Blockchain: Research and Applications 4 (2023) 100133

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

Research Article

The cost of privacy on blockchain: A study on sealed-bid auctions

Menelaos Kokaras[a], Magda Foti[b,*]

[a] Department of Electrical and Computer Engineering, University of Thessaly, Volos, 38221, Greece
[b] UBITECH Research Department, UBITECH Ltd, 15231, Athens, Greece

A R T I C L E I N F O

Keywords:
Privacy
Anonymity
Blockchain
Distributed ledger technology
Auction

A B S T R A C T

In order to preserve privacy in a blockchain ecosystem, the main objective is to keep a transaction's data private, such as the sender, the receiver, and the amount transferred. The current work studies the cryptographic tools commonly used to achieve this type of privacy, primarily focusing on the Ethereum blockchain. Such tools usually require many computational and storage resources, leading to additional fees. An anonymous auction protocol was developed as a case study to explore these costs, where hiding the identity and the amount of the bids utilizes a variety of cryptographic primitives. The proposed implementation was compared against three sealed-bid auction protocols, which utilize similar cryptographic tools for preserving privacy throughout the auction process. The results show that providing an additional level of anonymity, such as hiding someone's identity, can increase the gas cost significantly, up to 2.5 times, depending on the choice of the cryptographic tools, which determine the usage of the blockchain's storage and computational resources. By adjusting the level of decentralization on the application level by moving some operations off-chain and maintaining the role of the auctioneer, we show that we can maintain anonymity while reducing the gas cost by 40%.

1. Introduction

On a peer-to-peer e-cash system like Bitcoin [1], public keys ensure the pseudonymity of its users by mapping a user with a unique address. However, pseudonymity does not imply anonymity. The critical difference is that on a pseudonymous blockchain, an adversary might be able to link transactions made by an address to a real-world identity [2].

There are various mechanisms and blockchains to preserve the privacy of its users, such as ZCash [3] and Monero [4], utilizing cryptographic techniques like zero-knowledge proofs (ZKPs) and ring signatures to shield transactions. Even though e-cash systems like Zcash and Monero provide a satisfactory amount of privacy, Ethereum's leading cause is not to act as a currency but as a machine that executes programs called smart contracts. In Ethereum, cryptographic tools can be integrated on a smart contract level to increase privacy. Interacting with smart contracts requires a fee, and computationally demanding operations like cryptographic operations increase this fee significantly. Nevertheless, Ethereum is natively pseudonymous, and specific information is available for every transaction, including the sender's address, the receiver's address, and the amount transferred. Hiding this information is quite challenging, requiring a variety of cryptographic tools.

The current work serves as a benchmark for frequently used cryptographic tools and encryption schemes on the Ethereum blockchain. A payment mechanism called Anon-Zether [5] is extended and utilized, offering anonymous and confidential transactions in Ethereum. Through Anon-Zether, an anonymous auction protocol is implemented, which provides bidder anonymity inside a set, meaning that an observer can only distinguish that a bidder is participating in the auction and keeps the bids hidden at all times. The proposed scheme is compared with similar auction protocols, which use different approaches, and their cost is measured, providing a metric for similar decentralized applications. Also, to enable the verification of the auction procedure, we propose a process to make the auction verifiable using ZKPs.

The rest of the paper is organized as follows. Section 2 presents related work and existing tools related to anonymity on the Ethereum blockchain. Section 3 sets the required background. Section 4 discusses in detail the anonymous auction protocol and a process to ensure its fairness. Section 5 describes a procedure that enables the public to verify the auction process using ZKPs. Section 6 presents the cost of the auction process compared to other auction protocols, specifying the cryptographic mechanisms used in each. Finally, Section 7 concludes this work and discusses future research and development steps.

* Corresponding author.
E-mail addresses: mkokaras@uth.gr (M. Kokaras), mfoti@ubitech.eu (M. Foti).

https://doi.org/10.1016/j.bcra.2023.100133
Received 24 June 2022; Received in revised form 5 February 2023; Accepted 10 February 2023

2. State of the art

2.1. Blockchain auctions

The immutability of smart contracts and the integrity and transparency of blockchain ecosystems inspired a new direction in financial systems such as auctions or banking services based on blockchain. For instance, e-auctions, a well-researched topic in Ethereum, have a form of smart contract hosted on blockchain, and the process is based on traditional auction schemes. The most common auction schemes are

- First-price sealed-bid auctions. Bidders submit their encrypted bids to the auctioneer, who is responsible for announcing the winning bidder after the auction is finished.
- Second-price sealed-bid auctions. These auctions are also known as Vickrey Auctions, and the process is similar to first-price auctions, except that the winner pays the second-highest bid.
- English Auctions. This type is the most famous, although it is not the fairest. The bidders submit their bids openly and can increase the amount if another bidder offers more. This particular auction scheme is beneficial to the auctioneer because the bidders compete with each other and might raise their bids to win the auction.
- Dutch Auction. The auctioneer sets a starting price, decreasing it gradually until someone is willing to pay this amount.

The most common auction protocol on Ethereum is the sealed-bid auction. The main advantage is that a bidder offers the price he thinks resembles the actual value of the good. Sealing a bid can be implemented with various cryptographic schemes, such as commitments that resemble a real-life envelope. In this work, the proposed auction protocol, which is sealed-bid, is compared with other sealed-bid protocols.

Galal and Youssef [6] presented a smart contract for a first-price sealed-bid auction on Ethereum. This particular auction protocol contains five different phases. The first phase initiates the smart contract and its parameters, such as the public key of the auctioneer, the maximum number of bidders, the time intervals for the auction process, and the amount the bidders and the auctioneer must submit to have a monetary incentive to treat honestly. In the second phase, the bids are submitted as a commitment using Pedersen commitments [7], and the bidders submit the initial amount, which is deducted by the auction process in case of malicious action. In the third phase, bidders send the encrypted opening of the commitments to the auctioneer. This way, only the auctioneer is aware of the winning bidder. However, this raises a particular concern: if the auctioneer is the only one capable of determining the winner, how are the participants or other observers sure about the fairness of the auction process? A fair auction process on blockchain is often called verifiable. To make an auction verifiable, the auctioneer and the bidders must confirm the smart contract that they followed the process correctly without revealing the bid. This verification occurs in phase four, where the auctioneer must verify the correctness of all the submitted ciphertexts using ZKPs, and he also announces the winner without opening the bids. In phase five, the auction is terminated, the auctioneer receives the highest bid, and the honest bidders get a refund equal to the initial deposit. There are also additional scenarios where a bidder or the auctioneer might cheat, described in detail in their paper.

In Ref. [8], a sealed-bid auction protocol is achieved using commitment schemes and ring signatures. A ring is a set of possible signers, each one owning a public and a secret key. An outside observer, i.e., a verifier, is only sure that a ring member signed something, but it cannot tell who. This protocol works slightly the same as Galal and Youssef [6] in terms of phases and utilizes ZKPs to ensure the blockchain's fairness. Also, the participants submit a small number of cryptocurrencies to the smart contract, motivating them to follow the auction steps correctly. In the bidding phase, the bidders submit ring-signed commitments [9] of their bids and their identities to the auctioneer. On bid opening, the bidders reveal their bids to the auctioneer only using Public Key Encryption (PKE). The auctioneer declares the winner without knowing the bidders' identity, and finally, the winner reveals himself to the auctioneer so the auction process can be finalized.

In Ref. [10], the sealed-bid auction protocol operates without the need for an auctioneer, and the smart contract is responsible for the auction process. The auctioneer is converted into the “owner of the goods”, thus having no authority over the bidders. In the first phase, the participants register themselves to the smart contract getting the appropriate data used throughout the auction process. In the “publish” phase, the owner of the goods sends information about the goods, the reserve price, and the auction starting and ending time. Also, similar to the other auction protocols, the bidders and the owner of the goods submit an initial number of cryptocurrencies to have a monetary incentive to treat honestly. On “bid”, the smart contract verifies the bidders, the bidder provides the encrypted bids and generates Pedersen Commitments of the bids. On opening, the smart contract sorts the bids, checks the commitments, and sends the committed bids back to the bidders. On verification, the bidders use Bulletproofs to prove that the winning price is higher than their bid. On “finish”, the participants are refunded, and the winning bidder pays the rest of the amount to the owner of the goods.

All of the above-mentioned studies, together with other recently published articles [11,12], are suitable for anonymous auctions, but in practice, they rely on the assumption that there exists a blockchain providing anonymity and confidentiality on the transaction level, or they rely on trusted third parties or consortium blockchains [13].

2.2. Zokrates

Zokrates [14] is a set of tools utilized to create and integrate ZKPs in Ethereum, specifically zk-SNARKs, which is a family of ZKPs widely used in Ethereum. Zokrates consists of a high-level language where a user specifies a program that proves knowledge of a statement without revealing the witness. Then, two keys are generated: a verifier key and a prover key. The verifier key is used to generate a smart contract in Solidity, which is capable of verifying the proofs generated. The prover key is used to generate proof of the statement. The Zokrates toolbox consists of a compiler responsible for flattening the code, i.e., generating a list of variable definitions in an efficient form for future steps. The flattened code requires a witness, which is found using the witness generator. The witness on ZKP is the value someone wants to prove knowledge without revealing it. Then, using the program and its public arguments, the proving and verification keys are generated, which can be used for proof creation and verification using the libsnark library [15]. After a witness has been found, the proving key can be utilized to generate short proofs. These proofs can be verified on the smart contract level using a particular verification function generated using the previously generated verification key. The keys can be used an arbitrary number of times.

2.3. Aztec

The Aztec protocol is a protocol for fully confidential transactions on Ethereum, implying that the transaction amount is kept hidden.

The Aztec protocol tries to break free from the account-based system and introduce confidential UTXOs [16] to Ethereum, where a UTXO can be considered a banknote. The main intuition behind this is that cash is natively more secure than a public account system, where the state of an account is updated on each transaction. When bank notes change ownership, the only ones aware are the two parties.

A simple transfer in Aztec occurs by destroying notes and creating new ones. The sum of the values of the notes destroyed is equal to the sum of the values of the notes created, and one of the notes created has a value equal to the amount the sender wants to transfer to the receiver. This transaction is published encrypted under the receiver's public key. Of course, the values of the notes are not publicly revealed.

This simple transaction sparks problems, such as double spending or even generating notes with a much higher value than the ones destroyed. This is solved by utilizing ZKPs to prove the equality between the destroyed and newly generated notes without revealing the actual values.

Aztec also utilizes Merkle Trees [17], one to store all the generated notes, called note-tree, and one for the destroyed notes, called nullifier-tree. Owning a note indicates that the note exists on the note-tree but not on the nullifier-tree. Proving ownership also requires a user to submit a ZKP, which proves that inside Aztec's note-tree, there is a note with a particular value owned by this user.

2.4. Tornado cash

Tornado Cash is a protocol built in Ethereum that improves transaction privacy by breaking the on-chain link between a sender and a recipient. For instance, a user can deposit some funds in Tornado Cash and withdraw them to another account, with Tornado Cash as a middleman, acting similarly to blockchain mixers. This protocol also utilizes zk-SNARKs to hide sensitive information while proving various statements.

To use Tornado Cash, the first step is to deposit some funds. The user generates a secret random note automatically via Tornado Cash, and its hash is submitted to the Tornado Cash smart contract along with the amount he wishes to deposit. If the deposit is accepted, the contract stores it in a Merkle Tree, similar to Aztec. These deposits have a particular value because otherwise, an outside observer could determine the ownership of the "withdrawing" account by tracking the amount. The intuition is to generate an anonymity set, and the Tornado Cash contract holds the amounts deposited.

To withdraw, a user should prove to Tornado Cash that he owns a secret note to an unspent hash from the list of deposits. This is where zk-SNARKs come in handy, proving that an exact deposit matches the secret note without revealing which deposit. To prevent double-spend attacks, the user must provide a nullifier so he will not be able to use this secret again. This nullifier is a unique ID connected with the hash provided in the deposit. However, there is no way to identify which nullifier is assigned to which deposit hash.

2.5. Anon-Zether

The main paper of Zether [18] described Zether as a smart contract used either individually or from other smart contracts to exchange confidential amounts of an ERC20 Token called ZTH. Another version of Zether called Anon-Zether [5] contains a few changes to the implementation and enables anonymous transactions. On Anon-Zether, anonymous and confidential transactions are achieved thanks to ZKPs. Also, a new mechanism is proposed called $\Sigma-$*Bullets*, which enhances the interoperability of $\Sigma-$*protocols* [19] and Bulletproofs [20]. Using this mechanism, Zether can efficiently combine Bulletproofs-based range proofs with ElGamal encryptions, and more information is provided in Section 3. In brief, Zether keeps accounts inside its storage and the balances corresponding to the accounts. Accounts are stored as ElGamal public keys, and the balances are encrypted based on those keys. If a user wants to make a transaction to transfer ZTH or withdraw ZTH, he needs to provide a ZKP, and after the verification returns true, the transaction proceeds.

3. Background

3.1. Zero-knowledge proofs

Addressing ZKPs and other cryptographic tools requires knowledge of a few algebraic terms used throughout this paper. The description below is not strictly mathematical but aims to provide the fundamentals for understanding the notations used.

- A cyclic group G is called a finite cyclic group and is equal to $\{1, g^1, g^2 \ldots g^{p-1}\}$. The value g is called the generator point, and p is called the order of the group and can be considered a large prime number [21].
- The arithmetic inside this group is *modulo p* for possible wrap-around. For example, $g^i \times g^j = g^{(i+j)}$ *modulo p*.
- Z_p denotes the integers *modulo p* (its range is $[1, p-1]$). Z_p^* denotes the set of integers that have a multiplicative inverse *modulo p*, implying that $gcd(i, p) = 1$ for every integer i on the Z_p^*. We use $x \leftarrow \$S$ to denote that x is sampled uniformly at random from a set S. Throughout this work, when a random number generation is mentioned, it is drawn from these types of groups, and g corresponds to a generator point of a cyclic group.
- The decisional Diffie-Hellman (DDH) [22] assumption holds in G. The DDH states that a tuple $(g, g^a, g^b, g^{a \cdot b})$ is computationally indistinguishable from (g, g^a, g^b, g^c), where a, b, c are random. It also implies the discrete logarithm assumption, so for a value $y = g^r$, where $r \leftarrow \$ Z_p^*$ by knowing y and g, someone cannot determine the value r. In this work, the ElGamal key pair is a (x, y) tuple, where $y = g^x$ and x is a random integer drawn from Z_p^*, and by knowing (y, g) someone cannot determine the private key x.

In cryptography, a ZKP allows a prover to convince a verifier that a statement is true while concealing certain information about this statement. For example, as described in Ref. [19], a prover convinces a verifier that he knows a witness w such that $h = g^w$. The protocol suggested works like that.

- The prover generates a random number r and sends $a = g^r$ to the verifier.
- The verifier generates a random challenge e and sends it back to the prover.
- The prover creates a response $z = r + ew$. Note that the secret w, the random number r, and the challenge e are not visible.
- The verifier checks if $g^z = ah^e$. This equation will hold because $g^z = g^{r+ew} = g^r \times g^{ew} = a \times (g^w)^e = a \times h^e$.

The protocol described above is a $\Sigma-$*Protocol*. Both the transfer and burn proofs described in Section 2.5 as well as the balance proof described in Section 4.3 utilize these protocols. The above example is interactive, which means that the prover and the verifier must be present. To make the protocol non-interactive, the Fiat-Shamir heuristic [23] can be used to replace the challenge sent by the verifier with a hash function, allowing the prover to produce a challenge that he cannot predict. All the $\Sigma-$*Protocols* used in this work are non-interactive.

3.1.1. Range proofs

Range proofs [24] are ZKPs that can convince a verifier that a value v belongs to an interval without disclosing the value. In brief, the prover can prove that $0 \le v \le 2^n$. For this equation to hold, v must be represented in n bits. Also, the prover must prove that the bits are 0 or 1. Bulletproofs [20] is a ZKP protocol that is well-suited for efficient range proofs, improving performance in terms of proof size and verification time. However, it also allows proving other statements than range proofs.

3.1.2. Applications in blockchain

ZKP is a well-researched topic [25] on blockchain. The first widespread application of ZKP was ZCash, and by utilizing a ZKP model called zk-SNARKs, ZCash managed to implement anonymous transactions. Other notable mechanisms are Tornado Cash [26], which improves transaction privacy by breaking the on-chain link between the sender and the recipient; Aztec [27], which implements confidential transactions on Ethereum; and Zether, which implements confidential or anonymous transactions on Ethereum. ZKP is a trending topic in Ethereum, where zk-Rollups [28] are used to develop scalability solutions. Zk-Rollups bundle transactions off-chain into a single ZKP and publish it on the

Ethereum blockchain, thus reducing fees per user transfer. Instead, for example, when mining hundreds of transactions, they get bundled or "rolled up" into a single one of them. These transactions were executed off-chain, often called Layer 2 (L2) transactions.

3.2. Elliptic curves

Elliptic Curve Cryptography (ECC) [29] is a form of cryptography utilized in a variety of applications [30]. In this work, elliptic curves are used to construct the ElGamal ciphertexts as described in Section 3.5, and an encryption scheme based on ECC the Elliptic Curve Integrated Encryption Scheme (ECIES) [31] provides data encryption. In Bitcoin and Ethereum, an ECC-based algorithm called the Elliptic Curve Digital Signature Algorithm (ECDSA) is responsible for the digital signatures used in signing transactions [32]. An elliptic curve is a set of points that satisfy an equation of the form $y^2 = x^3 + ax + b$. What makes the elliptic curve suitable for cryptography is its properties. Note that elliptic curves similar to other forms of cryptography "squeeze" themselves to a fixed range by making all operations *modulo q*, where q is a big prime number. Elliptic curve properties are: 1) Elliptic curves are symmetric on the x-axis. 2) Every non-vertical line meets the curve at three or fewer points. Understanding ECC requires understanding elliptic curve addition and multiplication. To visually add a point in an elliptic curve with another point, draw a line between them. This line will also intersect the curve to one more point. The result of the addition is the exact point on the opposite side of the x-axis. A graphical representation of the elliptic curve addition is provided in Fig. 1.

For the multiplication operation, multiplying a number n with a point A yields $B = nA$, which is equal to n additions. So, to reach point B, the addition operation happened n times. So, by knowing only B and A, someone cannot distinguish how many times the addition operation occurred, keeping the value n encrypted.

3.3. RSA encryption

RSA encryption [33] is a public key encryption scheme where a message is encrypted under a public key, and the only way to decrypt this message is by knowing the private key. The RSA relies on the fact that given a number n, a product of two large prime numbers, it is very hard to find those two prime numbers. However, knowing one of the large prime numbers makes it relatively easy to find the other prime number. The procedure of generating a public key and a private key, encrypting a message, and decrypting it works as follows.

- Pick two prime integers p and q. These integers, for security purposes, are very large and must not be too close to each other. These prime numbers can be found using a primality test, such as the Rabin-Miller primality test [34,35]. The prime numbers are also kept secret.
- Compute the modulus $n = pq$.
- Compute $\lambda(n)$, where λ corresponds to Carmichael's totient function [36]. The output is the lowest common multiple of the prime numbers p and q. The $\lambda(n)$ is kept hidden.
- Generating a public key also requires specifying a prime number e which is in the range $[1, \lambda(n)]$. The numbers n and e are published as a part of the public key. Encrypting a number m produces a ciphertext $c = m^e \text{mod}(n)$.
- To decrypt the ciphertext and fetch the message, the private key corresponding to the public key is required. The private key is specified as $d = e^{-1}\text{mod}(\lambda(n))$. For decryption, the following formula is used: $m = c^d\text{mod}(n)$, where m is the encrypted message.

Encrypting messages other than numbers, like strings, requires converting the message into a potentially very large number, resulting in a bigger ciphertext. In this work, another PKE is used to encrypt the messages. However, additional measurements can be done to determine if the current scheme is more efficient than RSA, especially in the verification phase, where the encryption scheme must be implemented on-chain for public key encryption.

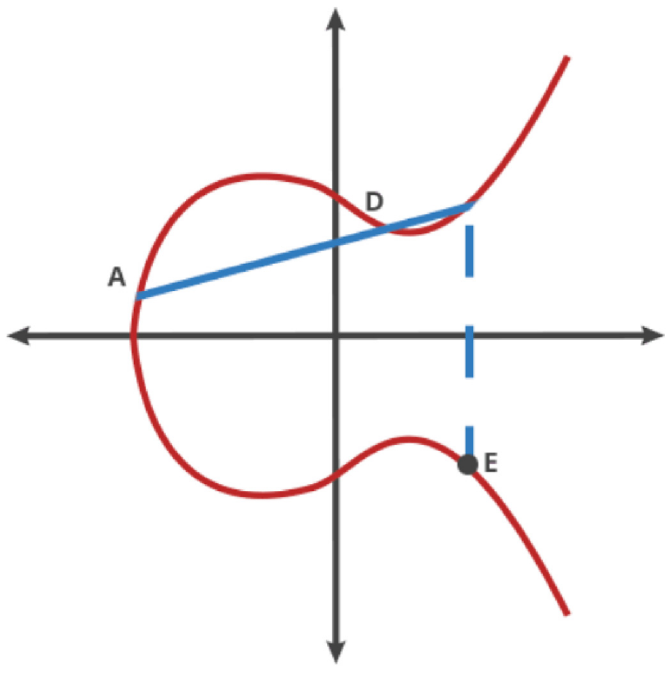


Fig. 1. Elliptic curve addition.

3.4. ElGamal encryption

For someone to understand the mechanisms of Zether is crucial to understand how ElGamal encryption works [37]. ElGamal encryption is a secure public key encryption scheme. A random number, x, acts like a private key for a user, and $y = g^x$ acts like a public key, where g is a generator point. Anon-Zether uses a variation of ElGamal encryption using elliptic curves, as described in detail in Section 3.5. To encrypt a balance let's say b.

- Find a random number r.
- Calculate y^r.
- Calculate g^r.
- Publish ciphertext $(g^b y^r, g^r)$.

Decrypting these ciphertexts requires holding the private key x that corresponds to y. To decrypt

- Calculate $(g^r)^x = (g^x)^y = y^r$.
- Now divide $g^b y^r$ with y^r and get the encrypted value g^b. The fact that the value b is in a certain range, in Anon-Zether's case [0, 4294967295], allows us to iterate through this range and find the value b from g^b.

Anon-Zether deals with accounts and accounts have balances that are encrypted in the form of $(C_L = g^b y^r, C_R = g^r)$. In a transaction, the balance of a user is updated. Let's say a user Alice wants to send some funds b' to Bob. The value b' needs to be encrypted under Bob's public key y. Therefore, based on the ElGamal encryption she publishes $(C_L{}' = g^{b'} y^{r'}, C_R{}' = g^{r'})$. Thanks to the additive homomorphic property of ElGamal, encrypted values can be added together without decryption by multiplying the ciphertexts: $(C_L C_L{}', C_R C_R{}') = (g^{b+b'} y^{r+r'}, g^{r+r'})$.

3.5. Implementation of ElGamal using elliptic curves

Anon-Zether utilized elliptic curves because of the existence of pre-compiled contracts [38] for elliptic curve operations, which make the operations cheaper in terms of gas. The differences from traditional ElGamal [39] are

1. Instead of $y = g^x$ as the public key is now $Y = xG$, where G is the generator point.
2. Encryption for a value b:
   - Find a random number r.
   - Calculate number rY instead of y^r.
   - Calculate number rG instead of g^r.
   - Publish ciphertext $(rY + bG, rG)$.
3. Decryption:
   - Calculate $x(rG) = r\,(xG) = rY$.
   - Now subtract $rY + bG - rY = bG$, which results in the encrypted data.
4. Additive Homomorphism:

   For the traditional ElGamal:

$$(C_L C_L', C_R C_R') = (g^{b+b'} y^{r+r'}, g^{r+r'})$$

With elliptic curves this property can be expressed as:

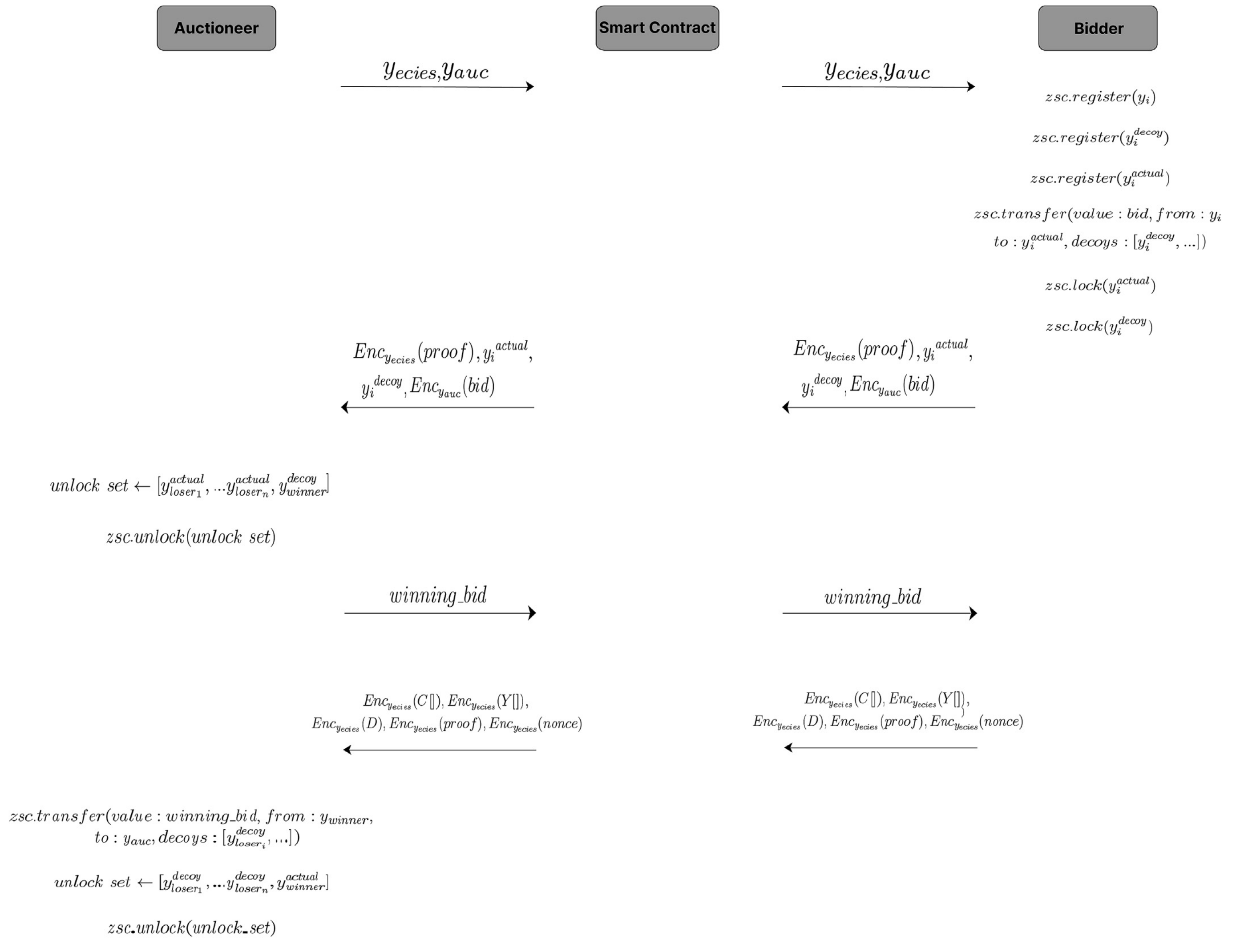
$$(C_L C_L', C_R C_R') = (rY + bG + r'Y + b'G,\ rG + r'G)$$
$$= ((r + r')Y + (b + b')G,\ (r + r')G).$$

4. Design

Based on Anon-Zether [5], this work proposes a sealed-bid and anonymous auction protocol. This protocol is a smart contract called AUC, responsible for handling the auction process and its various phases, including bid submission, winning bid declaration, and transferring funds. Anonymous Zether, a fundamental part of the protocol, provides anonymous transfers, and the lock/unlock functionality is suitable for sealed-bid auctions as proposed in Zether. Because the auction protocol is based on Anon-Zether, it must follow particular rules and mechanisms like epochs and have the necessary interval between function calls.

Through this protocol, bidders, which are externally controlled accounts, remain anonymous within a set despite the auction's outcome. The auctioneer and the bidders inside the set are the only entities aware of the winner's identity.

Fig. 2 provides an illustration of the overall process, giving details on the specific calls on the protocol functions by the involved actors and their time sequence.



Auctioneer
Smart Contract
Bidder
y_{ecies}, y_{auc}
y_{ecies}, y_{auc}
$zsc.register(y_i)$
$zsc.register(y_i^{decoy})$
$zsc.register(y_i^{actual})$
$zsc.transfer(value : bid, from : y_i$
$to : y_i^{actual}, decoys : [y_i^{decoy}, ...])$
$zsc.lock(y_i^{actual})$
$zsc.lock(y_i^{decoy})$
$Enc_{y_{ecies}}(proof), y_i^{actual},$
$y_i^{decoy}, Enc_{y_{auc}}(bid)$
$Enc_{y_{ecies}}(proof), y_i^{actual},$
$y_i^{decoy}, Enc_{y_{auc}}(bid)$
$unlock\ set \leftarrow [y_{loser_1}^{actual}, ...y_{loser_n}^{actual}, y_{winner}^{decoy}]$
$zsc.unlock(unlock\ set)$
$winning\_bid$
$winning\_bid$
$Enc_{y_{ecies}}(C[]), Enc_{y_{ecies}}(Y[]),$
$Enc_{y_{ecies}}(D), Enc_{y_{ecies}}(proof), Enc_{y_{ecies}}(nonce)$
$Enc_{y_{ecies}}(C[]), Enc_{y_{ecies}}(Y[]),$
$Enc_{y_{ecies}}(D), Enc_{y_{ecies}}(proof), Enc_{y_{ecies}}(nonce)$
$zsc.transfer(value : winning\_bid, from : y_{winner},$
$to : y_{auc}, decoys : [y_{loser_i}^{decoy}, ...])$
$unlock\ set \leftarrow [y_{loser_1}^{decoy}, ...y_{loser_n}^{decoy}, y_{winner}^{actual}]$
$zsc.unlock(unlock\_set)$

Fig. 2. Auction process.

4.1. Lock/unlock

In the Zether paper [18], a sealed-bid auction protocol is presented using the lock/unlock mechanism proposed in the same paper. The current work extends this protocol by providing an extra step of privacy by hiding the bidders inside a set and keeping their bids encrypted at all times. However, the lock/unlock mechanism has not been implemented in the Anon-Zether [40]. Zether provides the pseudocode for these mechanisms guiding the integration of Anon-Zether. Anon-Zether contains an account state and a pending state. Locking accounts require two similar states: a locked state to indicate if an account is locked and a pending locked state, which is a mechanism to prevent front-running. Registering an account initializes the locked and pending locked states to “unlocked”. To lock an account, the user provides an address (can be either an EOA or a contract) to which he hands over access, and the locked state changes. After locking, the user has no right to operate this account, unlike the address where he locked the account, which can be considered the “master”. Of course, the “master” of the account cannot withdraw or transfer funds because he needs the appropriate ZKP, which requires knowledge of the private key of the corresponding account. The master can only initiate the unlock function. RollOver is also used in those operations. The lock function requires a signature to verify the ownership of the account, which can be implemented with a Schnorr Signature [41]. The variable *addr* is the address of the master, y is the prover's ElGamal public key, and *epoch* is the current Anon-Zether's epoch. The equation $c^* == c$ will hold because $t^* = g^s y^{-c} = g^{r+cx} y^{-c} = (g^x)^c g^r y^{-c} = y^c y^{-c} t = t$.

$$
\begin{array}{ccc}
\underline{\textbf{Prover}} & & \underline{\textbf{Verifier}} \\
r \leftarrow \$Z_q & & \\
t = g^r & & \\
txt = addr \mid y \mid t \mid epoch & & \\
c = H(txt) & & \\
s = r + cx & & \\
 & \xrightarrow{(c,s)} & \\
 & & t^* = g^s y^{-c} \\
 & & txt^* = address \mid y \mid t^* \mid epoch \\
 & & c^* = H(txt^*) \\
 & & \text{Check } c^* == c
\end{array}
$$

4.2. Auctioneer registration

The auctioneer will be the one to deploy the auction contract. He initially registers in the already deployed Anon-Zether via the auction contract AUC, resulting in an Anon-Zether account and an ElGamal key pair for this account. Then, he generates an ECIES keypair that is going to be used by the bidders to encrypt their data. Both the ECIES (y_{ecies}) and ElGamal (y_{auc}) public keys will be published to the blockchain by the auctioneer.

4.3. Bid preparation and submission

Each bidder has to register in Anon-Zether. The lock function is utilized to submit a bid, which locks an account with a bid in an external smart contract, in this case, AUC. Then, he creates two more accounts, one for his actual bid and the other for his decoy bid, which will be zero. An outside observer cannot distinguish which one is who. He then transfers the amount he wishes to bid into his actual account and puts the decoy account in the anon-set in the Anon-Zether transfer function. That way, someone cannot tell how many ZTHs were sent and who received them, making it unclear if the holder of the funds is now the decoy or the actual account. The bidder then locks the accounts to the AUC by using its address. He also fetches the auctioneer's public keys from the blockchain to encrypt the data required throughout the auction. By locking the accounts, the bids remain sealed. Then he discloses his bid to the auctioneer by storing the encrypted balance proofs and data required for verification in the smart contract, ensuring the auctioneer that his account contains the amount he claims. The balance proof must correspond to the current state of the account. Ensuring the auctioneer about the correct balance can be achieved with a simple $\Sigma-$*Protocol* proving that an account has a specific balance. The protocol is defined as follows:

$$
\begin{array}{ccc}
\underline{\textbf{Prover}} & & \underline{\textbf{Verifier}} \\
r \leftarrow \$Zq & & \\
A_{C_R} = {C_R}^r & & \\
 & \xrightarrow{A_{C_R}} & \\
 & & c \leftarrow \$Zq \\
 & \xleftarrow{c} & \\
s = r + cx & & \\
 & \xrightarrow{s} & \\
 & & {C_R}^r = A_{C_R}\left(\frac{C_L}{g^b}\right)^c
\end{array}
$$

This interactive $\Sigma-$*Protocol* is converted into non-interactive via the Fiat-Shamir heuristic. The bidder generates a random challenge c by hashing his public key and A_{C_R}. Then, he submits $c|s$ as the proof. For verification, the auctioneer needs to reconstruct A_{C_R}. Then, he hashes A_{C_R} and the bidder's public key resulting in a new challenge c'. If $c' == c$, the proof is valid, and the auctioneer is sure that the account contains the balance g^b, which is submitted encrypted under the auctioneer's public key y_{auc}.

4.4. Bid comparison and winner declaration

Once the specified time has passed and the auction is closed, the auctioneer will fetch the data and the balance proofs published by the bidders. Then, he decrypts the data and determines if the statement about the account's balance is true or false using the off-chain balance proof verification algorithm. Because the bidders have provided two accounts, the balance proof should be verified correctly for only one of them, allowing the auctioneer to determine the actual and decoy accounts. As for the verified proofs, he simply compares the bids and identifies the winner. The auctioneer now unlocks the losers' actual accounts and the winner's decoy account. An outside observer cannot identify the winning bidder as he is unaware if an account is an actual or decoy account. He also stores the winning bid to the smart contract, letting the bidders know if they won the auction.

4.5. Transfer proof submission

Bidders must still participate in the auction after the outcome has been determined to preserve the winner's anonymity. The winning bidder must create a transfer proof for the amount he bid, which will be used to transfer ZTH from himself to the auctioneer. The losing bidders must also create transfer proofs. Otherwise, the winning bidder's identity would be revealed. Also, the anon-set used in transfer proofs must contain some decoy accounts. The transfer proofs and the data needed for the verification are then stored encrypted under the auctioneer's ECIES key to the smart contract.

4.6. Transfer earnings

The auctioneer fetches the transfer proofs and the data, decrypts them, and transfers his earnings to himself using the transfer function via an external call from the AUC contract. The external call is essential because the locked accounts cannot be operated from addresses other than the AUC. Finally, the auctioneer unlocks the remaining accounts.

5. Security analysis

In the previous section, we assumed that the participants are honest and behave according to the auction rules. In this section, we define scenarios on how a participant can act maliciously in the different phases of the auction and provide a preliminary discussion on how ZKP can be used to deal with these scenarios, making the protocol fully secure. The protocol then will also be verifiable, meaning that the participants and outside observers will be sure about the correctness of the auction process. A participant, auctioneer, or bidder can act maliciously in the following ways.

1. A malicious bidder can submit a false balance proof for the account he locked in the bidding process to the AUC contract or submit an unlocked account. On the other hand, a malicious auctioneer can claim that an honest bidder was dishonest even though he submitted valid proof.
2. A malicious auctioneer can declare a false winning bid or unlock false accounts to attack a bidder.
3. The losing bidders can submit false transfer arguments that reveal they are not the winning bidders, thus exposing the actual winner.
4. A malicious winning bidder can submit transfer proof for a smaller amount than he declared during the bidding process. A malicious auctioneer can claim that an honest winning bidder was dishonest despite submitting valid transfer proof and valid arguments or keep some of the remaining accounts locked.

To address these issues, the bidders and the auctioneers must have a reason to behave honestly. In our case (also in similar protocols), there is a monetary incentive by forcing them to lock some funds before the process. On Zether's auction protocol, this incentive was the locked account, which would remain locked if its owner was dishonest. However, this approach does not fit the proposed scheme. Also, each auction phase must have a pre-determined duration. In the following scenarios, when the auctioneer is punished, the AUC will keep his funds, the bidders will get a refund, and the auction will be terminated. When a bidder is punished, the AUC will keep his funds and remove him from the auction.

For the first scenario, a similar ZKP as the one proposed in Ref. [6] can be used. The protocol can force the auctioneer to verify the balance proofs' correctness once submitted using a ZKP. If a proof is invalid, the auctioneer declares that on the AUC. The honest bidder can deny this claim by revealing the proof and the balance. The AUC encrypts those values under the auctioneer's ECIES public key y_{ecies}. If the outcome ciphertext is found to be equivalent to the previously submitted ciphertext and the balance proof is valid, the auctioneer is proven malicious, and the AUC punishes him. In any other case, the bidder is malicious, and the AUC punishes him. Note that the bidder's accounts must also be locked to the AUC, which Anon-Zether's locked state can easily prove. If they are not locked to the AUC, the bidder is punished.

For the second scenario, the auctioneer has to prove that the *winning_bid* is indeed the maximum bid. By using Bulletproofs [20], he can prove that the accounts locked inside the AUC are in the range [0, *winning_bid*]. Also, he needs to prove the existence of an account inside the locked set with a balance equal to the *winning_bid*, which can be done by using one one-out-of-many proofs [42]. If the auctioneer decides to keep locked account(s) that contain losing bids, the bidders can reveal the balance of the decoy and the actual account to the AUC. The locked accounts of an honest, losing bidder should have a balance of zero and a balance of the declared bid. If the auctioneer keeps the actual account locked, the AUC will punish him. In any other scenario, the bidder is punished. This attack is not that severe because once the auction process is over, the AUC will release all the accounts.

For the third scenario, the same procedure is followed as the first one. The auctioneer needs to verify the correctness of the transfer proofs and arguments submitted. A similar ZKP used in *transfer* operation on Anon-Zether can be utilized to achieve this. This ZKP only matters for the anon-set where the winner will be hidden, much smaller than the number of bidders, reducing the gas fees.

For the final scenario, the auctioneer must reveal his balance before transferring and after transferring (or reveal the balance difference). If the balance difference is not equal to *winning_bid* an honest winning bidder must reveal the proof and the arguments. The AUC contract encrypts those values under the auctioneer's ECIES public key y_{ecies} and checks if they are equal to the previously submitted ciphertexts. In the case of equivalence, the AUC punishes the bidder. If they are not equal, the auctioneer has generated his transfer proof to attack the bidder, so the AUC punishes him. Also, the auctioneer must unlock the remaining accounts, which can be easily proven by the submitted accounts.

6. Results

In the current section, the measurements of the implemented auction protocol are presented compared to other implementations of auction protocols that aim to protect the privacy of their participants. These protocols use various mechanisms like commitment schemes [43], public key encryption, ElGamal encryption, elliptic curves, and ZKPs. All of them are very common in privacy-preserving protocols, and exploring their cost through auction protocols serves as a benchmark for more general protocols.

The gas cost was measured using the web3js [44] library on a ganache [45] private Ethereum blockchain. The measurements below are as of May 2022 with a gas price equal to 20 gwei and an Ethereum price equal to $2000 [46].

6.1. The framework

Table 1 contains the measurements for the Anon-Zether operations used in the anonymous auction as well as the cost of the deployment, which only happens once. For the transfer transaction, the anon-set size was 4. Note that the anon-set has to be a power of 2. Ganache has a default gas limit of 6721975. Measuring the gas cost for larger sets requires an increased gas limit. Otherwise, the gas cost will exceed the limit, and the transaction will fail. Also, Anon-Zether relies on epochs where an epoch duration has a predetermined value. Increasing the size of the anon-set increases the duration of the verification for the ZKP, which may result in the need to increase the epoch length. However, excessively large epochs will lead to large stalling between operations, affecting the auction protocol. The relation between the duration of the verification process and the anon-set size is presented in Fig. 3. The relation between the two variables is linear, and the duration of the verification process increases by 87 ms for every user that is added to the anon-set.

Table 2 contains the measurements for the proposed anonymous auction for three bidders. From these operations, only the *submitTransferArgs*, *unlockAccounts*, and *unlockRemainingAccounts* depend on the number of bidders. For each bidder, the *unlockAccounts* and *unlockRemainingAccounts* gas costs increase by 56k. The *submitTransferArgs* gas cost is proportional to the size of the anon-set, scaling linearly for bigger

Table 1
Gas cost for Anon-Zether operations.

Operation	Gas Cost	Gas(gwei)	in $
deployment	11700k	234000k	468
register	173k	3460k	6.92
deposit	220k	4400k	8.8
withdraw	2300k	46000k	92
transfer	5700k	114000k	228
lock	69k	1380k	2.76
unlock	56k	1120k	2.23

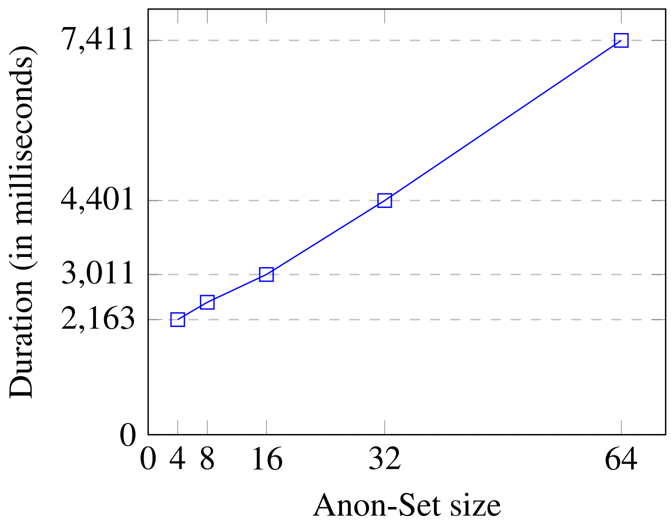


Fig. 3. Duration of the verification process.

Table 2
Gas cost for Auction operations.

Operation	Anon-Zether Operation	Gas Cost	Gas(gwei)	in $
deployment	–	2000k	40000k	80
submitKeys	–	125k	2500k	5
revealBids	–	417k	8340k	16.68
unlockAccounts	unlock	214k	4280k	8.56
transferEarnings	transfer	5800k	116000k	232
submitTransferArgs	–	5400k	108000k	216
unlockRemainingAccounts	unlock	145k	2900k	5.8

sets. The gas cost for *submitTransferArgs* is almost as high as the transfer transaction. The auctioneer needs to transfer the winning bid to himself, and bidders must provide the encrypted arguments of the *transferEarnings* operation. One of those arguments is the encrypted ciphertexts $C[]$ containing encrypted elliptic curve points, one for each bidder. Elliptic curve points in Solidity are two-dimensional points where each dimension is a bytes32 variable. The construction of these ciphertexts happens off-chain, requiring the generation of a random value r. Bidders can send r to the auctioneer encrypted, so he can construct these ciphertexts locally to reduce the gas cost from 5400k to 4050k for four members in the anon-set. This approach will also affect scaling because the amount of storage used inside the blockchain will be the same despite the number of bidders. However, it also raises security concerns and needs further investigation. Even though sending r reduces the gas cost, the data will be used in the *submitTransferArgs*, which is just an Anon-Zether *transfer* operation, where a bigger anon-set implies a bigger gas cost. The gas costs for various anon-set sizes were provided in the Anon-Zether paper. As it can be seen in Table 3, the gas cost increases with the anon-set size with a correlation coefficient of 0.99 and complexity $\mathcal{O}(n)$. The *RevealBids* function, as mentioned in a previous section, uses just a $\Sigma-$*Protocol* to prove that an account holds a certain amount of ZTH. Using burn proofs in the bidding process, as proposed in the Zether paper, costs 1800k gas (using Anon-Zether's burn proofs) in contrast with the proposed implementation, costing 417k.

Table 3
Gas cost for transfer function.

Anon-Set	Gas Cost	Gas(gwei)	in $
4	5700k	114000k	228
8	7300k	146000k	292
16	10900k	218000k	436
32	18700k	374000k	748
64	36000k	720000k	1440

6.2. The auction protocol

Each auction protocol contains various phases that might differ from each other, but the standard steps of each auction are: *Bid, Reveal Bid, Announce Winner*, and *Withdraw*. For the proposed scheme, these steps correspond to

- Bid: *lock* with a gas cost of 69k.
- Reveal Bid: *revealBids* with a gas cost of 417k.
- Announce Winner: *unlockAccounts* with a gas cost of 606k and submitting the winning bid.
- Withdraw: *transferEarnings* with a gas cost of 5800k.

Galal and Youssef [6] proposed a verifiable sealed-bid auction scheme. The bidding process requires storing a Pedersen commitment on the blockchain. Also, a public key encryption scheme is utilized to encrypt the outcome based on the auctioneer's public key. In Ref. [8], the bidders use ring signatures to sign commitments and their identities to the auctioneer, which are stored on-chain. A PKE is also used to encrypt the revealed bids. Li and Xue [10] and Galal and Youssef [6] utilized Pedersen commitments and PKE to seal the bids. Also, in Ref. [10], the need for an auctioneer is eliminated as the smart contract is responsible for the outcome of the auction process, which utilizes Bulletproofs to compare the bids without revealing the actual price. The auction scheme in Ref. [8], to preserve anonymity, relies on the assumption that Ethereum is confidential and anonymous. It also mentions Zether as a potential solution to preserve anonymity, increasing the gas cost significantly.

The scheme proposed also preserves the anonymity of the bidders inside a set by using Anon-Zether's operations in contrast to the other schemes where the bids only remain sealed. This implies that an outside observer can only determine the participants (without their bids). Even after the auction process is finished, the observer can only be sure that the winner belongs to a set of bidders.

Table 4 presents the gas costs for the three auction schemes for ten bidders, except the scheme of Li and Xue [10], which contains the average costs for a different number of bidders. Fig. 4 provides a visualization of these results. In Ref. [10], additional costs occur because the standard steps also contain ZKP verification, which requires elliptic curve operations. These verification costs are not mentioned and visualized in Table 4 and Fig. 4 for the other three schemes. For example, the two ZKP verifications require nearly 2600k gas in Ref. [6]. Also, the proposed implementation will incur additional costs because of the *TransferEarnings* function. The *Total* row is not a sum of the previous rows, but it contains the total cost of each auction protocol considering each protocol's unique phases.

Sharma's [8] *Bid* is the most expensive, scaling linearly for the number of bidders. This happens because the bid function takes as input a ring signature and the users that define this ring. In Ref. [6] the bid is stored as a Pedersen Commitment, which has a fixed size. In Ref. [10], the bidders

Table 4
Gas costs of the auction protocols.

Operation	Proposed Scheme	Galal and Youssef [6]	Sharma [8]	Li and Xue [10]
Deployment	2000k	3100k	1800k	–
Bid	69k	130k	1940k	2200k
Reveal Bid	417k	133k	3000k	6300k
Announce Winner	606k	166k	66k	–
Withdraw	5800k	47k	34k	4300k
Total	14000k	6000k	12000k	19000k

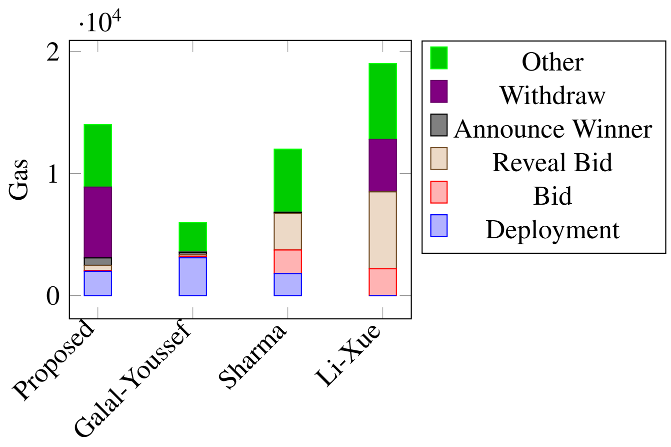


Fig. 4. Gas costs of the auction protocols.

provide encrypted bids and encrypted commitments. The proposed scheme uses Anon-Zether's lock functionality, which is cheap in terms of gas cost.

In the proposed scheme, the *Reveal bid* sends the encrypted balance to the auctioneer as an elliptic curve point and the encrypted proof (under the ECIES public key) for the ZKP. The output of the encrypted proof is a byte variable. Something similar occurs in Ref. [6], which reveals the Pedersen Commitment opening to the auctioneer but requires less storage. In Ref. [8], the commitment opening scales linearly to the number of bidders. In Ref. [10], revealing bids also determine the winner by utilizing Bulletproofs for bid comparison.

Announce Winner in the proposed scheme requires unlocking the bidders' actual accounts and the winner's decoy account, scaling linearly for more bidders. Sharma [8] and Galal and Youssef [6] stored the winning commitment on the blockchain.

Finally, *Withdraw* is much more costly in the proposed scheme because this is where anonymity is preserved by using the *transfer* function, which requires a lot of elliptic curve operations. In Refs [6,8], the contract refunds the initial deposits and the auctioneer receives the winning bid. In Ref. [10], this step is more complicated and leads to additional costs: the contract notifies the winning bidder to deposit the payment, the bidders receive the initial deposits, the owner of the goods

Table 5
Detailed costs of the implementations.

		Proposed scheme	Galal and Youssef [6]	Sharma [8]
Cryptographic Tools	Bid	–	Pedersen Commitments	Ring Sig, Commitment, PKE
	Reveal Bid	PKE, ZKP	PKE	Commitment, PKE
	Announce Winner	–	ECC	Commitment
	Withdraw	ZKP	–	–
ECC costs	Bid	–	–	–
	Reveal Bid	–	–	–
	Announce Winner	–	100k	–
	Withdraw	5000k	–	–
Storage Variables (bytes)	Bid	–	64	5k
	Reveal Bid	492	128	4.5k
	Announce Winner	–	–	32
	Withdraw	–	–	–

ECC: Elliptic Curve Cryptography, PKE: Public Key Encryption, ZKP: Zero-knowledge Proof.

Table 6
Comparison of the different protocols.

	Proposed	Galal and Youssef [6]	Sharma [8]	Li and Xue [10]
Auctioneer	✓	✓	✓	×
Anonymity	✓	×	×	×
Sealed Bids	✓	✓	✓	✓
Verification	✓	✓	✓	✓

receives the winning bid, and finally, the process is finished.

Table 5 summarizes the cryptographic tools used, the elliptic curve operations costs, and the encrypted storage variable size used on each phase of the three auction protocols. Note that the costs are not deterministic and may vary because of different storage variable lengths, especially in Sharma's [8] implementation, which uses ring signatures. Each protocol utilizes different tools and cryptographic schemes at each step. Elliptic curve operations are used in Pedersen Commitments ZKPs validation. Also, passing data between the auctioneer and the bidders require storing it inside the auction smart contracts. These data are mainly encrypted by using a PKE, which increases their size in bytes making their storage more expensive, and most of the time, storing these data is the most expensive task.

Apart from the differences in the computational cost between the presented works, there are also some fundamental similarities and differences between them. Table 6 compares the related protocols based on four important metrics for blockchain auction protocols: the existence of the auctioneer, the anonymity of the participants, the sealing of the bids through the bidding process, and the verification of the auction process from the blockchain. The only work that has removed the need for an auctioneer is that of Li and Xue [10], all the rest have a dedicated actor who acts as an auctioneer. The current work is the only one that does not rely on the assumption that the blockchain protocol provides anonymity on the transaction level, but by-design protects the sender and the receiver of each transaction, and this is why it is the only one to provide anonymity. Finally, all the works presented hide the bid information and provide a verification mechanism for results.

7. Conclusions and future work

The current work has explored and evaluated various cryptographic mechanisms that could potentially be used on Ethereum and other blockchain platforms. It also introduced an anonymous auction protocol based on Anon-Zether, preserving the anonymity of the bidders and their bids. The AUC smart contract, which handles the auction process, is implemented in Solidity. An auctioneer distinguishes the outcome of the auction. He, as well as the bidders, have a monetary incentive to treat honestly. Also, the comparison of the proposed scheme with other similar protocols provides a good metric for the cryptographic tools regarding gas cost and the level of privacy that can be achieved through them. To the best of our knowledge, the proposed protocol is the first auction protocol that provides anonymity on the P2P transaction exchange level, protecting not only the bid values but also the information related to the sender's identity. The set anonymity of bidders achieved in the proposed auction protocol uses additional blockchain resources, and in some cases, significantly increases the gas cost compared to other sealed-bid auction schemes proposed in the literature. However, the proposed protocol decreases the gas cost when compared with protocols that remove the need for an auctioneer.

Further indicative research and development plans include the following: (i) optimization of the AUC smart contract to minimize the gas cost, (ii) implementation of a web interface to improve the user experience when using the proposed distributed application, and (iii) implementation of the process proposed in Section 5 to make the auction publicly verifiable.

The AUC smart contract could possibly be optimized in terms of gas cost by using fewer storage variables or more efficient methods of storage. The transfer transaction cost can also be significantly reduced with a pre-compiled contract [38] that reduces the gas cost of elliptic curve operations. Also, the auction process could be split into phases based on *block.timestamps* but it should also take into consideration Anon-Zether's epoch mechanism. A Dapp [47] could be implemented to host the auction process to use this protocol in practice.

As for making the auction process publicly verifiable, the process proposed can be implemented as a smart contract.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system. http://Bitcoin.org/Bitcoin.pdf, 2008. (Accessed 3 June 2022).
[2] S. Meiklejohn, M. Pomarole, G. Jordan, et al., A fistful of Bitcoins: characterizing payments among men with no names, Commun. ACM 59 (4) (2016) 86–93, https://doi.org/10.1145/2896384.
[3] E.B. Sasson, A. Chiesa, C. Garman, et al., Zerocash: decentralized anonymous payments from bitcoin, in: Proceedings of the 2014 IEEE Symposium on Security and Privacy, IEEE, 2014, pp. 459–474, https://doi.org/10.1109/SP.2014.36.
[4] Monero Means Money: private, decentralized cryptocurrency that keeps your finances confidential and secure, https://www.getmonero.org/. (Accessed: 31 May 2022).
[5] B.E. Diamond, Many-out-of-many proofs and applications to anonymous Zether, in: Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), IEEE, 2021, pp. 1800–1817, https://doi.org/10.1109/SP40001.2021.00026.
[6] H.S. Galal, A.M. Youssef, Verifiable sealed-bid auction on the Ethereum blockchain, in: A. Zohar, I. Eyal, V. Teague, et al. (Eds.), Financial Cryptography and Data Security, Springer, Berlin, Heidelberg, 2019, pp. 265–278, https://doi.org/10.1007/978-3-662-58820-8_18.
[7] D. Demirel, J. Lancrenon, How to Securely Prolong the Computational Bindingness of Pedersen Commitments, Cryptology ePrint Archive, 2015 preprint, https://eprint.iacr.org/2015/584.
[8] G. Sharma, D. Verstraeten, V. Saraswat, et al., Anonymous sealed-bid auction on Ethereum, Electronics 10 (19) (2021) 2340, https://doi.org/10.3390/electronics10192340.
[9] R.L. Rivest, A. Shamir, Y. Tauman, How to leak a secret: theory and applications of ring signatures, in: O. Goldreich, A.L. Rosenberg, A.L. Selman (Eds.), Theoretical Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 164–186, https://doi.org/10.1007/11685654_7.
[10] H. Li, W. Xue, A blockchain-based sealed-bid e-auction scheme with smart contract and zero-knowledge proof, Secur. Commun. Network. 2021 (2021) 5523394, https://doi.org/10.1155/2021/5523394.
[11] Z. Ye, C.-L. Chen, W. Weng, et al., An anonymous and fair auction system based on blockchain, J. Supercomput. 79 (2022) 13909–13951, https://doi.org/10.1007/s11227-023-05155-w.
[12] Q. Zhang, S. Cao, X. Zhang, Enabling auction-based cross- blockchain protocol for online anonymous payment, in: Proceedings of the 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2021, pp. 715–722, https://doi.org/10.1109/ICPADS53394.2021.00095.
[13] J. Xiong, Q. Wang, Anonymous Auction Protocol Based on Time-Released Encryption Atop Consortium Blockchain, arXiv, 2019, preprint. arXiv: 1903.03285.
[14] J. Eberhardt, S. Tai, Zokrates—scalable privacy-preserving off-chain computations, in: Proceedings of the 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2018, pp. 1084–1091, https://doi.org/10.1109/Cybermatics_2018.2018.00199.
[15] C++ library for zkSNARKs, https://github.com/scipr-lab/libsnark. (Accessed: 18 June 2022).
[16] Unspent transaction output. https://en.wikipedia.org/wiki/Unspent_transaction_output. (Accessed 19 June 2022).
[17] R.C. Merkle, A digital signature based on a conventional encryption function, in: C. Pomerance (Ed.), Advances in Cryptology — CRYPTO '87, Springer, Berlin, Heidelberg, 1988, pp. 369–378, https://doi.org/10.1007/3-540-48184-2_32.
[18] B. Bünz, S. Agrawal, M. Zamani, et al., Zether: towards privacy in a smart contract world, in: J. Bonneau, N. Heninger (Eds.), Financial Cryptography and Data Security, Springer, Cham, 2020, pp. 423–443, https://doi.org/10.1007/978-3-030-51280-4_23.
[19] I. Damgård, On Σ-protocols, 2, 2010. https://www.cs.au.dk/~ivan/Sigma.pdf. (Accessed 19 June 2022).
[20] B. Bünz, J. Bootle, D. Boneh, et al., Bulletproofs: short proofs for confidential transactions and more, in: Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, pp. 315–334, https://doi.org/10.1109/SP.2018.00020.
[21] D. Boneh, V. Shoup, A Graduate Course in Applied Cryptography, 2017. https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_4.pdf. (Accessed 19 June 2022).
[22] D. Boneh, The decision diffie-hellman problem, in: J.P. Buhler (Ed.), Algorithmic Number Theory, Springer, Berlin, Heidelberg, 1998, pp. 48–63, https://doi.org/10.1007/BFb0054851.
[23] A. Fiat, A. Shamir, How to prove yourself: practical solutions to identification and signature problems, in: A.M. Odlyzko (Ed.), Advances in Cryptology—CRYPTO' 86, Springer, Berlin, Heidelberg, 1987, pp. 186–194, https://doi.org/10.1007/3-540-47721-7_12.
[24] E. Morais, T. Koens, C. vanWijk, et al., A survey on zero knowledge range proofs and applications, SN Appl. Sci. 1 (2019) 946, https://doi.org/10.1007/s42452-019-0989-z.
[25] X. Sun, F.R. Yu, P. Zhang, et al., A survey on zero-knowledge proof in blockchain, IEEE Netw 35 (4) (2021) 198–205, https://doi.org/10.1109/MNET.011.2000473.
[26] Tornado.cash, https://tornado.cash/. (Accessed: 31 March 2022)..
[27] Confidential transactions on Ethereum, https://aztec-protocol.gitbook.io/aztec-documentation/guides/an-introduction-to-aztec/confidential-transactions-have-arrived. (Accessed: 5 April 2022).
[28] Zk-rollups – ethhub, https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/#simple-overview. (Accessed: 5 April 2022).
[29] N. Sullivan, A (Relatively Easy to Understand) Primer on Elliptic Curve Cryptography, 2013. https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/. (Accessed 13 March 2022).
[30] J.W. Bos, J.A. Halderman, N. Heninger, et al., Elliptic curve cryptography in practice, in: N. Christin, R. Safavi-Naini (Eds.), Financial Cryptography and Data Security, Springer, Berlin, Heidelberg, 2014, pp. 157–175, https://doi.org/10.1007/978-3-662-45472-5_11.
[31] V. Gayoso Martínez, L. Hernández Encinas, G. Sánchez Ávila, A survey of the elliptic curve integrated encryption scheme, J. Comput. Sci. Eng. 2 (2) (2010) 7–13.
[32] S. Saqan, Explanation of Bitcoin's Elliptic Curve Digital Signature Algorithm, 2022. https://suhailsaqan.medium.com/explanation-of-bitcoins-elliptic-curve-digital-signature-algorithm-6603f951863a. (Accessed 22 May 2022).
[33] R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM 21 (2) (1978) 120–126, https://doi.org/10.1145/359340.359342.
[34] M.O. Rabin, Probabilistic algorithm for testing primality, J. Number Theor. 12 (1) (1980) 128–138, https://doi.org/10.1016/0022-314X(80)90084-0.
[35] G.L. Miller, Riemann's hypothesis and tests for primality, J. Comput. Syst. Sci. 13 (3) (1975) 300–317, https://doi.org/10.1145/800116.803773.
[36] Carmichael function, https://en.wikipedia.org/wiki/Carmichael_function. (Accessed: 18 June 2022).
[37] T. Elgamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Trans. Inf. Theor. 31 (4) (1985) 469–472, https://doi.org/10.1109/TIT.1985.1057074.
[38] Precompiled Contracts and Confidential Assets, 2019. https://blog.qtum.org/precompiled-contracts-and-confidential-assets-55f2b47b231d. (Accessed 17 March 2022).
[39] B. Buchanan, Elgamal and Elliptic Curve Cryptography (ECC), 2019. https://medium.com/asecuritysite-when-bob-met-alice/elgamal-and-elliptic-curve-cryptography-ecc-8b72c3c3555e. (Accessed 3 April 2022).
[40] Consensys/anonymous-zether: A private payment system for Ethereum-based blockchains, with no trusted setup, https://github.com/ConsenSys/anonymous-zether. (Accessed: 1 March 2022)..
[41] Schnorr signature, https://en.wikipedia.org/wiki/Schnorr_signature. (Accessed: 22 March 2022).
[42] J. Groth, M. Kohlweiss, One-out-of-many proofs: or how to leak a secret and spend a coin, in: E. Oswald, M. Fischlin (Eds.), Advances in Cryptology - EUROCRYPT 2015, Springer, Berlin, Heidelberg, 2015, pp. 253–280, https://doi.org/10.1007/978-3-662-46803-6_9.
[43] I. Damgård, Commitment schemes and zero-knowledge protocols, in: EEF School 1998: Lectures on Data Security, Springer, Berlin, Heidelberg, 1999, pp. 63–86, https://doi.org/10.1007/3-540-48969-X_3.
[44] web3.js - Ethereum JavaScript API, https://web3js.readthedocs.io/en/v1.7.3/. (Accessed: 1 March 2022).
[45] Ganache. https://trufflesuite.com/ganache/. (Accessed 1 March 2022).
[46] 22 gwei — Ethereum gas tracker, https://etherscan.io/gastracker. (Accessed: 22 May 2022).
[47] Decentralized applications (dApps), https://ethereum.org/en/dapps/. (Accessed: 3 June 2022).