ZeroAuction: Zero-Deposit Sealed-bid Auction via Delayed Execution

Haoqian Zhang[1], Michelle Yeo[2], Vero Estrada-Galinanes[1], and Bryan Ford[1]

[1] École Polytechnique Fédérale de Lausanne
{haoqian.zhang,vero.estrada,bryan.ford}@epfl.ch
[2] National University of Singapore
mxyeo@nus.edu.sg

Abstract. Auctions, a long-standing method of trading goods and services, are a promising use case for decentralized finance. However, due to the inherent transparency property of blockchains, current sealed-bid auction implementations on smart contracts requires a bidder to send at least two transactions to the underlying blockchain: a bidder must first commit their bid in the first transaction during the bidding period and reveal their bid in the second transaction once the revealing period starts. In addition, the smart contract often requires a deposit to incentivize bidders to reveal their bids, rendering unnecessary financial burdens and risks to bidders. We address these drawbacks by enforcing delayed execution in the blockchain execution layer to all transactions. In short, the blockchain only accepts encrypted transactions, and when the blockchain has finalized an encrypted transaction, the consensus group decrypts and executes it. This architecture enables ZeroAuction, a sealed-bid auction smart contract with zero deposit requirement. ZeroAuction relies on the blockchain enhanced with delayed execution to hide and bind the bids within the encrypted transactions and, after a delay period, reveals them automatically by decrypting and executing the transactions. Because a bidder only needs to interact with the blockchain once instead of two times to participate in the auction, ZeroAuction significantly reduces the latency overhead along with eliminating the deposit requirement.

Keywords: Commit-and-Reveal · Auction · Blockchain · Decentralized Finance.

1 Introduction

The auction, an ancient way of negotiating the exchange of goods and services, enters the world of blockchains and decentralized finance powered by general smart contracts [27]. While the blockchain provides an ideal platform for open auctions in which every bidder can observe others' bids during the bidding period, it is notably challenging to implement sealed-bid auctions due to the inherent transparent property of blockchains.

A sealed-bid auction should hide the bids during the bidding phase, bind them so they can not be modified, and reveal the bids during the revealing phase. To implement a sealed-bids auction under a transparent blockchain, auctioneers often rely on a commit-and-reveal approach in which a bidder first sends a commit transaction which contains the hash of their bid during the bidding phase, and the bidder propagates the reveal transaction to disclose their bid once the bidding phase is over. Although this simple approach hides the bids, bidders can choose not to pay if they win. In addition, it allows bidders to only reveal the bids that financially benefit them, *e.g.*, by committing multiple bids, but only revealing the smallest one that can win the auction. Altogether, these types of misbehaviour negatively impact the *fairness* of the protocol. To financially discourage these actions, a sealed-bid auction smart contract often requires a deposit from bidders during the bidding phase and returns the deposit when a bidder honestly finishes the auction. Section 2 provides an example of such a smart contract simplified from previous work [5,16,17].

However, this approach still has several drawbacks: (a) a bidder needs to interact with the blockchain for at least two rounds causing excessive latency overhead; (b) the smart contract needs to store the commitment of each bid leading to unnecessary storage overhead; (c) the smart contract requires deposits to incentivize the bidders to reveal their bids even for a bidder who only wants to bid a little, rendering avoidable financial burdens; (d) as the smart contract keeps the deposit, there are security risks of the deposits, such as the DAO attack [18]; (e) it is challenging for the auctioneer to decide the required deposit [23]; (f) bidders can choose never to reveal their bids; (g) due to a network congestion event [15] or deliberate denial-of-service (DoS) attack [10], the blockchain might fail to include the revealed transaction; (h) the number of auctions in which a bidder can participate is limited by the deposit requirement.

To overcome these drawbacks, we adopt *delayed execution* [29], a feature embedded in the blockchain architecture. When executing transactions under delayed execution, consensus nodes must execute all transactions with a global delay time parameter, and consensus nodes should not observe the content of any transaction during the delay time period to ensure the effectiveness of the delay execution; thus, the blockchain has to accept the encrypted transactions and the decryption and execution of the transaction can only happen after the delay. A blockchain enhanced with delayed execution enables the optimization of the commit-and-reveal scheme because the underlying blockchain, rather than users, handles the bid reveals.

In reality, a recipient should only accept a transaction once the blockchain finalizes the transaction after T block confirmations to mitigate double-spending attacks. For example, Ethereum requires 64 block confirmations (2 epochs) for a transaction to be finalized [11]. Our delayed execution adopts the same delay time parameter as the required T block confirmations for all transactions so that the blockchain executes and finalizes a transaction at the same time after a T block delay, and the delayed execution does not increase the transaction latency.

We demonstrate ZeroAuction, a sealed-bid auction smart contract with *zero* deposit requirement under our delayed execution. In ZeroAuction, a bidder only needs to send one transaction in which the bidder pays for their bid if it is the current highest bid. The blockchain under delayed execution automatically hides transactions; thus, bids are private during the bidding phase. When entering the revealing phase, the blockchain executes the transactions one by one, charging the current highest bid while returning other funds. Because a bidder only needs to send one transaction instead of two to participate in the auction, ZeroAuction reduces the latency for bidders by half and eliminates the deposit requirement. We present ZeroAuction in detail in Section 3.

We stress that ZeroAuction mitigates the drawbacks mentioned above: (a) a bidder only needs to interact with the blockchain once instead of two, significantly reducing the latency overhead; (b) ZeroAuction does not need to store the bids commitment from bidders in the smart contract; (c)-(e) ZeroAuction eliminates the deposit requirement; (f)-(g) the blockchain guarantees the revealing of all bids regardless of the bidders' behaviors and network environment; (h) a bidder can use the same funds to bid in multiple auctions, as there is no deposit requirement. To our knowledge, our solution is the first blockchain-based sealed-bid auction solution without a deposit requirement. However, we note that these advantages also come with an associated cost: auctioneers can not set up their preferred bidding period for more than the fixed global delayed time.

Previously, F3B utilized the idea of delayed execution to mitigate front-running attacks with a negligible latency overhead [29]. Tuxedo uses delayed execution to scale computations on blockchains [9]. This paper demonstrates the potential of delayed execution from the point of optimizing sealed-bid auction smart contracts.

Finally, we discuss a number of promising approaches for implementing delayed execution. We conclude that the threshold encryption method achieves the best trade-off within the existing toolbox in terms of achieving the properties of delayed execution, providing compatibility with existing blockchains, and inducing a reasonable latency overhead [29,19].

2 Preliminaries

In this section, we briefly introduce the properties of a sealed-bid auction, the commit-and-reveal scheme with a sealed-bid auction example and the delayed execution abstraction.

2.1 Sealed-bid Auction Properties

We require a sealed-bid auction smart contract to satisfy at least the following properties (formal definitions in Appendix A):

- **Hiding:** No bidder knows the bid of any other bidder during the bidding period.
- **Binding:** A bidder can not change their bid once the blockchain finalizes the bidding transaction.
- **Revealing:** All the sealed bids will be revealed during the revealing period.

Algorithm 1: Commit-and-reveal auction smart contract

```
1  Init Upon creating the auction smart contract:
2      Set $d$ as required deposit for the auction
3      $highest \leftarrow 0$, $winner \leftarrow \varnothing$, $hash \leftarrow []$
4
5  Bid Upon receiving $i$'s commitment $c_i$ first time in bidding period:
6      Assert($i$ transfers $d$)
7      $hash[i] \leftarrow c_i$
8
9  Reveal Upon receiving $i$'s bid $b_i$ and salt $r_i$ first time in revealing period:
10     Assert(Hash$(b_i, r_i) = hash[i]$)
11     Assert($b_i \leq d$)
12     if $b_i > highest$ then
13         Distribute $highest$ to $winner$ when $winner \neq \varnothing$;
14         Distribute $d - b_i$ to $i$
15         $highest \leftarrow b_i$
16         $winner \leftarrow i$
17     else
18         Distribute $d$ to $i$;
19     end
```

– **Non-malleability:** No bidder can alter any encrypted bid from others into another form such that the plaintext of the altered encrypted bid is related to the original bid.

The non-malleability property ensures that simply observing one bidder's encrypted bid does not give another bidder an unfair advantage, for example, to prevent a malicious bidder from altering an existing bid's ciphertext to bid 1 coin more than the value in the encrypted bid. Specifically, we consider the notion of NM-CPA security commonly used in the context of sealed-bid auctions [7,20,6]. Intuitively, NM-CPA security states that the plaintext decryptions of encrypted bids produced by an adversarial bidder must be indistinguishable.

To illustrate the benefit of delayed execution, we did not consider the posterior privacy property, which hides the losing bids from the public. We note that additional cryptographic tools like Zero-Knowledge Proofs (ZKP) or Multi-Party Computation (MPC) are needed for sealed-bid auctions ensuring the posterior privacy property [13,12,3].

2.2 Commit-and-Reveal Smart Contract for Sealed-bid Auction

Algorithm 1 illustrates implementing a commit-and-reveal smart contract simplified from real-world examples on the blockchain for a sealed-bid auction [5,16,17]. The smart contract contains two phases: (a) the bidding phase (**Bid** function), where each bidder submits their hidden bid commitment. This implements the commit stage of the commit-and-reveal scheme. (b) the revealing phase (**Reveal** function), where every bidder reveals his bid and the contract determines the winning bidder. This implements the reveal stage of the commit-and-reveal scheme.

In the bidding phase, bidders submit their hidden bid commitment c_i, and to ensure that bidders have enough funds to pay for what they bid, each bidder needs to transfer a deposit d. For simplicity, the contract requires any bid must be equal or smaller than the required deposit[3]. The array *hash* stores the commitments for all bidders.

In the revealing phase, each bidder submits their bid b_i and random salt r_i. The smart contract first checks that the hash of the bid and salt is identical to the commitment c_i sent in the bidding phase. If the hash is not equal to the hash of c_i, the function terminates with the bidder losing their deposit. Otherwise, if b_i is the current highest bid, the contract keeps b_i of the bidder's deposit and returns the remainder $d - b_i$ portion of the deposit to the ith bidder and the fund of the previous *winner*, if any. If b_i is not the current highest bid, the contract returns all his deposit d. When all bidders reveal their bids, the contract determines the *winner*, which is the bidder who submitted the highest bid and can pay for their bid.

This commit-and-reveal auction smart contract can satisfy the hiding, binding, and non-malleability properties, but it only partially satisfies the revealing property. The hiding and binding property directly follows from the underlying cryptographic commitment scheme. The non-malleability property may or may not be satisfied depending on the underlying commitment scheme (*i.e.*, whether the commitments are non-malleable [8]). Finally, the revealing property is enforced by fact that the deposit held by the smart contract is larger than the bids as well as the check done by the smart contract (Line 10) to ensure that the smart contract will slash the deposit if bidders do not reveal their bids or reveal an incorrect bid. Therefore, rational bidders will choose to reveal their bids during the reveal phase. Nevertheless, this revealing guarantee *does not hold* for malicious bidders.

We note, however, that this contract has several notable drawbacks:

(a) Each bidder has to interact with the blockchain for two rounds, increasing the latency overhead.
(b) The contract needs to store the bid commitment for each bidder, which increases the storage overhead.
(c) The deposit acts as an additional financial threshold for participating in the auction. Even a bidder who only wants to bid a small amount of funds could have to pay a high deposit to even participate in the auction.
(d) As the smart contract keeps the deposit, the deposits are exposed to security risks, such as the DAO attack [18].
(e) The auctioneer must set up the required deposit as an upper bound for all bids. Determining a minimal upper bound is extremely difficult when initi-

[3] By allowing a bidder to bid more than his deposit, they may choose not to pay for their bid at a cost of losing his deposit.

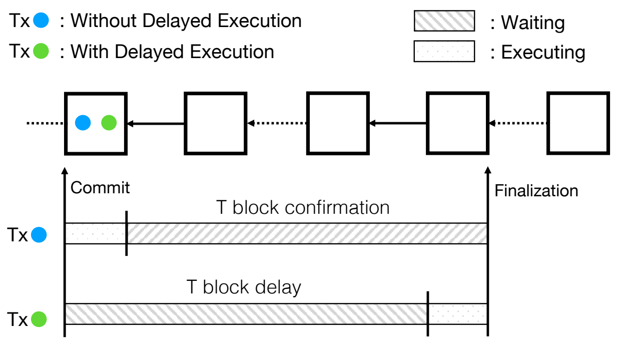

Fig. 1. In a blockchain without delayed execution, the consensus nodes execute the blue transaction upon its commitment, but recipients must wait for T block confirmation until its finalization. In the blockchain with delayed execution, the consensus nodes first wait for a T block delay before decrypting and executing the green transaction when the blockchain finalizes the transaction. Both transactions have the same finalization time; thus, the delayed execution does not increase transaction latency.

ating the contract as it relies on external knowledge of bidders' preferences and solvency status[23].

(f) Users can choose never to reveal their bids. In particular, unless we impose stronger assumptions on the adversarial model of the users in the system (e.g., deposit slashing with only rational users, which would make it irrational to withhold revealing of bids), there is no guarantee that Algorithm 1 can implement a commit-and-reveal smart contract with revealing property.

(g) The revealed bid might be missed due to network congestion [15] or a deliberate denial-of-service (DoS) attack [10], violating the revealing property.

(h) The deposit requirement limits the number of auctions in which a bidder can participate, as the coins used in the deposit for one auction cannot be used as a deposit for another auction.

2.3 Delayed Execution Abstraction

A delayed execution is an abstraction on the execution layer to ensure blockchains execute transactions with a delay [29,9]. We require that consensus nodes must execute all transactions with a fixed delay time. Furthermore, consensus nodes should not observe the content of any transaction during the delayed period to ensure the effectiveness of the delay execution. Hence, the blockchain has to accept *encrypted transactions*, and the decryption and execution of the transactions can only happen after the delay.

We formally define an abstraction of a delayed execution protocol on an underlying blockchain as follows:

Definition 1 (Delayed Execution Abstraction). *A delayed execution abstraction Π of some other protocol Π' is a tuple $(T_0, T, T', \Pi', (\mathsf{Enc}, \mathsf{Dec}))$ where*

$\infty > T' > T > 0$, *T_0 is the time of execution of Π', and* $(\mathsf{Enc}, \mathsf{Dec})$ *are the encryption and decryption functions of a committing encryption scheme. Π takes the same inputs as Π' and encrypts the inputs using* Enc. *Π ensures that its outputs are the same as Π', and that any decryption of encrypted ciphertexts and outputs occur before $T_0 + T$ w.p. ϵ for some negligible $\epsilon > 0$ and after $T_0 + T'$ w.p.* 1.

In the context of blockchains, Π' could refer to the execution of transactions or smart contracts running on the blockchain. For instance, Π' can be an open auction smart contract, and T_0 would denote the block's height in the blockchain containing the committed smart contract. The T parameter represents the delay time in blocks to delay the transaction outputs. T' denotes the upper bound on the execution time of Π, *i.e.*, the time it takes to obtain the outputs of Π' considering the delay time and the execution time of the underlying protocol Π'.

Choosing Confirmation Time as Delay Time: Although transactions are delayed as ciphertexts, we can choose a specific delay time so that adding delayed execution does not increase the transaction latency of the underlying blockchain. In practice, we note that all blockchains require recipients to wait for certain number of block confirmations before accepting a transaction to mitigate double-spending attacks. For instance, Ethereum requires 64 block confirmations (2 epochs) for a transaction to be finalized [11]. Without loss of generality, we assume that our underlying blockchain requires T block confirmation[4] to finalize a transaction into the blockchain. If we adopt the same T for the delayed time, the blockchain with delayed execution can finalize a transaction at the same time as the underlying blockchain. Figure 1 illustrates this process: in the underlying blockchain without delayed execution, the consensus nodes immediately execute the blue transaction upon its commitment, but then recipients must wait for T block confirmation until its finalization. In contrast, for the blockchain with delayed execution, the consensus nodes first wait for the T block delay before decrypting and executing the green transaction when the blockchain finalizes the transaction. Therefore, both transactions have the same finalization time, and the delayed execution with a T block delay does not increase transaction latency.

3 Auction Smart Contract with Delayed Execution

This section introduces ZeroAuction, an auction smart contract under the delayed execution, with a pseudocode and running examples. We argue how ZeroAuction satisfies the properties of the sealed-bid auction informally and show how ZeroAuction can overcome the drawbacks mentioned in Section 2.

3.1 Pseudocode

Algorithm 2 describes the ZeroAuction smart contract. When the auctioneer creates the smart contract, the consensus nodes run the **Init** function, which

[4] T can be 1 for the blockchains with instant finalization.

Algorithm 2: ZeroAuction smart contract with delayed execution

```
1  Init Upon creating the auction smart contract:
2      $highest \leftarrow 0$, $winner \leftarrow \varnothing$
3
4  Bid Upon receiving $i$'s bid $b_i$ in the bidding period:
5      if $b_i > highest$ then
6          Assert($i$ transfers $b_i$)
7          Distribute $highest$ to $winner$ when $winner \neq \varnothing$;
8          $highest \leftarrow b_i$
9          $winner \leftarrow i$
10     end
```

initializes two valuables: *highest*, which indicates the value of the current highest bid, and *winner*, which records the current winner of the auction. During the biding period, each bidder can submit his bid by calling **Bid** function, which encapsulates both commit and reveal phases. The function checks whether this bid is more than the current *highest*. If so, the smart contract asks the bidder to transfer the amount of his bid to itself, refunds the current *winner*, and finally update the current *highest* and *winner*. If not, the bidder loses the auction.

3.2 Under the Delayed Execution Environment

ZeroAuction, as presented in Algorithm 2, implements an open auction, as none of the bids are hidden. However, it becomes a sealed-bid auction when employed in a delayed execution environment. Bidders submit their bids within the delay time, so the delayed execution guarantees hiding the bids during the bidding period and revealing them during the revealing period.

Requirements: We require the delay time in the delayed execution to be the same as the confirmation time T of the underlying blockchain. Thus, when the blockchain with delayed execution decrypts and executes the transaction, it also finalizes the transaction without extra latency overhead. We also require the bidding time in any sealed-bid auction to be $\leq T$. Observe that a bidding time of more than T reveals the plaintext of encrypted bids submitted at the start of the bidding period to other bidders before the bidding period is over. We further demand that the blockchain delay executes all transactions by T time, including non-auction transactions, such as transfer transactions. This requirement ensures that no user can make quick transfers to another account during the auction to affect the outcome of the auction, given information revealed about others' bids during the revealing period.

Non-malleability: To guarantee the non-malleability property of the auction, a bidder first encrypts their transaction using a symmetric non-malleable encryption scheme, and the delayed execution ensures the release of the symmetric key after the delay so that consensus nodes can decrypt and execute the transaction. Specifically, when committing a ZeroAuction contract under delayed

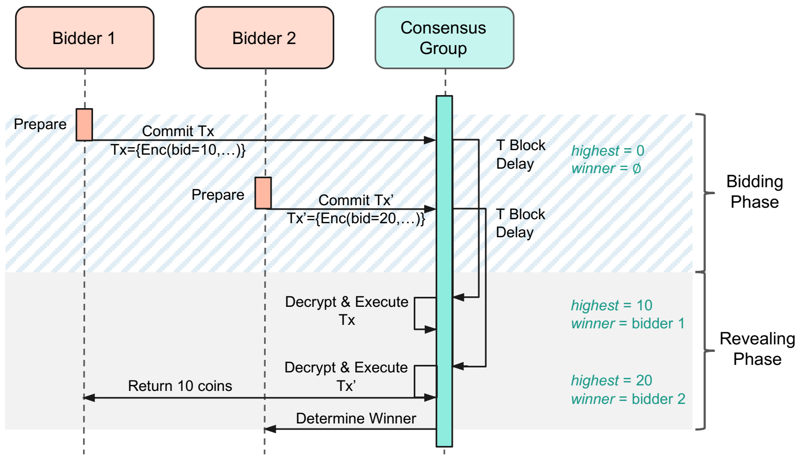

Fig. 2. A running example with two bidders. Both bidders commit their encrypted transactions to the blockchain during the bidding phase. After T block delay of each transaction, the consensus group decrypts and executes the transaction. At the end of the execution phase, the smart contract can determine the final auction winner.

execution, participating bidders first need to encrypt their entire original signed transaction (which contains their bid) under a symmetric CPA-secure encryption scheme (*e.g.*, block cipher with CBC mode of operation) and then append a strongly-unforgeable message authentication code (MAC) onto the resulting ciphertext. This encrypt-then-MAC composition is NM-CPA secure under these assumptions on the encryption scheme and MAC [2]. Next, bidders also further encrypt all relevant keys used to encrypt-then-MAC their bid transaction under the delayed execution (further discussion in Section 4.1) and send the encrypted symmetric keys as well as the encrypted transactions to the blockchain. Finally, the delayed execution decrypts and releases the symmetric keys after T blocks so that consensus nodes can decrypt and execute the transaction.

3.3 Single Auction Running Examples

This subsection provides examples with two bidders under the delayed execution with T as the global delayed parameter. For simplicity, we assume the bidding period is T in this example. We logically create two phases: the first T blocks as the bidding phase and the second T blocks as the revealing phase.

Two successful bids: In the first example, we demonstrate two legitimate bidders, presented in Figure 2. Assuming there are two bidders, bidder 1 and bidder 2, with their bids being 10 and 20 coins, respectively. We further assume they have enough cryptocurrency in their account balances to support their bids if they win. During the bidding phase, both bidders must seal and hide their bids by encrypting their signed transactions. We assume that both bidders prepare their encrypted transactions, and the blockchain successfully commits them, with bidder 1's transaction ordered before that of bidder 2.

T blocks after committing the encrypted transactions on the blockchain, consensus nodes begin to decrypt and execute the transactions. Upon executing bidder 1's transaction, the value *highest* is 0, and bidder 1's bid is more than *highest*; thus, the following things happen: (a) Bidder 1 needs to transfer 10 coins to the smart contract, (b) *highest* updates to 10, (c) *winner* changes to bidder 1. Upon executing bidder 2's transaction, the value *highest* is 10 and, bidder 2's bid is more than *highest*; thus, the following things happen: (a) Bidder 2 needs to transfer 20 coins to the smart contract, (b) Bidder 1 receives the refund of 10 coins from the smart contract, (c) *highest* updates to 20, (d) *winner* changes to bidder 2. At the end of the revealing phase, the blockchain has executed all the bidding transactions. The value *winner* records bidder 2 as the auction's final winner, and bidder 2 has already paid for what he bids.

One failed bid: In the second example, we assume that bidder 2 does not have enough balance to support his bid. The procedure is the same as before until the blockchain executes the bidder 2's transaction. However, as bidder 2 does not have enough balance to transfer 20 coins to the smart contract, the transaction failed in the assertion, and consensus nodes then revert and discard the transaction. At the end of the revealing phase, we can be sure that bidder 1 is the final winner, and bidder 1 has already paid what he bids.

3.4 Auction with Transfer Transactions

As the consensus nodes cannot verify the transaction until its decryption and execution, the bidder could submit a bid with a value that is more than their balance during the bidding period. If the bidder wants to make the bidding transaction valid, they can make transactions to transfer more coins to their balance. However, the bidder must commit the transfer transaction before the commitment of the bidding transaction so that the blockchain executes the transfer transaction before the bidding transaction, as the blockchain delays all transactions by T blocks. In contrast, if, by the time of the execution, the bidder still does not have enough funds, the bidding transaction will fail as in Line 6 of Algorithm 2.

When the revealing phase starts, the bidder can observe the bids from other bidders and accordingly make a transfer transaction to change their balance. However, we argue that the bidder can not benefit from it. Because the blockchain delays all transactions by T blocks, it will execute the transfer transaction committed during the revealing phase after the execution of the bidding transaction. Thus, the transfer transaction does not affect the auction.

3.5 Multiple Auctions

Because ZeroAuction does not need deposit, a bidder can use the same funds to participate in multiple auctions. For example, suppose there are two auctions, with auction A happening before auction B. A bidder thus can bid all their funds for both auctions. If the bidder wins auction A, they will not have enough funds to support their bid for auction B; thus, auction B fails the assertion (Line 6). If the bidder does not win auction A, they can still use the same funds for auction B. The blockchain delaying all transactions guarantees that, even though the bidder's available fund for auction B may change because of auction A, such change happens without knowing any of the bids in auction B.

3.6 ZeroAuction Properties

We informally reason how the ZeroAuction achieves the sealed-bid auction properties. **Correctness:** After all executions, *winner* is the bidder who submits the highest bid with the ability to pay; Hence, *winner* is the legitimate winner of the auction. **Hiding:** As all transactions submitted during the bidding period are encrypted, no entity can observe the bids. **Binding:** Once the blockchain finalizes the bidding transaction, the consensus nodes firmly writes the transaction into the blockchain; thus, no entity can change the bid. **Revealing:** The blockchain guarantees the revealing of the bids during the revealing period. **Non-malleability:** As the encrypt-then-MAC composition used in our delayed execution is NM-CPA secure [2], no bidder can benefit from copying and changing other bidders' ciphertexts. We formally prove these properties in Section 5.

We acknowledge that ZeroAuction only achieves a weak version of binding in that a bidder can bid more than they have without consequence. We stress, however, that this is a feature and not a bug of ZeroAuction. In doing so, we eliminate the deposit requirement and allow a bidder to participate in multiple auctions using the same funds. Additionally, although the balance of the bidder may change before the execution of their bidding transaction and consequently alter the result of the auction due to the transactions committed before the bidding transaction, we note that such behavior only happens without the knowledge of other revealed bids due to all transactions being delayed for the same T blocks.

In addition, ZeroAuction addresses the drawbacks mentioned in Section 2:

(a) Each bidder only needs to interact with the blockchain once only during the bidding period (see **Bid** function in Algorithm 2), decreasing the latency overhead.

(b) ZeroAuction does not need to store bid commitments from each bidder as in the array of hashed commitments (*hash*) in Algorithm 1, reducing the storage overhead.

(c)–(e) ZeroAuction does not require any deposit; hence, ZeroAuction mitigates the security risk of the deposited funds and relieves the auctioneer from determining the deposit amount.

(f)–(g) The blockchain guarantees the revealing of bids regardless of the bidders' behaviors and network environment.

(h) As ZeroAuction does not require a deposit, a bidder can participate in multiple auctions using the same fund.

However, achieving those advantages brings an inflexibility: ZeroAuction cannot have a bidding period of more than T blocks.

4 Discussion

In this section, we discuss possible implementations of the delayed execution methods and apply the delayed execution to optimize all the commit-and-reveal smart contracts.

4.1 Possible Encryption Methods for Delayed Execution

There are multiple approaches to implement a delayed execution. This subsection briefly discusses those approaches and argues that threshold encryption, though imperfect, is the best approach to implement delayed execution within the existing toolbox.

Encryption and Decryption by a Centralized Authority: A centralized authority can help to reveal the transaction. When committing a transaction to the blockchain, a user first encrypts the transaction with a symmetric key and then uses the public key of the centralized authority to encrypt this symmetric key. After the delay, the centralized authority reveals the symmetric key. However, this approach brings the single-point-of-failure issue to the system that the centralized authority might be malicious or offline.

Time-lock Puzzle: Alternatively, a user can use a time-lock puzzle to blind the transaction. For example, blockchain can use a verifiable delay function [4,21] to implement a proof-of-elapsed-time, and consensus nodes have to compute the function to decrypt the transaction. However, it is still an open challenge to link the computational time to real-time, and the variance of the solving time can pose a security issue to delayed execution, *e.g.*, a lucky node solves the puzzle fast, thus observing the bid during the bidding period.

Trusted Execution Environment: Instead of trusting an authority, we can utilize a trusted execution environment achieved by hardware, such as Intel SGX [28]. The hardware guarantees the delayed execution of transactions. Nevertheless, this method is subject to a single point of failure or compromise [22,25].

Threshold Encryption: Instead of relying on a centralized authority, we can rely on a group of nodes to decrypt a transaction. Specifically, t out of n nodes can decrypt the transaction after the delay. However, $t-1$ out of n can not reveal anything about the transaction, even if they collude. While it is natural to deploy this method for a permissioned blockchain, it is unclear how to offer compatibility to a proof-of-work blockchain without changing its security assumption. In addition, running threshold decryption increases the overhead of the underlying blockchain system.

In conclusion, each method has its pros and cons, and none of the methods are entirely satisfactory. We hope that as scientific research advances in this domain, better methods will emerge. Nevertheless, from a practical perspective within the existing toolbox, we argue that the threshold encryption method achieves the properties of the delayed execution (guaranteed by its security assumption), provides maximum compatibility to various blockchain systems' security assumptions (not relying on any centralized component), and demonstrates a reasonable latency delay [29,19]. We note that existing work F3B uses delayed execution based on threshold encryption to mitigate front-running attacks, and its threshold encryption component only induces a negligible (0.026%) increase in transaction latency on Ethereum [29], demonstrating that the threshold encryption solution is practical in real-world scenarios.

4.2 Optimizing Other Commit-and-Reveal Smart Contracts

Although we only discuss optimizing the sealed-bid auction example under the delayed execution, such a technique can generally be applied in other smart contract on blockchains relying on the commit-and-reveal scheme, such as voting, quizzes, random number generations, and games. In general, a user no longer needs to interact with the blockchain for two rounds to commit and reveal the value, but the blockchain, by default, hides and reveals the value (and the entire transaction). Thereby, other commit-and-reveal smart contracts can benefit in many ways similar to ZeroAuction: (a) reducing the latency cost that a user only writes data to the blockchain once; (b) preventing the output bias that a user may choose not to reveal his value in the traditional commit-and-reveal approach; (c) eliminating the deposit requirement in the committing phase, as the blockchain guarantees the revealing of the secret value. We leave the proof and demonstration of this generalization as future work.

5 Analysis of Delayed Execution

In this section, we formally show that ZeroAuction achieves all the necessary properties of a sealed-bid auction. For ease of exposition, we defer the formal definition of a commit-and-reveal scheme as well as the non-malleability property in Appendix A.

Let Π be a delayed execution abstraction with delay parameter T and execution time upper bound T' of a ZeroAuction protocol Π' for a single bid. Recall that T_0 is the time of execution of the underlying protocol Π'.

Theorem 1. *Π is ϵ_1-hiding in the time period $[T_0, T_0 + T]$, ϵ_2-binding, and T'-revealing for $\epsilon_1, \epsilon_2 > 0$, ϵ_1, ϵ_2 negligible, and finite $T' > 0$.*

Proof. Recall that the ZeroAuction protocol Π' does not alter the input bid in any form and the output of Π' is simply the input bid.

We first show that Π is ϵ_1-hiding for the duration $[T_0, T_0 + T]$. Let m be the input of Π' (the bid). Π runs $\mathsf{Enc}(m, r) = c$ for input m and some random value r. As $(\mathsf{Enc}, \mathsf{Dec})$ is CPA secure, and decryption of c only occurs before time $T_0 + T$ w.p. ϵ, the probability ϵ_1 that the message m can be retrieved from the ciphertext c during the time interval $[T_0, T_0 + T]$ is $\epsilon_1 < \epsilon + \epsilon'$ where ϵ' is the probability of a PPT adversary breaking the underlying encryption scheme. Since ϵ and ϵ' are both negligible, ϵ_1 is also negligible.

We now show that $\mathit{\Pi}$ is ϵ_2-binding. Since $(\mathsf{Enc}, \mathsf{Dec})$ is a committing encryption scheme, for any distinct m, m', $\mathbb{P}[\mathsf{Dec}(\mathsf{Enc}(m)) = m'] < \epsilon_2$ for some negligible ϵ_2. Thus $\mathit{\Pi}$ is ϵ_2-binding.

Finally, we show that $\mathit{\Pi}$ is T'-revealing. From Definition $1, \mathit{\Pi}$ outputs the same outputs as $\mathit{\Pi}'$ almost surely within T' time from the execution of $\mathit{\Pi}$ for finite T'. Thus, $\mathit{\Pi}$ is T'-revealing. □

Theorem 2. *$\mathit{\Pi}$ with a CPA secure symmetric ecryption scheme and strongly unforgeable MAC is NM-CPA secure.*

Proof. Follows as per Theorem 4.4 in [2].

Theorem 3. *$\mathit{\Pi}$ satisfies all* 4 *sealed-bid auction properties.*

Proof. Follows directly as a consequence of Theorems 1 and 2.

6 Related Work

Many papers have proposed to implement sealed-bid auctions on blockchains. Depending on who reveals the bids, we can classify the sealed-bid auction designs into two types: relying on bidders or a trusted third party(*e.g.*, an auctioneer) or group.

For the auction designs relying on bidders to reveal the bids [5,16,17] (similar to our commit-and-reveal design in Section 2.2), the bidders must first commit their bids and then reveal them, with two rounds of interactions with the blockchain causing excessive latency overhead. Further, such design requires to incentivize bidders to reveal their bids.

A trusted third party(group) can help to reveal the bids. Galal and Youssef proposed a sealed-bid auction that relies on an auctioneer to determine the winner with verifiability using zero-knowledge proofs (ZKP) [13,12]. Nevertheless, it still requires bidders to interact with the blockchain for two rounds. Strain utilizes the two-party computation to compare pairs of bids and uses ZKP to prove the outcome is correct [3]. However, its complexity grows as bidders increase, and it is too costly to use on the Ethereum blockchain. Trustee [14] relies on Intel SGX [28] to implement a sealed-bid auction while reducing the gas cost on Ethereum. However, it brings the concerns of a single point of failure or compromise. Riggs uses time-based cryptography on the application layer to achieve a non-interactive auction, but the high gas cost makes it not practical on Ethereum [24].

We propose ZeroAuction to demonstrate how delayed execution optimizes sealed-bid auction smart contracts. Unlike previous work, a bidder does not need a deposit to participate in the auctions and only needs to interact with the blockchain once. However, our solution has an upper bound of possible bidding duration.

Limited works have explored the delayed execution on blockchains. Tuxedo uses delayed execution to scale computations on blockchains [9]. F3B utilizes delayed execution to mitigate front-running attacks with a negligible latency overhead and addresses the spamming and incentive issues when a blockchain adopts the delayed execution [29].

7 Conclusion

In this paper, we first introduce a technique that delay executes transactions on the blockchain and also outline a protocol named ZeroAuction that demonstrates the utility of our delay execution technique to sealed-bid auctions. We show that if the delay time is the same as the confirmation time of the underlying blockchain, delay executing transactions does not increase transaction latency. Further, we demonstrate that when executing ZeroAuction under delayed execution, bidders do not need a deposit and only require one round of interaction with the blockchain to participate in a sealed-bid auction. However, our solution brings a major inflexibility: the bidding period can be at most the global delay time.

References

1. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: FOCS. pp. 394–403. IEEE Computer Society (1997)
2. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. J. Cryptol. **21**(4), 469–491 (2008)
3. Blass, E.O., Kerschbaum, F.: Strain: A secure auction for blockchains. In: Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I 23. pp. 87–110. Springer (2018)
4. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Annual international cryptology conference. pp. 757–788. Springer (2018)
5. Chen, B., Li, X., Xiang, T., Wang, P.: Sbrac: Blockchain-based sealed-bid auction with bidding price privacy and public verifiability. Journal of Information Security and Applications **65**, 103082 (2022)
6. Choi, S.G., Dachman-Soled, D., Malkin, T., Wee, H.: A black-box construction of non-malleable encryption from semantically secure encryption. J. Cryptol. **31**(1), 172–201 (2018)
7. Coretti, S., Dodis, Y., Tackmann, B., Venturi, D.: Non-malleable encryption: Simpler, shorter, stronger. IACR Cryptol. ePrint Arch. p. 772 (2015)
8. Crescenzo, G.D., Katz, J., Ostrovsky, R., Smith, A.D.: Efficient and non-interactive non-malleable commitment. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 2045, pp. 40–59. Springer (2001)
9. Das, S., Awathare, N., Ren, L., Ribeiro, V.J., Bellur, U.: Better late than never; scaling computation in blockchains by delaying execution. arXiv preprint arXiv:2005.11791 (2020)
10. Eskandari, S., Moosavi, S., Clark, J.: Sok: Transparent dishonesty: front-running attacks on blockchain. 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 23. pp. 170–189. Springer (2020)

11. Gasper (2022), https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/gasper/, accessed: 2022-10-03
12. Galal, H.S., Youssef, A.M.: Succinctly verifiable sealed-bid auction smart contract. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology: ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings 13. pp. 3–19. Springer (2018)
13. Galal, H.S., Youssef, A.M.: Verifiable sealed-bid auction on the ethereum blockchain. In: International Conference on Financial Cryptography and Data Security. pp. 265–278. Springer (2018)
14. Galal, H.S., Youssef, A.M.: 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 23. pp. 190–207. Springer (2020)
15. Kharif, O.: Cryptokitties mania overwhelms ethereum network's processing (2017), https://www.bloomberg.com/news/articles/2017-12-04/cryptokitties-quickly-becomes-most-widely-used-ethereum-app
16. Król, M., Sonnino, A., Tasiopoulos, A., Psaras, I., Rivière, E.: Pastrami: privacy-preserving, auditable, scalable & trustworthy auctions for multiple items. In: Proceedings of the 21st International Middleware Conference. pp. 296–310 (2020)
17. Lu, G., Zhang, Y., Lu, Z., Shao, J., Wei, G.: Blockchain-based sealed-bid domain name auction protocol. In: Applied Cryptography in Computer and Communications: First EAI International Conference, AC3 2021, Virtual Event, May 15-16, 2021, Proceedings 1. pp. 25–38. Springer (2021)
18. Mehar, M.I., Shier, C.L., Giambattista, A., Gong, E., Fletcher, G., Sanayhie, R., Kim, H.M., Laskowski, M.: Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack. Journal of Cases on Information Technology (JCIT) **21**(1), 19–32 (2019)
19. Momeni, P.: Fairblock: Preventing blockchain front-running with minimal overheads. Master's thesis, University of Waterloo (2022)
20. Pass, R., Shelat, A., Vaikuntanathan, V.: Relations among notions of non-malleability for encryption. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 4833, pp. 519–535. Springer (2007)
21. Pietrzak, K.: Simple verifiable delay functions. In: ITCS. LIPIcs, vol. 124, pp. 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
22. Ragab, H., Milburn, A., Razavi, K., Bos, H., Giuffrida, C.: Crosstalk: Speculative data leaks across cores are real. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1852–1867. IEEE (2021)
23. Schwartzbach, N.I.: Deposit schemes for incentivizing behavior in finite games of perfect information. CoRR **abs/2107.08748** (2021)
24. Tyagi, N., Arun, A., Freitag, C., Wahby, R., Bonneau, J., Mazières, D.: Riggs: Decentralized sealed-bid auctions. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 1227–1241 (2023)
25. Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 991–1008 (2018)
26. Wee, H.: One-way permutations, interactive hashing and statistically hiding commitments. In: TCC. Lecture Notes in Computer Science, vol. 4392, pp. 419–433. Springer (2007)
27. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)

28. Xing, B.C., Shanahan, M., Leslie-Hurd, R.: Intel® software guard extensions (intel® SGX) software support for dynamic memory allocation inside an enclave. Proceedings of the Hardware and Architectural Support for Security and Privacy 2016 pp. 1–9 (2016)
29. Zhang, H., Merino, L.H., Qu, Z., Bastankhah, M., Estrada-Galiñanes, V., Ford, B.: F3B: A Low-Overhead Blockchain Architecture with Per-Transaction Front-Running Protection. In: Bonneau, J., Weinberg, S.M. (eds.) 5th Conference on Advances in Financial Technologies (AFT 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 282, pp. 3:1–3:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.AFT.2023.3, https://drops.dagstuhl.de/opus/volltexte/2023/19192

A Cryptographic primitives and definitions

A.1 Commit-and-Reveal Scheme

A commit-and-reveal scheme is a pair of algorithms $(\mathsf{Commit}, \mathsf{Reveal})$ run between two parties S and R in two stages, where the sender S wants to commit a message m to the receiver R. In the following, we adapt the notation and definition of commitment scheme from previous work [26]. Both parties receive a security parameter (1^n) as input. In the commit stage, $\mathsf{Commit}(m, r)$ takes as a plaintext message m from a message space $\mathcal{M}$, a random string r, and outputs a commitment string c. In the reveal stage, $\mathsf{Reveal}(c, r, m^*)$ takes as input a commitment string c, randomness r and an auxiliary parameter m^*[5]. The sender S sends a single message m^* as well as r to R and R either outputs m and accepts or rejects.

In the context of sealed-bid auctions, we require a commit-and-reveal scheme to satisfy 3 additional properties: ϵ_1-hiding, ϵ_2-cryptographic binding, and τ-revealing for positive but negligible values of ϵ_1 and ϵ_2, and some $\tau > 0$. We define these properties as follows:

Definition 2 (ϵ-hiding). *Let $\Pi = (\mathsf{Commit}, \mathsf{Reveal})$ be a commit-and-reveal scheme. Let us define the following game played between* PPT *R and honest S during the commit stage of the commit-and-reveal protocol.*

- *R chooses $m_0, m_1 \in \mathcal{M}$ and gives m_0, m_1 to S*
- *S runs $c_0 \leftarrow \mathsf{Commit}(m_0, r_0)$ and $c_1 \leftarrow \mathsf{Commit}(m_1, r_1)$*
- *S samples a random bit $b \overset{\$}{\leftarrow} \{0, 1\}$ and gives c_b to R*
- *R outputs a guess bit b'*

We say Π is ϵ-hiding if $\forall\ m_0, m_1$, $\mathbb{P}[b' = b] < \frac{1}{2} - \epsilon$ for some $\epsilon > 0$.

[5] We use m^* as a placeholder for auxiliary information that S might have to send R depending on how the commit-and-reveal scheme is actually implemented. For instance, in hash-based commitment schemes m^* would be the message that S committed to in the commit stage.

The ϵ-hiding property of commit-and-reveal schemes in the context of sealed-bid auctions guarantee that once a bid is committed, any adversary can only uncover the bid with negligible probability during the commit stage. This guarantees that the *bids are hidden from others during the bidding period.*

Definition 3 (ϵ-binding). *Let $\Pi = (\mathsf{Commit}, \mathsf{Reveal})$ be a commit-and-reveal scheme. We say Π is ϵ-cryptographic binding if for all PPT S, S succeeds in the following game with honest R with negligible probability.*

- *S produces a commitment c during the commit stage of Π*
- *S outputs two distinct messages m_0, m_1 such that for both $b = 0$ and $b = 1$, R on input (c, r_b, m^*) accepts and outputs m_b.*

The ϵ-binding property of commit-and-reveal schemes in the context of sealed-bid auctions guarantee that once a bid is committed, the commitment can only open to two different messages with negligible probability. This ensures that *committed bids cannot be modified.*

Definition 4 (τ-revealing). *Let $\tau > 0$ be some time parameter, S, R PPT, and $\Pi = (\mathsf{Commit}, \mathsf{Reveal})$ be a commit-and-reveal scheme. We say Π is τ-revealing if for all $m \in \mathcal{M}$, S and honest R running $\mathsf{Reveal}(\mathsf{Commit}(m, r), r, m^*)$ results in R accepting and outputting m in time at most τ.*

We are interested in commit-and-reveal schemes that satisfy the τ-revealing property for finite τ. This ensures that all committed bids will be revealed during the revealing period, and guarantees that *participants cannot selectively and indefinitely withhold bids.*

A.2 NM-CPA security

```
Init
$k \xleftarrow{\$} \mathcal{K}; b \xleftarrow{\$} \{0,1\}; S \leftarrow \emptyset$

LR($m_0, m_1$)
if pdec then $C \leftarrow \perp$
else $C \xleftarrow{\$} Enc(k, m_b); S \leftarrow S \cup \{C\}$
return $C$

Enc($m$)
$C \xleftarrow{\$} Enc(k, m)$
return $C$
```

```
Dec*($C^*$)
pdec $\leftarrow$ true
for $i \in [|C^*|]$:
if $C^*[i] \in S$ then $M^*[i] \leftarrow \perp$
else $M^*[i] \leftarrow Dec(k, C^*[i])$
return $M^*$

Finalise($d$)
Return $(d = b)$
```

Fig. 3. NM-CPA security game $\Gamma^{\text{nm-cpa}}$ for symmetric key encryption schemes

We use the definition of NM-CPA security for symmetric encryption schemes from [2], which we will reiterate here for completeness. NM-CPA security is defined via the following game $\Gamma^{\text{nm-cpa}}$(refer to Figure 3). The game begins with the initialization step that chooses a symmetric encryption key k randomly from some keyspace $\mathcal{K}$, as well as a fixed random bit b. The adversary is allowed access to a special left-or-right (LR) encryption oracle that takes in a pair of messages (m_0, m_1) and always encrypts the left (m_0) or the right (m_1) message depending on the choice of b [1]. In parallel, the adversary also has access to a one-time-use decryption oracle Dec^* which takes as input a vector of ciphertexts and outputs the plaintext decryption of each ciphertext. After querying Dec^*, the adversary does not have access anymore to the LR oracle; however, it continues to have access to the plaintext encryption oracle Enc. Finally, the adversary outputs a guess bit d, and we say the adversary wins the game (output of game is 1) if $d = b$. For any adversary $\mathcal{A}$, we define the advantage of the adversary to be the following: $Adv^{\text{nm-cpa}}(\mathcal{A}) = 2 \cdot \mathbb{P}[\Gamma^{\text{nm-cpa}} \text{ outputs } 1] - 1$.

NM-CPA security is shown by comparing the advantage of the adversary in the NM-CPA security game to an adversary with the same resources in another game and then using a known security reduction to show NM-CPA security. We omit the details in this write-up for clarity of exposition, but we refer the interested reader to [2] for a thorough overview.