Towards trustworthy and privacy-preserving decentralized auctions

Tiphaine Henry, Julien Hatin, Eloi Besnard, Nassim Laga, Walid Gaaloul

Towards trustworthy and privacy-preserving decentralized auctions

Tiphaine Henry[1,2,3*], Julien Hatin[2*], Eloi Besnard[3], Nassim Laga[2] and Walid Gaaloul[3]

[1]Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France.
[2]Orange Innovation, Paris, France.
[3]Telecom SudParis, Institut Polytechnique de Paris, Paris, France.

*Corresponding author(s). E-mail(s): tiphaine.henry@cea.fr; julien.hatin@orange.com;
Contributing authors: eloi.besnard@telecom-sudparis.eu; nassim.laga@orange.com; walid.gaaloul@telecom-sudparis.eu;

Abstract

Blockchain smart-contracts can be used as service mappers, connecting a contractor with the service provider best fitting desired service requirements (e.g., price or quality of service). The allocation consists of comparing competitive bids using a smart-contract. However, in competitive environments, service providers may be reluctant to share sensitive information offers with the blockchain as it makes any transaction implicitly public. To reconcile data privacy imperatives with the benefits of blockchain, we propose to leverage fully homomorphic encryption (FHE) for blockchain-based sealed-bid auctions. More precisely (i) FHE enables the processing of bids without decrypting them, (ii) smart-contracts gather and orchestrate bids comparison, and (iii) a computation oracle carries on comparisons over ciphered data. Collusion attempts may occur between bidders and the computation oracle. To prevent this, we combine FHE with hybrid RSA/AES encryption to preserve the privacy of the onchain bid contents. Hence, our protocol prevents information leakage onchain and on the service providers' side during bids comparison. We validate this approach through an implemented prototype.

Keywords: Blockchain, Fully Homomorphic Encryption, Privacy, Auctions

1 Introduction

Decentralized collaborations have risen in recent years, mainly fostered by progress in information systems technologies. Companies manage shared processes to coordinate themselves efficiently [1, 2]. Open tenders are an example of such decentralized collaborations, where several competitors submit their tenders to the platform to win the service. As an illustration, in logistic supply chains, a dedicated allocation system maps delivery requests to the best available carrier (Fig. 1). The allocation system may leverage the quality of service ratings by aggregating metrics such as average delay or customer satisfaction. The business process management system can manage service metrics and accounting and compliance aspects. The platform will then connect the winning tender with the service requester and manage the service enactment and settlement.

Automation needs arise in such a context to reduce administrative costs related to candidates' selection, contracting, and payment stages. The following requirements arise to foster automation and the adoption of autonomous allocation systems. First, the allocation protocol should be trustworthy by ensuring bid integrity and comparing all submitted bids based on predefined metrics. Second, auditability of the protocol and decision-making should be accessible to all participants to ease claim resolution if they occur. Lastly, the protocol should offer bid privacy to limit information asymmetries and collusion risks between actors [3], as they may comprise differentiating criteria.

The blockchain has emerged as a new tool for trustworthy decentralized data storage and decision making [4, 5] with the potential for lower operational and transaction costs [6, 7]. The blockchain ledger stores transactions in a tamper-proof fashion and scripts referred to as smart-contracts execute predefined protocols autonomously. Oracles can feed the blockchain with reliable external data: the blockchain connects to several oracles to triangulate the status of transactions outside the blockchain system. Hence, when combined with oracle entities, smart-contracts can provide automation gains during the allocation and process management stage [8, 9]. Smart-contracts can secure the allocation and settlement of the allocation service in an efficient, finite, and autonomous way [6, 10]. Such scripts can take action regarding the bids evaluation, service completion, and payment without requiring a trusted third party. Such a system can even, for accounting and compliance purposes, (1) record all settled transactions in a tamper-proof fashion and (2) keep track of delivery information provided by different oracles and tracking sensors. This configuration ensures information transparency as the involved members can access past allocation history and the tamper-proof delivery information stored in the ledger.

Transaction logs stored on the ledger may reveal publicly sensitive data. Dealing with data privacy in a blockchain framework raises challenges regarding access control enforcement and processing of sensitive data [11–13]. Privacy-preserving methods can be used as a layer of protection. Zero-knowledge proof [14] has been used as a building block for ring signatures and mixing services. Additionally, encryption techniques such as homomorphic encryption

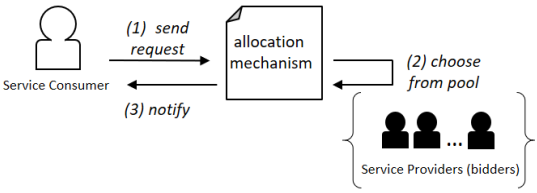


Fig. 1: Service allocation mechanism

have been used to carry on algebraic operations on ciphertexts [15–18]. Plaintexts are thus not disclosed to participants. A trusted third party is often a single point of attack or failure. This configuration may only be possible in some use cases, for it may result in concerns of collusion of the trusted third party with the other stakeholders.

Another approach combines secure multi-party computing, homomorphic encryption, and zero-knowledge proof to compare offers while preserving privacy. During a pre-processing stage, the candidates compare pairwise their offers in private channels using zero-knowledge proof mechanisms[19, 20]. Results of the comparison are then processed on-chain. This approach addresses power asymmetry issues. Nonetheless, candidates need to interact with one another, and communication costs may be prohibitive [21].

Despite several publications of blockchain-based allocation protocols to map services and their procurement, there needs to be more research on multi-objective service allocation in a privacy-preserving fashion [22]. Fully homomorphic encryption (FHE) preserves algebraic operations over ciphered data. Research is currently lacking regarding its potential for trustworthy privacy-preserving computations on-chain.

Our contributions. This study sets out to answer the following question: *(RQ) How to leverage blockchain as a trustworthy service allocation tool while preserving the confidentiality of offers?*

This paper introduces a new mechanism for trustworthy, decentralized, and privacy-preserving service allocation using ciphered multi-objective offers.

Fig. 2 depicts the system components and their interactions. Our solution leverages the blockchain to compare ciphered offers in a trustworthy fashion without needing trusted third parties. The blockchain ledger keeps track of all transactions and offers tamper-proof storage facilities for service agreements. Smart-contracts provide a reliable, decentralized, trustworthy allocation facility, managing the open tendering and the confidential service settlement. A blockchain oracle service compares ciphered offers on-chain on behalf of the smart contract for scalability reasons.

Fig. 2 presents the mechanism. Upon a service request (step 1), service providers request the RSA public key of the comparison oracle to the smart contract (steps 2-3). This key will be used to carry on hybrid RSA/AES encryption. This encryption layer prevents collusion between participants wishing to access other bid contents. A service provider authority generates a ciphering FHE key and forwards it to the bidders (steps 4-5). Service providers will publish their

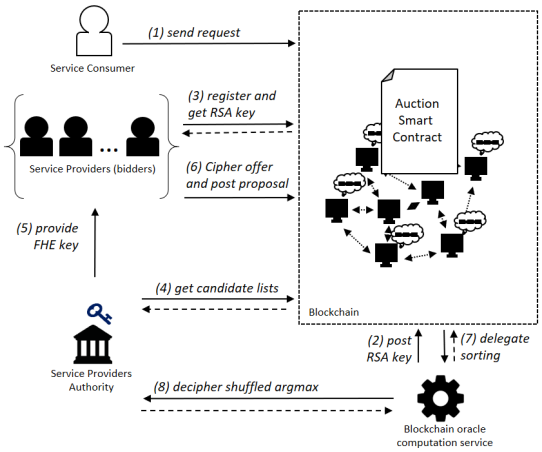


Fig. 2: Overview of the system

bids on-chain once ciphered with FHE and the hybrid RSA/AES ciphering layer (step 6). As mentioned, this hybrid encryption layer protects bid content from the service provider authority or other bidders. The latter have access to the FHE key and the data stored on-chain. The comparison oracle retrieves and compares the offers ciphered in FHE (step 7). The comparison output, the ciphered argmax, is shuffled and provided to the service provider authority for deciphering. Shuffling the argmax prevents the service providers' authority from gaining knowledge of the bidders' identity. There is no linkage between bidding requests posted on-chain and the ordering of the argmax. The service providers' authority finally hands out the winning bid position to the smart contract. The smart contract then enacts the binding using the offer linked to the argmax.

Our approach interest comes from using FHE techniques to carry on bid comparison in such an ecosystem: any calculation is theoretically accessible to compare offers while not needing trusted hardware having access to plaintext. Additionally, as in [23, 24], this approach ensures bidders' non-interactivity. It also offers a distributed authority because several entities manage the bid comparison (the service providers' authority, the blockchain, and the ciphered computation oracle). Finally, our approach differs from other literature by using an oracle to delegate FHE comparisons while preventing smart-contracts' rising transaction costs.

To summarize, we propose the following contributions:

- We propose a trustworthy mechanism for managing tendering leveraging blockchain smart contracts. We leverage FHE and RSA/AES ciphering in such blockchain context for privacy purposes.
- We leverage Fermat's Little Theorem introduced in [25, 26] for onchain comparison of FHE offers. To prevent collusion, we ensure a separation of powers in bidding and using FHE so that the comparison oracle does not know the actual value of the argmax.

- We demonstrate the feasibility of the mechanism using a proof-of-concept that maps services to service providers in a decentralized and privacy-preserving fashion.

We organize the remainder of this paper as follows. Section 2 introduces key concepts around blockchain and cryptography algorithms leveraged in the article. Section 3 reviews related work. Then, section 4 presents our motivating example. Section 5 provides an abstract description of the system's actors and trust relationships. In section 6, we describe the proposed mechanism. In section 7, we validate our approach by building and experimenting on a prototype. We discuss the results and conclude the paper with section 8.

2 Preliminaries

This section introduces key concepts about blockchain, smart-contracts, and oracles before introducing the two cryptography algorithms used in this paper, namely the hybrid RSA/AES encryption scheme, and FHE.

2.1 Blockchain, smart-contracts, oracles, and IPFS

The blockchain is a distributed ledger organized into linked blocks of transactions. A set of nodes (or users) hold a copy of the ledger and update it following a set of rules, a consensus protocol. A consensus protocol defines the protocol followed to verify and append new transactions to the chain. Proof-of-work and proof-of-stake are among the best-known consensus protocols. With proof-of-work, a puzzle needs to be solved by miners to append a new block and get a financial fee. With proof of stake, the node responsible for adding a new block of validated transactions has the most assets. These rules help avoid malicious nodes or invalid transactions. Merkle trees ensure the ledger's integrity by linking transactions using cryptography rules. Cryptography technologies such as Elliptic Curve Digital Signature Algorithm (ECDSA) provide each participant with a public address and a private key to sign transactions [27]. This scheme ensures participants' pseudo-anonymity. Anonymity is not guaranteed as the ledger links user public addresses to the transactions they are involved with. Blockchain networks can be permission-less (any node can join or leave the network) or permissionned (nodes must be approved to join the network; leaving the network may be prone to more difficulties).

Blockchains comprise executable scripts, the smart-contracts. These smart-contracts can store the states of defined variables and execute functions on-demand. They can run the business logic and manage business processes in an autonomous fashion [8]. Some smart-contract services depend on external data. To answer this need, oracle services bridge the closed blockchain network [28]. Oracle services repeatedly trigger the same API and triangulate the results to prevent malicious behavior on the API side. They forward the requested external data to the smart-contract services. An alternative use for oracles is to ask APIs to carry on heavy computations, otherwise too expensive to carry on

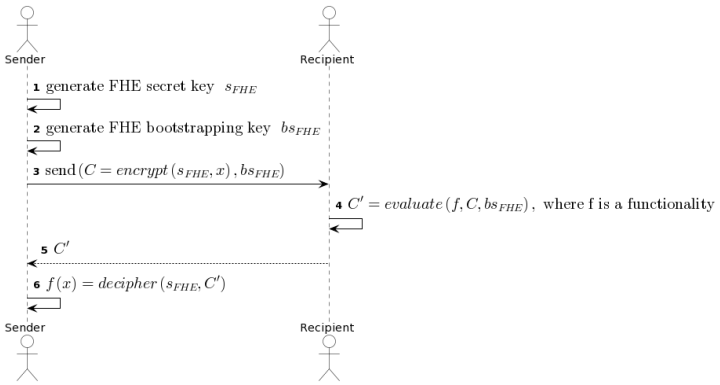


Fig. 3: FHE Scheme

directly on smart-contracts. Chainlink and Provable are examples of blockchain oracle solutions.

IPFS is an open peer-to-peer file-sharing network providing high throughput and low latency [29]. Two main building components are the distributed hash tables and a Merkle DAG. First, distributed hash tables provide high throughput and low latency for data distribution: nodes of the IPFS network can store and share data in a decentralized fashion. Second, Merkle DAG provides content addressing and tamper-resistance: data are identified uniquely and permanently stored with integrity. In the blockchain context, IPFS provides a reliable and low transaction cost storage capacity [30], which is useful for blockchains with proof-of-work consensus such as Ethereum, where storing data on smart-contracts is costly.

2.2 FHE and Hybrid RSA/AES encryption schemes

Homomorphic encryption, first proposed by Rivest et al. in 1978 [31] is a ciphering algorithm preserving algebraic operations: one can carry computations on encrypted data without requiring a decryption key. This encryption protocol often helps address the millionaire problem where two millionaires want to compare their wealth without disclosing the amount of money they own [32]. Building on this idea, several works defined partial homomorphic encryption algorithms where only one type of algebraic operation is possible (e.g., Paillier algorithm [33]).

The first fully homomorphic encryption scheme, aiming at evaluating any arbitrary function, has first been proposed by Craig Gentry in his dissertation [34]. Fig. 3 illustrates a high-level view of the FHE scheme. Let's consider a number x and the corresponding ciphertext c. Consider an algebraic operation f on x such that $f(x) = y$. In an FHE scheme, decrypting $f(c)$ will give us y. Hence, one can carry on an algebraic operation on a ciphertext with homomorphic encryption without having access to plaintext.

In more details, it consists into the evaluation of an arbitrary boolean circuit composed of binary gates (e.g., AND, OR, XOR, NAND, or NOR gates) over encrypted data without revealing any information on the data The encryption first consists of translating a plaintext into a binary. Afterward, the binary goes through a set of boolean circuits. Each circuit corresponds to an operation (addition, multiplication, etc.). Successive operations may trigger noise; hence the deciphered output of the computation may not be exact. Bootstrapping operations help reduce this noise [35].

The literature includes several iterations of FHE schemes building on Gentry's proposal. The most representative are BGV [36], FV [37], TFHE [38], and CKKS [39]. A set of FHE libraries, addressing the different FHE scheme generations, are available for research purposes. Examples of such libraries are HeLIB, TFHE, PALISADE, or FHEW. A set of compilers and accelerators are also available to facilitate the use of these tools.

Several technical limitations still hinder real-life uses, such as slow computation speed or accuracy problems, and a lack of common stack embodying features proposed in the various iterations (matrix multiplication, non-linear function evaluation, fast bootstrapping, etc). Research is currently being led to overcome these limitations, especially through the length of hardware acceleration. The survey of Marcolla et al [40] provides a more in-depth overview of FHE technics and challenges.

The RSA/AES hybrid encryption protocol, first proposed in [41], leverages RSA and AES. RSA (which is asymmetric) is computationally expensive, and cannot be applied on large data. Hence, the large data is ciphered with AES (which is symmetric). The secret AES key is ciphered with RSA and shared conjointly with the data.

3 Related work

Multi-party computation, alongside zero-knowledge-proof [42], can be used to conduct sealed-bid auctions. With zero-knowledge-proofs, a prover can demonstrate knowledge on a piece of information without leaking information directly to the verifier. In the context of sealed-bid auctions, the non-interactive zero-knowledge proof variant, which consists of one-way communication between the prover and the verifier (c.f. zk-SNARKs [43]), is often used by bidders to compare offers pairwise in private channels without revealing the content of the bids [19, 20, 44]. They reveal the result of each smart-contract comparison using evidence techniques with zero knowledge disclosure. The main limitation is that bidders need to interact with each other. Hence, issues may arise if one bidder is unwilling to participate, takes more time than needed, or if too many bidders need to interact. Additionally, the smart-contract can reconstruct the bids ordering (e.g., from the most to the least expensive if the auctions are on a price), reducing the bids' privacy.

In [45], a hybrid public/private blockchain scheme, combined with encryption techniques, is proposed to carry on privacy-preserving auctions. The public

blockchain gathers bids, and once the auction terminates, the auctioneer can access the content of the bids in the private blockchain. Such architecture answers the need for low auction costs and low latency. Nonetheless, the auctioneer must orchestrate and deploy the auction on the private chain. It may reduce the benefits of smart-contracts as a trustworthy and autonomous third party and reintroduces security downsides. Moreover, scalability is limited due to the public section of the mechanism [24].

Other approaches use a smart-contract to gather offers and compare them onchain using a trusted execution environment or enclave [17, 18, 23, 24, 46–50]. A smart-contract bridges the gap between customers and the enclave. Approaches using partial homomorphic encryption gather offers onchain. An enclave then deciphers offers offchain, as in [46]. In [24], a trusted execution environment computes allocations in a blockchain environment, using an oracle to track node preferences. However, in these approaches, the enclave has access to the content of offers, which can result in a single point of failure. Additionally, partial homomorphic encryption, used e.g., in [17, 18] does not allow the combination of different operations (addition, subtraction, multiplication, division), which is necessary to carry on typical aggregation strategies used to compute a multi-objective comparison of offers. Some papers also build on group signature schemes to ensure bidders anonymity [47, 49, 50]. Similarly, hardware-based trusted execution environments can identify the winning bid. However, in our cross-organizational setting where partners may collaborate for the remaining process activities, we do not wish to reach bidders full anonymity. Moreover, bids content are made accessible to the competitors. Hence, this technique does not meet our motivating example need.

In summary, carrying on privacy-preserving auctions on blockchain with multi-party-computing reintroduces bidders interactivity [23]. Additionally, the use of the hybrid public/private blockchain scheme in [45] weakens trustless governance due to the active role of the auctioneer. Furthermore, using trusted execution environments reintroduces a single point of failure as the enclave accesses the content of offers.

4 Motivating example

We introduce the main concepts and problem using a simple example that we revisit in the sequel of this paper. Freight exchange procurement platforms are an example of service allocation platforms. Shippers ask for a delivery service, and several carriers registered into the platform compete for the service. The carrier choice is paramount as the quality of the delivery directly impacts customer satisfaction and may have significant financial consequences.

Suppose four carriers, A, B, C, and D. They answer a service delivery request made by a shipper through a procurement platform. The shipper sets a maximum price threshold and a minimum capacity for offers to be eligible: the truck capacity should be at least $5m^3$, and the service price below $20\$/m^3/km$.

Additionally, if two or more bids are suitable, the offer must be allocated to the carrier with the optimal bid.

Table 1: Competing carriers offers

Carrier	Availability	Location	Price[1]	Capacity[1 2]
A	D1	C1	10\$/m^3/km	10
B	D1	C1	5\$/m^3/km	5
C	D2	C1	10\$/m^3/km	5
D	D2	C1	10\$/m^3/km	8

[1]Sensitive metrics that should remain private.

[2]Truck capacity is normalized to scale 10.

Table 1 presents the candidates' bids, with a normalized truck capacity. The underlying allocation mechanism and data processed should be trustworthy to discourage actors' collusion and data tampering. Additionally, sensitive information encapsulated into carriers' offers should remain confidential (i.e., not accessible to other carriers).

Our use case is a cross-organizational business process (i.e., involving two companies or more). In this setting, the business process management system, managing tasks shared between actors (such as resource allocation) should ensure (1) a distributed governance, (2) an open allocation system to avoid any power imbalance, (3) transactions auditability for trust purposes, and (4) the possibility to automate business process management tasks. Blockchain is a technology that offers, by design, these characteristics. Hence, blockchain is critical to answer the need for a trustless business process management system in our cross-organizational context.

Preserving bids privacy in the blockchain ledger arises as a requirement for the adoption of such system. Hence, we propose to leverage blockchain-based resource allocation schemes with algebraic operations computed on ciphered bids.

5 System context

This section gives an abstract description of the system's actors, their interactions, and their trust relationships.

5.1 Entities and data flows

- the bidders (also referred to as service providers)
- the service provider authority who has the FHE keys,
- the blockhain with (i) the ledger for a distributed non tamperable storage where bidders can collect information without talking to each other, (ii) the smart contract that acts as a trustworthy intermediary between bidders to

manage the tendering. It delegates the bids comparison to the comparison oracle.
- the comparison oracle, triggered by the smart contract, that does the comparison of the ciphertexts by delegation of the smart contract for scalability reasons, using FHE
- IPFS for distributed storage of the bids, used for blockchain scalability purposes. Each IPFS hash is stored onchain for non repudiability of the bids.

Our mechanism comprises the following data: (1) the FHE key, accessible by the service provider authority and bidders only; (2) the RSA public key, accessible to all as it is stored onchain, and its corresponding RSA private key accessible to the comparison oracle only, (3) bidders AES keys, that can be accessed by the comparison oracle, (4) bidders offers, all ciphered with (i) the same FHE key, and (ii) the unique AES key, which in turn is ciphered in RSA to prevent deciphering attempts, (5) and finally the ciphered argmax issued from the bids comparison.

5.2 Threat model

We define the actors' behaviors as follows:

- We suppose bidders are dishonest;
- We suppose the service provider authority and the computation oracle are honest but curious;

We now describe the possible collusions between actors:

- *All bidders colluding together:* Bidders have no interest in colluding together as they are competitors;
- *One bidder colluding with the service provider authority:* they will hold the content of the bidder offer, its AES key, and the FHE key. This collusion does not allow the access to other bidders contents; and hence has no interest;
- *One bidder colluding with the oracle:* The bidder brings the FHE key, and the oracle the offers of the other bidders. Hence they can access the offer contents of all bidders;
- *the service provider authority colluding with the oracle:* The bidder brings the FHE key and the oracle the other bidders' offers. Similarly, they can access all the offers of the bidders.

In the following, We hypothesise that the comparison oracle, and the service providers authority have an honest but curious behavior: they will not try to change the protocol, hence not collude with a bidder. In this context, we propose a trustless mechanism, as each entity only has access to no information or partial information regarding the tendering.

6 The approach

Notation. First we introduce the notations used in the following sections. We refer to $k\_$ as the ciphering key for the symmetric encryption of type _. We refer to $(s\_, p\_)$ as the couple of respectively secret and public keys for the asymmetric encryption algorithm of type _. Let x a natural number. $C\_$ denotes the ciphering function of type _; $c = C\_(x, p\_)$ denotes the ciphered version of x obtained after applying the ciphering protocol of type _; D_ denotes the ciphering function of type _, and $x = D(c, s\_)$. For symmetric encryption, $k\_$ denotes both ciphering and deciphering. We define a service provider offer O as a vector of offers, where $O[i]$ is an array comprising sensitive metrics to evaluate. We refer to O_{enc} as the array where each element of O has been ciphered with the public key of the encryption algorithm of type _: with $i \in [0, size(O) - 1]$, $O_{enc}[i]$=$C(O[i], p\_)$. We refer to L as the list of ciphered aggregated offers.

6.1 Overall approach

We now provide an overall view of the approach before diving into the details of each stage. First, we initialize the ciphering keys for the AES, RSA, and FHE algorithms. To preserve the privacy of sensitive offers during the allocation protocol, we separate the management of the encryption keys between five stakeholders. The second stage consists of the smart-contract gathering the ciphered sensitive information of eligible candidates and the RSA-ciphered AES key. The third stage consists of deciding on the best offer: the oracle receives the ciphered keys and bids, deciphers the key, then uses the deciphered key to access FHE-ciphered offers. As a final stage, the oracle computes the best offer by comparing the FHE-ciphered data. The protocol sets the final binding, and service management can begin[51]. We describe each stage in more detail hereinafter.

The privacy-preserving allocation smart-contract holds a list of registered service requests. Each registered service request comprises the blockchain addresses of the service requester and allocated service provider (null at first). It also comprises the open tendering and the service settling status. The open tendering comprises the list of submitted bids, each attached to its issuer blockchain address, and the open tendering status. The latter can be open: the smart-contract accepts new bids. It can be pending: the smart-contract no longer accepts bids and proceeds to the bids comparison. It can finally be closed: the comparison has terminated, and the service is allocated. In this case, the blockchain address of the service provider is populated with the address of the winning bid issuer.

6.2 Key initialization

In our approach, we leverage three encryption algorithms: FHE, AES, and RSA. FHE is used to compare offers that are encrypted without the need to reveal any information on the offers. The service provider authority and service providers hold the same FHE key. Hence, they can decipher bids stored onchain. An

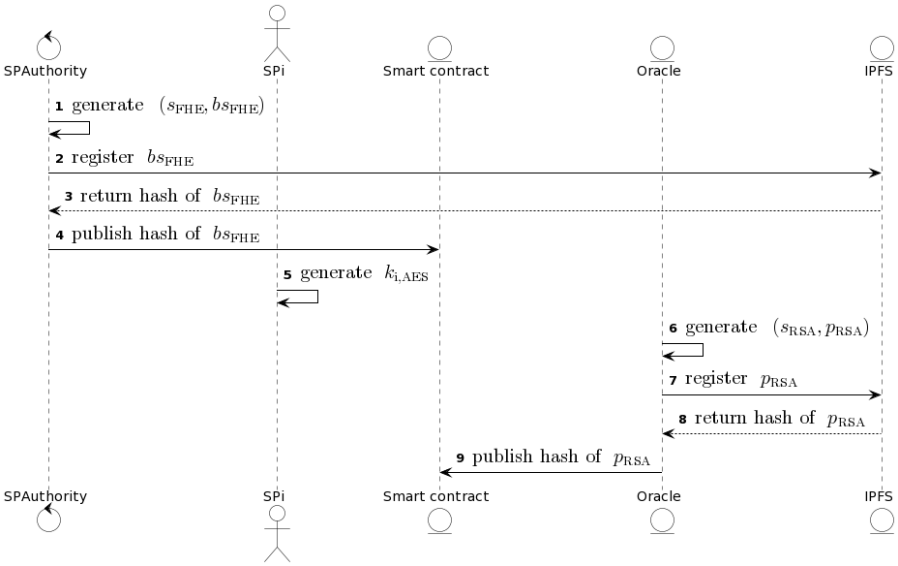


Fig. 4: Initialization of cipher keys (SP=service provider)

asymmetric encryption layer is thus necessary to preserve offers confidentiality. A standard asymmetric encryption algorithm is RSA. Nonetheless, the FHE ciphertext length is too large for RSA, but not for AES. As AES is symmetric, the issue of confidentiality rises again, and it must be combined with an asymmetric scheme. To do so, we use the hybrid AES/RSA encryption scheme presented in [41]. Each bidder generates its own AES key and ciphers the FHE offer with the AES encryption. Hence only bidders can decipher their own offers. Meanwhile, bidders apply the RSA encryption on their AES key. Hence only the RSA key holder (the comparison oracle) can access the AES key.

Fig. 4 presents the sequence diagram of the initialization of the three RSA, AES, and FHE ciphering keys. In this approach, three separate entities manage the RSA, AES, and FHE key generation to avoid any deciphering attempt by one of the entities.

- The service providers' authority generates the FHE key k_{FHE} (step 1) that will be forwarded to the service providers once the auction time finishes. The service providers will use this key to cipher their offers.
- Each service provider generates its own AES encryption key (steps 2).
- The oracle generates the pair of private and public RSA keys s_{RSA} and p_{RSA} used to cipher and decipher messages with the RSA algorithm (step 3). The private key s_{RSA} remains secret for the oracle to retrieve offers ciphered with the RSA algorithm. The public key p_{RSA} is saved in IPFS to reduce storage costs in the smart-contract (step 4). The IPFS hash used to retrieve the public key on the IPFS network is forwarded to the oracle (step 5). The oracle then publishes the hash of p_{RSA} onchain (step 6).

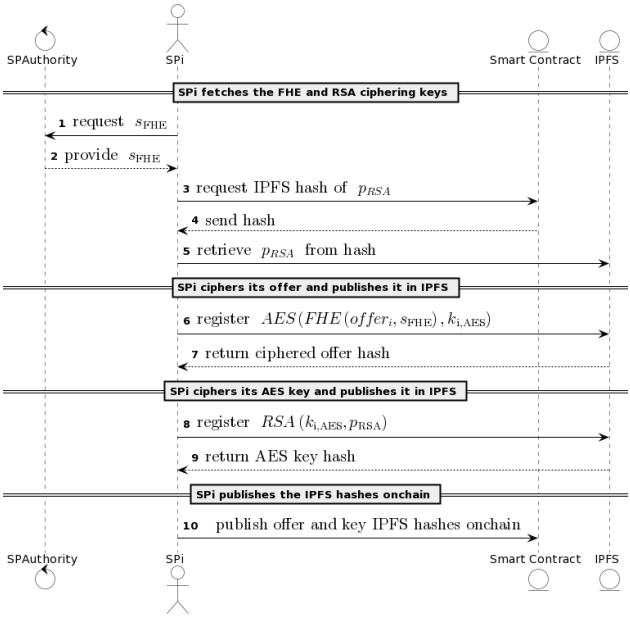


Fig. 5: Ciphering and gathering offers

Let's consider the delivering procurement allocation of our motivating example. The carriers' authority generates k_{FHE}. The four bidders A, B, C, and D of our motivating example all subscribe to the smart-contract to participate in the bid. They each generate a private AES encryption key k_{AES}. The oracle generates the keyset $\{s_{RSA}, p_{RSA}\}$ and publishes the public RSA key to IPFS. It then publishes the hash of p_{RSA} to the smart-contract.

6.3 Forwarding FHE-ciphered offers to the smart-contract

Fig. 5 then depicts the ciphered offers comparisons stage.

In a first stage, each service provider SPi fetches the FHE and RSA ciphering keys. Each candidate requests the FHE key to the service providers' authority (step 1). The service providers' authority forwards the FHE key to each candidate via private channels (step 2). Each candidate SPi then triggers the smart-contract to retrieve the IPFS hash of p_{RSA} (step 3). The smart-contract returns the hash to SPA (step 4). SPi connects to IPFS to retrieve p_{RSA} (step 5). In a second stage, each service provider ciphers its offer and publishes it in IPFS. SPi ciphers her offer O_i twice using FHE encryption first and then AES encryption. She applies the following ciphering algorithms: $C_{AES}(C_{FHE}(O_i, k_{FHE}), k_{i,AES})$ using her personal AES key $k_{i,AES}$. She saves it to IPFS (step 6). The corresponding hash is forwarded back to SPi (step 7). In a third stage, SPi also ciphers $k_{A,AES}$ using the RSA algorithm and

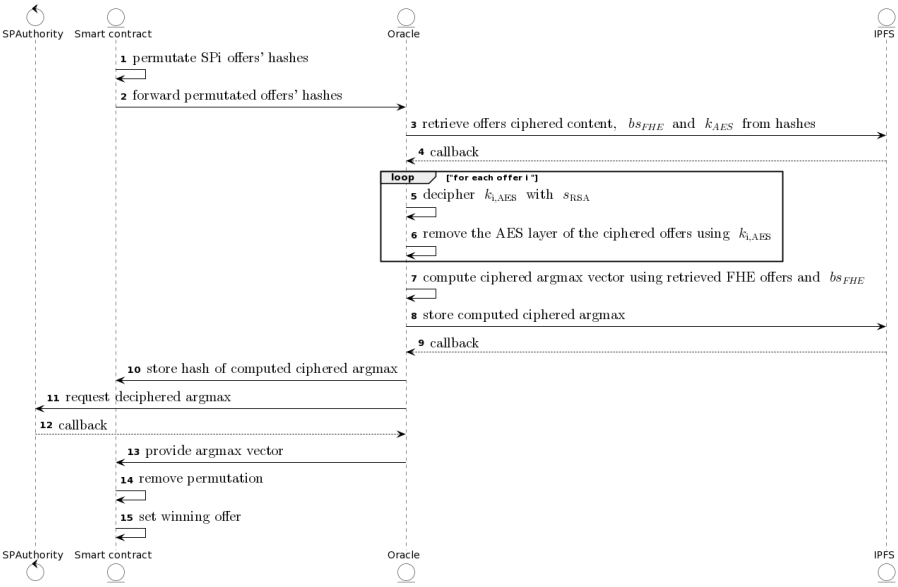


Fig. 6: Ciphered comparison and allocation

forwards it to IPFS (step 8). The corresponding hash is forwarded back to SPi (step 9). SPi finally publishes the hashes of her ciphered offer and key to the smart-contract (step 10).

In our motivating example, delivering driver A computes her ciphered AES key is expressed as $C_{RSA}(k_{A,AES}, p_{RSA})$. She also processes her ciphered offer as follows: $O_{A_{enc}} = \begin{bmatrix} C_{AES}(C_{FHE}(price, k_{FHE}), k_{A,AES}) \\ C_{AES}(C_{FHE}(capacity, k_{FHE}), k_{A,AES}) \end{bmatrix}$.

6.4 Service allocation

Once all offers are forwarded, the smart-contract delegates the offers' comparison to the oracle. Delegation occurs to avoid intensive computations carried on the blockchain network. Indeed, the more calculation on the blockchain, the more expensive the transaction's time and transaction fees.

Through this mechanism, we propose to compare ciphered offers directly. Such mechanism has been missing in the literature, as oftentimes the oracle has directly access to clear offers to realize the comparison, hence privacy leakages could occur. What is more, to the best of our knowledge, there is a research gap regarding the argmax computation of an array of elements ciphered in FHE.

Shuffling and forwarding ciphered offers

The smart-contract interchanges the randomly received ciphered offers (Fig.6, step 1). By so doing, we prevent an honest but curious oracle behavior that could lead to an information leakage on the order of candidates' submissions and

hence on the winning offer. The smart-contract then forwards the interchanged offers to the oracle for an argmax computation (Fig.6, step 2).

To do so, the smart-contract sends an API request to the oracle, specifying the ID of the bid comparison to analyze. Meanwhile, the smart-contract sends an event comprising the list of hashes of the ciphered offers and AES keys.

In the following, we refer to the interchanged offers as π_A and π_B.

In our motivating example, the smart-contract interchanges the received ciphered offers $[O_{A_{enc}}, O_{B_{enc}}]$ randomly and forwards the interchanged offers to the oracle for an argmax computation.

Retrieving FHE offers and computing the mean

We now propose to leverage oracle services to carry on the offers comparison. The oracle first retrieves the list of ciphered offers and keys hashes specified in the comparison request event. It then connects to IPFS to fetch the offers and keys from the hashes (Fig.6, step 3-4). For each offer i, the oracle uses the private key s_{RSA} to decipher $k_{i,AES}$ (Fig.6, step 5). It then uses each AES key to process the corresponding FHE offers (Fig.6, step 6): it will first decipher the offers to retrieve the FHE offers and then compute each offer's ciphered mean.

In our motivating example, the oracle deciphers the RSA layer of the AES key of A. It will then use this AES key to remove the AES encryption layer of π_A. The oracle then computes the mean for the offer submitted by A: it obtains $avg(\pi_A) = (C_{FHE}(capacityA, k_{FHE} + C_{FHE}(priceA, k_{FHE})/2$. The computation of $avg(\pi_A)$ gives $C_{FHE}(8, k_{FHE})$. The computation of $avg(\pi_B)$ gives $C_{FHE}(5.5, k_{FHE})$.

As a side note, we chose a simple average as a possible use. Alternative computations can be carried on to aggregate offer metrics. For example, Pareto optima could be computed, or weighted mean with weighting factors defined in the smart-contract in the service request.

Comparing offers

An oracle service now carries the pairwise comparison of the processed offers (Fig.6, step 7).

Finding the argmax of a set of two or more ciphered numbers is challenging, as the result of the maximum between two ciphered numbers is also ciphered [52–54]. It cannot be reused to be compared with a third number without deciphering it due to noise addition during ciphering and comparison [54].

Algorithm 1 circumvents this issue using pairwise comparisons between offers [54], and equality testing using the Fermat little theorem, which has been already applied for FHE in the context of cloud environments [25, 26]. We leverage these approaches here in a blockchain context.

B will store the argmax of the ciphered offers (Algorithm 1, line 2). A pairwise comparison is launched on L (Algorithm 1, line 3-11). More precisely, the maximum m between two elements $L[i]$ and $L[j]$ is assessed (Algorithm 1, line 6). An equality assessment between $L[i]$ and m is then determined (Algorithm 1, line 7): if $L[i]$ is maximum, the function *testEquality* returns a number c_n

Algorithm 1 Ciphered numbers comparison

Input: L the list of ciphered aggregated offers, c_1 and c_n ciphered 1 and n, and p a prime number large enough to fulfill the condition $p > max(L)$.
Output: B the binary argmax vector

```
1:  procedure cipherCompare(O)
2:      var B ← []
3:      for i ∈ [0, length(L)] do
4:          for j ∈ [0, length(L)] do
5:              if i != j then
6:                  var m ← c_max(L[i], L[j])
7:                  var b ← testEquality(L[i], m, c_1, c_n, p)
8:                  B[i] ← B[i] + b
9:              end if
10:         end for
11:     end for
12:     return B
13: end procedure
```

where $n > 1$. Otherwise, *testEquality* returns c_0. We increment the argmax $B[i]$ with the output of each comparison (Algorithm 1, line 8). Once all pairwise comparisons are performed, B comprises the ciphered argmax of L. The argmax will be the index with the maximum value.

In our motivating example, the forwarded ciphered offers are [π_A, π_B]. The respective indexes of π_A and π_B are 0 and 1. We suppose n=10. The oracle computes the comparison for [$mean(\pi_A)$, $mean(\pi_B)$] and [$mean(\pi_B)$,$mean(\pi_A)$]. The argmax of the comparisons is B=[c_0,c_{10}].

Deciphering the argmax and retrieving the best offer

After the argmax computation, the oracle asks the service provider authority to decipher the argmax (Fig. 6, step 8). The service provider authority uses its FHE key to decrypt the argmax. It then forwards the deciphered argmax to the oracle (Fig. 6, step 9). The oracle then transfers the argmax vector to summarize the pairwise comparisons as an array to the smart-contract (Fig. 6, step 10). The smart-contract reverts the shuffling applied in step 1 on the received array (Fig. 6, step 11). The smart-contract sets the winning offer (Fig. 6, step 12).

If there is only one maximum, the winner is the service provider whose index is mapped to the array's maximum. If there are several offers, some mechanism to break the tie can be implemented. We use the time of submission stored in the blockchain ledger to break the tie (c.f., Fig. 5, step 10).

The smart-contract emits an event to notify service providers of the output of the auction and sets the blockchain address of the service provider with the address of the winning bid issuer.

In our motivating example, the oracle sends the ciphered argmax to the carriers' authority for deciphering. The carrier authority deciphers the argmax

and obtains L_{dec}=[10,0]. She forwards L_{dec} to the oracle, which sends L_{dec} to the smart-contract. The smart-contract reverts the shuffling to the original order on the deciphered argmax: it finds that B is the winning offer. The smart-contract sets the blockchain address of the service provider with the address of the winning bid issuer. It also sends a notification to A to inform her that her bid has not been successful.

In summary, we have combined three ciphering algorithms to carry on a privacy-preserving comparison of offers. This bidding stage can be followed by smart contract-based payment, and business process management to a larger extent to benefit from the blockchain characteristics. The use of FHE instead of partial homomorphic encryption enables the comparison of multi-objective offers. Meanwhile, the hybrid RSA/AES algorithm enables a confidential transfer of FHE-ciphered offers in the blockchain. The content of offers is tamper-proof and cannot be read by other competitors as they do not hold the AES key. The only participant able to decipher the offers is the comparison oracle, that holds the RSA secret key. Moreover, the content of offers remains confidential, even for the comparison oracle, as it is ciphered with FHE, and can only be deciphered by FHE key holders.

The bidding protocol could be extended to other confidential metrics stored onchain. For example, truck volume should be considered as packages should fit inside the different type of vehicles. Additionally, a fleet of vehicles could be proposed to reach low CO_2 emissions leveraging electric trucks and bikes with carts. Hence, several algebraic operations would leverage carriers data. During service allocation, the smart contract would filter trucks based on the available volume (dividing the truck volume by package unitary volume), or compute CO_2 emissions based on the fleet of trucks proposed. We could also imagine a price based on an hourly rate. Hence, at payment time, the smart contract should compute the hourly rate price by number of hours for payment. Hence, fully homomorphic encryption makes it possible to compute ciphered data during the process life-cycle.

Smart-contract provides trust into the allocation system by acting as a trustworthy intermediary to collect offers. The hashes of the keys and offers are tamper-proof, and hence ensure an objective comparison of the offers, while limiting collusion risks between different participants. We also ensure public auditability as the following information is available on the ledger: IPFS hashes containing ciphered offers, bidders participation, and the winning result blockchain address.

7 Implementation and evaluation

This section aims at validating our approach experimentally. We build and evaluate a decentralized privacy-preserving resource-binding protocol for delivery driving services.

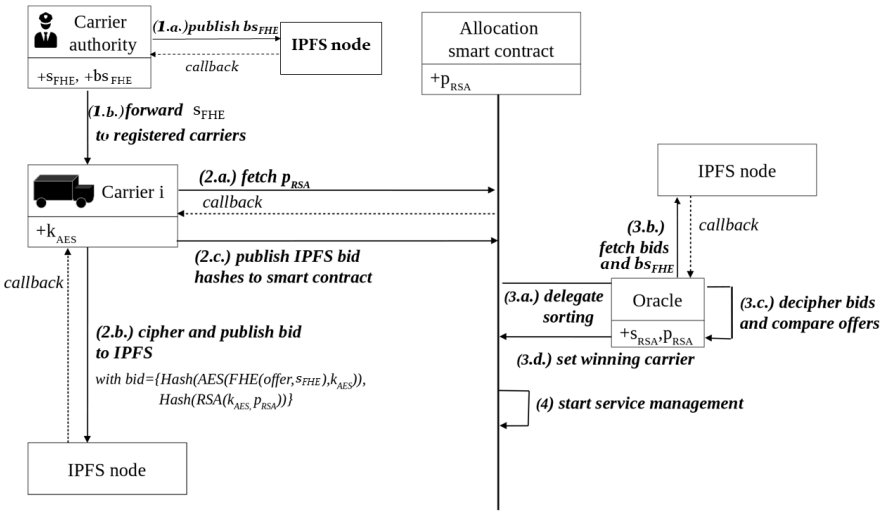


Fig. 7: Protocol and key holders

7.1 Implementation

To demonstrate the approach's feasibility, we build a C++ API that gathers ciphered offers hashes and retrieves IPFS hashes and compares ciphered offers using FHE. To do so, we also leverage the TFHE C++ library [38] that implements FHE, as well as the Cryptopp C++ encryption library[1] to provide RSA and AES ciphering facilities. The C++ API holds the following functions: FHE, RSA, and AES key generation, registration in IPFS, offers registration that populates a JSON file comprising offers holding the hash of the AES+FHE ciphered offer, and the ciphered AES public key, and offer comparison using the content of the provided JSON file. Code is accessible here: https://github.com/tiphainehenry/fhe_oracle. We leverage the C++ API to implement the proposed mechanism in the Ethereum network. We deploy our smart-contract on a local test network to assess the approach. We use Infura[2] as our API gateway to IPFS. By this mean, we can store the ciphered offers and keys into IPFS, and recover them using IPFS hashes. The comparison API is managed with a personal computer with an Intel i5 core CPU, 4GB of RAM.

Our prototype comprises two parts. The first part demonstrates the offers registration part of the approach using Ethereum and Infura. It covers RSA, AES, and FHE key generation. To mimic real-life behavior, we initially generate FHE and RSA key pairs to mimic the key creation stage and provide private keys to the service providers' authority and oracle, respectively. We publish the RSA public key to the smart-contract to simulate the initial oracle behavior, and transfer the FHE key to carriers (Fig. 7, step 1.a.). We also store the bootstrapping key in IPFS (Fig. 7, step 1.b.). Two elements motivated the use of

[1] https://cryptopp.com/
[2] https://infura.io/docs/ipfs

a public blockchain for this experimental design setup: (1) We consider a use case where carriers operate in a uberised market. (2) Ethereum is oftentimes used in the literature, and these results can be used for future baseline comparisons.

Our prototype covers bidders' local ciphering of offers: each bidder asks the RSA key to the smart-contract (Fig. 7, step 2.a.), generates the AES key pair locally, then ciphers the offer using FHE and AES private keys, and the AES public key using the RSA public key. Bidders then publish the offers hash into IPFS using Infura (Fig. 7, step 2.b.). They then register their offer in the smart-contract (Fig. 7, step 2.c.). The smart-contract takes care of the registration of competing offers. For bidders to interact with the smart-contract, the smart-contract implements the following functionalities: bid initialization, offer registration, and comparison launch. As a side note, the size of the RSA key is set to 1024 bit, AES and TFHE key sizes follow the library standards (128 bit each).

The second part of the mechanism demonstrates the comparison part of the approach, using a toy array of ciphered offers stored in IPFS. It mimics the smart-contract, the oracle, and the service providers' authority interactions on the reception of a JSON file populated with ciphered submissions. The API receives the IPFS hashes of ciphered offers and fetches the data (Fig. 7, step 3.b.) and the bootstrapping key. Then, it proceeds to bids deciphering and comparison following the strategy presented in section 6.4 (Fig. 7, step 3.c.). The winning bidder is allocated to the service request (Fig. 7, step 3.d.), and the service management and later on settlement can proceed (Fig. 7, step 4)

7.2 Evaluation

We evaluate the protocol using three experiments: the ciphering file sizes, the smart-contract transaction costs, and the ciphered comparison processing time.

7.2.1 Ciphering key and file sizes

Table 2: Size of files generated during the protocol (1 Mbit = 125 KB). Acronyms: IS= information system

Context	Participant	Storage location	Data type	File size
Initialization	FHE key auth.	Competitors' IS	k_{FHE}	109MB
	FHE key auth.	Competitors' IS	FHE parameters	418B
	Tender Initiator	IPFS	p_{RSA}	160B
	Tender Initiator	Oracle	s_{RSA}	634B
Bid ciphering	Carrier	Carrier	k_{AES}	128B
	Carrier	IPFS	IV	16B
	Carrier	IPFS	$C_{AES}(C_{FHE}(offer))$	40KB

Table 2 gathers the size of the files generated during the execution of the motivating example scenario. The largest file is the FHE key file k_{FHE} (109MB). The size depends on the initial parameters used to generate the FHE key, that impact the multiplicative depth chosen to carry on bootstrapping operations, and noise reduction [38, 55]. In our approach, the size of the FHE key is not a bottleneck, as it is forwarded by the service providers' authority to the bidders in private channels. The files to be stored on IPFS are (1) the RSA public key, (2) offers ciphered content, and (3) offers IV. The AES key file and IV file sizes are 128 and 16B. The ciphered offer file is larger with 40KB. It is to note that the length of ciphertext is around a hundred times the plaintext size. Nonetheless, these file sizes range in the file sizes accepted to be stored in IPFS (100MB per request with Infura at the time of writing).

7.2.2 Smart-contract execution time

Table 3: Smart-contract transaction costs.

Context	Participant	ETH Tx. fee
RSA Key IPFS hash storage	Tender Initiator	0.00265454ETH
Offer creation	Tender Initiator	0.00350124ETH
Offer registration IPFS hash storage	Competitor[1]	0.00417978ETH
Comparison request	Tender Initiator	0.152ETH

[1]Average transaction fees for competitors registration, where competitors offers are described in the motivating example (see Section 4)

Table 3 presents the transaction costs required to perform smart-contract executions on the Ethereum blockchain. At the time of writing, 1ETH=2842.09€. The transaction costs needed to store the RSA key on the blockchain are worth 0.00265454ETH (7.54€). It is worth 0.00350124ETH (9.95€) for the offer creation. The average cost paid by each competitor to register the IPFS hash of the ciphered offer is worth 0.00488606ETH (13.89€). Finally, the most expansive transaction consists of the comparison request performed by the tender initiator when the auction terminates. It is worth 0.152ETH (432€). This transaction fee does not depend on the number of offers to be compared, as offer content is sent via an event to the oracle, and event triggers do not require transaction fees. Instead, price is fixed by the oracle, here Provable, and derives from the need to compensate the oracle for its computing power.

7.2.3 Ciphered comparison time

Finally, we investigate the time taken to compare integers ciphered following the proposed encryption approach. To do so, we generate random numbers between 0 and 9. We cipher each using FHE and then launch the comparison algorithm.

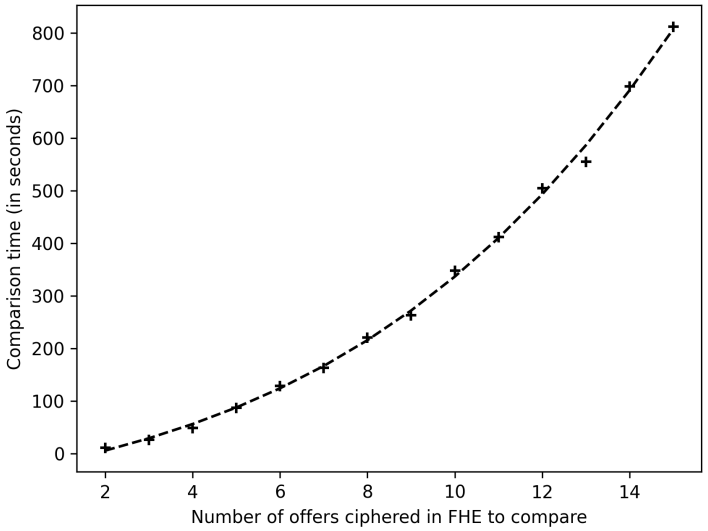


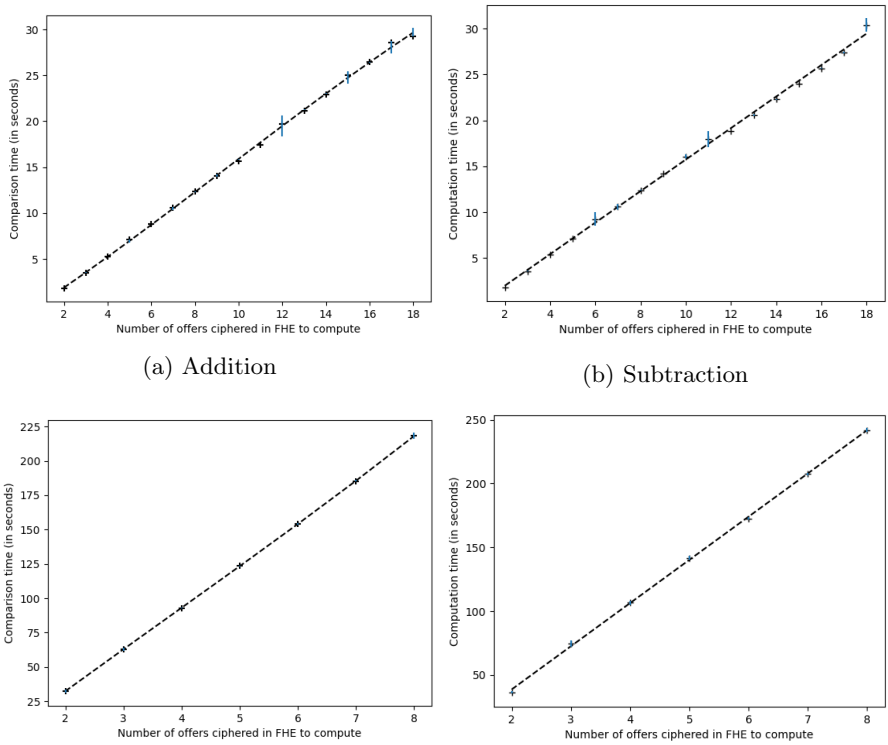
Fig. 8: Comparison time according to the number of ciphered FHE offers submitted

We measure the time needed to perform the pairwise FHE comparison. Fig. 8 depicts the comparison times recorded for arrays of ciphered offers of increasing size. The time necessary to perform a comparison depends on the number of offers at stake. The dotted line in the figure represents a polynomial of degree 3 and illustrates the complexity in x^3 of our algorithm.

7.2.4 Extending the computation oracle with basic operations

We enrich the experiments section by investigating the time taken to perform the following basic operation requests on ciphered data: addition, subtraction, multiplication, and division. Indeed, as mentioned earlier, these operations could be leveraged for enriching the bidding protocol (e.g., by computing the truck volume necessary for a given package), or afterwards, e.g., to compute the total service price based on an hourly rate. We thus investigate the computation time taken to perform each operation based on the number of inputs. For each number of input tested, we compute five times the time taken to perform the computation, and derive the mean and standard deviation. Figure 9 present the results.

Fig. 9a presents the time taken to perform addition on an increasing number of inputs, and Fig. 9b the time taken to perform subtraction. For both operations, we test between two and twenty input numbers, and record the processing time. The time taken to perform additions or subtractions grows linearly with the number of inputs. Adding two inputs take 1.89s, whereas adding twenty inputs take 33s. Similarly, two inputs take 1.81s to be subtracted,



(a) Addition

(b) Subtraction

(c) Multiplication

(d) Division

Fig. 9: Computation time according to the number of ciphered numbers submitted

whereas twenty inputs take 34s. These computation time results are equivalent as ciphered numbers go through equivalent circuits for addition or subtraction.

Fig. 9c presents the time taken to perform multiplication on an increasing number of inputs, ranging until eight input numbers. Two inputs take 32s to be multiplied, whereas eight inputs take 220s. This is significantly longer than the addition or subtraction results. This difference could be explained by the circuit complexity. Multiplication circuits consist of repeated addition circuits. Hence, the computation time to perform a multiplication comprises several addition computation times. Additionally, our algorithm loops over the maximum number of bits used to describe an input: we do not optimize the computation time based on the input bit size.

Fig. 9d presents the time taken to perform divisions on an increasing number of inputs, also ranging until eight inputs numbers. The result is always the integer part. Two inputs take 36s to be divided, whereas eight inputs take 241s (i.e., 4m01s). Like in the multiplication circuit, the size of the inputs are not taken into account.

In summary, these operations remain costly regarding computation time, as they grow linearly with the number of inputs. To lower this computation time, several solutions could be considered such as the parallelization of the computation over shared memory (OpenMP), distributed memory (OpenMPI) or using GPUs (CUDA).

7.2.5 Threat model analysis

Our goal is to prove that bidders cannot view the content of other bids, and that the comparison oracle and the service provider authority cannot access bid contents. We propose the following threat model analysis:

- **Bidders cannot access other bidders values** as (i) the bids are encrypted between each bidder and (ii) the comparison oracle uses hybrid encryption (RSA + AES).
- **The comparison oracle cannot access bidders value, nor the offers argmax content**. Indeed, the comparison oracle can only partially decrypt the bids, which are still encrypted in FHE. Hence the comparison oracle cannot access bids value. Additionally, it can only compute a ciphered argmax to identify the winning bid. As it does not hold the FHE key, it is not possible either to access the argmax.
- **Service providers authority can only know whether one or several best offers exist.** The service provider authority cannot access bids contents as FHE-ciphered offers are encapsulated with a RSA/AES ciphering layer. Moreover, the smart contracts forwards shuffled offers to the comparison oracle, that forwards to the service provider authority a ciphered argmax to decipher. Hence, the deciphered argmax is also shuffled by extension: the service provider authority cannot know who sent what, only that one or several best offers have been found.

We ensure a separation of powers in bidding and using FHE so that the comparison oracle does not know the actual value of the argmax. The two authorities, the comparison oracle, and the service provider authority, are trusted not to collude. However, even if they would collude, the oracle would only gain access to the information that one or several best offers have been found.

By using blockchain, we provide a public auditability of the tendering as the following information is available on the ledger: IPFS hashes containing ciphered offers, bidders participation, and the winning result blockchain address.

As a side note, an incentive mechanism shall be included in future work to discourage the oracle from colluding with a bidder.

8 Discussion and conclusion

This paper proposes a mechanism to balance trust and privacy in a cross-organizational system. To do so, we leverage the blockchain as a business process management system as this technology offers by design (1) a distributed

governance, (2) an open allocation system to avoid any power imbalance, (3) transactions auditability for trust purposes, and (4) the possibility to automate business process management tasks. We propose to carry on trustworthy auctions using blockchain smart contracts while preserving the confidentiality of sensitive data values in bids.

Our solution comprises a computation oracle and a service provider authority managing encryption keys. Smart-contracts manage autonomously and transparently encryption keys attribution, offers registration, and bids allocation. This distributed authority preserves bids privacy and bidders non interactivity. The pseudo-anonymity of blockchain users also ensures bidders' privacy requirements. The comparison of offers, managed by a comparison oracle, occurs on FHE-ciphered data. As service providers and the service provider authority both hold the FHE ciphering key, they could decipher the offers stored onchain. We propose a mechanism leveraging hybrid RSA/AES encryption to prevent this. In our mechanism, privacy is ensured as: (1) bidders cannot decipher other offers content due to the RSA/AES ciphering, (2) the comparison oracle cannot decipher the offers contents as offers are ciphered in FHE, and (3) the service provider authority cannot decipher offers as it does not possess AES keys, nor retrieve information regarding winning bids as the argmax comes shuffled. Additionally, we ensure public auditability of the sealed-bid auction as the following information is available on the ledger: IPFS hashes containing ciphered offers, bidders participation, and the winning bidder blockchain address.

We build a sealed-bid allocation prototype to demonstrate the feasibility of the approach. Experiments show the approach's feasibility, though scalability and latency issues arise. The transaction costs required to launch a comparison request depend on the oracle chosen, and using Provable may be prohibitive to perform a ciphered comparison. The choice of public versus private blockchain depends on the use case, and we will expand this study for private blockchain settings in future work. Moreover, the change in consensus of Ethereum and alternative designs such as Ethereum rollups may address the high cost and scalability limitations of public blockchains. Future work will focus on these alternative designs.

Additionally, the experiments show that the protocol requires tuning FHE parameters for noise addition. Such tuning can impact FHE key, ciphered offers size, and comparison time. It is to note that the FHE ecosystem is still maturing, and the integration of FHE to the Solidity ecosystem shall be more accessible in the years to come, thanks to better language interfaces and libraries.

Our approach assumes that the service providers' authority does not collude with the comparison API. Nonetheless, collusion risks may arise in our system, as oracles and the service provider authority bring centralization to the system. First, using oracles implies a re-centralization and requires trust in the output. Triangulation methods from several oracles could partially answer the risk of data tampering from a malicious node. Additionally, a collusion risk may exist between the service providers' authority managing FHE keys and the computation oracle. The oracle possesses FHE offers, and the service providers'

authority has the decryption key. Hence, removing interactions between competitors comes at the expense of a centralized authority responsible for both key issuance and argmax deciphering. We aim to limit information leakage using offer permutation on the smart-contract size to prevent linking offer content to the pseudonymous identities of participants.

Moreover, public verifiability implies providing access to any blockchain member (1) the legitimacy of the bidder, (2) the validity of the offer, and (3) the accuracy of the auction issue. Bidder legitimacy and offers validity are ensured with IPFS hashes containing ciphered offers, and bids onchain registration. The public verifiability of the auction issue is a limitation of our approach, as we leverage a trusted execution environment referred to as our computation oracle. Nonetheless, ensuring bids privacy (and in a broader scope computing sensitive data) is necessary in an industrial context. It should be enforced both onchain and in the computation oracle environment. Hence minimum verifiability comes as a consequence of this requirement. To mitigate this issue, the selection algorithm proposed should not be centralized, but implemented in several servers.

Another limitation of this approach concerns the latency of the comparison request of ciphered offers. The computation time follows the comparison algorithm complexity of x^3, induced by the ciphered maximum computation of arrays: the comparison time is directly proportional to the cube to the number of offers compared. The running time per gate seems to be a limiting factor for FHE schemes, reducing our solution's practical use for more complex operations. Ongoing research in the field of FHE may help solve this latency issue [56].

We implement the comparison asynchronously, using smart-contract events, as oracle requests have a maximum callback time often exceeded with the FHE comparison. Hence, this approach does not apply to time-dependent applications, especially if many offers are at stake.

Finally, scalability issues arise. The random access memory of the machine limits the number of comparisons. In our testing configuration, with a personal computer with an Intel i5 core CPU, 4GB of RAM, comparison works with a maximum of 15 offers. Additionally, public IPFS storage can be limited if offer sizes are consequent with multiple metrics. To circumvent this issue, one could use a private or permissioned IPFS channel to store offers in a decentralized fashion.

Our approach interest comes from FHE techniques for the bids comparison in such an ecosystem: any calculation is theoretically accessible to compare offers while not needing trusted hardware having access to plaintext. Hence, FHE encryption provides more possibilities for more complex computations: allocation protocols can evolve dynamically based on new allocation rules without causing data privacy leakage, as data is ciphered.

An avenue for future work is to work on the formal verification of the proposed algorithms. We also plan to leverage FHE for privacy-preserving business process management systems, especially for activities involving data aggregation of sensitive metrics. Other protocol variants such as hybrid encryption

for service provider authority are also to be studied. Finally, another exciting avenue is ensuring payment stage privacy at settlement time. Indeed, a payment in cryptocurrency reveals the hidden content in the privacy-preserving auction. Nonetheless, the winning service provider may not be willing to reveal the content of its bid to competitors. To circumvent this issue, future approaches may use oracle banks, single-use tokens linked to a random value, and ciphered payment facilities such as Zether [57].

References

[1] Bermbach, D., Maghsudi, S., Hasenburg, J., Pfandzelter, T.: Towards auction-based function placement in serverless fog platforms. In: 2020 IEEE International Conference on Fog Computing (ICFC), pp. 25–31 (2020). IEEE

[2] Zhang, Y., Lee, C., Niyato, D., Wang, P.: Auction approaches for resource allocation in wireless systems: A survey. IEEE Communications Surveys Tutorials **15**(3), 1020–1041 (2013). https://doi.org/10.1109/SURV.2012.110112.00125

[3] Alvarez, R., Nojoumian, M.: Comprehensive survey on privacy-preserving protocols for sealed-bid auctions. Computers & Security **88**, 101502 (2020)

[4] Wood, G., *et al.*: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)

[5] Mendling, J., Weber, I., Aalst, W.V.D., Brocke, J.V., Cabanillas, C., Daniel, F., Debois, S., Ciccio, C.D., Dumas, M., Dustdar, S., *et al.*: Blockchains for business process management-challenges and opportunities. ACM Transactions on Management Information Systems (TMIS) **9**(1), 1–16 (2018)

[6] Pan, S., Trentesaux, D., McFarlane, D., Montreuil, B., Ballot, E., Huang, G.Q.: Digital interoperability in logistics and supply chain management: state-of-the-art and research avenues towards physical internet. Computers in Industry **128**, 103435 (2021)

[7] Saripalli, S.H.: Transforming government banking by leveraging the potential of blockchain technology. Journal of Banking and Financial Technology **5**(2), 135–142 (2021)

[8] Mendling et al., J.: Blockchains for business process management - challenges and opportunities. ACM Transactions on Management Information Systems **9**(1), 1–16 (2018). https://doi.org/10.1145/3183367. Accessed 2019-10-07

[9] Henry, T., Laga, N., Hatin, J., Beck, R., Gaaloul, W.: Hire me fairly:

towards dynamic resource-binding with smart contracts. In: 2021 IEEE International Conference on Services Computing (SCC), pp. 407–412 (2021). IEEE

[10] Oranburg, S., Palagashvili, L.: The gig economy, smart contracts, and disruption of traditional work arrangements. Smart Contracts, and Disruption of Traditional Work Arrangements (October 22, 2018) (2018)

[11] Pintado, O.L.: Challenges of blockchain-based collaborative business processes: An overview of the caterpillar system. Blockchain and Robotic Process Automation, 31–42 (2021)

[12] Dasgupta, D., Shrein, J.M., Gupta, K.D.: A survey of blockchain from security perspective. Journal of Banking and Financial Technology **3**(1), 1–17 (2019)

[13] De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Livraga, G., Paraboschi, S., Samarati, P.: Distributed query evaluation over encrypted data. In: IFIP Annual Conference on Data and Applications Security and Privacy, pp. 96–114 (2021). Springer

[14] Feng, Q., He, D., Zeadally, S., Khan, M.K., Kumar, N.: A survey on privacy protection in blockchain system. Journal of Network and Computer Applications **126**, 45–58 (2019)

[15] Du, M., Chen, Q., Xiao, J., Yang, H., Ma, X.: Supply chain finance innovation using blockchain. IEEE Transactions on Engineering Management **67**(4), 1045–1058 (2020)

[16] Tso, R., Liu, Z.-Y., Hsiao, J.-H.: Distributed e-voting and e-bidding systems based on smart contract. Electronics **8**(4), 422 (2019)

[17] 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 (2018). Springer

[18] Baranwal, P.R.: Blockchain based full privacy preserving public procurement. In: International Conference on Blockchain, pp. 3–17 (2020). Springer

[19] Blass, E.-O., Kerschbaum, F.: Strain: A secure auction for blockchains. In: European Symposium on Research in Computer Security, pp. 87–110 (2018). Springer

[20] Ma, J., Qi, B., Lv, K.: Fully private auctions for the highest bid. In: Proceedings of the ACM Turing Celebration Conference-China, pp. 1–6 (2019)

[21] Zhou, J., Feng, Y., Wang, Z., Guo, D.: Using secure multi-party computation to protect privacy on a permissioned blockchain. Sensors **21**(4) (2021). https://doi.org/10.3390/s21041540

[22] Mammadzada, K., Iqbal, M., Milani, F., García-Bañuelos, L., Matulevičius, R.: Blockchain oracles: A framework for blockchain-based applications. In: International Conference on Business Process Management, pp. 19–34 (2020). Springer

[23] Sonnino, A., Król, M., Tasiopoulos, A.G., Psaras, I.: Asterisk: Auction-based shared economy resolution system for blockchain. arXiv preprint arXiv:1901.07824 (2019)

[24] Keizer, N.V., Ascigil, O., Psaras, I., Pavlou, G.: Flock: Fast, lightweight, and scalable allocation for decentralized services on blockchain. In: 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–9 (2021). IEEE

[25] Xiang, G., Cui, Z.: The algebra homomorphic encryption scheme based on fermat's little theorem. In: 2012 International Conference on Communication Systems and Network Technologies, pp. 978–981 (2012). IEEE

[26] Tan, B.H.M., Lee, H.T., Wang, H., Ren, S., Aung, K.M.M.: Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. IEEE Transactions on Dependable and Secure Computing **18**(6), 2861–2874 (2020)

[27] Xiong, H., Jin, C., Alazab, M., Yeh, K.-H., Wang, H., Gadekallu, T.R., Wang, W., Su, C.: On the design of blockchain-based ecdsa with fault-tolerant batch verification protocol for blockchain-enabled iomt. IEEE journal of biomedical and health informatics **26**(5), 1977–1986 (2021)

[28] Al-Breiki, H., Rehman, M.H.U., Salah, K., Svetinovic, D.: Trustworthy blockchain oracles: review, comparison, and open research challenges. IEEE Access **8**, 85675–85685 (2020)

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

[30] Huang, H., Lin, J., Zheng, B., Zheng, Z., Bian, J.: When blockchain meets distributed file systems: An overview, challenges, and open issues. IEEE Access **8**, 50574–50586 (2020)

[31] Rivest, R.L., Adleman, L., Dertouzos, M.L., *et al.*: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978)

[32] Lin, H.-Y., Tzeng, W.-G.: An efficient solution to the millionaires' problem based on homomorphic encryption. In: International Conference on Applied Cryptography and Network Security, pp. 456–466 (2005). Springer

[33] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 223–238 (1999). Springer

[34] Gentry, C.: A Fully Homomorphic Encryption Scheme. Stanford university, ??? (2009)

[35] Chen, H., Han, K.: Homomorphic lower digits removal and improved fhe bootstrapping. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 315–337 (2018). Springer

[36] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)

[37] Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)

[38] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)

[39] Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 360–384 (2018). Springer

[40] Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F.H., Aaraj, N.: Survey on fully homomorphic encryption, theory and applications (2022)

[41] Mahalle, V.S., Shahade, A.K.: Enhancing the data security in cloud by implementing hybrid (rsa & aes) encryption algorithm. In: 2014 International Conference on Power, Automation and Communication (INPAC), pp. 146–149 (2014). IEEE

[42] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on computing **18**(1), 186–208 (1989)

[43] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14), pp. 781–796 (2014)

[44] Wang, D., Zhao, J., Wang, Y.: A survey on privacy protection of blockchain: The technology and application. IEEE Access **8**, 108766–108781 (2020)

[45] Desai, H., Kantarcioglu, M., Kagal, L.: A hybrid blockchain architecture for privacy-enabled and accountable auctions. In: 2019 IEEE International Conference on Blockchain (Blockchain), pp. 34–43 (2019). IEEE

[46] Galal, H.S., Youssef, A.M.: Trustee: full privacy preserving vickrey auction on top of ethereum. In: International Conference on Financial Cryptography and Data Security, pp. 190–207 (2019). Springer

[47] Enkhtaivan, B., Takenouchi, T., Sako, K.: A fair anonymous auction scheme utilizing trusted hardware and blockchain. In: 2019 17th International Conference on Privacy, Security and Trust (PST), pp. 1–5 (2019). IEEE

[48] 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)

[49] Devidas, S., Rao YV, S., Rekha, N.R.: A decentralized group signature scheme for privacy protection in a blockchain. International Journal of Applied Mathematics and Computer Science **31**(2) (2021)

[50] Zhang, S., Pu, M., Wang, B., Dong, B.: A privacy protection scheme of microgrid direct electricity transaction based on consortium blockchain and continuous double auction. IEEE access **7**, 151746–151753 (2019)

[51] Henry, T., Brahem, A., Laga, N., Hatin, J., Gaaloul, W., Benatallah, B.: Trustworthy cross-organizational collaborations with hybrid on/off-chain declarative choreographies. In: International Conference on Service-Oriented Computing, pp. 81–96 (2021). Springer

[52] Bourse, F., Sanders, O., Traoré, J.: Improved secure integer comparison via homomorphic encryption. In: Cryptographers' Track at the RSA Conference, pp. 391–416 (2020). Springer

[53] Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: NDSS, vol. 4324, p. 4325 (2015)

[54] Togan, M., Pleşca, C.: Comparison-based computations over fully homomorphic encrypted data. In: 2014 10th International Conference on Communications (COMM), pp. 1–6 (2014). IEEE

[55] Bonnoron, G.: A journey towards practical fully homomorphic encryption. PhD thesis, Ecole nationale supérieure Mines-Télécom Atlantique (2018)

[56] Chatterjee, A., Sengupta, I.: Sorting of fully homomorphic encrypted cloud data: Can partitioning be effective? IEEE Transactions on Services Computing **13**(3), 545–558 (2017)

[57] Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security, pp. 423–443 (2020). Springer