew Beams[1], Pedro Moreno-Sanchez[1,2], Yibin Yang[3], Srinivasan Raghuraman[1,4], Panagiotis Chatzigiannis[1], and Mahdi Zamani[1]

[1] Visa Research
[2] IMDEA
[3] Georgia Institute of Technology
[4] MIT

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 $O(k \log n)$. 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 acquire any information about the bidders' identities and their respective bids, except for the winner and the winning bid amount. [5]

1 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 [1,14]. 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 between each other (*non-interactivity*) and the overall on-chain cost 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 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 represents 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 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 $O(k \log n)$. This improves over existing solutions that require $O(n)$ on-chain complexity even if a single bidder deviates from the protocol (see Section 2).

We have implemented and deployed our protocol on a private EVM chain using Hyperledger Besu. Our implementation demonstrates that our auction protocol offers significant efficiency improvements compared to existing on-chain solutions, resulting in minimal on-chain interactions and enhanced scalability. Moreover, we have used zkSnark to ensure 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.

2 Related Work

On-chain Auction. Sealed bid auctions can be implemented directly on Layer-1 as in [7,34,33,6,42,11,13,35,18,20,21,26,27,45]. 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

Table 1. 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	$O(1)$	$O(k \log n)$	$O(n)$	Yes

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 [2,38,23,22,17,16] 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.[6] Note 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) [37,43,47]. 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

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

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.

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) [44], 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) [46], 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 $O(k \log n)$ 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.

3 Preliminaries

3.1 Cryptographic Building Blocks

Notation. We denote by 1^λ the security parameter and by $\mathtt{negl}(\lambda)$ 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.

Standard Cryptographic Building Blocks. We consider a family of collision resistant hash function H, an *existential unforgeability* digital signature scheme, $\Sigma = (\texttt{KeyGen}, \texttt{Sign}, \texttt{SigVerify})$ and a *binding, hiding, non-interactive, non-malleable* commitment scheme, $\Gamma = (\texttt{P}_{\texttt{com}}, \texttt{V}_{\texttt{com}})$. We also consider a zero-knowledge Succinct Non-interactive ARgument of Knowledge (zkSnark), $\Pi = (\texttt{Setup}, \texttt{Prove}, \texttt{Verify})$. We consider Merkle tree as an instance for authenticated data structure for set membership. Merkle tree consists of these following algorithms $T = (\texttt{Init}, \texttt{Prove}, \texttt{Verify}, \texttt{Replace})$. We refer to Appendix A for the formal definitions of these cryptographic primitives. To achieve UC security, we will need UC secure versions of the above primitives (e.g., [24,28]).

3.2 Programmable Payment Channel (PPC) and State Channel

Programmable Payment Channel (PPC). A PPC [46] 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 [46], 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 [46] for an explanation of how to compile any two-party covenant contract into two interlocked promises. For completeness, we include in Appendix B the detailed construction of PPC supporting covenants.

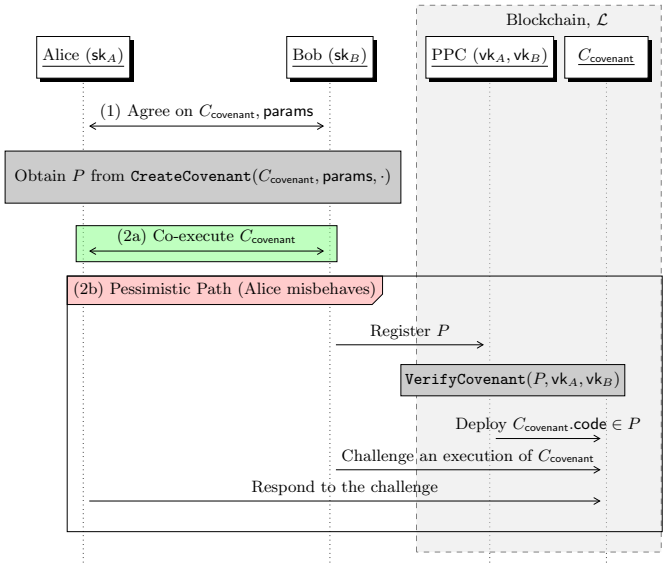

Fig. 1. An overview of creating and executing promises in PPC. Step (2a) indicates the optimistic path, while step (2b) indicates a pessimistic path where an on-chain resolution is needed.

4 Building off-chain Auction from PPC: An Overview

4.1 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 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 $\mathsf{seller} \in R$.

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.

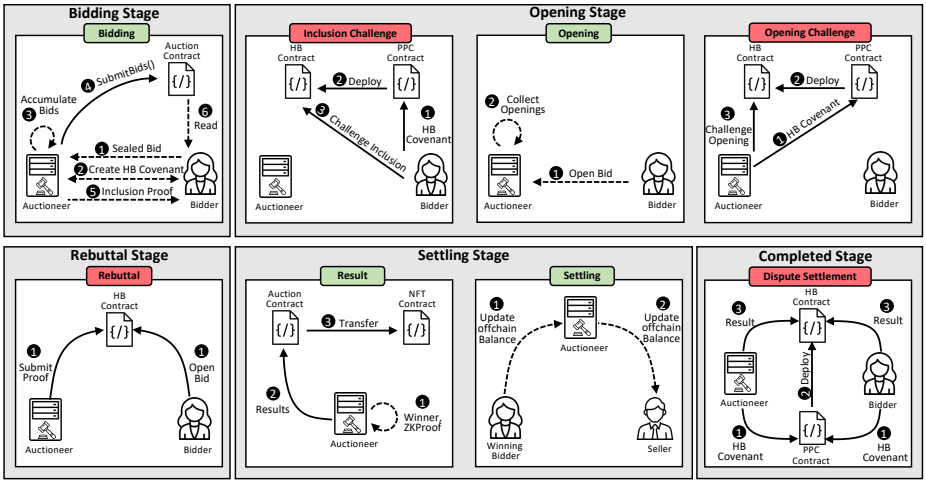

Fig. 2. An overview of the auction protocol. The solid line denotes on-chain actions, and the dashed line denotes off-chain actions. The dispute settlement between Seller and Hub is identical to the dispute settlement between seller and bidder.

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 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[7]; *(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

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

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 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.

4.2 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 2. 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 considered 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 $\mathsf{cm}' = \mathsf{P}_{\mathtt{com}}(m+1, r)$ from $\mathsf{P}_{\mathtt{com}}(m, r)$). 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 financially penalized, and honest parties will be refunded.

Threat Model. We assume that the cryptographic primitives (cf. Section 3.1) are secure. Additionally, 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.

5 Off-chain PPC Auction Construction

5.1 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 Appendix F, 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 $\Gamma = (\mathsf{P}_{\mathtt{com}}, \mathsf{V}_{\mathtt{com}})$ for the auction protocol. In our auction, the sealed bid amount will be computed as $\mathsf{bid} = \mathsf{P}_{\mathtt{com}}(\mathsf{amt}; r)$. 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. 8) is finished, Hub can start sharing the cryptographic parameters, $\mathsf{params}_{\mathtt{Crypto}}$, with all participants.

5.2 Auction Protocol

Auction Creation (Hub $\leftrightarrow$ 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. 3). Subsequently, a covenant, as defined in Fig. 4, 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. 9.

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 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

`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`"
3 : **if** $\overline{\mathsf{numBids}} = 0$: Require $\mathsf{root}_{\mathtt{bids}} = \overline{\mathsf{root}_{\mathtt{bids}}}$ // Default root with 0 bids
4 : Set $(\mathsf{numBids}, \mathsf{root}_{\mathtt{bids}}) \leftarrow (\overline{\mathsf{numBids}}, \overline{\mathsf{root}_{\mathtt{bids}}})$
5 : Set accSubmitted $\leftarrow$ "`True`"

`RevealWinner`($\overline{\mathsf{amt}_{\mathtt{winner}}}, \overline{\mathsf{winner}}, \pi, \mathsf{UnopenedBids}$) /*After Opening Stage*/

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 Π.`Verify`($\mathsf{vk}_{\mathtt{winner}}$, [$\mathsf{root}_{\mathtt{bids}}$, winner, $\mathsf{amt}_{\mathtt{winner}}$, numBids], π) $= 1$
7 : Set $(\mathsf{amt}_{\mathtt{winner}}, \mathsf{winner}) \leftarrow (\overline{\mathsf{amt}_{\mathtt{winner}}}, \overline{\mathsf{winner}})$
8 : Invoke nftAddress.`TransferFrom`(`address`(this), winner, tokenID)
9 : Set resultSubmitted $\leftarrow$ "`True`"

`DeclareFailed()` /*Auction Failed*/

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

`GetStage()` /* Auction Stages; Durations = $[\mathsf{T}_{\mathsf{bidding}}, \mathsf{T}_{\mathsf{bidSubmit}}, \mathsf{T}_{\mathsf{opening}}, \mathsf{T}_{\mathsf{rebuttal}}, \mathsf{T}_{\mathsf{settle}}]$*/

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

`UpdateRoot`(UnopenedBids) /*Remove Unopened Bids*/

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

Fig. 3. 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 5.1) 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; Moreover, if any line of the code fails (e.g., invoking other contracts' functions) the state of the contract is rolled back.

```
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.amt_winner;
```

Fig. 4. A covenant contract, $C_{\mathsf{HSCovenant}}$, between Hub and Seller, initialized with $\mathsf{params}_{\mathsf{HSCovenant}} = (\mathsf{maxBid}, C_{\mathtt{auction}}.\mathsf{addr}, \mathsf{salt})$.

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. 5, 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 face penalties.

Opening (Hub $\leftrightarrow$ 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. 5). 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 the bidder.

Rebuttal (Hub $\leftrightarrow$ 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 Step. 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).

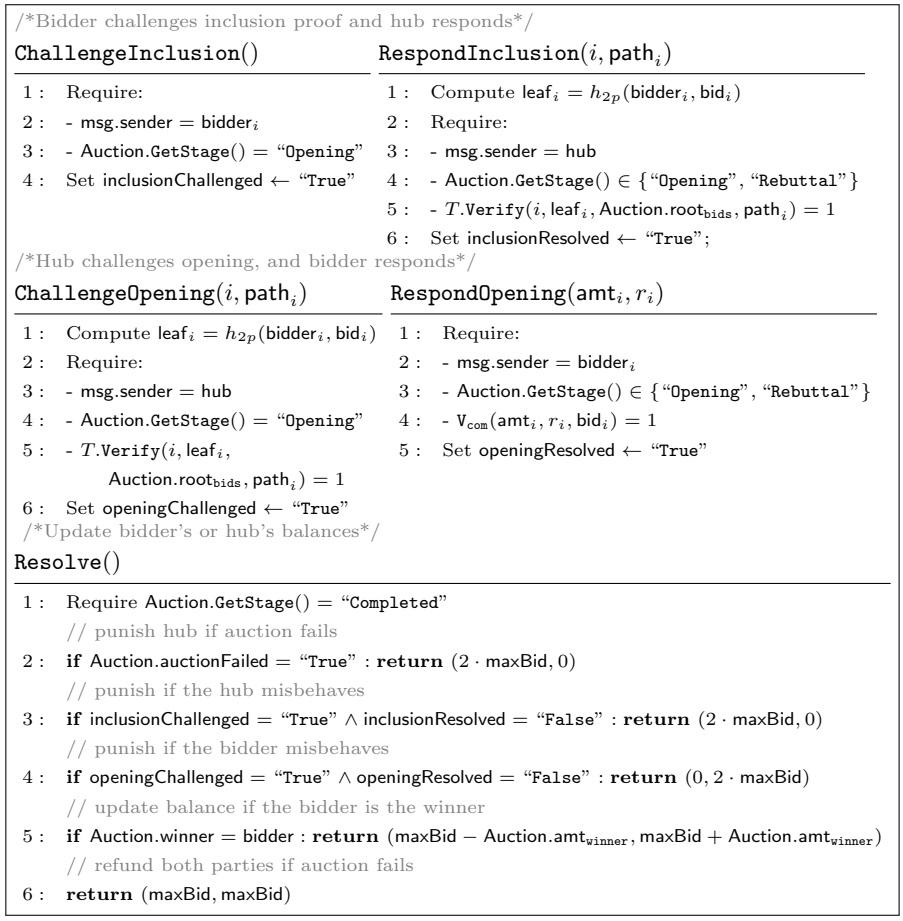

Fig. 5. A covenant contract, $C_{\mathsf{HBCovenant}}$, between Hub and Bidder, initialized with $\mathsf{params}_{\mathsf{HBCovenant}} = (\mathsf{maxBid}, \mathsf{bidder}_i, \mathsf{hub}, C_{\mathtt{auction}}.\mathsf{addr}, \mathsf{bid}_i, \mathsf{salt}, \sigma_i)$. Here $\mathsf{Auction} = C_{\mathtt{auction}}.\mathsf{addr}$.

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): Off-chain Updating Channel Balances. 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 [46], 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 following *completed* stage.

Settling (Seller ↔ Hub, Hub ↔ Bidders): On-chain Updating Channel Balances. 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 detailed protocol description can be found in Appendix C.

Security Analysis. We analyze our protocol in the UC framework. We formally state our theorem as follows. 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 [46] for definitions of these. Recall that [46] 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 Appendix D.

Theorem 1. *Assuming the existence of non-interactive UC secure commitments and non-interactive UC secure zkSnark and collision-resistant hash functions, there exists a protocol that UC-realizes $\mathcal{F}_{\mathsf{auctions}}$ in the $(\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{ledger}})$-hybrid world. Furthermore, when the hub is honest, the maximum number of on-chain calls (made via $\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{ledger}}$) is $O(k)$ with total size $O(k \log n)$ where k denotes the number of adversarial bidders and n denotes the total number of bidders.*

Corollary 1. *Assuming all underlying cryptographic primitives are secure, our auction protocol achieves all properties defined in Section 4.2.*

6 Evaluation

6.1 Parameters and PPC Implementation

Choices of cryptographic primitives. We use Groth's zkSnark [30] due to its efficiency in terms of proofs' size and the verifier's cost. For cryptographic hash functions, we use Poseidon hash function [29] 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, the Poseidon hash function is not only designed specifically for zkSnark applications but also highly efficient for smart contract applications in terms of gas costs. 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 [31] to construct the circuit, C_{winner}, described in Equation (1). We use Groth's zkSnark proof system implemented by the `snarkjs` library [32] to perform the trusted setup for obtaining the proving and evaluation keys during the auction setup (cf. Fig. 8) 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

Table 2. 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 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 4. This auction process takes place off-chain and is secured by collateral held in PPC established between the hub and participating parties. To assess the auction protocol, we have deployed a modified version of PPC detailed in [46]. The pseudocode of the contract used for the payment channel can be found in Fig. 7. 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 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.

6.2 Performance

Off-chain costs: zkSnark setup and proving cost. We evaluated our auction protocol at various tree depths. Note that greater tree depths allow for accommodating a larger number of bidders. Table 3 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 2 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 Designs. We also implemented two other versions of the auction protocol. First, an on-chain auction where bidders have to bid and open their bids on-chain. Second, a naive version of our construction where the hub does not leverage zero-knowledge proof in revealing the winner. Figure 6 gives a comparison of these protocols.

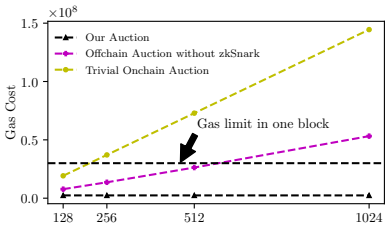

Fig. 6. Total gas costs for different approaches.

Tree depth	#Contraints (C_{winner})	**Setup Time**	**Key Sizes** ($\mathsf{ek}_{\mathsf{winner}}, \mathsf{vk}_{\mathsf{winner}} =$ 4.0KB)	**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 3. Off-chain Costs: zkSnark Setup and Proving. For a tree of a depth d, it can support up to 2^d bidders.

7 Conclusion

Auction protocols serve a crucial function in both real-world and blockchain systems, as they facilitate the discovery of the true value of digital assets and goods while promoting fairness. Despite their significance, there is an absence of efficient auction protocols in blockchain, primarily due to the considerable gas costs and limited scalability associated with on-chain transactions. Furthermore, such auctions lack adequate privacy protections for bidders, as the amounts of their bids and their identities are exposed on the public blockchain.

In this work, we presented a novel multi-party off-chain auction protocol, which is built upon a two-party programmable payment channel. Our implementation demonstrates that our auction protocol is significantly more efficient compared to existing on-chain solutions, resulting in a minimal on-chain interaction and improved scalability. Additionally, in terms of privacy, we employed zkSnark to ensure that the on-chain contract and other participants do not gain any knowledge about the bidders and their bids, except for the identity of the winner and the winning bid amount.

Disclaimer

Case studies, comparisons, statistics, research and recommendations are provided "AS IS" and intended for informational purposes only and should not be relied upon for operational, marketing, legal, technical, tax, financial or other advice. Visa Inc. neither makes any warranty or representation as to the completeness or accuracy of the information within this document, nor assumes any liability or responsibility that may result from reliance on such information. The Information contained herein is not intended as investment or legal advice, and readers are encouraged to seek the advice of a competent professional where such advice is required.

These materials and best practice recommendations are provided for informational purposes only and should not be relied upon for marketing, legal, regulatory or other advice. Recommended marketing materials should be independently evaluated in light of your specific business needs and any applicable laws and regulations. Visa is not responsible for your use of the marketing materials, best practice recommendations, or other information, including errors of any kind, contained in this document.

References

1. eBay revenue and usage statistics. `https://www.businessofapps.com/data/ebay-statistics/`.
2. State channels - ethereum.org. `https://ethereum.org/en/developers/docs/scaling/state-channels/`.
3. ebay. `https://www.ebay.com/`, 2023.
4. Opensea. `https://opensea.io/`, 2023.
5. 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*, pages 287–304, 2015.
6. Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439. Springer, 2014.
7. Erik-Oliver Blass and Florian Kerschbaum. Strain: A secure auction for blockchains. In *European Symposium on Research in Computer Security, ESORICS 2018*, 2018.
8. Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.5*, 2020.
9. Sean Bowe, Ariel Gabizon, and Matthew D. Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. In *Financial Cryptography and Data Security*, pages 64–77, 2019.
10. Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017.
11. Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *Financial Cryptography and Data Security*, pages 423–443, 2020.
12. Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2075–2092, 2019.
13. Biwen Chen, Xue Li, Tao Xiang, and Peng Wang. SBRAC: blockchain-based sealed-bid auction with bidding price privacy and public verifiability. *J. Inf. Secur. Appl.*, 65:103082, 2022.
14. James Chen, Gordon Scott, and Amanda Bellucco-Chatham. Dutch auction: Understanding how it's used in public offerings. `https://www.investopedia.com/terms/d/dutchauction.asp`.
15. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 738–768, Cham, 2020. Springer International Publishing.
16. Tom Close. Nitro protocol. *Cryptology ePrint Archive*, 2019.
17. Jeff Coleman, Liam Horne, and Li Xuanji. Counterfactual: Generalized state channels. *Acessed: http://l4.ventures/papers/statechannels.pdf*, 4:2019, 2018.
18. Theodoros Constantinides and John Cartlidge. Block auction: A general blockchain protocol for privacy-preserving and verifiable periodic double auctions. In *2021 IEEE International Conference on Blockchain*. IEEE, 2021.
19. Giovanni Di Crescenzo, Jonathan Katz, Rafail Ostrovsky, and Adam 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, EUROCRYPT '01, page 40–59, Berlin, Heidelberg, 2001. Springer-Verlag.
20. Harsh Desai and Murat Kantarcioglu. SECAUCTEE: securing auction smart contracts using trusted execution environments. In *2021 IEEE International Conference on Blockchain*, pages 448–455. IEEE, 2021.
21. Harsh Desai, Murat Kantarcioglu, and Lalana Kagal. A hybrid blockchain architecture for privacy-enabled and accountable auctions. In *IEEE International Conference on Blockchain*, pages 34–43. IEEE, 2019.
22. Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-party virtual state channels. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 625–656. Springer, 2019.
23. Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 949–966, 2018.
24. Marc Fischlin, Benoît Libert, and Mark Manulis. Non-interactive and re-usable universally composable string commitments with adaptive security. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 468–485. Springer, 2011. URL: `https://www.iacr.org/archive/asiacrypt2011/70730457/70730457.pdf`, `doi:10.1007/978-3-642-25385-0_25`.
25. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.
26. Hisham S. Galal and Amr 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*, volume 10958 of *Lecture Notes in Computer Science*, pages 265–278. Springer, 2018.
27. Hisham S. Galal and Amr M. Youssef. 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*, volume 11599 of *Lecture Notes in Computer Science*, pages 190–207. Springer, 2019.
28. Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable snarks. In *Advances in Cryptology – EUROCRYPT 2023*, pages 315–346, Cham, 2023. Springer Nature Switzerland.
29. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. Cryptology ePrint Archive, Report 2019/458, 2019.
30. Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.
31. Iden3. Circom: Circuit compiler for zksnark. `https://github.com/iden3/circom`.
32. Iden3. Snarkjs: Javascript and pure web assembly implementation of zksnark schemes. `https://github.com/iden3/snarkjs`.
33. Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multiparty computation using a global transaction ledger. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 705–734. Springer, 2016.

34. Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.
35. Michal Król, Alberto Sonnino, Argyrios G. Tasiopoulos, Ioannis Psaras, and Etienne Rivière. PASTRAMI: privacy-preserving, auditable, scalable & trustworthy auctions for multiple items. In *Middleware '20: 21st International Middleware Conference*, pages 296–310, 2020.
36. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah 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*, CCS '19, page 2111–2128, 2019.
37. Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. Sok: Validating bridges as a scaling solution for blockchains. *Cryptology ePrint Archive*, 2021.
38. Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *Financial Cryptography and Data Security*, pages 508–526. Springer International Publishing, 2019.
39. Andrew Miller and Sean Bowe. Zcash MPC Setup. `https://www.zfnd.org/blog/powers-of-tau/`.
40. Phillip Rogaway and Thomas 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*, pages 371–388. Springer, 2004.
41. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.
42. Jan Christoph Schlegel and Akaki Mamageishvili. On-chain auctions with deposits. *CoRR*, abs/2103.16681, 2021.
43. Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10, 2022.
44. Pengcheng Xia, Haoyu Wang, Zhou Yu, Xinyu Liu, Xiapu Luo, and Guoai Xu. Ethereum name service: the good, the bad, and the ugly. *CoRR*, abs/2104.05185, 2021.
45. Jie Xiong and Qi Wang. Anonymous auction protocol based on time-released encryption atop consortium blockchain. *CoRR*, abs/1903.03285, 2019.
46. Yibin Yang, Mohsen Minaei, Srinivasan Raghuraman, Ranjit Kumaresan, and Mahdi Zamani. Off-chain programmability at scale. Cryptology ePrint Archive, Paper 2023/347, 2023.
47. Bennet Yee, Dawn Song, Patrick McCorry, and Chris Buckland. Shades of finality and layer 2 scaling. *arXiv preprint arXiv:2201.07920*, 2022.

A Cryptographic Building Blocks

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 [40].

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:*

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

- $\mathsf{cm} \leftarrow \mathtt{P}_{\mathtt{com}}(m, r)$ *accepts a message m and a secret randomness r as inputs and returns the commitment string* cm*.*
- $0/1 \leftarrow \mathtt{V}_{\mathtt{com}}(m, r, \mathsf{cm})$ *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*. In particular, *hiding* reveals nothing about the committed data, *binding* implies that no adversary can modify the committed data after it is committed, and *non-malleability* implies that given a commitment, $\mathsf{cm} = \mathtt{P}_{\mathtt{com}}(m, r)$, an adversary cannot output $\mathsf{cm}' = \mathtt{P}_{\mathtt{com}}(m', r')$ on a message m' that is related to m. Finally, *non-interaction* means there is no interaction between participants. For the formal definitions of these properties, we refer readers to [19].

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 [41] 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 [41]). *A zero-knowledge Succinct Non-interactive ARgument of Knowledge for arithmetic (zkSnark) circuit satisfiability is a triple of efficient algorithms* $(\mathtt{Setup}, \mathtt{Prove}, \mathtt{Verify})$*:*

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

Apart from *Correctness, Soundness,* and *Zero-knowledge* properties, a zkSnark requires two additional properties, *Succinctness* and *Simulation Extractability* [30].

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

Definition 4 (Merkle Tree [8]). *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:*

- $\mathsf{root} \leftarrow \mathtt{Init}(1^\lambda, X)$ *takes the security parameter and a list* $X = (x_1, \ldots, x_n)$ *as inputs and outputs a root,* root*.*
- $\mathsf{path}_i \leftarrow \mathtt{Prove}(i, x, X)$ *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 [8].

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. In our protocol, this allows us to maintain the auction protocol's continuity, even if certain bidders decline to disclose their bids.

- $\mathsf{root}'/\perp \leftarrow \mathtt{Replace}(i, x, \mathsf{root}, \mathsf{path}_i, x')$: 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 i 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.

B Programmable Payment Channel Construction (PPC)

For completeness, in this section, we provide the detailed construction of PPC contract as described in [46].

`CREATE2` **Opcode.** PPC relies on a new Ethereum Virtual Machine opcode, `CREATE2`, that was introduced in EIP-1014. The `CREATE2` opcode allows to predict the contract's address before the contract is deployed. It computes the address of the created contract as, $\mathsf{addr} = H(\mathtt{0xFF}, \mathsf{sender}, \mathsf{salt}, C)$ where Sender is the contract's creator, H is Keccak-256, salt is a 256-bit value chosen by the creator, and the output addr is the address of the contract C.

PPC Contract. We present the detailed implementation of programmable payment channels contract in Fig. 7. The programmable payment channel contract is initialized with a channel id *cid*, the parties' public keys vk_A and vk_B, and an expiry time $\mathsf{claimDuration}$ by which the channel settles the amounts deposited. We track the deposit amount and the credit amount (which will be monotonically increasing) for the two parties. In addition, we also track a *receipt* id and an accumulator value `Acc`. We will describe what these are for below, but for now think of receipts as keeping track of received promises that have been resolved off-chain, and the accumulator as keeping track of received promises that have not yet been resolved.

From PPC to Two-party State Channel. Yang *et al.* [46] showed how PPC can realize a state channel by compiling a contract into two interlocked promises (i.e., covenant) that can be used with the PPC contract. This is done by precomputing the on-chain addresses of both promises and initialized each other with the addresses included inside the parameters. In this work, we omit these technical details and refer the reader to [46] for this generic compiler. We call the two interlocked promises a covenant, and it works as follows:

(1) Opening a PPC Both parties can unilaterally open a PPC by deploying an instance of a PPC contract (cf. Fig. 7) on-chain that includes data of both parties (e.g., their addresses $\mathsf{vk}_A, \mathsf{vk}_B$). At any point of time, both parties can top up their balance via an on-chain function call, `Deposit`().

`Deposit`(amt)

1 : Require status = "`Active`" ∧ caller.vk ∈ {`A`.addr, `B`.addr}
2 : **if** caller.vk = `A`.addr : `A`.deposit ← `A`.deposit + amt
3 : **if** caller.vk = `B`.addr : `B`.deposit ← `B`.deposit + amt

`RegisterReceipt`(R)

1 : Require status ∈ {"`Active`", "`Closing`"}
2 : **if** status = "`Active`" : chanExpiry ← now + claimDuration ∧ status ← "`Closing`"
3 : Require caller.vk ∈ {`A`.addr, `B`.addr}
4 : **if** caller.vk = `A`.addr :
    - Require `SigVerify`($R.\sigma$, [cid, R.idx, R.credit, R.`Acc`], `B`.addr)
    - Set `A`.rid ← R.idx, `A`.credit ← R.credit, `A`.`Acc` ← R.`Acc`
5 : **else** :
    - Abort if `SigVerify`($R.\sigma$, [cid, R.idx, R.credit, R.`Acc`], `A`.addr)
    - Set `B`.rid ← R.idx, `B`.credit ← R.credit, `B`.`Acc` ← R.`Acc`

`RegisterPromise`(P)

1 : Require status ∈ {"`Active`", "`Closing`"}
2 : **if** status = "`Active`" : chanExpiry ← now + claimDuration, status ← "`Closing`"
3 : Require caller.vk ∈ {`A`.addr, `B`.addr}
4 : Require [P.addr, P.receiver] ∉ unresolvedPromises
5 : **if** P.sendr = `A`.addr : sendr ← `A`, receiver ← `B`
6 : **else** : sendr ← `B`, receiver ← `A`
7 : Require `SigVerify`($P.\sigma$, [cid, P.rid, P.sendr, P.receiver, P.addr], sendr.addr)
8 : **if** caller.vk = receiver.addr ∧ P.rid < receiver.rid :
      Require: `ACC.VerifyProof`(`Acc`, P.addr, P.`proof`)
9 : Invoke `Deploy`(P.byteCode, P.salt)
10 : Set unresolvedPromises.`push`([P.addr, receiver])

`Close`()

1 : Require caller.vk ∈ {`A`.addr, `B`.addr}
2 : **if** caller.vk = `A`.addr : `A`.closed ← `T`; **else** `B`.closed ← `T`
3 : **if** `A`.closed ∧ `B`.closed, status ← "`Closed`"
4 : **if** status = "`Active`" : chanExpiry ← now + claimDuration, status ← "`Closing`"

`Withdraw`()

1 : Require status ∈ {"`Closing`", "`Closed`"}
2 : **if** status = "`Closing`", Require now > chanExpiry
3 : For each (addr, receiver) ∈ unresolvedPromises :
      receiver.credit ← receiver.credit + addr.`resolve`()
4 : Let `total` = `A`.deposit + `B`.deposit
5 : Invoke `transfer`(`A`.addr, min(`total`, max(0, `A`.deposit + `A`.credit − `B`.credit)),
6 : Invoke `transfer`(`B`.addr, min(`total`, max(0, `B`.deposit + `B`.credit − `A`.credit))

Fig. 7. PPC contract

(2) Creating a Covenant Both parties can engage in an interactive, off-chain protocol to mutually agree on the contract's logic (denoted by C_{covenant}) that they want to execute on their previously opened PPC. In a bit more detail:

- $P \leftarrow \langle \texttt{CreateCovenant}(C_{\mathsf{covenant}}, \mathsf{params}, \mathsf{sk}_A), \texttt{CreateCovenant}(C_{\mathsf{covenant}}, \mathsf{params}, \mathsf{sk}_B)\rangle$: This is a two-party protocol to mutually agree on a contract C_{covenant}. Once both parties agree on parameters (e.g., $\mathsf{params} = (\mathsf{amt}_A, \mathsf{amt}_B, \mathsf{salt})$) to be supplied to the contract C_{covenant}, they both sign the payload consisting of the smart contract code and its initialization parameters. Finally, both parties will receive $P = (C_{\mathsf{covenant}}, \mathsf{params}, \sigma_A, \sigma_B)$ which acts as authorization from both parties.
  Remark 1. We note that this is a two-step protocol, where one party needs to initiate the covenant creation step. It's possible for the other party to decline a response with a signature yet subsequently register the covenant on-chain. Nevertheless, once the covenant is registered on-chain, the initiating party will receive the signature of the non-initiating party.
- $0/1 \leftarrow \texttt{VerifyCovenant}(P, \mathsf{vk}_A, \mathsf{vk}_B)$: This function takes as input the covenant P, and two signature verification keys, vk_A and vk_B, and returns 1 if the covenant P was successfully authorized by both Alice and Bob.

(3) Executing a Covenant Upon agreeing on a covenant P, both parties can begin executing functions specified in the covenant contract, C_{covenant}, off-chain (i.e., the *optimistic path*). During the protocol, if one party fails to respond, the other party can bring the covenant on-chain via the PPC contract. Upon receiving a valid covenant P, the PPC contract deploys the C_{covenant} encapsulated within P to the blockchain. After that, the other party can bring the latest state on-chain and wait for the unresponsive party to respond (i.e., the *pessimistic path*). Any covenant P deployed through PPC is considered *unresolved*.

(4) Closing a PPC At any point in time, either party can request to close the channel. When the channel is in a closing stage, no new covenants are created and parties bring their not yet executed covenants to the PPC contract (see above). After submitting all outstanding covenants (and their on-chain execution), the PPC contract queries all *unresolved* covenants to settle their balances.

C 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.

C.1 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

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

1 : Sample $h_{2p} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$

2 : Choose $d \in \mathbb{Z}_{>0}$, Let $X = \{x_1, \ldots, x_{2^d}\}$ where $x_i = 0$ for all $x_i \in X$

3 : Let T be a Merkle tree instance parameterized with h_{2p} :

– Run $\mathsf{root}_{\mathtt{bids}} = T.\texttt{Init}(1^\lambda, X)$,

4 : Let $\Gamma = (\mathtt{P}_{\mathtt{com}}, \mathtt{V}_{\mathtt{com}})$ be a commitment scheme.

5 : Construct $C_{\mathtt{winner}}$ for the statement described in Eq. (1)

6 : Let Π be a zkSnark instance :

– Run $(\mathsf{ek}_{\mathtt{winner}}, \mathsf{vk}_{\mathtt{winner}}) \leftarrow \Pi.\texttt{Setup}(1^\lambda, C_{\mathtt{winner}})$

7 : 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. 8. Auction's cryptographic parameters setup

in the auction during the bidding stage, (*ii*) the amount committed by the winner is the highest amount among *all* bids.

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

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.

C.2 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, $C_{\mathtt{auction}}$ (cf. Fig. 3), the cryptographic parameters ($\mathsf{params}_{\mathtt{crypto}}$), the auction parameters, $\mathsf{params}_{\mathtt{auction}}$, consisting of the seller's address, seller, the hub's address, hub, the address of the item, $\mathsf{tokenID}$, the address of the NFT contract, $\mathsf{nftAddress}$, 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, $C_{\mathtt{auction}}$ and the parameters, $\mathsf{params} = (\mathsf{params}_{\mathtt{crypto}}, \mathsf{params}_{\mathtt{auction}})$

(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., $\mathsf{params}_{\mathtt{crypto}}, \mathsf{params}_{\mathtt{auction}}$).
(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. 4) and parameters for the contract $\mathsf{params}_{\mathsf{HSCovenant}} = (\mathsf{maxBid}, C_{\mathtt{auction}}.\mathsf{addr}, \mathsf{salt})$ 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, $\mathsf{tokenID}$, 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, $\mathsf{nftAddress}$ will transfer the ownership of $\mathsf{tokenID}$ 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 9 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 3.2 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 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., $C_{\mathtt{auction}}$). 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. 10, and Fig. 11 outlines how this settlement step works.

C.3 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.

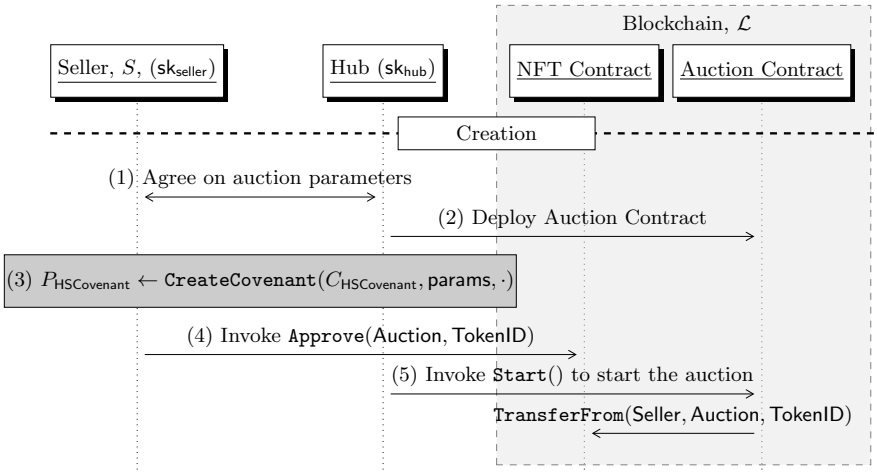

Fig. 9. PPC Auction: Subprotocol between seller and hub.

(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. 5), 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 $\mathsf{bid}_i = \mathtt{P}_{\mathtt{com}}(\mathsf{amt}_i, r_i)$. The bidder specifies the logic of $C_{\mathsf{HBCovenant}}$ initialized with the following parameters, $\mathsf{params}_{\mathsf{HBCovenant}} = (\mathsf{maxBid}, \mathsf{bidder}_i, \mathsf{hub}, C_{\mathtt{auction}}.\mathsf{addr}, \mathsf{bid}_i, \mathsf{salt}, \sigma_i)$ where maxBid and $C_{\mathtt{auction}}.\mathsf{addr}$ are from $C_{\mathtt{auction}}$, 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 $\mathtt{CreateCovenant}()$ 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 $\mathsf{root}_{\mathtt{bids}}$ to the auction contract (cf. function $\mathtt{SubmitSealedBids}()$). This step minimizes the data to be included on-chain.

Remark 2. 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.

Create Auction Protocol

Preliminaries. seller wishes to auction its NFT tokenID in contract nftAddress, with the following parameters: maxBid price 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. 4), $C_{\mathtt{auction}}$ be the publicly available contract Fig. 3, $\mathsf{params}_{\mathtt{crypto}}$ be the public cryptographic parameters used the Auction contract shown in Fig. 8, `CreateCovenant`() be an idealized protocol that allows both hub and seller to agree on the covenant. `CreateCovenant`() can be instantiated using two interlocked promises in PPC.

Protocol.

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

Fig. 10. Detailed subprotocol between seller and hub

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 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.

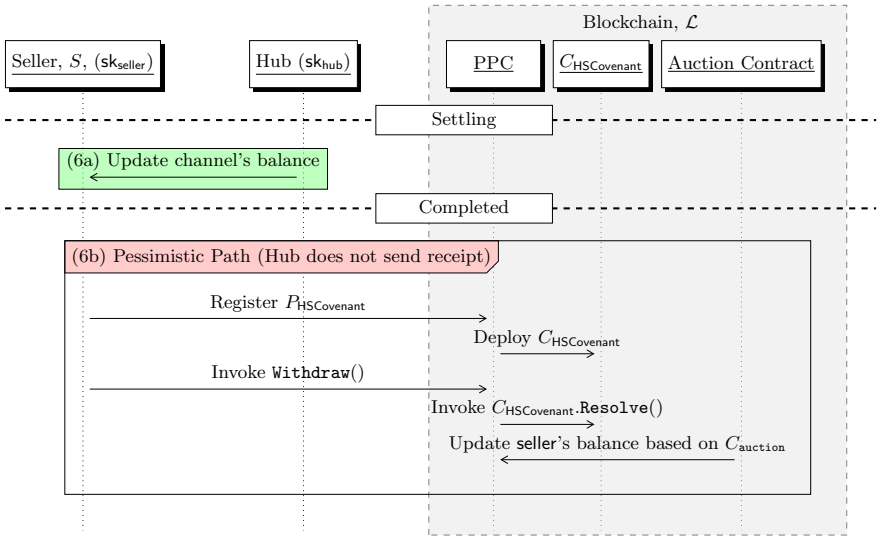

Fig. 11. PPC Auction: Protocols between seller and hub. Settling stage: step (6a) indicates the optimistic path where Hub is honest and sends the receipt and step (6b) is when hub does not send the receipt

(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 $\mathsf{root}_{\mathtt{bids}}$. 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. `ChallengeInclusion`()) the hub to provide the inclusion proof on-chain (via `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. `ChallengeOpening`()), and the bidder will have to respond to the challenge on-chain (via `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. `ChallengeInclusion` or `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. `RespondInclusion`() and `RespondOpening`()) from the other party.

Step (1)-(6) captures three stages of our protocol, namely, the *bidding, opening* and *rebuttal* stage. Figure 13 illustrates the protocol between the hub and bidders.

Remark 3. 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 12, 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. 5), $C_{\mathtt{PPC}}$ be the contract of the PPC channel between hub and bidder, $C_{\mathtt{auction}}$ be the publicly available contract Fig. 3, $\mathsf{params} = (\mathsf{params}_{\mathtt{auction}}, \mathsf{params}_{\mathtt{crypto}})$ 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 $C_{\mathtt{auction}}.\mathtt{GetStage}() =$ "$\mathtt{Bidding}$" 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 $\mathsf{root}_{\mathtt{bids}} = T.\mathtt{Init}(1^\lambda, \{\mathsf{leaf}_1, \ldots, \mathsf{leaf}_{\mathsf{numBids}}, 0, \ldots\})$
   (d) Invoke $\mathtt{SubmitSealedBids}(\mathsf{numBids}, \mathsf{root}_{\mathtt{bids}})$;
   (e) Compute path_i for bidder_i, send $[\mathbf{InclusionProof}, (i, \mathsf{path}_i)]$ to bidder_i.
3. Upon receiving $[\mathbf{InclusionProof}, (i, \mathsf{path}_i)]$ 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 $T.\mathtt{Verify}(i, \mathsf{leaf}_i, \mathsf{root}_{\mathtt{bids}}) =$ "$\mathtt{False}$": 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 $[\mathbf{Opening}, (\mathsf{amt}_i, r_i)]$ to the hub.
4. Upon receiving $[\mathbf{Opening,}\ (\mathsf{amt}_i, r_i)]$ from a bidder_i, hub performs the following steps:
   (a) If $\mathtt{V}_{\mathtt{com}}(\mathsf{amt}_i, r_i, \mathsf{bid}_i) =$ "$\mathtt{False}$": 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. 12. Bidder and Hub bidding protocol

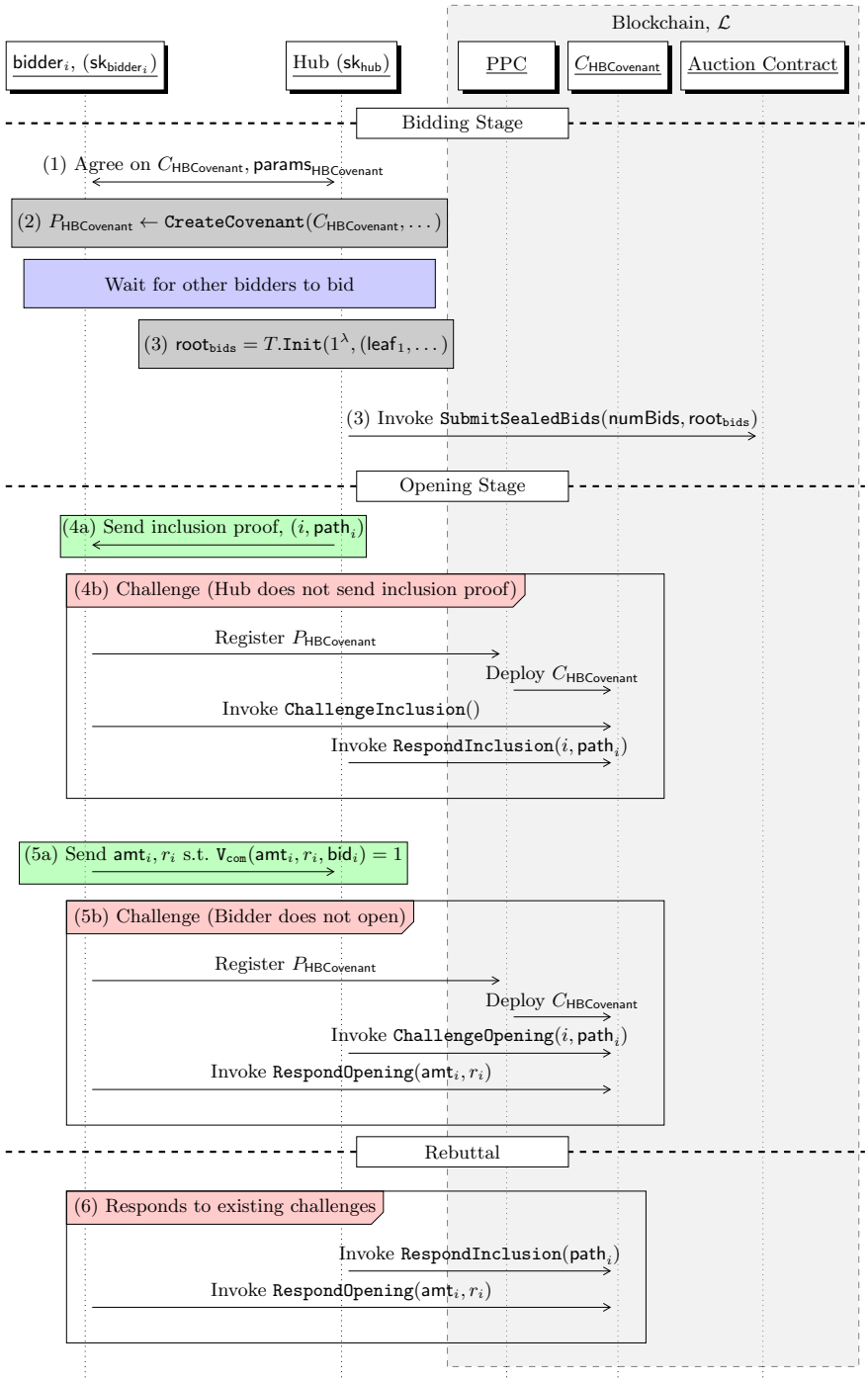

Fig. 13. PPC Auction: Subprotocol between bidder and hub.

D Formalization

The ideal functionality $\mathcal{F}_{\mathsf{auction}}$ is presented in Figures 14 and 15. The ideal functionality somewhat closely follows the structure of the protocol itself, and has similar stages whose durations are specified by $\mathsf{Durations} = [\mathsf{T}_{\mathsf{bidding}}, \mathsf{T}_{\mathsf{opening}}, \mathsf{T}_{\mathsf{rebuttal}}, \mathsf{T}_{\mathsf{settle}}]$. The other parameters in the auction are the usual parameters $\mathsf{hub}, \mathsf{seller}, \mathsf{nft}$ and maxBid. For ease of notation we define the cumulative durations $\mathsf{T}^{\star}_{\mathsf{bidding}}, \mathsf{T}^{\star}_{\mathsf{opening}}, \mathsf{T}^{\star}_{\mathsf{rebuttal}}, \mathsf{T}^{\star}_{\mathsf{settle}}$ as in Figure 14. We assume that the ideal functionality has access to the $\mathsf{balances}$ hashmap through the ideal functionality $\mathcal{F}_{\mathsf{ledger}}$.

Transfer stage. At a high level, the ideal functionality $\mathcal{F}_{\mathsf{auctions}}$ operates as follows. It waits to receive the **transfer** message from the seller specifying the nft that it wishes to auction off. On receiving it, the ideal makes itself the owner of the nft (until the auction concludes) and locks up maxBid from the balance of the seller. (Note that we will be abstracting out the covenants from the ideal world.) This concludes the Transfer stage, and $\mathsf{T}_{\mathsf{start}}$ is initialized to the current time (indicating the start of the auction).

Bidding stage. In the bidding stage, the bidders submit their bids to the ideal functionality. After doing some sanity checks (including checking if the bidder has sufficient balance), the ideal functionality forwards the bidder identity bidder_i (and *not* the bid amount) to the hub. If the hub responds with **ok**, then this would correspond to a successful covenant creation in the real world, and consequently in the ideal, we would lock up maxBid amount from the hub as well as bidder_i. No bid submission is allowed after this stage.

Opening stage. In the opening stage, all the bids (in particular, from those parties that the hub agreed to create a covenant with) are revealed to the hub.

Settling stage. In this stage, we expect the simulator to send a message of the form $(\mathbf{reveal}, S)$ and a message of the form $(\mathbf{settle}, S_1, S_2)$. Briefly speaking, the set S corresponds to the set of bidders among whom the winner is selected. Note that when the hub is honest, we enforce that the set S includes all honest bidders. (When the hub is dishonest, some honest bidders may be excluded.) The set S_1 corresponds to the set of parties for whom the hub compensates with a balance increase of maxBid in the covenant. The set S_2 corresponds to the set of parties who compensate the hub with a balance increase of maxBid in the covenant. Note that the ideal functionality $\mathcal{F}_{\mathsf{auction}}$ ensures that $S_2 \subseteq C$ thereby guaranteeing that the honest parties never compensate the hub. Likewise the ideal enforces that when the hub is corrupt, either honest parties get included in the auction, i.e., the set S or they get compensated, i.e., included in the set S_1. On the other hand, when the hub is honest, it is guaranteed that all honest parties get included in the auction, i.e., in the set S, and also that the hub never compensates any (dishonest) party (i.e., the set $S_1 = \emptyset$). The ideal $\mathcal{F}_{\mathsf{auction}}$ then computes the winner of the auction, and sets $\mathsf{nft.owner}$ to the winner.

Resolve stage. In this stage, each of the participants are allowed to send a **resolve** message to the ideal. This corresponds to the parties resolving the

covenants in the real world. First, when the seller submits a resolve message, then if the winner is properly defined, the balances are adjusted to reflect the money transfer from the hub to the seller corresponding to the winning bid. When the bidder or the hub submits a resolve message, then the balances of the corresponding bidder and the hub are adjusted depending on the auction winner and the sets S_1, S_2, as shown in Figure 15. At a high level, if the auction winner is not decided (i.e., the auction failed), then the bidder that sent a resolve message will get an additional maxBid (remember we locked up maxBid from the bidder's balance in the bidding stage) as compensation. Next, if the bidder happens to be the winner then the original deposit of maxBid minus the winning bid is returned to the bidder. Likewise for the hub, the original deposit plus the winning bid is returned. Finally, if the bidder is not the winner, then the hub and the bidder are returned their original deposit of maxBid. This concludes the description of the ideal functionality in Figures 14 and 15.

We now state our formal theorem. Our formal theorem is stated in terms of a hybrid world involving hybrid ideal functionalities $\mathcal{F}_{\mathsf{ledger}}$ and $\mathcal{F}_{\mathsf{SC}}$ and we refer the reader to [46] for definitions of these. Recall that [46] shows how the (pairwise) state channel functionality $\mathcal{F}_{\mathsf{SC}}$ can be emulated via PPC channels.

$\mathcal{F}_{\mathsf{auction}}$

Preliminaries. The parties involved are the bidders bidder_i, $i \in [n]$, the hub hub, and the seller seller. The NFT being auctioned is nft. We assume $\mathcal{F}_{\mathsf{auction}}$ has access to nft through the ideal functionality $\mathcal{F}_{\mathsf{ledger}}$. The maximum bid price set by the seller is maxBid. The durations of the various stages of the auction are defined by $\mathsf{Durations} = [\mathsf{T}_{\mathsf{bidding}}, \mathsf{T}_{\mathsf{opening}}, \mathsf{T}_{\mathsf{rebuttal}}, \mathsf{T}_{\mathsf{settle}}]$. Let t denote the current time and the starting time $\mathsf{T}_{\mathsf{start}} = \infty$. The auction parameters $\mathsf{params}_{\mathtt{auction}}$ are $(\mathsf{hub}, \mathsf{seller}, \mathsf{nft}, \mathsf{maxBid}, \mathsf{Durations})$. We define cumulative times (after $\mathsf{T}_{\mathsf{start}}$ has been reinitialized from ∞ in **Transfer** below) as $\mathsf{Durations}^\star = [\mathsf{T}^\star_{\mathsf{bidding}}, \mathsf{T}^\star_{\mathsf{opening}}, \mathsf{T}^\star_{\mathsf{rebuttal}}, \mathsf{T}^\star_{\mathsf{settle}}]$, where

- $\mathsf{T}^\star_{\mathsf{bidding}} = \mathsf{T}_{\mathsf{start}} + \mathsf{T}_{\mathsf{bidding}}$,
- $\mathsf{T}^\star_{\mathsf{opening}} = \mathsf{T}^\star_{\mathsf{bidding}} + \mathsf{T}_{\mathsf{opening}}$,
- $\mathsf{T}^\star_{\mathsf{rebuttal}} = \mathsf{T}^\star_{\mathsf{opening}} + \mathsf{T}_{\mathsf{rebuttal}}$,
- $\mathsf{T}^\star_{\mathsf{settle}} = \mathsf{T}^\star_{\mathsf{rebuttal}} + \mathsf{T}_{\mathsf{settle}}$

The bids submitted by the parties are maintained by the ideal functionality in the hashmap bids. (For ease of presentation, we assume $\mathcal{F}_{\mathsf{auction}}$ has access to the hashmap $\mathsf{balances}$ through the ideal functionality $\mathcal{F}_{\mathsf{SC}}$.) Without loss of generality, we define $\mathsf{bidder}_0 = \mathsf{seller}$.

Transfer. When the seller seller first submits a message of the form $(\mathbf{transfer}, \mathsf{nft})$:

- If $\mathsf{nft.owner} \neq \mathsf{seller}$, return $\perp$.
- Send the message $(\mathbf{transfer}, \mathsf{seller})$ to the hub hub.
  - If hub responds with $(\mathbf{ok}, \mathsf{seller})$, then set
    - $\mathsf{nft.owner} = \mathcal{F}_{\mathsf{auction}}$,
    - $\mathsf{balances}[\mathsf{hub}] = \mathsf{balances}[\mathsf{hub}] - \mathsf{maxBid}$,
    - $\mathsf{bids}[0] = 0$,
    - $\mathsf{T}_{\mathsf{start}} = t$,

    and return **success**.
  - Else, return $\perp$.

Bid. When bidder bidder_i first submits a message of the form $(\mathbf{bid}, \mathsf{amt}_i)$:

- If $t \notin [\mathsf{T}_{\mathsf{start}}, \mathsf{T}^\star_{\mathsf{bidding}}]$, return $\perp$.
- If $\mathsf{balances}[\mathsf{bidder}_i] < \mathsf{maxBid}$, return $\perp$.
- If $\mathsf{amt}_i > \mathsf{maxBid}$, return $\perp$.
- Send the message $(\mathbf{bid}, \mathsf{bidder}_i)$ to the hub hub.
  - If hub responds with $(\mathbf{ok}, \mathsf{bidder}_i)$, then set
    - $\mathsf{bids}[i] = \mathsf{amt}_i$,
    - $\mathsf{balances}[\mathsf{bidder}_i] = \mathsf{balances}[\mathsf{bidder}_i] - \mathsf{maxBid}$,
    - $\mathsf{balances}[\mathsf{hub}] = \mathsf{balances}[\mathsf{hub}] - \mathsf{maxBid}$,

    and return **success**.
  - Else, return $\perp$.

Fig. 14. Ideal auction functionality $\mathcal{F}_{\mathsf{auction}}$ (Part 1 of 2)

$\mathcal{F}_{\mathsf{auction}}$ (contd.)

Open. When bidder bidder_i first submits the message **open**:

- If $t \notin [\mathsf{T}^{\star}_{\mathsf{bidding}}, \mathsf{T}^{\star}_{\mathsf{opening}}]$, return $\bot$.
- If $\mathsf{bids}[i] = \bot$, return $\bot$.
- Send the message $(\mathbf{open}, \mathsf{bidder}_i, \mathsf{bids}[i])$ to the hub hub, and return **success**.

Settle. When the simulator first submits the message $(\mathbf{reveal}, S)$, and additionally submits the message $(\mathbf{settle}, S_1, S_2)$:

- If $t \notin [\mathsf{T}^{\star}_{\mathsf{rebuttal}}, \mathsf{T}^{\star}_{\mathsf{settle}}]$, return $\bot$.
- If the hub is honest and ($H \not\subseteq S$ or $S_1 \neq \emptyset$), where H is the set of honest parties, return $\bot$.
- If the hub is corrupt and $H \not\subseteq S \cup S_1$, where H is the set of honest parties, return $\bot$.
- If $S_2 \not\subseteq C$, where C is the set of corrupt parties, return $\bot$.
- If $0 \notin S$, return $\bot$.
- Set $\mathsf{winner} = \arg\max_{j \in S} \mathsf{bids}[j]$.
- Set $\mathsf{nft.owner} = \mathsf{bidder}_{\mathsf{winner}}$.
- Send the message $(\mathbf{winner}, \mathsf{winner})$ to the simulator and all parties, and return **success**.

Resolve. When the seller seller first submits the message **resolve**, or the hub hub submits the message $(\mathbf{resolve}, \mathsf{seller})$:

- If $t \leq \mathsf{T}^{\star}_{\mathsf{settle}}$, return $\bot$.
- If $\mathsf{winner} = \bot$, set $\mathsf{balances}[\mathsf{seller}] = \mathsf{balances}[\mathsf{seller}] + \mathsf{maxBid}$.
- If $\mathsf{winner} \neq \bot$, set
  - $\mathsf{balances}[\mathsf{hub}] = \mathsf{balances}[\mathsf{hub}] + (\mathsf{maxBid} - \mathsf{bids}[\mathsf{winner}])$,
  - $\mathsf{balances}[\mathsf{seller}] = \mathsf{balances}[\mathsf{seller}] + \mathsf{bids}[\mathsf{winner}]$
- Return **success**.

When the bidder bidder_i first submits the message **resolve**, or the hub hub submits the message $(\mathbf{resolve}, \mathsf{bidder}_i)$:

- If $t \leq \mathsf{T}^{\star}_{\mathsf{settle}}$, return $\bot$.
- If $\mathsf{winner} = \bot$, set $\mathsf{balances}[\mathsf{bidder}_i] = \mathsf{balances}[\mathsf{bidder}_i] + 2 \cdot \mathsf{maxBid}$ and return **success**.
- If $i \in S_1$, set $\mathsf{balances}[\mathsf{bidder}_i] = \mathsf{balances}[\mathsf{bidder}_i] + 2 \cdot \mathsf{maxBid}$ and return **success**.
- If $i \in S_2$, set $\mathsf{balances}[\mathsf{hub}] = \mathsf{balances}[\mathsf{hub}] + 2 \cdot \mathsf{maxBid}$ and return **success**.
- If $\mathsf{winner} = \mathsf{bidder}_i$, set
  - $\mathsf{balances}[\mathsf{hub}] = \mathsf{balances}[\mathsf{hub}] + (\mathsf{maxBid} + \mathsf{bids}[\mathsf{winner}])$,
  - $\mathsf{balances}[\mathsf{bidder}_i] = \mathsf{balances}[\mathsf{bidder}_i] + (\mathsf{maxBid} - \mathsf{bids}[\mathsf{winner}])$
- If $\mathsf{winner} \neq \mathsf{bidder}_i$, set
  - $\mathsf{balances}[\mathsf{hub}] = \mathsf{balances}[\mathsf{hub}] + \mathsf{maxBid}$,
  - $\mathsf{balances}[\mathsf{bidder}_i] = \mathsf{balances}[\mathsf{bidder}_i] + \mathsf{maxBid}$
- Return **success**.

Fig. 15. Ideal auction functionality $\mathcal{F}_{\mathsf{auction}}$ (Part 2 of 2)

Theorem 1. *Assuming the existence of non-interactive UC secure commitments and non-interactive UC secure zkSnark and collision-resistant hash functions, there exists a protocol that UC-realizes* $\mathcal{F}_{\mathsf{auctions}}$ *in the* $(\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{ledger}})$*-hybrid world. Furthermore, when the hub is honest, the maximum number of on-chain calls (made via* $\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{ledger}}$*) is* $O(k)$ *with total size* $O(k \log n)$ *where* k *denotes the number of adversarial bidders and* n *denotes the total number of bidders.*

Proof sketch. We now sketch the proof of the theorem. We start by sketching the description of the simulator with some brief inline arguments to prove indistinguishability of the simulation from the real execution.

Case 1: Hub is Corrupt. We first consider the case when the hub is corrupt.

In the real protocol, the seller and hub agree on the auction parameters. Then the hub sends a message to $\mathcal{F}_{\mathsf{ledger}}$ to deploy the auction contract. The simulator acting as $\mathcal{F}_{\mathsf{ledger}}$ accepts the auction contract, and checks if the parameters are as agreed by the seller. Then the seller begins by notifying the hub of its intent to create a covenant via $\mathcal{F}_{\mathsf{SC}}$ (note that the covenant hardcodes the auction contract address). The simulator acting as $\mathcal{F}_{\mathsf{SC}}$ waits for such a message, and upon receiving it forwards it to the hub. If the hub also responds to creating the covenant, then the simulator responds with $(\mathbf{ok}, \mathsf{seller})$ message to $\mathcal{F}_{\mathsf{auction}}$.

In a similar way to the above, we handle covenant creation with the bidders. In more detail, if the simulator receives the message $(\mathbf{bid}, \mathsf{bidder}_i)$, then the simulator acting as $\mathcal{F}_{\mathsf{SC}}$ sends a message to the hub notifying that the covenant creation process has been initiated by the bidder bidder_i. If the hub responds with **ok** for the covenant creation, then the simulator respoinds with $(\mathbf{ok}, \mathsf{bidder}_i)$ to $\mathcal{F}_{\mathsf{auction}}$. Note that the covenant has the sealed bid (i.e., commitment to the bid) encoded in it. For this purpose, the simulator will use a commitment to the zero string. (This step will be indistinguishable from the protocol due to the hiding property of the commitment.) For covenants submitted to $\mathcal{F}_{\mathsf{SC}}$ by malicious bidders, the simulator uses the extractor of the noninteractive UC-secure commitment scheme in order to find the bids of the malicious bidders, and then submits the message $(\mathbf{bid}, \mathsf{amt}_i)$ to $\mathcal{F}_{\mathsf{auction}}$ in the **Open** stage.

Following this, the hub submits the accumulator to $\mathcal{F}_{\mathsf{ledger}}$. The simulator acting as $\mathcal{F}_{\mathsf{ledger}}$ accepts the value. Then acting as honest bidders, the simulator will request inclusion proof from the hub. Let $S_{\mathsf{chall\text{-}inc}}$ denote the set of honest bidders for whom the hub did not provide an inclusion proof. For each bidder in $S_{\mathsf{chall\text{-}inc}}$, the simulator simulates a call to the covenant on $\mathcal{F}_{\mathsf{SC}}$ whereby the bidder is challenging inclusion on the covenant. If for a given bidder, the hub responds by submitting a valid inclusion proof to the covenant on $\mathcal{F}_{\mathsf{SC}}$, then the simulator adds this bidder to the set $S_{\mathsf{rebut\text{-}inc}}$.

Next, in the real protocol, the hub may send messages to the covenant on $\mathcal{F}_{\mathsf{SC}}$ challenging opening for some bidders. For every honest bidder, the simulator acting as the hub to $\mathcal{F}_{\mathsf{auction}}$ would have received the bids in the clear. Therefore, when the hub challenges opening for honest bidders, the simulator uses these values to equivocate the commitment values specified in the covenant. Denote the set of dishonest bidders for whom the hub challenged opening on the covenant

on $\mathcal{F}_{\mathsf{SC}}$ by $S_{\mathsf{chall\text{-}open}}$. Acting as $\mathcal{F}_{\mathsf{SC}}$, the simulator waits to receive messages from the adversary acting as dishonest bidders on the covenant. If the adversary sends a successful opening to the challenges on the covenant, then the simulator adds these dishonest bidders to the set $S_{\mathsf{rebut\text{-}open}}$.

Finally, the simulator, acting as $\mathcal{F}_{\mathsf{ledger}}$ waits to receive a UC secure zkSnark proof π from the dishonest hub on the auction contract. Then using the extractor implied by UC security, the simulator obtains the witness which yields a set S corresponding to the set of bidders among whom the winner is calculated (cf. Equation 1). The simulator then submits the message $(\mathbf{reveal}, S)$ and $(\mathbf{settle}, S_1 = S_{\mathsf{chall\text{-}inc}} \setminus S_{\mathsf{rebut\text{-}inc}}, S_2 = S_{\mathsf{chall\text{-}open}} \setminus S_{\mathsf{rebut\text{-}open}})$ to $\mathcal{F}_{\mathsf{auction}}$ in the **Settle** stage.

The simulation in the **Resolve** stage is fairly straightforward. If the simulator acting as $\mathcal{F}_{\mathsf{SC}}$ receives a resolve message from the hub then the simulator forwards this message to $\mathcal{F}_{\mathsf{auction}}$. On the other hand, for every honest bidder bidder_i in S_1, the simulator submits the message $(\mathbf{resolve}, \mathsf{bidder}_i)$ to $\mathcal{F}_{\mathsf{auction}}$. This completes the description of the simulator.

It is straightforward to see that this simulation can be proven indistinguishable from the real execution.

Case 2. Hub is Honest. Next we consider the case when the hub is honest.

In the real protocol, the seller and hub agree on the auction parameters. Then the simulator simulating the honest hub sends a message to $\mathcal{F}_{\mathsf{ledger}}$ to deploy the auction contract, and notifies the seller. When the seller begins by notifying the hub of its intent to create a covenant via $\mathcal{F}_{\mathsf{SC}}$ (note that the covenant hard-codes the auction contract address), the simulator acting as $\mathcal{F}_{\mathsf{SC}}$ waits for such a message, and upon receiving it sends the message $(\mathbf{transfer}, \mathsf{nft})$ to $\mathcal{F}_{\mathsf{auction}}$.

Next, we handle covenant creation with the bidders. In more detail, if the simulator acting as $\mathcal{F}_{\mathsf{SC}}$ receives a message to initiate the covenant creation from a dishonest bidder, then it extracts the bid amount (thanks to the simulation extractability of the UC-secure noninteractive commitment scheme; note that the covenant has the sealed bid encoded in it). Then the simulator sends $(\mathbf{bid}, \mathsf{amt}_i)$ to $\mathcal{F}_{\mathsf{auction}}$, and emulates the hub in completing the covenant creation.

In the next step, the simulator will use a commitment to the zero string for sealed bids corresponding to the honest bidders, and then creates an accumulator root combining the dishonest bids along with the (honest) zero bids. Acting as $\mathcal{F}_{\mathsf{ledger}}$, the simulator submits the accumulator root to the auction contract, and notifies the adversary. (This step will be indistinguishable from the protocol due to the hiding property of the commitment.)

Then when the dishonest bidders request an inclusion proof from the simulator (acting as honest hub), it will provide the correct inclusion proofs (recall that the accumulator does contain all the dishonest bids).

Next, the simulator acting as the honest hub will request openings from the dishonest bidders. If the bidder provides an opening which is different from what the simulator extracted earlier, then the simulator aborts and outputs $\perp$. (Note that this will happen with negligible probability due to the binding property of the commitment.) If the bidder refuses to provide an opening, then

the simulator acting as $\mathcal{F}_{\mathsf{SC}}$ will submit a challenge opening (i.e., on behalf of a request from the emulated hub) to the dishonest bidder and adds it to the set $S_{\mathsf{chall\text{-}open}}$. If the bidder responds to the challenge, then the simulator acting as $\mathcal{F}_{\mathsf{SC}}$ will receive the opening. As before, if the opening is different from what the simulator extracted, then the simulation is aborted. Otherwise, the bidder is added to the set $S_{\mathsf{rebut\text{-}open}}$. The simulator constructs the set S by adding all the honest bidders to it, and also all the dishonest bidders who provided the expected opening (either directly or on the covenant to $\mathcal{F}_{\mathsf{SC}}$). The simulator then sends the message $(\mathbf{reveal}, S \cup \{0\})$ and additionally submits the message $(\mathbf{settle}, S_1 = \emptyset, S_2 = S_{\mathsf{chall\text{-}open}} \setminus S_{\mathsf{rebut\text{-}open}})$ to $\mathcal{F}_{\mathsf{auction}}$.

The simulator then receives the identity of the winner from $\mathcal{F}_{\mathsf{auction}}$. Now the simulator has all the information that it needs to provide a simulated proof according to Equation 1, i.e., using the zero-knowledge simulator implied by the UC-security of the employed zkSnark scheme.

Once all this is done, the simulation in the **Resolve** stage is fairly straightforward. If the simulator acting as $\mathcal{F}_{\mathsf{SC}}$ receives a resolve message from any of the dishonest bidders, then the simulator forwards this message to $\mathcal{F}_{\mathsf{auction}}$. This completes the description of the simulator.

It is straightforward to see that this simulation can be proven indistinguishable from the real execution.

Finally, we briefly discuss the onchain complexity of our protocol. Excluding the update root operation, the number (and size) of onchain interactions wrt auction contract is $O(1)$ (transfer to and fro of the nft, and submission of the accumulator root and the zkSnark proof). When the hub is honest, the number of update root operations is bounded by the number of adversarial bidders, and since each update operation requires $\log n$ on-chain operations, the maximum size of such operations is bounded by $O(k \log n)$. When the hub is honest, there are no disputes between the hub and the honest bidders, and thus all the interactions happen offchain via $\mathcal{F}_{\mathsf{SC}}$. Between the hub and a dishonest bidder, the disputes may be resolved onchain via $\mathcal{F}_{\mathsf{SC}}$. However, the dispute resolution process is limited to challenging and rebutting inclusions and openings. The maximum onchain calls is therefore $O(k)$, and with total data size $O(k \log n)$.

E Achieved Properties

In this part, we provide an informal discussion on how $\mathcal{F}_{\mathsf{auction}}$, and consequently our auction protocol satisfies all desired properties defined in Section 4.2.

Auction Correctness. That the ideal functionality $\mathcal{F}_{\mathsf{auction}}$ correctly implements auction is straightforward. The winner is chosen as the one who has the maximum bid among a set of bidders S submitted by the simulator. When the hub is honest, this set must include all the honest parties. On the other hand, when the hub is dishonest, we ensure that either an honest party is included in the auction or it gets compensated by the hub.

Verifying the correctness of our auction protocol is straightforward by examining the logic of our contracts. By assuming that the underlying blockchain

and cryptographic primitives are secure and correct, our protocol ensures that once the covenants are created between participants, they must adhere to the execution of the protocol.

Privacy. That the ideal functionality $\mathcal{F}_{\mathsf{auction}}$ achieves privacy is straightforward. In particular, only the winning bid is revealed to all parties—all the other (honest) bids are hidden from every other bidder and the seller. Note that the hub is made aware of all the bids during the **Opening** stage. However, this is after the **Bidding** stage, so the malicious bids are independent of the honest bids. Note that a malicious hub can reveal the Therefore, if the hub is honest, we get *post-auction privacy* as well.

Our protocol achieves the desired privacy properties when both the hub and bidders follow the protocol. In particular, our protocol achieves *auction privacy* because of the hiding property of the commitment scheme. Hub will not learn anything about the committed amounts from users' sealed bids. Under an optimistic case, our protocol also achieves *post-auction privacy* due to the zero-knowledge property of the zkSnark scheme. Other bidders will not learn any information about the witness, aside from the amount of the winning bid and the address of the winner. The only time other bidders will learn about the amount committed to the sealed bid is when either the hub or a bidder is malicious, and an on-chain challenge is triggered.

Efficiency. That the ideal functionality $\mathcal{F}_{\mathsf{auction}}$ is efficient in terms of on-chain operations is relatively straightforward. To see why, we need to keep track of interactions of $\mathcal{F}_{\mathsf{auction}}$ with $\mathcal{F}_{\mathsf{ledger}}$. Such interaction happens only in the context of nft transfers, once in the **Transfer** stage (when the nft is transferred from the seller to $\mathcal{F}_{\mathsf{auction}}$), and once in the **Settle** stage (when the nft is transferred from $\mathcal{F}_{\mathsf{auction}}$ to the winner of the auction). The interactions between an honest hub and an honest bidder happens completely off-chain. When either the hub or the bidder is malicious, the balance adjustment in the **Resolve** stage may happen onchain, but this also means that the corresponding $\mathcal{F}_{\mathsf{SC}}$ (or PPC) channel will end up being closed (i.e., onchain).

Our auction protocol achieves efficiency due to the scalability property of PPC. In particular, bidders do not need to issue any on-chain transactions if both the hub and bidders follow the protocol. The only on-chain actions required from the hub are to start the auction, submit the Merkle root, and reveal the winner. Similarly, between the seller and the hub, in the optimistic case where both parties follow the protocol, the only on-chain action needed from the seller is to approve the NFT transfer on-chain. Finally, the on-chain cost is independent of the number of bidders due to the properties of the underlying zkSnark scheme. This scheme enables the smart contract to verify the validity of a statement with a constant cost, regardless of the size of the witness set (in our case, the number of bidders). Thus, scalability is preserved even as the number of participants in the auction increases.

Liveness. That the ideal functionality $\mathcal{F}_{\mathsf{auction}}$ ensures liveness just follows from inspection.

Our construction ensures liveness for two reasons. Firstly, the covenant requires bidders to open bids either off-chain or on-chain; otherwise, they may face financial penalties. Secondly, if malicious bidders are willing to pay the cost and do not open their sealed bids, we introduce a replacement mechanism that allows the hub to remove unopened bids from the Merkle tree. This enables the hub to issue a correct zkSnark proof based on all existing opened bids.

Security. That the ideal functionality $\mathcal{F}_{\mathsf{auction}}$ provides security is also straightforward. In particular, we wish to re-iterate that the dishonest bids are independent of the honest bids. This is true even when the hub is honest, whereby the hub must submit all the bids before the honest bids are revealed to it. It is also straightforward to see that the parties can bid exactly once.

Our construction directly satisfies *non-malleability* due to the non-malleable property of the underlying commitment scheme. This prevents a malicious hub from colluding with other bidders to issue a bid related to honest bidders. Similarly, the *bid binding* property is ensured by the binding property of the underlying commitment scheme.

Financial Fairness. That the ideal functionality $\mathcal{F}_{\mathsf{auction}}$ is financially fair follows from the fact that the honest parties' balances are never decreased (except of course when the winner happens to be an honest party). In particular, in the **Settle** stage, we ensure that $S_2 \subseteq C$ (recall that S_2 is the subset of bidders who get their balances decreased, i.e., the hub's balance is increased (on the pairwise channel)) and also that that when the hub is corrupt, $H \subseteq S \cup S_1$, and finally when teh hub is honest, we ensure that $H \subseteq S$ and that $S_1 = \emptyset$. Recall that S_1 is the subset of bidders whose balance is increased, and S is the subset of bidders who are eligible to be the winner of the auction.

Our auction protocol ensures financial fairness due to the covenant contract agreement. Specifically, once both parties mutually agree on the covenant contract, it enforces adherence to the subsequent steps of the protocol for each party. As a result, any deviation from the agreed-upon process by either party will lead to a financial penalty.

F Discussion

Trusted Setup in zkSnark. The use of Groth16's zkSnark requires a trusted setup to generate the evaluation and proving keys for the circuit. However, 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 [9,5,10] have proposed protocols for such trusted setups, demonstrating that as long as at least one participant is honest, the zkSnark instance remains secure. For example, the Zcash team performed an MPC setup for their protocol parameters in 2017 [39]. A second approach is to use universal setup zkSnark constructions [36,12,25,15] 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.

Duration Lengths. In Section 5, 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.

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[8] 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.[9]. 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, the generation of zk proofs in the *settling* stage heavily depends on the number of bidders (exact numbers are provided in Table 3), impacting the overall timing.

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

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